Sanchr

Privacy Without Compromise

A Technical White Paper on
End-to-End Encrypted Communication

Version 1.1 | April 2026

Prepared by the Zynclave Team

Table of Contents

Executive Summary

Sanchr is a privacy-first encrypted messaging platform that addresses fundamental gaps in how existing secure messengers handle media privacy, user experience friction in privacy controls, and contact discovery metadata exposure.

While established platforms have normalized encrypted messaging, each makes trade-offs that leave users exposed — from metadata sharing with advertising infrastructure, to plaintext contact uploads, to encryption that is not enabled by default. Sanchr introduces three architectural innovations:

- **Vault** — an encrypted media storage system with guaranteed server-side object deletion via dual-mechanism TTL enforcement, eliminating the server as a persistence vector for expired media.
- **Vync Mode** — a single-toggle privacy state that atomically activates maximum privacy across the entire application, eliminating the multi-step configuration burden that causes most users to leave privacy features unused.
- **Hashed Contact Discovery** — a contact matching system that operates on SHA-256 hashed phone numbers without hardware trust assumptions, significantly reducing the contact information exposed to the server compared to plaintext-upload approaches.

The platform is built on a Rust backend architecture using the Signal Protocol for end-to-end encryption, a high-throughput distributed database for message storage, and a hybrid modular monolith architecture that separates real-time calling infrastructure from the core messaging relay.

Feature Maturity Status

Feature	Status
Signal Protocol E2EE (X3DH + Double Ratchet)	Available
1:1 Messaging with encrypted relay	Available
Authentication (phone + OTP + OAuth)	Available
Vault with server-enforced object deletion	Available
Vync Mode (atomic privacy toggle)	Available
Hashed contact discovery	Available
1:1 Voice/Video calls with SRTP	Available
Push notifications	Available
Media upload/download (client-side E2EE)	Available
Sealed sender (metadata protection)	Planned — V2
Group messaging	Planned — V2
Encrypted backups	Planned — V2
Group calls (SFU)	Planned — V3
Traffic padding (timing analysis resistance)	Research
Server open source	Planned

Part I: Why Sanchr Exists

The Privacy Paradox in Modern Messaging

Over 3 billion people use encrypted messaging applications daily. Yet the vast majority of these users operate under a false sense of security. The encryption marketing of major platforms obscures critical gaps in their privacy architectures.

Consider what happens when a user sends a photo through a typical encrypted messenger. The message content is encrypted in transit. But the photo may be automatically saved to the recipient's camera roll. The metadata — who sent it, when, to whom, how large the file was — may be collected and shared with advertising infrastructure. The user's entire contact list may have been uploaded in plaintext during onboarding. The photo persists indefinitely on both devices with no automatic expiration.

The encryption of message content, while important, addresses only one dimension of privacy. True privacy requires protecting content, metadata, media lifecycle, and user behavior patterns simultaneously.

What Users Actually Need

Media that disappears from controlled infrastructure. Users sharing sensitive documents, personal photos, or confidential business materials need assurance that shared media will be deleted from the server and its storage backends — not just hidden from the UI while persisting in server-side storage. Existing "disappearing message" features operate on best-effort client-side deletion with no server-side enforcement. Sanchr's Vault guarantees server-side object deletion, though it cannot control what happens on recipient devices, in OS-level backups, or via screen capture.

Privacy that does not require expertise. Security researchers can configure a messenger for maximum privacy by toggling a dozen settings across multiple screens. The average user never discovers these settings exist. Privacy features that require configuration are privacy features that go unused.

Contact discovery that minimizes exposure. When a messaging app uploads your contacts to match against its user database, your entire social network becomes known to the service provider. This data is subpoena-able, hackable, and in some platforms, commercially exploitable. Perfect zero-knowledge contact discovery remains an open research problem; Sanchr's approach substantially reduces exposure compared to plaintext upload, while being transparent about its limitations.

Sanchr's Design Philosophy

The server is the adversary. Every architectural decision assumes the server could be compromised, subpoenaed, or operated by a hostile actor. The server never holds plaintext content, private keys, decryption capability, or raw contact information. It is designed to be a relay with minimal useful information even under complete compromise.

