Security Audit

dunno

Version 1.1.0

Privacy-Focused Android Messaging Application

Document: DUNNO-SEC-2026-001

Date: 05 August 2026

Classification: Confidential

Scope: Full source code audit (Android/Kotlin)

This report presents an independent security audit of the dunno v1.1.0 source code, conducted from the perspective of a published release build (.aab/.apk). Build configuration, debug logs, and internal repository details are out of scope.

Table of Contents

1. Executive Summary

This report presents a security audit of dunno v1.1.0, an Android privacy-focused messaging application. The audit covers the full Kotlin/Java source code (~130+ files), configuration files, and build configuration from the perspective of a published release build (.aab/.apk).

The application employs a strong, multi-layered security architecture. All messaging traffic is end-to-end encrypted using NaCl/libsodium and routed exclusively over the Tor network. Local storage is encrypted with SQLCipher, and a user-configurable app PIN with Argon2id key derivation protects data at rest. This version introduces several significant security improvements: device-bound key storage (DeviceKeyWrapper), Keystore-wrapped identity keys (IdentityKeyWrapper), a session-key architecture that minimizes plaintext key exposure in RAM (Inbox v4), runtime anti-tamper monitoring (DeviceIntegrity), and obfs4 bridge support for censored networks (BridgeConfig).

The core messaging functionality — text chat, PTT walkie-talkie, and file attachments — contains no critical or high-severity findings. One HIGH finding (H-01) relates to the wipe-PIN verification storage, which lacks the device-binding protection that other key material enjoys.

The application also includes two optional WebRTC-based features: Fast Call (audio) and Video Call. These features deliberately trade some privacy guarantees for functionality. Before use, the application presents an explicit security warning requiring user consent. Findings related to these features are documented separately in Section 7.

i This audit evaluates the application as it would appear to an end user after publication. Build configuration files, debug logging (stripped by ProGuard in release builds), and internal repository details are excluded from scope.

2. Scope & Methodology

In Scope

- Full source code audit of all Kotlin/Java production files (~130+ classes)
- AndroidManifest.xml, proguard-rules.pro, network_security_config.xml
- Cryptography, networking, storage, authentication, and UI security components
- WebRTC audio (Fast Call) and video call functionality

Out of Scope

- Server-side infrastructure (TURN/STUN relay, video signalling server)
- Debug and log code — stripped by ProGuard in release builds
- Build configuration files not present in the published artifact
- Third-party library vulnerabilities (unless caused by misconfiguration)
- Runtime/dynamic analysis (fuzzing, network traffic analysis)
- Physical device security

Methodology

Manual code review with architectural analysis. Each security-relevant component was assessed against criteria of confidentiality, integrity, authentication, authorization, and robustness. The OWASP Mobile Top 10 (2024) served as a reference framework. Findings are classified using a four-tier severity scale: CRITICAL, HIGH, MEDIUM, LOW.

3. Application Architecture

dunno is an Android (Kotlin/Jetpack Compose) messaging application with the following security architecture layers:

- Transport: All communication routed through Tor hidden services (SOCKS5). A local NanoHTTPD server (127.0.0.1 only) receives inbound messages. Optional obfs4 bridge support for censored networks.
- End-to-end encryption: Transport v2 uses ECDH (X25519) with per-message derived keys via NaCl cryptoSecretBoxEasy. Ed25519 detached signatures provide authenticity. Legacy v1 sealed-box format retained for backward compatibility.
- Local storage: SQLCipher (AES-256-CBC) with Keystore-wrapped passphrase. Message bodies use Inbox v4 format combining ECDH shared secret with HMAC-derived SessionKey. UserDataKey is device-bound via DeviceKeyWrapper (Android Keystore) and zeroed from RAM after unlock.
- App access control: PIN-based unlock with Argon2id key derivation (opsLimit=10, ~64 MB memory). Biometric unlock via AndroidX Biometric. Separate wipe-PIN for emergency data destruction. Identity secret keys are Keystore-wrapped with user authentication (IdentityKeyWrapper).
- Runtime protection: DeviceIntegrity monitors for Frida, root, and other tampering. Detection triggers session key wipe and forced app lock. ProGuard obfuscates crypto classes in release builds.
- Emergency wipe: 13-phase procedure with secure delete (zero-overwrite before deletion), Keystore entry cleanup, Tor hidden-service key removal, and crash recovery via persistent markers.

