REVERIE

Dream Diary

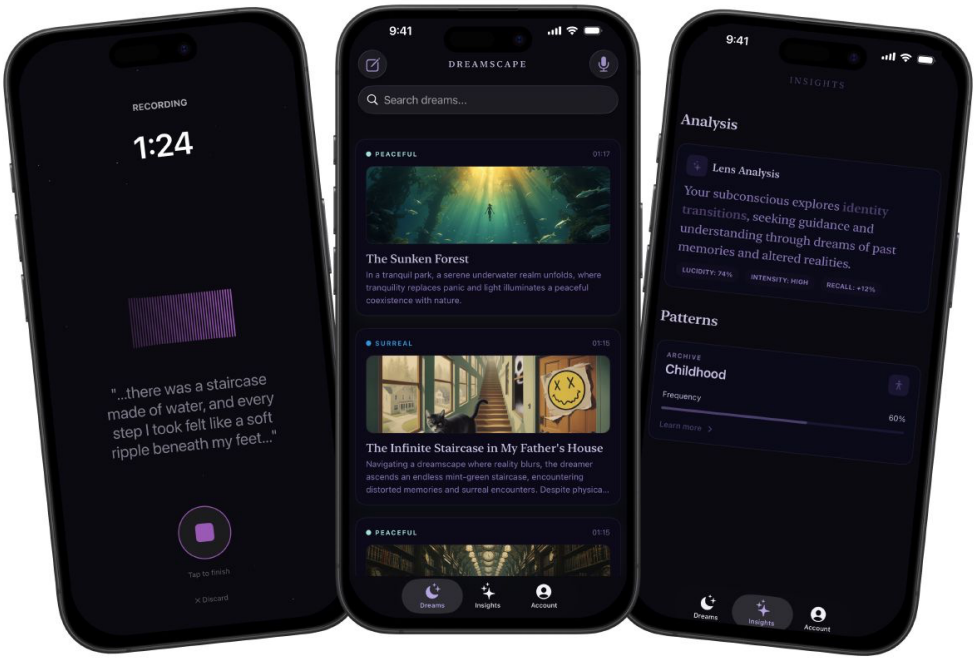
Security & Privacy Whitepaper

Zero-Knowledge Architecture, Cryptographic Design, and On-Device AI Processing

Version 0.1 [DRAFT] | April 2026
Classification: Public

Mindful Slumber

www.mindfulslumber.com/reverie



Executive Summary

Reverie is a privacy-native dream journaling application designed for iOS. This whitepaper is intended for two audiences: prospective users evaluating whether Reverie meets their privacy expectations, and the information security community conducting independent scrutiny of our design decisions.

The central design premise of Reverie is simple: the developers of this application should be technically incapable of reading a user's dream journal. This is not a policy commitment — it is a cryptographic guarantee enforced by the architecture itself. Every piece of sensitive user content is encrypted on the device before it ever touches network infrastructure, using a key that only the user possesses.

This document details the full cryptographic stack underpinning that guarantee, the threat model it is designed to address, the specific primitives and parameters chosen and why, and the areas where the current architecture has known limitations or accepted trade-offs.

Key Properties

End-to-end encryption of all user data using AES-256-GCM • Keys derived from user password via PBKDF2-HMAC-SHA256 (600,000 iterations) • Master Key never transmitted to backend in plaintext • On-device AI processing: no audio or content sent to third-party APIs • Hardware-backed key storage via iOS Secure Enclave • BIP39-style deterministic recovery phrase for vault recovery

1. Introduction

1.1 The Privacy Problem with Dream Journals

A dream journal is among the most intimate digital records a person can maintain. Dreams surface unconscious thoughts, fears, desires, and creative imagery that individuals may never articulate in waking life. Unlike a general diary, dream journals are often recorded immediately upon waking — in an unguarded, semi-conscious state — and contain raw psychological content that their authors would rarely share with any other person.

