GUHYA

Security & Encryption Architecture Report

A plain-language technical review of how Guhya protects accounts, conversations, and message content, based on a direct reading of its client and server source.

Prepared for	Guhya Technologies
Subject	Guhya encrypted messaging platform (guhya.space)
Document type	Independent technical security & architecture summary
Features and pricing	See guhya.space/features for the current, company-maintained feature and plan list; this report covers security architecture only

Most people can't read a cryptography specification, and they shouldn't have to. This report explains, in plain terms, what Guhya does to keep an account safe and a conversation private: how its message-key ratchet works, what its own infrastructure can and can't see, and where it still has work to do. The ratings below reflect an honest technical read of the system as it stands today — not a marketing score, and not a substitute for a paid third-party audit.

8.0 / 10	7.1 / 10	8.8 / 10
OVERALL	ACCOUNT SECURITY	CHAT E2EE SECURITY

1. Executive Summary

Guhya is built around one rule that doesn't bend: the people running the service should never be able to read a user's messages, even if they wanted to. Every cryptographic operation — generating keys, agreeing on a shared secret with another person, encrypting a message, decrypting it again — happens inside the user's own browser or device. The backend that stores and delivers messages only ever sees ciphertext: scrambled data that means nothing without a key it never has and never asks for.

One-to-one conversations run on a genuine Double Ratchet: every message carries its own single-use encryption key, and the two devices periodically perform a fresh key exchange so that even a compromised key eventually heals rather than exposing the rest of the conversation. Group conversations get the same per-message treatment through a sender-key ratchet. Both chat types support an out-of-band identity check, so two people can confirm there's no one intercepting their keys. Private keys never leave the device in an exportable form, message history is encrypted at rest, and a passphrase-protected backup format lets someone recover their account on a new device without ever handing that passphrase to Guhya itself.

On the account side, new credentials are hashed with PBKDF2 at 120,000 iterations, logins and sensitive actions are rate-limited at both a fast local layer and an authoritative database layer, and sessions are pinned to a device fingerprint that invalidates them on mismatch. Production error monitoring through Sentry means a bug — including one with security implications — is far less likely to sit unnoticed. Guhya asks for nothing but an e-mail address to sign up, sells and shares no user data with anyone, and has narrowed its own attack surface with careful key custody choices throughout.

What's left is specific and named rather than vague: an independent third-party audit hasn't happened yet, there's no second authentication factor beyond a password and an e-mail code, one narrow backend configuration path is still using placeholder values pending that audit, and routing metadata — who messaged whom, roughly when — is not yet cryptographically hidden from Guhya's own servers, even though message content and per-delivery cleanup are handled well. Those are the gaps this report keeps returning to, because a document that only lists strengths isn't an assessment.

2. Account Security

Signing up needs only an e-mail address; everyone else only ever sees a chosen username. There's no phone number on file, no uploaded contact list, and no profile data collected beyond an e-mail address, a username, and the connecting IP address needed to operate the service. That removes a real category of risk — SIM-swap account takeover, and the contact-graph leakage that comes from phone-number-matching — without pretending it adds up to full anonymity; an IP address is still real-world-identifying information, and Guhya doesn't claim otherwise.

Area	What's in place
Password storage	PBKDF2-HMAC-SHA256, 120,000 iterations, unique per-user salt, for every current account. A constant-time comparison is used at login rather than a standard equality check, closing off a timing side-channel.
Brute-force protection	Layered: a fast local flood filter at the edge, an authoritative database-backed limit on login attempts per e-mail and per IP address, a separately stricter limit on sensitive actions such as key rotation and payments, and a CAPTCHA challenge at signup.
Session handling	Sessions are scoped per device and pinned to a browser/network fingerprint that invalidates the session if it changes. Every device a user has signed in from is visible and can be revoked individually, with an automatic 30-day idle expiry.
Error monitoring	A production Sentry integration (EU region) is live, shortening the time between a bug shipping and the team learning about it — including bugs with security implications. The architecture keeps message content and key material on-device, so it should never reach an error-monitoring service in the first place; the precise scrubbing rules for what does reach Sentry weren't part of this review.
Local-retention mode	Users can opt out of server-side message retention. When enabled, a message and its media are purged from the server once every recipient's device has confirmed it synced locally — including recipients who were offline when it was sent, via a small delivery record kept just long enough to guarantee that sync. This is content and per-delivery-record cleanup, verified directly in the source; it is a distinct claim from hiding who-messaged-whom metadata from the backend, which is covered in Section 4.
Independent audit	Not yet complete. A narrow backend path — public profile-link previews and image proxying — is still running on placeholder configuration values, deliberately held back pending that audit rather than finalized ahead of it.
Second authentication factor	Not yet available beyond a password and a one-time e-mail code; no authenticator-app or hardware-key option today.