Privacy must be the default, not the option. Features like Vync Mode exist because the safest configuration should be one toggle away, not buried in a settings hierarchy that users never navigate.

Server-side deletion must be enforced, not requested. The Vault system uses infrastructure-level TTL deletion mechanisms, ensuring that expired media is removed from server-controlled infrastructure regardless of client behavior. This does not and cannot guarantee deletion from recipient endpoints.

Part II: Technical Architecture

System Overview

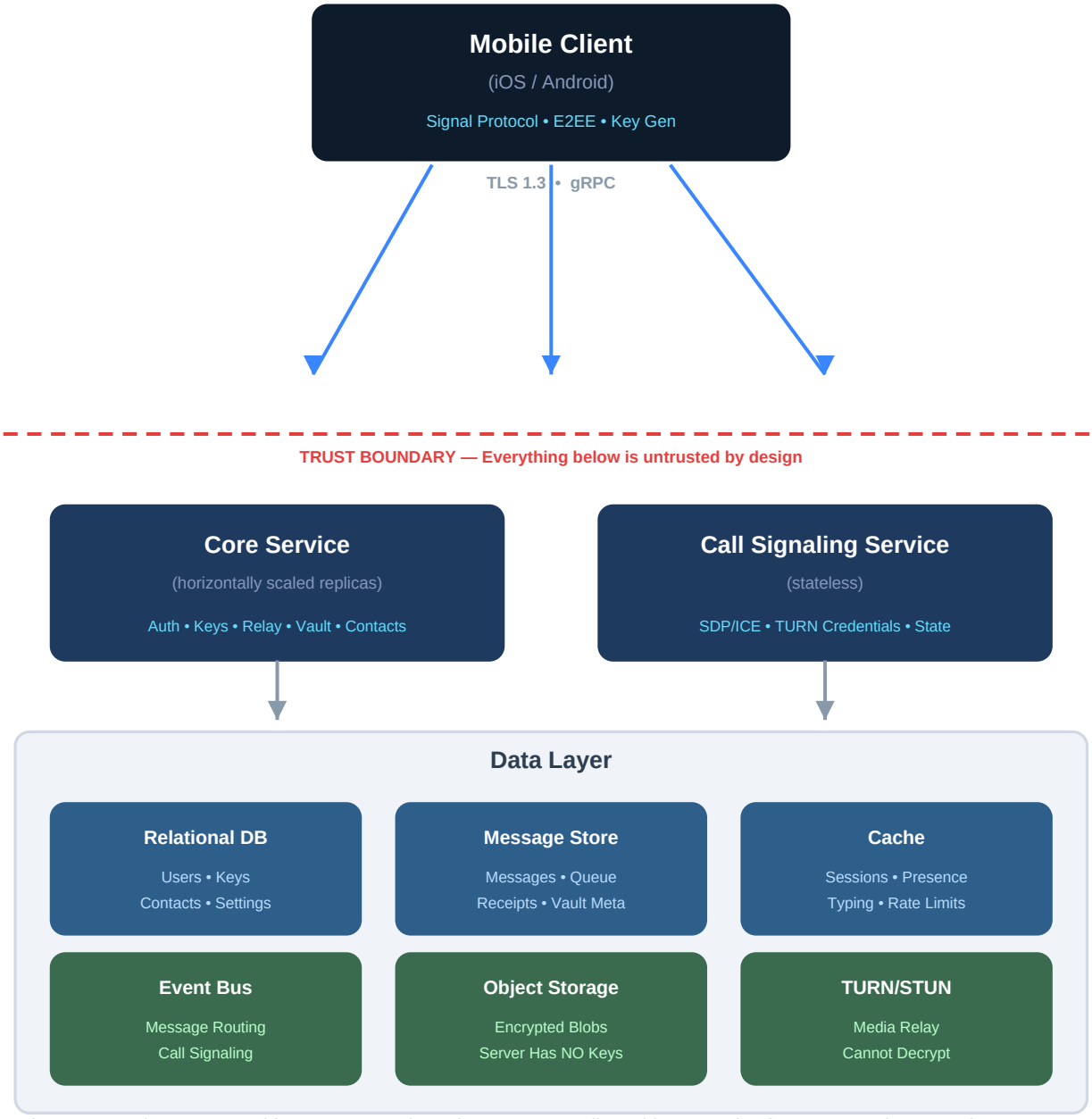


Figure 1: Sanchr System Architecture — trust boundary separates client-side encryption from untrusted server relay

The client encrypts before sending. The server relays opaque ciphertext.