Yet the majority of digital journaling applications store user content in plaintext on server infrastructure, protected only by access controls and transport encryption. This means the application vendor, their cloud provider, and any attacker who compromises their systems can read every entry. For dream journals specifically, this represents a serious and underappreciated privacy exposure.

1.2 Reverie's Design Philosophy

Reverie was architected from first principles around a single constraint: user data must remain opaque to everyone except the user. This constraint drove every major design decision:

- Content is encrypted on-device before persistence or sync, not after transmission.
- Encryption keys are derived from a user-controlled password and never transmitted.
- AI features (transcription, tagging, image generation) operate within the iOS sandbox using on-device models, never streaming content to external APIs.
- The backend is designed to store and serve opaque ciphertext blobs, not plaintext data.

1.3 Scope of This Document

This whitepaper covers the security and privacy architecture of Reverie for iOS. It addresses:

- The threat model and the attacks we protect against (and those we do not).
- The cryptographic primitives, parameters, and their rationale.
- Key management, derivation, envelope design, and recovery mechanisms.
- Local device safeguards and hardware security integration.
- On-device AI sandboxing and data isolation.
- Known limitations, accepted trade-offs, and future work.

2. Threat Model

2.1 Assets Under Protection

The following data types are treated as sensitive assets requiring encryption at rest and in transit:

Asset	Description
Dream Entries	Full text transcriptions of dream records, including timestamps, tags, and structured metadata.
Audio Recordings	Raw audio captured at time of journaling, before and after transcription.
AI-Generated Insights	Semantic analyses, recurring theme tags, emotional annotations generated from entries.
Cover Art	AI-generated images associated with individual dream entries.
Vault Password	The user's master authentication secret. Never stored or transmitted.
Master Key (MK)	The 256-bit AES key used to encrypt all of the above. Protected via envelope encryption.

2.2 Adversaries and Attack Vectors

Reverie's architecture is explicitly designed to resist the following threat actors and vectors:

2.2.1 Complete Backend Compromise

A sophisticated attacker who achieves full read access to Reverie's backend infrastructure — databases, object storage, application servers, and key management services — will obtain only encrypted ciphertext. The backend holds no plaintext content, no Master Keys, and no Vault Passwords. Recovery is limited to rekeying user envelopes, not decrypting stored content.

2.2.2 Cloud Provider Access

Reverie uses cloud infrastructure for ciphertext storage and envelope delivery. A cloud provider with unilateral access to stored data will observe only opaque binary blobs with no semantic content. This includes compelled access scenarios (e.g., subpoenas or warrants directed at Reverie's cloud provider).

2.2.3 Network Interception

All client-server communication uses HTTPS/TLS. However, Reverie does not rely on TLS as the primary confidentiality control for user data. Even if TLS were compromised — through a rogue certificate authority, SSL stripping, or protocol downgrade attack — transmitted content would remain AES-GCM encrypted and unreadable to a network adversary.

2.2.4 Unauthorized Physical Device Access

Reverie integrates with iOS Data Protection APIs and the Secure Enclave to protect locally stored Master Keys. A physical adversary who obtains an unlocked device is limited by the optional biometric app lock. On a locked or rebooted device, the Master Key is inaccessible until the user authenticates, leveraging the iOS Data Protection Class A/B boundary.

2.2.5 Malicious Insiders

Reverie engineers and operations staff have no technical capability to decrypt user content. They may access ciphertext blobs and Master Key Envelopes, but cannot derive the plaintext without the user's Vault Password — which is never transmitted to Reverie's servers.

2.3 Out-of-Scope Threats

The following threats are explicitly acknowledged as outside the current security guarantee:

- Compromised iOS device (jailbroken, malware-infected, or device with OS-level compromise). In this scenario, an attacker with kernel-level access can in principle extract the Master Key from the Secure Enclave or intercept decrypted data in memory.
- User credential theft. If a user's Vault Password is observed directly (e.g., shoulder surfing, keylogger, or phishing), the Master Key Envelope can be decrypted.
- Recovery phrase interception. If the 12-word recovery phrase is photographed, copied, or obtained, the Master Key can be reconstructed.
- Coercion of the user. Reverie provides no plausible-deniability or duress-password features in the current release.
- Legal compulsion of the user. Reverie cannot protect against legal orders compelling the user to produce their Vault Password or recovery phrase.