4. Security Analysis — Core Application

4.1 Cryptography & Key Management

The cryptographic implementation is of high quality. All primitives are provided by libsodium (LazySodium bindings) — no custom cryptography is implemented.

- Transport v2: ECDH via cryptoBoxBeforeNm + SHA-256 key derivation with domain separation ('dunno-transport-v2'). Per-message 256-bit key + 24-byte random nonce via cryptoSecretBoxEasy.
- Inbox v4: SessionKey = HMAC-SHA256(UserDataKey, 'dunno-session-v1' || generation). Combines ECDH shared secret with SessionKey. UserDataKey is zeroed after unlock — only the SessionKey remains in RAM. A TEE compromise alone cannot decrypt stored messages without the PIN.
- PIN key derivation: Argon2id via libsodium crypto_pwhash. Fixed parameters (opsLimit=10, memLimit=64 MB). Domain separation between unlock-PIN and wipe-PIN. PIN chars are converted to bytes without an intermediate immutable String. Arrays.fill() zeroing after use.
- DeviceKeyWrapper: Hardware-backed AES-256-GCM key in Android Keystore (alias 'pm3_device_bind'). Binds the UserDataKey blob to the physical device — a disk image copied to another device cannot decrypt it.
- IdentityKeyWrapper: User-authenticated Keystore key (4h timeout, BIOMETRIC_STRONG | DEVICE_CREDENTIAL). Identity secret keys are never stored in plaintext on disk. Auto-migration from plaintext at first unlock.

Finding: WipeVerifier stores the raw Argon2id output of the wipe-PIN as a PinKeyBlob (MAGIC='PINB') without additional protection. Unlike the UserDataKey (protected by DeviceKeyWrapper) and identity secret keys (protected by IdentityKeyWrapper), the wipe-PIN verification hash has no device-binding layer. An attacker with a forensic disk image can perform unlimited offline brute-force attempts against this hash.

Impact: An attacker with physical access to the device (or a forensic image) can attempt unlimited wipe-PIN guesses without rate-limiting or device-binding. A successful guess triggers a full data wipe — a denial-of-service risk for the legitimate user.

Recommendation: Apply DeviceKeyWrapper (already present in the codebase) to the wipe-PIN blob, identical to how UserDataKeyManager v2 protects the UserDataKey with a dual layer (DeviceKey + PIN-wrap). This makes offline brute-force impossible without TEE access.

4.2 Networking & Transport

All core messaging traffic (chat, PTT, attachments, Fast Call signaling) is routed through Tor hidden services via a custom HTTP/1.1 client (OnionHttpClient) over SOCKS5. The Network Security Config only permits cleartext for localhost and .onion domains.

- Tor-only architecture: HTTP over SOCKS5 to .onion addresses. OkHttp is used exclusively for the optional Video Call signalling feature (see Section 7).
- Connection pooling: Maximum 3 concurrent connections per peer for messages, 2 for call signaling.
- Bridge support: BridgeConfig with 12 embedded obfs4 bridges, automatic rotation on bootstrap failure, persistence of working bridges, and iat-mode filtering (only 0 or 2 accepted).

Finding: OnionHttpClient implements a hand-written HTTP/1.1 parser (readAsciiLine, parseStatusCode, readHeaders) rather than using a proven library. This introduces risk of parsing errors or HTTP request smuggling vulnerabilities.

Impact: Potential parsing vulnerabilities when communicating with malicious Tor hidden services. Impact is limited because peers are trusted contacts (E2EE-verified).

Recommendation: Consider using a lightweight HTTP client library with SOCKS5 support, or isolate the parser in a thoroughly-tested module with fuzz testing.

