Security Review Report — Revision 3

Keylay

Protocol Implementation Review: Cryptographic Design, Transport Security, Relay Trust Claims, and Annotated Code Map

Subject:	Keylay Encrypted QR Relay — index.html, server.js
Security page:	security.html (keylay.org/security)
Audit scope:	Cryptographic correctness, protocol claims, relay trust model, server hardening
Methodology:	Manual source review, claim-vs-code differential analysis
Original review:	April 2, 2026
This revision:	April 27, 2026
Changes since v2:	Six open items resolved; two documentation discrepancies corrected; security.html updated
Status:	Internal review — not a formal third-party security audit

Revision 3 closes all remaining open findings. Four operational gaps resolved: server rate limiting and payload cap (F-02), session expiry (F-06), client pre-handshake buffer cap, and truncated-IP logging. Two documentation discrepancies identified and corrected: peer-key-change behavior description (D-04) and CSP claim scope (D-05). security.html updated to match the deployed code on both points. No open findings remain.

Table of Contents

1. Executive Summary

This revision reviews the Keylay encrypted QR relay at version 0.7, incorporating all changes applied since Revision 2 (April 2, 2026). All six previously open findings have been resolved and verified. No open findings remain.

Four previously open operational findings were closed through code changes: server rate limiting and payload cap (F-02), server session expiry (F-06), client pre-handshake buffer cap, and truncated-IP logging.

Two documentation discrepancies were identified during this review — cases where security.html described the implementation inaccurately — and both have been corrected. D-04 identified that the peer-key-change behavior described in security.html no longer matched the deployed code: the page described a session reset that the code no longer performs. D-05 identified that the CSP confirmation overstated protection: the page claimed XSS no longer had access to in-memory session state, which was false while script-src 'unsafe-inline' remained present. Both claims have been rewritten in security.html to accurately describe what the current implementation does and does not protect.

The core cryptographic protocol — HMAC-authenticated handshake, ephemeral X25519 key exchange, HKDF session key derivation, AES-256-GCM message encryption, and counter-bound replay protection — remains sound against the deployed code.

2. Scope and Methodology

File	Role	Lines
`index.html`	Client application — all cryptographic logic, UI, WebSocket client	~12,896
`server.js`	WebSocket relay — session routing, role management, rate limiting	331
`security.html`	Published security claims page — basis for claim comparison	~545

This review traces every specific, testable claim in security.html to the corresponding code path and evaluates each for accuracy. The server is audited independently. This revision re-examines all previously-open locations to confirm fixes, audits the new server.js rate-limiting and session-timer logic, and compares the current handleHelloMessage() behavior against the security page description.

3. Protocol Architecture

Keylay routes coordination data (PSBTs, public keys, UR sequences) between two peers through a WebSocket relay. The relay is kept blind: it routes ciphertext and sees only a derived channel identifier, never the raw session code or plaintext payload.

1	**Code entry**	User enters or generates a session code. The generator uses a 31-character unambiguous alphabet (no 0/O/1/I/l) at 10 characters, with rejection sampling (threshold 248). Effective entropy: log2(31^10) ~ 49.84 bits. Manual entry accepted up to 32 characters.
2	**Channel derivation**	SHA-256(code) is computed; the first 32 hex characters (128 bits) are used as the WebSocket channel name. The relay sees this hash, not the raw code.
3	**HMAC key derivation**	The raw code is stretched via PBKDF2-SHA256, salt 'keylay-v1-hmac', 300,000 iterations, producing an HMAC-SHA256 signing key that authenticates hello messages.
4	**Ephemeral X25519 key generation**	An ephemeral X25519 key pair is generated per session via crypto.subtle.generateKey. Private key is set to null after handshake.
5	**Signed public-key exchange**	Each peer HMAC-signs its base64-encoded ephemeral public key and sends {type:'hello', pubkey, sig}. The relay forwards it. The receiver verifies the HMAC before accepting the key.
6	**HKDF session key derivation**	Both peers compute X25519(myPrivate, theirPublic). Shared bits are imported as HKDF key material. HKDF info encodes 'keylay-v1-session|' + sorted public key pair. No key material transits the wire.
7	**AES-GCM message encryption**	Each message is encrypted with a fresh 12-byte random IV and AAD 'keylay-v1|' + counter. The receiver enforces strictly monotonic counter ordering; counter is advanced only after successful decryption.
8	**Handshake state machine**	States: IDLE -> HELLO_SENT -> ACTIVE. Concurrency lock prevents simultaneous hello processing. While ACTIVE, a hello carrying a different peer public key is rejected — active session continues. Legitimate reconnects require a page reload.