3. Cryptographic Architecture

3.1 Design Principles

Reverie's cryptographic design follows three core principles: use only well-audited, standardized primitives; avoid inventing novel cryptographic constructions; and ensure that every sensitive operation defaults to fail-closed behavior (i.e., decryption failure surfaces an error rather than silently passing corrupted data).

3.2 Symmetric Encryption: AES-256-GCM

All user-generated content is encrypted using AES in Galois/Counter Mode (GCM) with a 256-bit key.

Rationale for AES-GCM

AES-GCM is an Authenticated Encryption with Associated Data (AEAD) scheme. It provides both confidentiality and integrity in a single pass, meaning that any tampering with the ciphertext — even a single bit flip — will cause decryption to fail with an authentication error. This eliminates a class of attacks (e.g., padding oracles, chosen-ciphertext attacks) that affect encryption-only modes like AES-CBC.

> **Technical Note — AEAD**
> AES-GCM produces a 128-bit authentication tag appended to the ciphertext. During decryption, the tag is verified before any plaintext is returned. If the tag does not match, decryption fails entirely. Reverie uses a unique 96-bit random nonce per encryption operation. Nonce reuse with the same key under GCM is catastrophic (it can expose the authentication key), so Reverie generates a fresh nonce for every encrypt call via Apple's SecRandomCopyBytes, which is backed by the Secure Enclave RNG.

Key Size

The 256-bit key length provides a security margin well in excess of any known classical or near-term quantum cryptanalytic attack. NIST's post-quantum guidance notes that AES-256 retains approximately 128 bits of effective security against Grover's algorithm on a quantum computer — a level considered adequate for long-horizon security planning.

3.3 Key Derivation: PBKDF2-HMAC-SHA256

The Master Key is never stored in plaintext anywhere. Instead, it is protected by a Password-Derived Key (PDK) generated from the user's Vault Password via PBKDF2.

Parameter	Value	Rationale

Algorithm	PBKDF2-HMAC-SHA256	NIST SP 800-132 compliant; universally supported on Apple platforms via CommonCrypto.
Iterations	600,000	Meets OWASP 2024 recommendation of 600,000 for PBKDF2-SHA256; calibrated to ~300ms on a 2022 iPhone.
Salt	32 bytes (256-bit), random	Generated via SecRandomCopyBytes at registration. Stored alongside the Master Key Envelope on the backend.
Output length	32 bytes (256-bit)	Directly used as the AES-256 key for envelope decryption.

The salt and iteration count are stored in plaintext alongside the Master Key Envelope on the backend. This is intentional: neither value is secret, and their availability is required to re-derive the PDK on a new device. An attacker who obtains these values still cannot derive the PDK without the Vault Password.

Why Not Argon2id?

Argon2id is the current Password Hashing Competition winner and is generally preferred for new designs due to its memory-hardness, which resists GPU- and ASIC-accelerated brute-force attacks. Reverie uses PBKDF2-HMAC-SHA256 for the current release due to its native availability in Apple's CommonCrypto framework without additional dependencies. Migrating to Argon2id is listed as a priority item for a future release (see Section 8, Future Work).

3.4 Master Key Envelope

The Master Key Envelope is the mechanism by which the Master Key can be reconstituted on a new device without ever transmitting it in plaintext.

At registration:

- A 256-bit Master Key (MK) is generated via SecRandomCopyBytes.
- The user defines a Vault Password.
- A 32-byte random salt is generated.
- The PDK is derived: PDK = PBKDF2-HMAC-SHA256(VaultPassword, salt, 600000, 32).
- The Envelope is computed: Envelope = AES-GCM-Encrypt(PDK, MK).
- The Envelope, salt, and iteration count are transmitted to and stored on the backend.
- The Vault Password and PDK are discarded from memory immediately after use.