LOW L-01: XOR obfuscation of relay .onion address in TurnCredentialClient

Finding: TurnCredentialClient uses XOR encryption to hide the TURN relay .onion address in the APK. The code explicitly notes this is not a secret — only intended to defeat naive string/regex scans. This is security through obscurity.

Impact: Negligible — the address is trivially extractable from the APK.

Recommendation: Accept that the .onion address is publicly known. The current approach adds complexity without meaningful security benefit.

4.3 Data Storage & Database

Local storage is protected with multiple encryption layers:

- SQLCipher (AES-256-CBC): Full database encrypted. 32-byte passphrase via KeystoreHelper (KeyStore-wrapped AES-256-GCM). @Synchronized prevents race conditions.
- Field-level encryption: Message bodies via Inbox v4 (ECDH + SessionKey). Contact names, conversation previews, and attachment metadata via SensitiveDataCrypto (AES-256-GCM + AAD).
- UserDataKey v2: Dual wrapping — PIN-wrap (AES-256-GCM) + DeviceKeyWrapper. Auto-migration from v1 at first unlock after update.
- Emergency wipe: 13 phases with secure delete (zero-overwrite before delete). Keystore aliases and Tor hidden-service keys are cleaned up. Resume-after-crash support.

The UserDataKey is wrapped with two layers: first AES-256-GCM with the PinWrappingKey (Argon2id-derived from the user's PIN), then AES-256-GCM with a hardware-backed Android Keystore key (DeviceKeyWrapper). A disk image copied to another device cannot decrypt the outer layer. Automatic migration from v1 (PIN-only, no device binding) to v2 at first unlock after update.

STRONG S-02: Inbox v4: SessionKey architecture minimizes RAM exposure

The v4 inbox format uses SessionKey = HMAC-SHA256(UserDataKey, context || generation). The UserDataKey is zeroed immediately after unlock — only the SessionKey persists in RAM. This dramatically reduces the forensic attack surface. Even a full TEE compromise cannot recover locally stored message content without the PIN.

STRONG S-03: Emergency wipe with secure delete and Keystore cleanup

EmergencyWipeManager overwrites files with zeros before deletion. New phases include Tor hidden-service key removal and Android Keystore entry cleanup (5 aliases: identity wrap, SQLCipher passphrase, attachment wrap, device bind, biometric). Database files use 3 retries with deleteOnExit fallback. Crash recovery via persistent wipe-in-progress marker.

4.4 Authentication & Access Control

The authentication model is layered and robust:

- App PIN: Argon2id (opsLimit=10, memLimit=64 MB) with domain separation (DUNNO_UNLOCK_V1 vs DUNNO_WIPE_V1). Constant-time comparison. Arrays.fill() zeroing after use.
- Biometric: BiometricKeyManager with 4-hour timeout, BIOMETRIC_STRONG.
- Wipe PIN: Separate PIN for emergency data destruction. Works at any time (no delay).
- Exponential backoff: FailedAttemptManager with escalating delay (5s, 10s, 20s, 40s cap).
- DeviceIntegrity: Runtime anti-Frida/anti-root monitoring. Detection wipes the session key and forces app lock. ProGuard obfuscates crypto classes in release builds as an additional anti-Frida measure.

Finding: The code documents a 1-hour inactivity reset for the failed-attempt counter, but does not store a timestamp. The shouldResetDueToInactivity() method always returns true if any failed attempt has ever occurred. The counter is never automatically reset by inactivity.

Impact: A legitimate user who makes 6+ typos retains the delay permanently until a successful unlock or app restart. The documented 1-hour reset behavior does not function.

Recommendation: Store a timestamp of the last attempt in AppSettingsEntity, and check whether more than 1 hour has elapsed on each new attempt.