4. Verified Security Claims

Each security property described in security.html was traced to a specific code path and evaluated for correctness against the current source. All claims pass.

Status	Claim	Finding
PASS	Relay sees only ciphertext and channel hash, not plaintext	Confirmed. Payload is AES-GCM ciphertext; channel name is SHA-256(code)[:32].
PASS	Relay MITM resistance without code knowledge	Confirmed. HMAC over ephemeral public key; forgery requires PBKDF2 derivation from code.
PASS	Forward secrecy: past sessions not decryptable after code disclosure	Confirmed. Session key derived from ephemeral X25519; private key nulled post-handshake.
PASS	No session key transmitted	Confirmed. Both peers derive AES key locally from shared DH bits via HKDF.
PASS	HKDF info binds key to sorted public key pair	Confirmed. info = "keylay-v1-session|" + [myPub,theirPub].sort().join("|").
PASS	AES-GCM with 12-byte random IV per message	Confirmed. crypto.getRandomValues(new Uint8Array(12)) per encrypt() call.
PASS	Counter-bound AAD prevents replay; counter enforced on receipt	Confirmed. AAD = 'keylay-v1|'+counter; recvCounter advanced only after successful decrypt.
PASS	Unencrypted data messages rejected	Confirmed. msg.encrypted === false causes immediate rejection.
PASS	Unexpected peer key while ACTIVE is rejected; session continues	Confirmed. Incoming hello with changed public key is discarded; active session key and counters preserved. security.html updated (D-04, resolved).
PASS	Session limited to exactly two peers	Confirmed. server.js refuses third connection and closes socket immediately.
PASS	Ephemeral private key discarded after handshake	Confirmed. ephemeralKeyPair = null immediately after deriveSessionKey().
PASS	HMAC verification uses Web Crypto constant-time verify	Confirmed. crypto.subtle.verify() used — not a string comparison.
PASS	No custody of wallet private keys	Confirmed. App handles only coordination data. No signing occurs.
PASS	CSP restricts data exfiltration; inline script limitation disclosed	Confirmed. connect-src limited to 'self' + wss://app.keylay.org. security.html updated to describe scope accurately and disclose 'unsafe-inline' (D-05, resolved).
PASS	No full IP addresses logged	Confirmed. truncateIp() reduces to /16 (IPv4) or /32 hextet (IPv6) before any log() call.
PASS	Server enforces rate and size limits	Confirmed. maxPayload 256 KB; token bucket burst 5/1 per sec; per-IP cap 10; global cap 100.

NOTE	Code knowledge treated as session membership	Correct as stated. Any party with the code can join — by design.

5. Discrepancies: Security Page vs. Deployed Code

All discrepancies identified across Revisions 1, 2, and 3 are shown here for the record. Each was a case where security.html described the implementation differently from how the deployed code actually behaved. All five have been corrected.

RESOLVED D-01 · PBKDF2 Iteration Count (Revision 1 -> 2)

Was: Page claimed the current deployment uses 100,000 iterations; target V1 is 300,000.

Now: Code used 300,000. Security page updated.

RESOLVED D-02 · Replay Protection Status (Revision 1 -> 2)

Was: Page claimed "Replay protection is not yet deployed."

Now: Counter-bound AAD and strict monotonic counter enforcement were already deployed. Security page updated.

RESOLVED D-03 · Hello Message Wire Format (Revision 1 -> 2)

Was: Page claimed hello was carried inside a data envelope with a JSON-stringified payload.

Now: Code sends type:"hello" messages directly with pubkey and sig fields. Security page updated.

New discrepancies identified and resolved in Revision 3:

RESOLVED D-04 · Peer-Key-Change Behavior (Revision 3)

Was: security.html stated: "If a peer reconnects with a different ephemeral public key while a session is active, the session key is cleared, counters are reset, and a full handshake is forced before data flows again."

Deployed code (lines 12318-12323):

```
if (handshakeState === 'ACTIVE' && theirPublicKeyB64 !== peerPublicKeyB64) { // Ignoring
hello — possible rogue joiner return; }
```

The change was deliberate: the prior reset-on-key-change behavior was a denial-of-service vector — any peer knowing the code could force continuous session resets. Silent ignore closes that vector. Legitimate reconnects require a page reload.

Now: security.html updated to accurately describe the current behavior: unexpected peer keys while ACTIVE are rejected and the active session continues unchanged.