Compromise of any server component yields encrypted data the attacker cannot read.

Sanchr's backend consists of two server components operating on shared infrastructure. The **Core Service** handles authentication, key management, message relay, contact management, settings, vault operations, and media URL generation. The **Call Signaling Service** handles WebRTC call signaling, ICE candidate relay, and TURN credential management.

Both components are stateless and horizontally scalable. All persistent state lives in the data layer: a relational database for structured data, a distributed high-throughput database for message storage, an in-memory cache for ephemeral state, and S3-compatible object storage for encrypted media.

End-to-End Encryption: The Signal Protocol

Sanchr implements the Signal Protocol, a widely-adopted cryptographic protocol for end-to-end encryption. The protocol combines three mechanisms:

Extended Triple Diffie-Hellman (X3DH) establishes a shared secret between two parties who have never communicated before, without requiring both to be online simultaneously. Each user registers a set of public keys with the server: a long-term identity key, a medium-term signed pre-key, and a set of one-time pre-keys. When a sender wants to message a recipient, they fetch the recipient's public key bundle from the server, perform the X3DH key agreement locally on their device, and derive a shared secret that the server never learns.

The Double Ratchet Algorithm provides forward secrecy and post-compromise security by continuously rotating encryption keys. Each message is encrypted with a unique key derived from the ratchet state. Compromising a single message key does not reveal past or future messages. The ratchet state exists only on the communicating devices — the server has no knowledge of it.

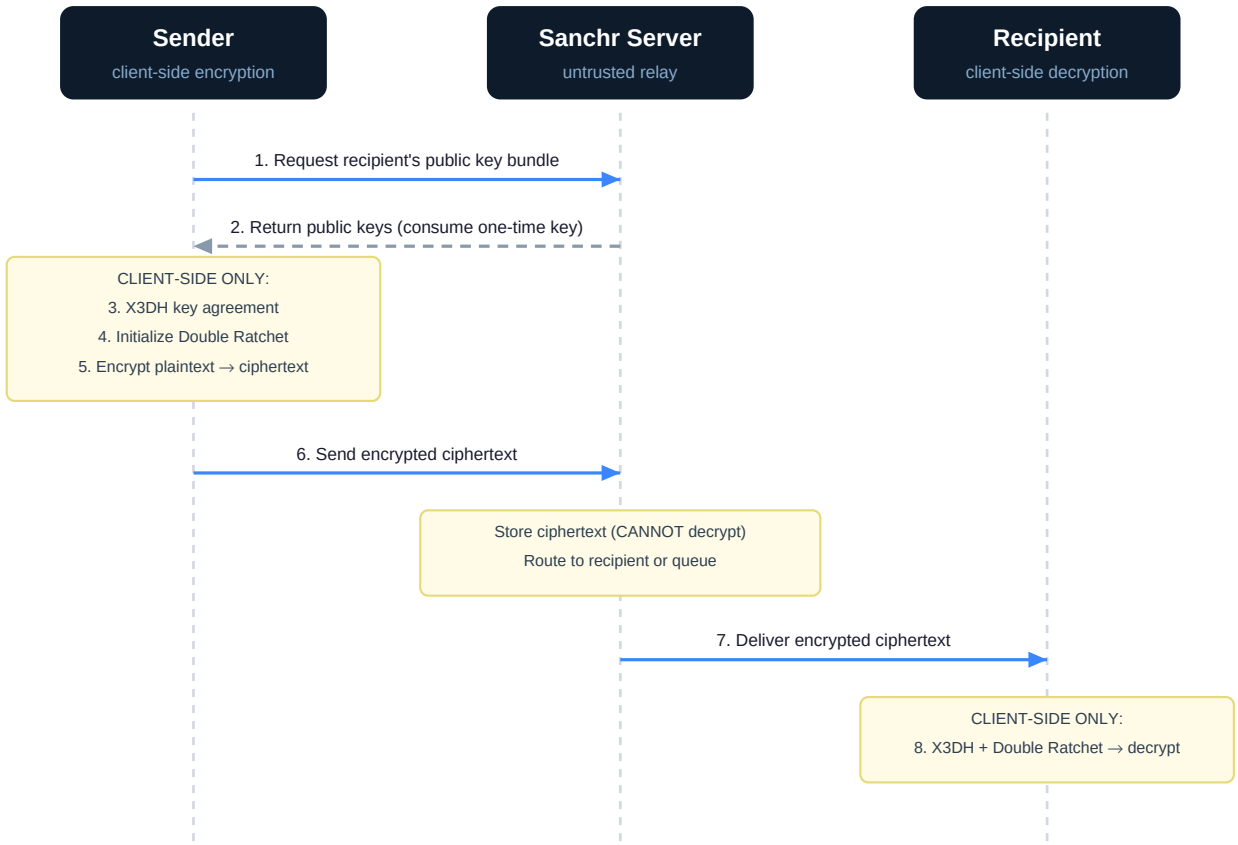
Sealed Sender (planned for V2) will extend metadata protection by encrypting the sender's identity inside the message ciphertext. The server will not be able to determine who sent a message — it will only deliver the opaque ciphertext to the addressed recipient.