4.5 Input Validation & Rate Limiting

Multi-layered input validation is consistent across all protocols:

- Body size limits: Messages 256 KiB, attachments/PTT/call 512 KiB. Response 64 KiB max.
- Rate limiting: InboundRateLimiter per peer (senderSignPublicKey).
- Timestamp validation: +/-30s skew, 5-minute maximum age. Prevents replay of old messages.
- Nonce validation: 16-byte random nonce per signed request. RequestNonceJanitor purges expired nonces.
- Signature verification: All inbound requests (messages, attachments, PTT, call signaling, video invites) verified with Ed25519. Failed verification returns uniform error responses.
- Thread isolation: Separate pools for reading (8), messages (4), signaling (2), and attachments (4).

Every inbound request passes through: envelope parsing, signature verification, timestamp check, nonce check, body-hash verification, and per-peer rate limiting. Faulty inputs receive consistent HTTP status codes. The layered validation makes it extremely difficult to inject invalid or malicious data.

5. Findings — Core Application

The following findings pertain to the secure core of the application: text chat, PTT walkie-talkie, and file attachments. No CRITICAL findings were identified in this area.

WebRTC audio and video calling features are assessed separately in Section 7, as these are optional features presented with an explicit security warning and user consent requirement.

Ref	Severity	Component	Title
H-01	HIGH	Crypto	Wipe-PIN hash stored without device-binding
M-01	MEDIUM	Networking	Custom HTTP/1.1 parser in OnionHttpClient
M-02	MEDIUM	Auth	FailedAttemptManager missing timestamp storage
L-01	LOW	Networking	XOR obfuscation of relay .onion address
L-02	LOW	Build	Missing ProGuard rules for OkHttp

Core application summary: 5 findings (1 HIGH, 2 MEDIUM, 2 LOW). The core messaging functionality is well-protected with no critical vulnerabilities. The HIGH finding (H-01) is a defense-in-depth gap affecting only the wipe-PIN path — the primary unlock-PIN and data encryption remain fully protected with device binding.

6. Security Strengths — Core Application

dunno v1.1.0 demonstrates a mature security development process. The following strengths deserve explicit recognition:

STRONG S-01: Transport v2: ECDH with per-message derived keys

SodiumCrypto has been modernized from asymmetric sealed boxes to ECDH-based per-message 256-bit keys via cryptoSecretBoxEasy. This is cryptographically cleaner (forward secrecy via ephemeral keys) and more efficient. Domain separation via 'dunno-transport-v2' context string.

STRONG S-02: Inbox v4: SessionKey architecture

The new v4 inbox format minimizes RAM exposure of key material. UserDataKey is zeroed immediately after unlock — only the SessionKey persists. A TEE compromise alone cannot recover stored messages.

STRONG S-03: DeviceKeyWrapper: hardware-backed device binding

AES-256-GCM key in Android Keystore (alias 'pm3_device_bind') binds the UserDataKey to the physical device. Disk images copied to another device cannot decrypt the blob.

STRONG S-04: IdentityKeyWrapper: Keystore-protected identity secrets

Identity secret keys (Ed25519 + Curve25519) are stored as Keystore-wrapped blobs with user authentication (4h timeout). Auto-migration from plaintext at first unlock. IdentityManager.clearCache() zeros plaintext copies on app lock.

STRONG S-05: DeviceIntegrity: runtime anti-tamper monitoring

Startup check plus periodic monitoring for Frida, root, and other runtime threats. Detection triggers session key wipe and forced app lock. ProGuard deliberately obfuscates crypto classes in release builds as an additional anti-Frida measure.

STRONG S-06: BridgeConfig: obfs4 bridge support

12 embedded obfs4 bridges with automatic rotation, persistence of working bridges, and iat-mode filtering. User-customizable bridge list. Enables use in censored networks.

STRONG S-07: EmergencyWipeManager with secure delete

Files are overwritten with zeros before deletion. Tor HS keys and Android Keystore entries (5 aliases) are cleaned up. Crash recovery via persistent markers.

STRONG S-08: ProGuard crypto obfuscation

Crypto classes are deliberately obfuscated in release builds (only security package retained for reflection). This complicates reverse engineering and Frida hooking.