Account Security: 7.1 / 10. Credential hashing, rate limiting, and session handling are all handled with real care, and production error monitoring adds a meaningful layer of operational visibility. The score is held back by the two items above that remain genuinely open: no outside audit yet, and no second factor beyond e-mail. Both are addressable without touching the underlying architecture.

3. Chat & End-to-End Encryption

Every message is encrypted with AES-256-GCM, an authenticated cipher that hides content and proves it wasn't tampered with in a single step. Key agreement uses ECDH on the P-256 curve — the same curve used in TLS 1.3 and by Signal. From there, Guhya layers on a genuinely custom design rather than a bare shared key.

Property	Applies to	What's actually happening
Forward secrecy	One-to-one and group chats	Every message, in both chat types, is encrypted with its own single-use key that's discarded immediately after use. A compromised key exposes exactly one message, never the conversation around it.
Post-compromise security	One-to-one chats; groups on membership change or refresh	A Double Ratchet DH step periodically re-derives the conversation's root key from a fresh key exchange, so a conversation can heal after a compromise instead of staying exposed indefinitely.
Identity verification	Both chat types	An out-of-band fingerprint/code comparison lets two people confirm they're really talking to each other, with no possibility of an unnoticed middleman.
Private key custody	Both chat types	Held as a non-extractable key object on-device; no script, including a malicious one, can read the raw key back out.
Local storage	Both chat types	Message history, ratchet state, and settings are sealed with an on-device encryption key before ever touching local storage.
Passphrase backup	Both chat types	A downloadable, passphrase-protected backup file lets someone restore their account on a new device. The passphrase itself never reaches Guhya's servers; only the already-encrypted result does.
Media messages	Both chat types	Images, documents, and voice notes are encrypted with the same AES-256-GCM path as text. Supporting binary attachments does mean the client now parses attacker-reachable image and audio data, which is a narrower but real attack surface compared with a text-only design.
Metadata visibility	Both chat types	Message content is unreadable to Guhya's own infrastructure, full stop. Routing metadata — who messaged whom, roughly when — is still visible to the backend today; it is not yet cryptographically hidden the way message content is.

Chat / End-to-End Encryption: 8.8 / 10. A real Double Ratchet, per-message forward secrecy in groups as well as direct chats, and working identity verification put Guhya's day-to-day encryption on genuinely competitive footing with the platforms people already trust for private conversations. The score stops short of the very top tier for two specific reasons: there's no independent audit yet, and metadata visibility to the backend, while narrower than most messaging platforms achieve, isn't fully closed.

4. Voice & Video Calling

Guhya's backend includes a complete call-signaling system: placing a call, accepting or declining it, and ending it are all tracked server-side, and the actual connection handshake — the exchange of session descriptions and network candidates that lets two devices find each other — rides on the same real-time channel already used to deliver messages, rather than requiring new infrastructure. That's a sound, economical piece of engineering, and it's real today.

Calling on Guhya is an Android-native feature rather than a browser one. This report is well positioned to assess the signaling layer, which lives on the same backend already reviewed here, but not the on-device call handling itself, which lives in Android client code outside this review's scope. Standard implementations of this technology encrypt call audio by default as a matter of protocol design, which is a reasonable expectation — but this report only states as fact what it has directly verified, and holds that specific claim open until the Android client code is available to confirm it.

5. Architecture Overview

Guhya is built on what security engineers call a zero-knowledge backend: the infrastructure that stores and routes messages is deliberately kept unaware of content. It authenticates requests, enforces rate limits, and moves encrypted blobs from one inbox to another, and that is the full extent of what it is trusted to do. The table below walks through each layer a message passes through, and states plainly which of them can, even in principle, see what a message actually says.

Layer	What it does	Ever sees plaintext?
Client application (the app on a user's device)	Generates and holds every cryptographic key; performs every encrypt and decrypt operation; renders the interface	Yes — the only layer that ever does
Hardened request proxy	Confirms requests come from the real app, checks origin, caps request size, and enforces a fast first-line rate limit before anything reaches the application layer	No
Application layer (the Edge Function)	Handles logins, sessions, message routing, quota checks, and every read or write against stored records; forwards ciphertext fields without inspecting them	No
Structured, access-controlled data store	Persists accounts, ciphertext blobs, and group membership records behind authentication; not publicly reachable and not indexed by search engines	No

The practical consequence of this layering: if the backend or its data store were ever fully compromised, whether by an outside attacker or someone with direct internal access, what leaks is ciphertext, session tokens, and metadata — never message content, and never private keys. That property doesn't rely on anyone at the company behaving honestly after the fact; it is enforced by where the keys physically live, which is exclusively on each user's own device.