The Server's Role in E2EE

The server is explicitly designed as an untrusted relay. It performs four functions:

- **Public key distribution.** The server stores users' public key bundles and serves them to other users who want to establish encrypted sessions. The server never holds private keys.
- **Encrypted blob relay.** The server receives ciphertext from senders and routes it to recipients. It cannot decrypt the content.
- **Offline message queuing.** When a recipient is offline, the server queues encrypted messages. When the recipient reconnects, the server delivers the queued ciphertext.
- **Pre-key lifecycle management.** The server tracks pre-key counts and alerts clients when replenishment is needed.

Message Flow: Sending an Encrypted Message



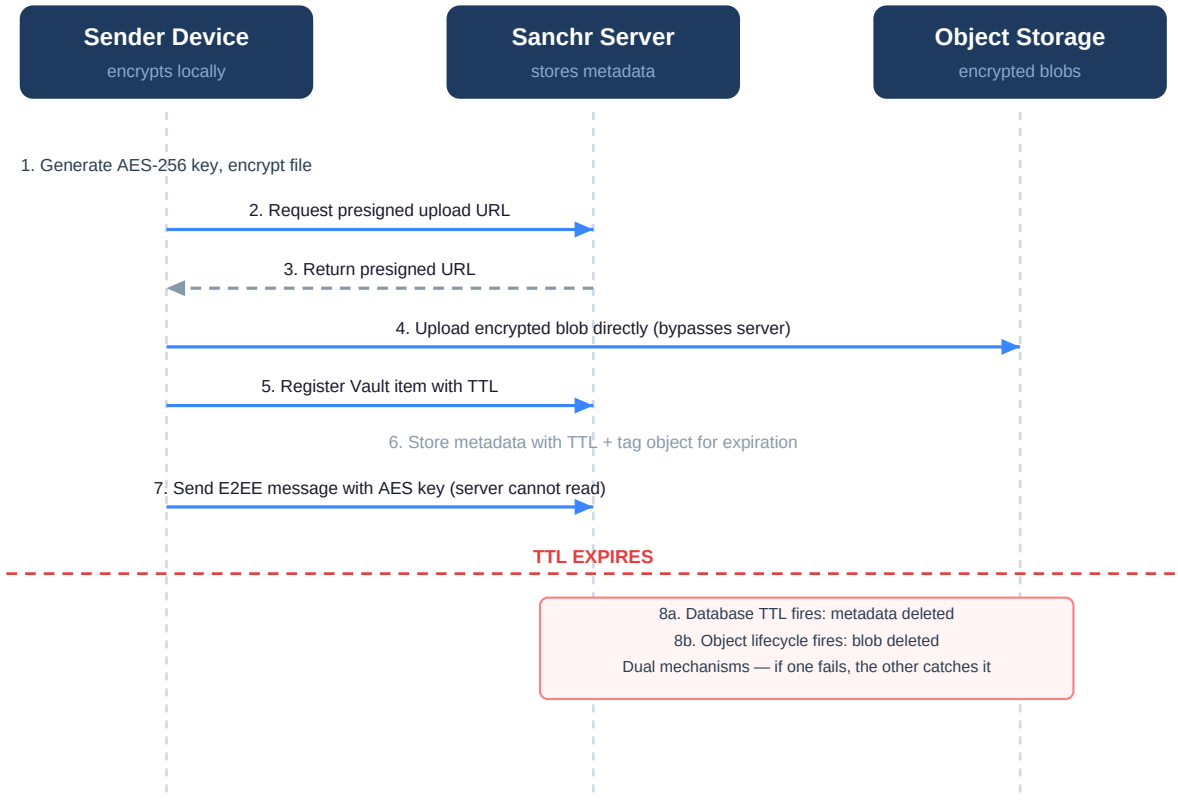
The server handles relay only (steps 6-7). It never performs encryption or decryption.