7. Optional Features — WebRTC Audio & Video Calling

NOTE: The WebRTC audio call (Fast Call) and video call features are OPTIONAL. Before use, the application presents an explicit security warning: these features operate in a reduced-security environment compared to the core Tor-only messaging. The user must explicitly accept this warning to enable the features. They are included alongside the secure PTT walkie-talkie to meet user demand for real-time calling functionality, despite the inherent privacy trade-offs.

7.1 WebRTC Audio Call (Fast Call)

Fast Call provides real-time audio via WebRTC. Signaling (SDP offer/answer, ICE candidates) is routed over Tor using Ed25519-signed messages (CallProtocol). TURN credentials are fetched via a signed request to a .onion relay (TurnCredentialClient). Media is encrypted with DTLS-SRTP (enabled by default, not disabled in code).

Security characteristics:

- Signaling: End-to-end encrypted and signed (Ed25519) over Tor
- TURN authentication: Signed request with timestamp and SHA-256 body hash
- Media encryption: DTLS-SRTP active (default WebRTC behavior, confirmed in code)
- Audio-only: No video track — OfferToReceiveVideo=false

Risks:

- ICE TransportsType.ALL: Gathers host, srflx (STUN), and relay (TURN) candidates. The STUN protocol reveals the user's public IP address to the TURN/STUN server operator.
- The TURN server is accessed via Tor (.onion), but the STUN payload itself contains the IP address.

Finding: WebRtcCallSession.kt uses IceTransportsType.ALL with the comment 'Do not force RELAY'. The WebRTC stack gathers STUN candidates that reveal the user's public IP address to the TURN/STUN server. While the server is accessed via Tor, the STUN protocol itself transmits the IP address in its payload. DTLS-SRTP is active (media encryption enabled).

Impact: The user's real IP address may be visible to the TURN/STUN infrastructure operator. This partially undermines Tor's anonymity guarantees. Mitigated by the mandatory user consent dialog shown before Fast Call can be used.

Recommendation: Consider offering a RELAY-only ICE configuration as an optional privacy mode for users who prioritize anonymity over connectivity. Maintain the existing user warning.

7.2 WebRTC Video Call

Video Call provides real-time audio+video via WebRTC. Unlike the core application, only the initial invitation (VideoCallTorSender) is routed over Tor. All subsequent communication — WebSocket signaling and STUN — uses clearnet servers. Camera initialization is separated from PeerConnection setup, allowing audio-only fallback if the camera fails.

Security characteristics:

- Invitation: Signed Ed25519 message over Tor (CallProtocol pattern)
- Media encryption: DTLS-SRTP active by default
- Camera fallback: Audio-only connection remains intact if camera initialization fails
- Isolated architecture: VideoCallViewModel is fully independent from CallController (Fast Call)

Risks:

- Clearnet signaling: OkHttp WebSocket to wss://assistant.eelke-sp.cc. IP address, call timing, and duration visible to the VPS operator.
- Clearnet STUN: stun:assistant.eelke-sp.cc:3470 reveals the user's IP address.
- No certificate pinning: Vulnerable to MITM via a compromised CA.

- Auth without nonce: Replay of a valid auth message is possible within the timestamp window.

Finding: VideoCallServer.kt connects to wss://assistant.eelke-sp.cc/video-call via OkHttp — a direct TCP connection outside Tor. STUN uses a clearnet endpoint. Only the initial invitation (via VideoCallTorSender) goes over Tor. No certificate pinning is configured.

Impact: The user's IP address, call timing, duration, and the fact of communication are visible to the VPS operator. This is a complete bypass of Tor's anonymity guarantees. Mitigated by the mandatory user consent dialog shown before Video Call can be used.

Recommendation: Option A: Route signaling through Tor (WebSocket over SOCKS5 is technically complex but possible). Option B: Document clearly that Video Call does not provide anonymity and maintain the warning. Option C: Implement certificate pinning for the WSS connection as a minimum improvement.