RESOLVED D-05 · CSP Claim Overstated XSS Protection (Revision 3)

Was: security.html stated: "A successful XSS injection no longer has unrestricted access to the DOM, session key variables, and decrypted payloads." With script-src 'unsafe-inline' present, an injected inline script can still execute and read in-memory cryptographic state. The claim was false.

Deployed CSP (index.html line 22):

```
script-src 'self' 'unsafe-inline'
```

What the CSP correctly restricts: connect-src limited to 'self' and wss://app.keylay.org; object-src, base-uri, and form-action all 'none'. An injected script cannot exfiltrate to an arbitrary external host — but it can read in-memory state.

Now: security.html updated to describe what the CSP restricts (exfiltration paths and resource loading) and to disclose the 'unsafe-inline' limitation.

6. Resolved Findings

All findings from Revisions 1, 2, and 3 are shown here, verified against the corresponding code path.

RESOLVED F-01 · Counter Advanced Before Decryption Verification (Revision 1 -> 2)

The client advanced its replay counter before decryption completed. A relay injecting a message with a fabricated counter value could consume the counter slot without a valid payload, causing subsequent legitimate messages to be rejected.

Verified fix:

```
// recvCounter now advanced only after successful decrypt: text = await
decrypt(msg.payload, sessionKey, msg.counter); recvCounter = msg.counter; // advanced
only on success
```

RESOLVED F-02 · Server: No Rate Limiting or Message Size Enforcement (Revision 2 -> 3)

The relay server imposed no restrictions on connection rate, message frequency, or payload size. The client-side messageBuffer also had no size cap, creating a memory exhaustion vector.

Verified fix:

```
const MAX_PAYLOAD_BYTES = 256 * 1024; const RATE_BUCKET_CAPACITY = 5; const
RATE_REFILL_PER_SEC = 1; const MAX_CONNECTIONS_PER_IP = 10; const MAX_GLOBAL_CONNECTIONS
= 100; // client: const MAX_BUFFERED_MESSAGES = 32;
```

RESOLVED F-03 · Server: Full Message Logging Including Channel Hash (Revision 1 -> 2)

The relay server was logging the channel hash and full message objects, creating a session-correlation record in server logs.

Verified fix:

```
// Log output now limited to type and peer count: log('msg type=' + message.type);
log('routed ' + (format||'data') + ' to ' + forwarded + ' peer(s)');
```

RESOLVED F-05 · No Content Security Policy (Revision 2 -> 3)

The page set no Content-Security-Policy header or meta tag. Added in v0.7 with connect-src limited to 'self' and wss://app.keylay.org, object-src 'none', base-uri 'none', form-action 'none'. Scope of protection accurately described in security.html (see D-05).

Verified fix:

```
// index.html line 22 (abbreviated): default-src 'none'; script-src 'self'
'unsafe-inline'; connect-src 'self' wss://app.keylay.org; object-src 'none'; base-uri
'none'; form-action 'none';
```

RESOLVED F-06 · No Server-Side Session Expiry (Revision 2 -> 3)

Sessions persisted indefinitely; no idle timeout or maximum lifetime was enforced.

Verified fix:

```
const SESSION_IDLE_MS = 15 * 60 * 1000; // 15-min idle const SESSION_MAX_MS = 60 * 60 * 1000; // 60-min hard cap // client mirrors both: SESSION_LIMIT_MS, IDLE_TIMEOUT_MS
```

RESOLVED F-07 · Modular Bias in Code Generator (Revision 1 -> 2)

The 31-character alphabet does not divide evenly into 256, causing the first 8 characters to appear slightly more often. The practical entropy difference was 0.11 bits.

Verified fix:

```
const threshold = 256 - (256 % alphabetLen); // 248 if (arr[i] < threshold) { code += alphabet.charAt(arr[i] % alphabetLen); }
```

RESOLVED IP logging · Full IP Addresses in Server Logs (Revision 2 -> 3)

Server logs contained full IP addresses. truncateIp() now reduces every address to its /16 prefix before any log call. Full IPs remain in-memory for the per-IP rate cap only.

Verified fix:

```
if (ip.startsWith('::ffff:')) ip = ip.slice(7); const v4 = ip.match(/^(\d+)\.(\d+)\.\d+\.\d+$/); if (v4) return `${v4[1]}.${v4[2]}`;
```

7. Open Findings

No open findings remain in the current implementation.

The following are hardening gaps — areas for future improvement, not bugs. Both are correctly disclosed in security.html.