Figure 2: End-to-end encrypted message flow — the server relays opaque ciphertext only

Vault: Encrypted Media with Server-Side Expiration

The Vault provides an organized, browsable repository of shared media (photos, videos, files) with infrastructure-enforced expiration on server-controlled storage.

Scope of the expiration guarantee. The Vault guarantees that encrypted media objects are deleted from Sanchr's server infrastructure when the configured TTL expires. This guarantee does not extend to recipient devices, OS-level backups, screenshots, clipboard contents, or any data that has left server-controlled infrastructure.



The server never holds the AES decryption key. It stores and eventually deletes an opaque encrypted blob.

Figure 3: Vault lifecycle — client-side encryption, direct upload, and dual-mechanism TTL deletion

Client-side encryption. Before uploading any media, the sending client generates a random AES-256-GCM key, encrypts the file and its thumbnail, and uploads the encrypted blob directly to object storage via a presigned URL. The server never possesses the decryption key. The AES key is transmitted to the recipient inside an encrypted message — it is never exposed to the server.

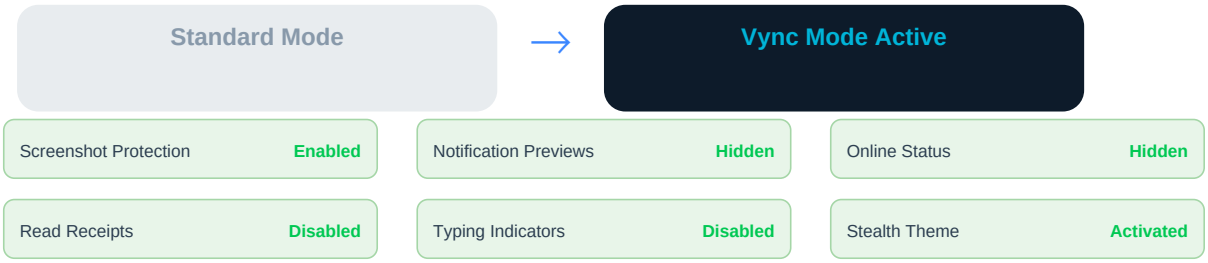
Dual-mechanism server-side deletion. Each Vault item has an expiration timestamp set at creation time. Expiration is enforced through two independent mechanisms: database TTL automatically deletes the metadata row, and object storage lifecycle rules automatically delete the encrypted blob. If one mechanism fails, the other catches it.

Browsable organization. Unlike disappearing messages that vanish from the chat timeline, Vault items are organized in a dedicated interface with filtering by media type, sender attribution, and remaining time until expiration. Users can explicitly save items to their device before expiration if desired.