At login on a new device:

- The backend returns the Envelope, salt, and iteration count.
- The user enters their Vault Password.
- The PDK is re-derived locally.
- The Master Key is recovered: MK = AES-GCM-Decrypt(PDK, Envelope).
- The MK is stored in the iOS Keychain with Secure Enclave protection (see Section 4).

At no point does the backend receive the Vault Password, the PDK, or the Master Key in plaintext. The backend's role is strictly to store and return the encrypted Envelope.

3.5 Data Encryption at Rest

Every piece of sensitive user data stored via Core Data or synced to cloud storage is individually encrypted with the Master Key before being written. Encryption is performed at the application layer, above the storage layer, meaning that database snapshots, iCloud backups, and object storage blobs all contain only ciphertext.

Each encryption operation produces an independent nonce, resulting in independent ciphertext even for identical plaintext inputs. This prevents an attacker from inferring when two entries share identical content.

4. Account Recovery

4.1 The Recovery Phrase Mechanism

Losing access to a Vault Password would result in permanent loss of access to all encrypted dream data. To mitigate this risk, Reverie employs a deterministic recovery phrase mechanism inspired by the BIP39 standard used in cryptocurrency wallets.

At the time the Master Key is generated, it is deterministically encoded as a 12-word phrase drawn from a custom wordlist. The phrase is a direct human-readable encoding of the 256 bits of Master Key entropy — it is not a password from which a key is derived; it is the key, encoded in a memorizable format.

Property	Detail
Phrase length	12 words
Entropy encoded	256 bits (full Master Key)
Encoding basis	Custom wordlist (BIP39-style), 2048 words
Derivation direction	Deterministic: MK → phrase and phrase → MK
Storage	Never stored by Reverie. User responsibility only.
Use case	Vault Password forgotten; reconstruct MK directly on device.

4.2 Security Properties of the Recovery Phrase

Because the recovery phrase encodes the Master Key directly, its security properties are equivalent to the Master Key itself: an attacker who obtains the phrase obtains full read access to all encrypted content. Users should treat their recovery phrase with the same care as a private key:

- Write it down on paper and store it in a physically secure location.
- Do not photograph it or store it digitally in any cloud-synced location.
- Do not transmit it via email, SMS, or any messaging application.
- Do not enter it into any application other than Reverie.

4.3 No Server-Side Recovery

Reverie provides no server-side account recovery mechanism beyond the recovery phrase. There is no "forgot password" email flow that resets vault access. If a user loses both their Vault Password and their recovery phrase, their encrypted data is permanently unrecoverable. This is an intentional design decision: any server-side recovery mechanism would necessarily require Reverie to hold a key capable of decrypting user content, which violates the zero-knowledge guarantee.

User Advisory

The recovery phrase is the single most important security artifact in Reverie's model. Loss of the phrase combined with loss of the Vault Password results in permanent, irrecoverable data loss. Reverie provides clear in-app guidance on secure phrase storage during onboarding and prompts users to verify their phrase before closing the setup flow.

5. Local Device Safeguards

5.1 iOS Keychain and Secure Enclave Integration

Once the Master Key has been derived or recovered, it must be stored securely on the device for use during normal app operation. Reverie stores the active Master Key in the iOS Keychain with the following configuration:

Keychain Attribute	Value and Intent
kSecAttrAccessControl	privateKeyUsage — requests hardware-backed protection via Secure Enclave.
kSecAttrAccessible	kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly — key is unavailable until first user unlock after reboot; not migrated to new devices via backup.
kSecAttrSynchronizable	false — prevents key from syncing to iCloud Keychain.
Protection class	Class B (Protected Until First User Authentication) — enforces the after-first-unlock constraint at the filesystem level.

The ThisDeviceOnly constraint is intentional. Because the Master Key Envelope is available from the backend, a new device can always re-derive the Master Key by re-authenticating with the Vault Password. Allowing the raw Master Key to migrate via backup or iCloud Keychain sync would expand the attack surface unnecessarily.