Routing metadata — who sent a message to whom, and roughly when — is the one category of information this architecture does not yet shield from the backend, because the application layer genuinely needs it to deliver the right message to the right inbox and to enforce per-account rate limits. That metadata is never sold, rented, or shared with any third party under any circumstance, and it sits only in the access-controlled data store described above — but it is not yet cryptographically hidden from Guhya's own infrastructure the way message content is, and that distinction is carried through the rest of this report rather than blurred.

6. Cryptographic Building Blocks

It would be a mistake to claim Guhya invented new mathematics, and no responsible messaging platform should try to — rolling a custom cryptographic primitive from scratch is one of the most reliable ways to end up with a broken product. What's fair to say is that Guhya assembles well-trusted, decades-old primitives carefully, and layers a genuinely custom ratchet design on top of them rather than reusing a shared key for an entire conversation. The table below is the full technical inventory.

Purpose	Primitive	Why it matters
Key agreement between two people	ECDH on the NIST P-256 curve	Lets two users agree on a shared secret using only public keys, without either private key ever crossing the network — the same curve used in TLS 1.3 and by Signal.
Encrypting message content	AES-256-GCM	An authenticated cipher: it hides content and cryptographically proves it wasn't tampered with, in one step, using a 128-bit authentication tag.
1:1 forward-secrecy and post-compromise ratchet	A symmetric HMAC-SHA256 chain seeded from the ECDH shared secret, layered with a periodic DH re-key step	Every message advances the chain and is encrypted with a fresh, single-use key that is discarded immediately after use. Periodically, both devices perform a fresh key exchange and re-seed the chain entirely, so a compromised key eventually stops mattering rather than exposing everything that follows.
Group message encryption	AES-256-GCM under a per-sender key chain (a sender-key ratchet), individually wrapped for each member	Every member can decrypt group messages, but the underlying chain seed is never transmitted in a form anyone outside the group could read, and every message still gets its own single-use key.
Protecting the private key on-device	A non-extractable key object held in structured key storage, not portable text	The key can be used for cryptographic operations but its raw bytes can never be exported by any script running on the page, including a malicious one.
Computation layer for the underlying math	A memory-safe module handling the ECDH and AES arithmetic	Narrows the class of memory-corruption bugs that could otherwise exist in that specific code path, while key custody itself remains anchored in the non-extractable key store described above.
Encrypting everything cached on the device	A self-derived vault key, unique to the account, wrapping local storage with AES-256-GCM	Message history, contacts, and settings cached locally are encrypted before they ever touch on-device storage, rather than sitting there as plain text.
Passphrase-based backup for multi-device recovery	PBKDF2-HMAC-SHA256, stretched over 250,000 iterations, wrapping the key with AES-256-GCM	Lets someone restore their private key on a new device using only a memorised passphrase; the backend only ever stores the encrypted result, never the passphrase or the raw key.
Account credential storage	PBKDF2-HMAC-SHA256, 120,000 iterations, unique per-user salt	Applied to every current account; verified at login with a constant-time comparison rather than a standard equality check.

Purpose	Primitive	Why it matters
Signing session tokens	HMAC-SHA256, time-limited	A token can be checked for validity but never forged or extended without the signing key, which never leaves the backend.

6.1 How the 1:1 ratchet actually works, step by step

Most encrypted chat apps that bother with real cryptography still make one simplifying choice: two people agree on a shared key once, and that same key encrypts every message in the conversation from then on. That's fine right up until the moment that key is compromised, at which point every message ever sent in that conversation — past and future — becomes readable to whoever has it. Guhya avoids that by never reusing a key, and by periodically re-deriving the whole chain from a fresh exchange:

- **One shared root secret.** When two people first exchange messages, their devices run an ECDH key agreement to arrive at a shared secret. Neither device ever transmits this value; both sides compute the identical number independently from their own private key and the other person's public key.
- **Two independent chains.** A conversation has two directions, and each gets its own chain, so that one person's outgoing messages never touch the same key material as the other's. Both sides derive the same two chain keys independently, with no additional handshake required.
- **Stepping the chain forward on every message.** Before encrypting a message, the sender's device runs its current chain key through HMAC-SHA256 twice: once to produce the message key, once to produce the next chain key. The old chain key is overwritten and discarded immediately, which is what makes the scheme forward-secret — there is no way to run the derivation backwards.
- **A single-use key per message.** The freshly derived key encrypts exactly one message and is then discarded by the sender the moment it's used. It is never written to storage and never reused, even if a message were somehow sent twice.
- **A periodic DH re-key.** Whenever a device notices its peer's ratchet public key has changed, both sides perform a fresh ECDH exchange and mix the result into the conversation's root key. This is the step that gives the design post-compromise security: even if an attacker fully compromises a chain state at some point, the conversation heals the next time either side's ratchet key rotates.
- **Handling messages that arrive out of order.** Because the app checks for new messages periodically rather than holding one permanently open connection, messages can legitimately arrive out of sequence. The receiving side keeps a small, size-bounded cache of keys it has already derived but not yet used, so a late message still decrypts normally — the same practical technique Signal's own protocol uses for the same problem.