PBKDF2 is not memory-hard: PBKDF2-SHA256 at 300,000 iterations raises brute-force cost but is GPU-parallelizable. Argon2id or scrypt would be more resistant to offline attacks. Upgrading the KDF is tracked in PLANNED_FEATURES.md.

Inline scripts not hash-pinned under CSP: 'unsafe-inline' in script-src limits XSS defense to exfiltration prevention rather than full execution prevention. Replacing it with sha256 hash entries requires a build step.

Session limit is policy, not cryptographic: A compromised relay could bypass the two-participant limit. No in-protocol mechanism cryptographically prevents a third party from joining if the relay permits it.

meta-tag CSP does not support frame-ancestors: frame-ancestors 'none' requires an HTTP response header. Setting it at the static-hosting layer is recommended to prevent clickjacking.

8. Server-Side Security Review

Session Isolation

Sessions are keyed by channel hash in a server-side Map. Third connections are refused and the socket closed immediately. Cross-session isolation relies on 128-bit channel identifier collision resistance.

Rate Limiting and Connection Caps

Five controls: (1) maxPayload 256 KB at the protocol layer; (2) per-connection token bucket (burst 5, refill 1/sec); (3) per-IP cap of 10; (4) global cap of 100; (5) liveness ping every 30 seconds. PONG_TIMEOUT_MS (60 s) is documented but enforcement latency is effectively up to ~30 s — no practical security consequence.

Role Assignment and Transfer

The first client becomes sender. Any peer can send {type:'claim'} to take the sender role. This is within the stated threat model and disclosed in security.html.

Session Lifetime

Idle timeout (15 min) bumped on each data or hello message. Max-lifetime timer (60 min) fires unconditionally. Both call closeSession(). Client mirrors both timers and wipes all key material via leaveChannel().

Logging

Output limited to message type and peer count. Full IPs never written. Channel hashes, public keys, HMAC signatures, and payload content do not appear in logs. ISO-8601 timestamped.

Input Validation

Malformed JSON is caught and discarded. Payload size enforced at the frame layer via maxPayload. String fields are not independently bounded but the frame cap limits their size.

9. Positive Findings

Web Crypto API exclusively	All cryptographic operations use crypto.subtle. No third-party primitives in the protocol path.
Ephemeral private key discarded post-handshake	ephemeralKeyPair = null immediately after deriveSessionKey(). Forward secrecy correctly implemented.
Constant-time HMAC verification	crypto.subtle.verify() — no timing side-channel.
Counter-bound AAD with correct ordering	recvCounter advanced only after successful AES-GCM decryption.
Uniform code generation	Rejection sampling (threshold 248) ensures 49.84-bit entropy. Provably uniform distribution.
HKDF info binds key to session participants	Sorted public key pair in HKDF info differentiates sessions sharing the same code but different ephemeral keys.
Handshake concurrency lock	helloProcessing flag prevents racing handleHelloMessage() executions on ephemeralKeyPair.
Rogue-key rejection while ACTIVE	Hellos with an unexpected peer key while ACTIVE are discarded without disrupting the active session.
X25519 browser capability check	checkX25519Support() gates the join flow. No silent fallback to weaker cryptography.
Pre-handshake buffer capped	MAX_BUFFERED_MESSAGES = 32; overflow drops oldest. Memory exhaustion path closed.
Session idle and max-lifetime timers	Client and server enforce 15-min idle and 60-min max independently.
Truncated IP logging	No full IPs persisted at any layer.
Five-layer server rate stack	maxPayload + token bucket + per-IP cap + global cap + liveness sweep.
Crypto setup before WebSocket open	joinChannel() derives all key material before creating the WebSocket. No race condition.
CSP exfiltration restriction accurately disclosed	connect-src limits exfiltration. security.html now accurately describes what the CSP does and does not protect.