Share controls. The sending user can configure whether the recipient is permitted to save or forward Vault items. These controls are enforced by the client application and provide meaningful friction against casual redistribution. They do not prevent a determined user from capturing content through external means.

Vync Mode: Atomic Privacy State

Vync Mode addresses a fundamental UX problem in privacy-focused applications: the gap between available privacy features and actually-configured privacy features.



One toggle atomically activates all six privacy settings in a single database transaction.

Figure 4: Vync Mode — a single toggle atomically activates six privacy settings

When a user activates Vync Mode, the server atomically updates six privacy-related settings in a single transaction: screenshot protection is enabled, notification previews are hidden, online status is hidden, read receipts are disabled, typing indicators are disabled, and a stealth theme is activated.

Screenshot protection limitations. Screenshot protection relies on OS-level APIs. These APIs are effective against casual screenshots within the app but do not prevent external cameras, screen recording on rooted/jailbroken devices, accessibility service abuse, or OS-level screen mirroring. Vync Mode's screenshot protection is a friction mechanism, not a cryptographic guarantee.

Research in usable security consistently demonstrates that single-action privacy controls achieve higher adoption rates than multi-step configurations.

Hashed Contact Discovery

Contact discovery — the process of determining which of your phone contacts also use the messaging app — presents a fundamental privacy challenge. Uploading your entire contact list to the server exposes your social graph to the service provider.

Sanchr's approach:

1. The client normalizes each phone number to E.164 format and computes its SHA-256 hash locally on the device.
2. The client sends only the hashed phone numbers to the server.
3. The server performs set intersection between the received hashes and its user database.
4. The server returns matched user profiles without ever learning the actual phone numbers queried.

What this approach provides. Compared to plaintext contact upload, hashed discovery prevents the server from trivially reading contact phone numbers. Compared to hardware enclave approaches, it avoids dependency on hardware security guarantees that have been subject to documented side-channel attacks.

What this approach does not provide. This is not "zero-knowledge" in the formal cryptographic sense. Phone numbers have low entropy — approximately 10 billion valid numbers globally. A motivated server operator could precompute a rainbow table of all possible phone number hashes. Rate limiting provides resistance against external attackers, but does not protect against a malicious server operator.

Optional and selective. Sanchr's contact sync is entirely optional. Users can skip it during onboarding or choose to sync only selected contacts.

Calling Architecture

Sanchr provides end-to-end encrypted voice and video calls using WebRTC with a self-hosted signaling layer.

Signaling. The call signaling service handles SDP offer/answer exchange and ICE candidate trickle between call participants. The signaling server manages call session state and enforces timeouts.

Media transport. For 1:1 calls, media flows directly peer-to-peer when network conditions permit. When direct connectivity is not possible, media is relayed through TURN servers at edge locations. TURN credentials are time-limited and generated using HMAC-based authentication.

Call E2EE. Sanchr's call encryption pre-exchanges SRTP key material through the existing Signal Protocol session rather than relying on standard DTLS-SRTP negotiation. This ensures that relay servers cannot decrypt media, since they never participate in key negotiation.

Implementation caveats under active development:

- **WebRTC stack integration** — injecting pre-shared SRTP keys requires platform-specific handling.
- **Rekeying** — the current design uses a single key pair per call; periodic rekey is planned.
- **Identity binding** — SRTP keys are bound to Signal session identity, with no additional media authentication.
- **Renegotiation** — call hold/resume must survive SDP renegotiation without falling back to DTLS-SRTP.

Data Storage Architecture

Data	Store Type	Purpose	Encryption
User accounts, keys, contacts, settings	Relational DB	ACID guarantees	Disk-level encryption
Messages, pending queue, receipts, vault index	Distributed store	High-throughput with TTL	E2EE ciphertext
Sessions, presence, typing, rate limits	In-memory cache	Ephemeral state	In-memory; encrypted persistence
Media files (photos, videos, documents)	Object storage	Encrypted blobs	Client-side AES-256-GCM

Messages are partitioned by conversation and clustered by timestamp in descending order, so fetching the most recent messages is a single-partition sequential read — the optimal access pattern for chat history pagination.