5.2 iOS Data Protection Classes

Reverie's Core Data store is configured to use iOS Data Protection Class A (NSFileProtectionComplete) for at-rest files. Under Class A, the file encryption key is derived in part from the device's UID key (fused at manufacturing) and the user's passcode, and is unavailable while the device is locked. This provides defense in depth: even if the AES-GCM encryption layer were somehow bypassed, an attacker with a locked device cannot access the underlying files.

5.3 Biometric App Lock

Reverie offers an optional biometric lock at the application layer. When enabled, the app's ReverieKeyManager withholds access to decryption routines until LocalAuthentication (LAContext) successfully evaluates a Face ID or Touch ID challenge. This protects against unauthorized access on a device that is unlocked but unattended (e.g., a phone left on a desk).

The biometric lock is implemented as an application-layer control, not a Keychain access control. This is a deliberate choice: Keychain-level biometric controls (via kSecAccessControlBiometryAny) can be disrupted by biometric enrollment changes and do not support the graceful fallback to Vault Password re-entry that Reverie requires when biometrics are unavailable.

5.4 Memory Handling

Sensitive key material (Master Key, PDK) is stored in Swift value types that are explicitly zeroed before deallocation wherever possible. Reverie avoids logging of key material and excludes sensitive Keychain items from crash reports and analytics payloads. The application does not retain the Vault Password in memory beyond the duration required for PDK derivation.

6. On-Device AI Processing

6.1 Privacy Implications of AI Features

AI features in consumer applications frequently require streaming user data to remote inference APIs. For a privacy-sensitive application like Reverie, this represents a significant risk: even if the AI vendor has strong privacy policies, the user's raw audio and entry content would leave the device, potentially traversing networks and residing on third-party servers.

Reverie eliminates this risk by executing all AI workloads locally on the device using Apple's Neural Engine and Core ML framework.

6.2 Audio Transcription

Property	Detail
Framework	WhisperKit (Core ML-optimized) or Apple Speech framework (on-device mode)
Processing location	Fully on-device; no audio data transmitted to external APIs
Model weights	Downloaded at setup; verified via bundle signature before use
Data lifecycle	Audio buffer → transcription string; raw buffer not persisted after transcription completes

WhisperKit is OpenAI's Whisper model converted to Core ML format for on-device execution. All inference runs within the iOS application sandbox. Audio data is captured, processed, and discarded within the device; the only output that survives is the text transcription, which is immediately encrypted with the Master Key before being written to Core Data.

6.3 Semantic Analysis and Tagging

Semantic analysis, recurring theme identification, and entry tagging are performed by a locally-executed language model running via Core ML. No entry content is transmitted to Reverie's servers or any third-party NLP API for analysis. Model weights are bundled with the application or downloaded on first launch from Reverie's CDN and verified against an expected hash before use.

6.4 AI Cover Art Generation

Per-entry cover art is generated using a DreamShaper Stable Diffusion model variant, also running entirely on-device via the Neural Engine. Image generation is triggered by a text prompt derived from the entry's tags — the prompt itself is derived on-device and the generation process does not require network access. Generated images are encrypted with the Master Key before storage, consistent with all other sensitive assets.

6.5 AI Pipeline Sandboxing

Reverie's application architecture enforces a strict boundary between the AI processing layer and the network layer. The AI pipeline components (transcription, analysis, image generation) are structurally prohibited from initiating network requests during user data processing. This is enforced at the module level: the AI pipeline has no access to URLSession instances or network-capable APIs.

Background download of Core ML model weight files (which do occur over the network at initial setup) is handled by a separate, segregated download manager that operates before the user's vault is unlocked and before any user data enters the processing pipeline.

Verifiable Claim

The claim that AI processing is on-device can be independently verified via network traffic analysis. Users can use a proxy tool such as Charles Proxy or an iOS network inspector to confirm that no outbound connections are made to AI inference APIs during journaling, transcription, or image generation workflows.