10. Summary Table

ID	Finding	Severity	Status
D-01	PBKDF2 iterations: page said 100k, code used 300k	**Discrepancy**	**Resolved (v2)**
D-02	Replay protection: page said absent, code had it	**Discrepancy**	**Resolved (v2)**
D-03	Hello wire format described incorrectly	**Discrepancy**	**Resolved (v2)**
D-04	Peer-key-change behavior changed; page not updated	**Discrepancy**	**Resolved (v3)**
D-05	CSP claim overstated XSS protection	**Discrepancy**	**Resolved (v3)**
F-01	Counter advanced before decryption — relay DoS vector	**Moderate**	**Resolved (v2)**
F-02	No rate limiting or message size enforcement (server)	**Moderate**	**Resolved (v3)**
F-03	Channel hash logged in server output	**Low**	**Resolved (v2)**
F-04	Unauthenticated role claiming — behavior disclosed	**Low**	**Resolved (v3)**
F-05	No Content Security Policy	**Low**	**Resolved (v3)**
F-06	No server-side session expiry	**Info**	**Resolved (v3)**
F-07	Modular bias in code generator	**Info**	**Resolved (v2)**
—	Client pre-handshake buffer unbounded	**Low**	**Resolved (v3)**
—	Full IP addresses in server logs	**Low**	**Resolved (v3)**

11. Conclusions and Recommendations

Revision 3 closes all remaining open findings. The core cryptographic implementation is sound: HMAC-authenticated handshake, ephemeral X25519 forward secrecy, HKDF session key derivation, AES-256-GCM encryption with counter-bound AAD, and strict replay protection all hold against the deployed code.

The two documentation discrepancies identified in this revision — D-04 and D-05 — have been corrected in security.html. The page now accurately describes what the implementation does and does not protect in both areas.

Priority	Item
Medium	Hash-pin inline script blocks: replace script-src 'unsafe-inline' with sha256 hash entries. Tightens XSS defense from exfiltration-only to full execution prevention.
Low	Add frame-ancestors 'none' as an HTTP response header at the static-hosting layer.
Future	Upgrade PBKDF2 to Argon2id or scrypt. Changing the KDF salt is a breaking protocol change requiring a migration plan.

This report was produced through manual source review with AI-assisted analysis. It constitutes a detailed technical review but not a formal third-party security audit.

Appendix A: Annotated Code Map

The following tables map every significant block in index.html and server.js to its function in the execution and security model, anchored to specific line numbers in the current source.

A.1 index.html — Structure Overview

Lines	Block	Role in execution / security model
1-13	Document bootstrap / meta tags	DOCTYPE, charset, viewport.
14-394	CSS style block	All UI styling. No cryptographic role.
396-495	HTML body / UI markup	Channel setup form, QR canvas, video element. maxlength="32" on code input is the only security-relevant attribute.
496-10599	jsQR v1.4.0 (inlined)	QR camera decoder. Not in encryption or handshake path. No known advisories.
10600-10613	qrcode v1.5.1 (inlined)	QR canvas renderer. Not in encryption or handshake path. No known advisories.
10614-10655	Application script / constants	GCM_IV_LENGTH, CHANNEL_CODE_LENGTH, CHANNEL_CODE_CHARS, SESSION_LIMIT_MS, IDLE_TIMEOUT_MS, MAX_BUFFERED_MESSAGES.
10656-10820	Cryptographic functions	All crypto primitives — see A.3.
10821-11060	BBQr utility functions	Base36, hex, Base32 encode/decode. No cryptographic role.
11061-11370	Format helpers / file handling	UR parsing, BBQr/UR send paths. Data enters encryption at sendEncrypted().
11371-11395	State variable declarations	All mutable session and handshake state — see A.2.
11396-11420	sendEncrypted()	Enforce ACTIVE before any send. Increments sendCounter.
11421-11665	joinChannel() / leaveChannel()	WebSocket lifecycle. Crypto setup precedes socket open. leaveChannel() wipes all key material.
11666-11960	QR scan loop and dispatch	Camera processing, BBQr/UR reassembly. Passes to sendEncrypted().
11961-12250	Sender display / file output	QR animation, file download. Downstream of decryption.
12251-12380	Handshake helpers	broadcastHello(), handleHelloMessage() — see A.4.
12381-12445	handleWebSocketMessage()	WebSocket event dispatch.
12446-12500	processDataMessage()	Replay protection and decryption.
12501-12800	UI update functions	No cryptographic role.
12801-12896	Event setup / DOMContentLoaded	Button handlers, X25519 capability check on load.

A.2 State Variables (lines 11371-11395)

Variable	Init	Security role
handshakeState	'IDLE'	Guards sendEncrypted() — messages cannot be sent until ACTIVE.
ephemeralKeyPair	null	X25519 CryptoKeyPair. Nulled after deriveSessionKey(). Private key inaccessible after handshake.
myPublicKeyB64	''	This peer's exported public key. Used in HMAC signing and HKDF info.
peerPublicKeyB64	''	Peer's public key. Used to detect and reject unexpected incoming keys.
hmacKey	null	HMAC-SHA256 key. Signs and verifies hello messages. Cleared on leaveChannel().
sessionKey	null	AES-256-GCM key. Never transmitted. Cleared on leaveChannel() and expiry.
messageBuffer	[]	Pre-handshake buffer. Capped at 32. Overflow drops oldest.
helloProcessing	false	Concurrency lock on handleHelloMessage().
sendCounter	0	Monotonic send counter. Included in AES-GCM AAD.
recvCounter	-1	Last accepted peer counter. Anything <= this is rejected.
idleTimer	null	Fires after IDLE_TIMEOUT_MS; calls leaveChannel().