Part III: Threat Model

Threat 1: Compromised Server

Scenario. An attacker gains full control of Sanchr's server infrastructure, including all databases, application code, and network traffic.

What the attacker learns:

- Encrypted ciphertext blobs — cannot be decrypted without device-held private keys
- Server timestamps of message send/receive events
- Conversation participant lists
- Content type hints ("text", "image", "video") but not actual content
- SHA-256 hashed phone numbers — reversible via dictionary attack (see Contact Discovery)
- Public key bundles — these are public by design
- Presence information and call metadata (who called whom, when, duration)

What the attacker cannot learn:

- Message plaintext, media content, or file contents
- Private keys or ratchet state (stored only on user devices)
- SRTP keys for call media or Vault item decryption keys (transmitted inside E2EE messages)

Threat 2: Man-in-the-Middle Attack

Scenario. An attacker intercepts the connection between a client and the server and attempts to substitute their own public keys.

Protection. The Signal Protocol's safety number verification mechanism allows users to compare their shared safety number out-of-band. If keys have been substituted, the safety numbers will not match.

Limitation. Safety number verification requires user action. Users who do not verify safety numbers are vulnerable to a persistent MITM attack. This is a fundamental limitation shared by all Signal Protocol implementations.

Threat 3: Device Seizure

Scenario. An attacker gains physical access to a user's unlocked device.

What is exposed: all decrypted message history, private keys and ratchet state, downloaded Vault items, and contact/conversation metadata.

Mitigations: Vync Mode with screenshot protection, screen lock with biometric authentication, Vault expiration to limit media persistence, and disappearing messages.

Limitation. An unlocked device with full disk access exposes all locally stored data. This is a fundamental limitation of any end-to-end encrypted system.

Threat 4: Metadata Analysis

Scenario. An attacker with access to server logs performs traffic analysis to infer communication patterns.

Available metadata includes: timing of sends/receives, message sizes, conversation participation, online/offline patterns, call metadata, push notification routing tokens, and client IP addresses. This is the area where Sanchr's current privacy guarantees are weakest.

Current mitigations: presence suppression via Vync Mode, minimal content type hints, and no message content in push notifications.

Planned mitigations: sealed sender (V2), ciphertext padding, IP log rotation, and push notification metadata minimization.

Threat 5: Legal Compulsion

Scenario. Sanchr receives a legal order to produce user data.

What can be provided: encrypted ciphertext (useless without device keys), account timestamps, hashed phone numbers, IP addresses (subject to retention policy), and timing/call metadata.

What cannot be provided: message content, private keys, or call audio/video content.

Threat 6: Pre-Key Exhaustion

Scenario. An attacker rapidly consumes all of a user's one-time pre-keys.

Impact. Exhausted pre-keys cause new sessions to fall back to the signed pre-key only, slightly reducing the forward secrecy guarantee.

Mitigations: rate limiting on key bundle requests, pre-key count monitoring with automatic client alerts, and batch replenishment of 100 pre-keys.

Threat 7: Endpoint Compromise

Scenario. Malware, a rooted/jailbroken OS, or a compromised accessibility service runs on the user's device.

A compromised endpoint can read all decrypted content, extract private keys, bypass screenshot protection, capture vault items, log keystrokes, and exfiltrate metadata. This is a fundamental limitation of all end-to-end encrypted systems.

Planned improvements: platform secure enclave integration for key storage, root/jailbreak detection, multi-device key sync with independent ratchets, and client app attestation.

Appendix A: Cryptographic Specifications

Key Types and Algorithms

Key	Algorithm	Size	Lifetime
Identity Key	Curve25519	256-bit	Permanent (per device)
Signed Pre-Key	Curve25519	256-bit	Rotated weekly
One-Time Pre-Key	Curve25519	256-bit	Single use (consumed)
Message Key	AES-256-GCM	256-bit	Single message
HMAC Key	HMAC-SHA256	256-bit	Per ratchet step
Vault Media Key	AES-256-GCM	256-bit	Per media item
SRTP Master Key	AES-128-CM	128-bit	Per call session
Password Hash	Argon2id	Variable	Stored as hash
Phone Hash	SHA-256	256-bit	Stored permanently