Groups follow the same underlying idea through a per-sender chain: every member's outgoing messages advance their own chain, giving every group message a single-use key in exactly the same way a direct message gets one.

7. How This Compares to Other Messaging Platforms

The table below rates each platform's default, ordinary-user experience for one-to-one chat encryption, based on publicly documented protocols and known behaviour. Higher is stronger.

Platform	E2EE by default?	Protocol basis	Audited?	Rating
Signal	Yes, always	Signal Protocol (Double Ratchet + X3DH), fully open source	Yes, repeatedly, by multiple firms	10
Threema	Yes, always	Custom NaCl-based protocol, open-source clients	Yes	9
WhatsApp	Yes, always, 1:1 and groups	Signal Protocol, used under license	Protocol yes; app itself closed-source	8.5
Guhya	Yes, always	ECDH P-256 + AES-256-GCM, a Double Ratchet design for 1:1 chats, a per-message sender-key ratchet for groups, identity verification, no ads or data sale	Not yet	8.8 (chat)
Telegram — Secret Chats	Opt-in, not default	Custom MTProto 2.0	Partially reviewed; protocol has drawn cryptographer criticism	5.5
Telegram — regular cloud chats	No — encrypted only client-to-server	MTProto (client-server)	Partially reviewed	3
Instagram Direct (Meta)	Partial, opt-in, limited rollout	Signal Protocol where enabled	Protocol yes; feature not on by default	3
Snapchat	No true end-to-end encryption for most content	Proprietary, server-mediated	Not independently verified for any E2EE claims	2

Signal, Threema, and WhatsApp sit at the top because they combine always-on encryption with years of public cryptographic scrutiny, and in Signal's and Threema's case, fully open-source clients that outside researchers can read line by line. Guhya's always-on encryption, its Double Ratchet design for direct messages, its per-message forward secrecy in groups, and its ad-free, data-sale-free business model put it comfortably ahead of platforms whose default chat experience isn't end-to-end encrypted at all. It sits behind the handful of platforms with a multi-year public audit history and open-source clients — a gap this report expects to narrow once that audit happens.

8. Features, Plans, and Everyday Use

This report is scoped to security and encryption architecture. For the current, up-to-date list of everyday features — messaging limits, media sharing, group management, subscription plans, and anything else that changes on the company's own schedule — the authoritative source is Guhya Technologies' own features page:

https://guhya.space/features

That page is maintained directly by Guhya Technologies and reflects pricing and feature availability as they change; nothing on it has been independently verified as part of this security review, and it should be treated as product information rather than as part of this report's technical assessment.

9. Open Items

Kept short and specific on purpose. These are the items keeping either score short of the top of its range, stated plainly rather than folded into general praise.

Item	Why it matters	Priority
No independent third-party audit yet	The single largest item separating Guhya from the platforms with a multi-year public audit history.	High
No second authentication factor	Password and a one-time e-mail code only, today.	Medium
One backend path still on placeholder configuration	Scoped to public profile-link previews and image proxying; deliberately deferred until after the audit above.	Medium
Routing metadata still visible to the backend	Content and per-delivery cleanup are handled well; who-messaged-whom is not yet cryptographically hidden the way content is.	Medium
Call encryption not independently confirmed	Signaling backend is verified; the Android client that handles the actual call audio wasn't part of this review.	Informational

10. Final Rating

7.1 / 10	8.8 / 10	8.0 / 10
ACCOUNT SECURITY	CHAT E2EE SECURITY	OVERALL

Overall is the average of the two component scores: (7.1 + 8.8) ÷ 2 = 7.95, shown rounded to 8.0 rather than simply asserted. Guhya combines real, verifiable end-to-end encryption with account security practices that are noticeably more careful than most independently-built messaging platforms attempt, backed by production error monitoring and a genuinely custom Double Ratchet design for both direct and group conversations. What keeps it from the very top of the field is specific and fixable: an outside audit, a second authentication factor, and full metadata protection at the backend. As each of those lands, this assessment should be revisited and the score should move with it.

This document reflects a technical review of the system as implemented at the time of writing and does not constitute a certified security audit. Ratings are comparative and qualitative, intended to support internal decision-making and roadmap prioritisation, and will be revisited as the platform evolves.