MEDIUM M-03 (WebRTC Video): Video Call server authentication lacks nonce — replay possible

Finding: VideoCallServer.sendAuth() uses signPkB64 + contactId + timestamp + signatureB64. No random nonce is included in the auth message, allowing replay of a valid signed message within the timestamp window.

Impact: An attacker who intercepts a valid auth message can replay it. The server must implement replay detection.

Recommendation: Add a random nonce to the auth message. Implement certificate pinning for the WSS connection.

7.3 Privacy Comparison

The following table compares the privacy properties of each communication mode:

Property	Chat + PTT (Core)	Fast Call (Audio)	Video Call
Transport	Tor (.onion)	Tor + STUN/TURN	Clearnet (WSS+STUN)
IP visible to server	No	Yes (STUN server)	Yes (VPS + STUN)
Signaling	Tor (E2EE)	Tor (E2EE)	Clearnet WSS
Media encryption	N/A	DTLS-SRTP	DTLS-SRTP
Metadata protected	Yes (Tor)	Partial	No (VPS sees all)
User warning required	No	Yes (explicit)	Yes (explicit)
Privacy level	Maximum	Reduced	Minimal

Conclusion: The core application (Chat + PTT) provides full anonymity via Tor. Fast Call partially compromises this (IP visible to the TURN operator). Video Call fully compromises this (IP, metadata, and signaling visible to an external VPS). Both WebRTC features are opt-in and require explicit user consent after a security warning.

Ref	Severity	Component	Title
C-01	CRITICAL	WebRTC Audio	ICE ALL exposes IP via STUN
C-02	CRITICAL	WebRTC Video	Clearnet signaling (WSS+STUN) bypasses Tor
M-03	MEDIUM	WebRTC Video	Server auth without nonce — replay possible

i All WebRTC features are opt-in. The application displays an explicit security warning before use, and the website documents the reduced privacy guarantees. These features exist alongside the secure PTT walkie-talkie to meet user demand for real-time calling functionality.

8. Conclusion & Recommendations

8.1 Overall Assessment

dunno v1.1.0 is a well-secured messaging application with a strong cryptographic foundation and a thoughtfully designed architecture. The addition of DeviceKeyWrapper, IdentityKeyWrapper, Inbox v4, Transport v2, DeviceIntegrity, BridgeConfig, and the enhanced EmergencyWipeManager demonstrate a mature security development process.

The core messaging functionality (chat, PTT, attachments) contains no critical vulnerabilities. One HIGH finding (wipe-PIN verification without device-binding) and two MEDIUM findings (custom HTTP parser, missing timestamp in FailedAttemptManager) are the only issues in the secure core. These are well-defined and straightforward to address.

The two CRITICAL findings relate exclusively to the optional WebRTC features (audio and video calling), which are protected by explicit user consent. The privacy trade-offs are clearly communicated to the user before these features can be activated.

8.2 Recommendations by Priority

- [HIGH] Apply DeviceKeyWrapper to the wipe-PIN blob (WipeVerifier) for device-binding, matching the protection already applied to the UserDataKey.
- [HIGH] Consider a RELAY-only ICE configuration as an optional privacy mode for Fast Call.
- [HIGH] Implement certificate pinning for the Video Call WSS connection.
- [HIGH] Add a random nonce to the Video Call auth message to prevent replay.
- [MEDIUM] Replace the custom HTTP parser in OnionHttpClient with a proven library, or add comprehensive fuzz tests.
- [MEDIUM] Implement actual timestamp storage in FailedAttemptManager for the 1-hour inactivity reset.
- [LOW] Add ProGuard keep-rules for OkHttp (com.squareup.okhttp3).
- [LOW] Document that the XOR obfuscation in TurnCredentialClient is not a security feature.

8.3 Final Remarks

The dunno development team has achieved an impressive level of security for a mobile messaging application. The use of proven cryptography (libsodium), Tor-only transport with bridge support, hardware-backed key storage (Android Keystore/TEE), and runtime integrity monitoring places this application among the strongest privacy-focused communication tools available. With the recommended improvements, the security posture can be further perfected.