A.3 Cryptographic Functions (lines 10656-10820)

Function	Security role
checkX25519Support()	Attempts crypto.subtle.generateKey. Gates join button. No silent fallback.
deriveChannelName(code)	SHA-256(code)[:32]. Relay receives hash, not raw code.
generateEphemeralKeyPair()	crypto.subtle.generateKey({name:'X25519'}, true, ['deriveBits']). Private key never exported.
deriveSessionKey(kp, myPub, theirPub)	X25519 DH -> HKDF (SHA-256, 32-byte zero salt, info='keylay-v1-session|'+sorted pair). Derives AES-256-GCM key. No key transits wire.
deriveHmacKey(code)	PBKDF2-SHA256, salt='keylay-v1-hmac', 300k iterations -> HMAC-SHA256 key. Not memory-hard.
signPublicKey / verifyPublicKey	crypto.subtle.sign/verify('HMAC',...). verify() is constant-time.
encrypt(plaintext, key, counter)	12-byte random IV; AAD='keylay-v1|'+counter; AES-GCM encrypt; IV prepended.
decrypt(b64, key, counter)	Slices IV; reconstructs AAD from counter; AES-GCM decrypt (throws on tag mismatch).
generateRandomCode()	Rejection sampling (threshold 248); % 31 into CHANNEL_CODE_CHARS. 49.84-bit entropy.

A.4 handleHelloMessage() — Handshake Receiver

Step	Code	Security role
Ignore own reflection	if (theirPub === myPublicKeyB64) return;	Relay broadcasts hellos to all peers including sender.
Verify HMAC	if (!theirSig || !(await verifyPublicKey(...))) return;	Rejects peers that cannot sign with a key derived from the session code.
Reject unexpected key while ACTIVE	if (ACTIVE && theirPub !== peerPub) return;	Discards rogue joiner hellos. No session reset. Active session preserved.
Derive session key	sessionKey = await deriveSessionKey(ephemeralKeyPair, ...);	X25519 DH + HKDF.
Advance to ACTIVE; discard private key	handshakeState='ACTIVE'; ephemeralKeyPair=null;	Forward secrecy begins here.
Send final hello	ws.send({type:'hello', pubkey, sig});	Ensures peer can complete handshake.
Flush message buffer	for (msg of buffered) await processDataMessage(msg);	Buffered messages decrypted in order.

A.5 server.js — Structure (331 lines)

Lines	Function	Role
1-34	Constants and bootstrap	MAX_PAYLOAD_BYTES, RATE_BUCKET_CAPACITY, RATE_REFILL_PER_SEC, MAX_CONNECTIONS_PER_IP, MAX_GLOBAL_CONNECTIONS, SESSION_IDLE_MS, SESSION_MAX_MS, PING_INTERVAL_MS.
36-52	sessions Map, ipConnections, globalConnections	Per-session and per-IP state. No persistence across restarts.
54-78	remoteIp(), truncateIp(), log()	IP extraction; /16 truncation; ISO-8601 timestamped log helper.
80-110	closeSession(), bumpIdle(), armSessionTimers()	Idle timer, max-lifetime timer, graceful peer notification.
112-230	connection handler	Admission control, rate bucket, liveness tracking, message dispatch, close handler.
232-250	liveness sweep	Pings all clients every PING_INTERVAL_MS; terminates non-ponging sockets.
252-262	handleJoin(ws, code)	First joiner -> sender. Second -> receiver. Third -> refused.
264-286	handleClaimSender(ws, code)	Unauthenticated sender role claim within session membership. Disclosed in security.html.
288-318	handleDataTransmission(ws, ...)	Forwards encrypted payload to all other peers. Does not inspect content.
320-331	handleHello(ws, code, pubkey, sig)	Forwards hello to all other peers. HMAC verification is the client's responsibility.

This report was produced through manual source review with AI-assisted analysis. It constitutes a detailed technical review but not a formal third-party security audit.