X3DH Key Agreement Parameters

- Curve: Curve25519
- Hash: SHA-256
- Info string: "Sanchr-X3DH"
- One-time pre-keys per batch: 100
- Replenishment threshold: 20

Double Ratchet Parameters

- Root key derivation: HKDF-SHA-256
- Chain key derivation: HMAC-SHA-256
- Message key derivation: HMAC-SHA-256
- Symmetric encryption: AES-256-GCM
- Maximum skip: 1000 message keys

Argon2id Password Hashing

- Memory: 64 MB
- Iterations: 3
- Parallelism: 4
- Salt: 16 bytes (random)
- Output: 32 bytes

Media Encryption

- Algorithm: AES-256-GCM
- Key: 256-bit random (per file)

- IV/Nonce: 96-bit random
- Authentication tag: 128-bit
- Key distribution: inside Signal Protocol encrypted message

SRTP Call Encryption

- Cipher: AES-128-CM (Counter Mode)
- Authentication: HMAC-SHA1-80
- Key derivation: from master key exchanged via encrypted message
- Rekeying: not currently implemented; planned for future release

Appendix B: Infrastructure Specifications

Server Components

Component	Technology	Purpose
Core service	Rust (async)	Auth, messaging, contacts, vault, media, settings
Call service	Rust (async)	WebRTC signaling, TURN credential management
Relational DB	PostgreSQL	Users, keys, contacts, conversations, settings
Message DB	ScyllaDB	Messages, pending queue, receipts, vault metadata
Cache	Redis-compatible	Sessions, presence, typing, rate limits
Event bus	NATS JetStream	Message routing, call signaling, push dispatch
Object storage	S3-compatible	Encrypted media blobs
TURN/STUN	Self-hosted	WebRTC media relay

Deployment

- Orchestration: Kubernetes with horizontal autoscaling
- TLS: TLS 1.3 external, mutual TLS internal
- CI/CD: automated formatting, linting, testing, build, and container delivery

Capacity Design Targets

These are architectural design targets, not benchmarked results. Actual production capacity will depend on deployment topology, hardware, and traffic patterns.

Metric	Design Target
Concurrent users	1M+
Message throughput	100K+ messages/second
Message delivery latency (p99)	< 500ms
Call setup latency	< 2 seconds
Vault expiration accuracy	Within 60 seconds of configured TTL
Pre-key replenishment latency	< 5 seconds from alert to upload

Appendix C: Assumptions and Non-Goals

Assumptions

1. **Client devices are trusted at the application layer.** Sanchr assumes the mobile OS provides basic process isolation and that the client application has not been tampered with.
2. **The Signal Protocol implementation is correct.** Sanchr uses the official Signal Foundation Rust crate. A bug in the protocol library would affect encryption guarantees.
3. **Standard cryptographic primitives are secure.** Security depends on the computational hardness of Curve25519 ECDH, AES-256-GCM, SHA-256, and Argon2id.
4. **Deployment infrastructure provides disk encryption and network isolation.** Server-side security assumes the hosting provider's protections function as documented.

Non-Goals

Sanchr explicitly does not claim to:

- **Protect against a compromised endpoint.** If the recipient's device is compromised, all content displayed on that device is exposed. E2EE protects the transport and server, not the endpoints.
- **Prevent screenshots by external means.** A camera pointed at a screen or a screen recording can capture displayed content regardless of software-level screenshot protection.
- **Provide anonymous communication.** Sanchr requires phone number registration and is a privacy tool, not an anonymity tool.
- **Fully hide communication metadata.** The server observes conversation membership, message timing, and call metadata. Sealed sender (V2) will reduce but not eliminate metadata exposure.
- **Guarantee media deletion on recipient devices.** Server-side expiration removes media from Sanchr's infrastructure only.
- **Replace legal or operational security practices.** Sanchr provides technical privacy controls, not protection against social engineering, coercion, or legal compulsion.

Sanchr is built by the Zynclave team. For security disclosures, contact security@sanchr.com.

This document is versioned and will be updated as the protocol and architecture evolve.