7. Backend Architecture and Data Flows

7.1 What the Backend Stores

Reverie's backend is explicitly designed to hold only data that carries no risk if disclosed. The following table summarizes what is and is not stored server-side:

Data Type	Stored Server-Side?	Form Stored
Dream entry content	Yes	AES-256-GCM ciphertext only
Audio recordings	Yes (optional sync)	AES-256-GCM ciphertext only
Cover art images	Yes (optional sync)	AES-256-GCM ciphertext only
Master Key Envelope	Yes	AES-GCM ciphertext of MK; encrypted with PDK
PBKDF2 salt	Yes	Plaintext (not secret)
PBKDF2 iteration count	Yes	Plaintext (not secret)
Vault Password	Never	N/A — never transmitted
Password-Derived Key (PDK)	Never	N/A — never transmitted
Master Key (plaintext)	Never	N/A — never exists on server
Recovery phrase	Never	N/A — user responsibility

7.2 Authentication

User authentication to Reverie's backend uses a standard token-based system (JWT). Critically, the authentication credential is separate from and unrelated to the Vault Password. A compromised authentication token allows an attacker to retrieve the user's Master Key Envelope and ciphertext blobs, but provides no capability to decrypt them without the Vault Password.

This separation of authentication from decryption is an important design property: the two keys that protect Reverie's security model (the Vault Password and the authentication token) are independent, meaning compromise of one does not imply compromise of the other.

7.3 Transport Security

All communication between the Reverie iOS client and backend services uses TLS 1.2 or higher with certificate pinning. Certificate pinning prevents a rogue or compelled certificate authority from issuing a certificate for Reverie's domain and performing a man-in-the-middle attack against users on compromised networks.

8. Known Limitations and Accepted Trade-offs

Reverie's architecture makes explicit trade-offs in favor of privacy that result in user experience constraints. These are documented transparently below.

8.1 Irrecoverable Vault Loss

If a user loses both their Vault Password and their recovery phrase, their encrypted content is permanently unrecoverable. This is the direct consequence of the zero-knowledge architecture: Reverie cannot decrypt user content even in exceptional circumstances. This trade-off is accepted as fundamental to the design.

8.2 PBKDF2 vs. Argon2id

The current release uses PBKDF2-HMAC-SHA256 for password derivation rather than Argon2id. PBKDF2 is not memory-hard, which means it is more susceptible to GPU-accelerated brute-force attacks than Argon2id. The 600,000-iteration parameter compensates partially for this weakness on consumer hardware, but Argon2id would provide stronger guarantees for equivalent user-perceived latency. Migration to Argon2id is planned for a future release pending evaluation of native Apple platform support.

8.3 Single Master Key Architecture

All user data is encrypted under a single Master Key. This simplifies the key management architecture but means that a Master Key compromise exposes all data — historical entries cannot be protected by forward secrecy under this design. A future rekeying mechanism (e.g., per-entry or per-period keys, wrapped by the Master Key) would provide granular access control and reduce the blast radius of a key compromise.

8.4 No Multi-Device Real-Time Sync

Because the Master Key is stored in a per-device Keychain item that does not sync, real-time sync between devices requires the user to authenticate (enter their Vault Password or use the recovery phrase) on each new device. This is a deliberate security decision but represents a usability constraint relative to applications that sync keys transparently via iCloud Keychain.

8.5 Metadata Leakage

While content is encrypted, certain metadata is not: entry timestamps, approximate entry frequency, and the size of encrypted blobs may be observable by a sophisticated backend attacker. Future work includes adding noise to timestamps and padding ciphertext blobs to fixed sizes to reduce metadata leakage.

8.6 On-Device AI Model Integrity

Core ML model weights are verified against a known hash after download but before first use. If a supply chain attack compromised the model weights prior to download, a malicious model could in principle exfiltrate data via a covert channel (e.g., steganographic encoding in generated images). Reverie mitigates this via signed model bundles and network isolation of the AI pipeline, but a sufficiently sophisticated supply chain attack against model weights is a residual risk.

9. Regulatory Alignment

Reverie's zero-knowledge architecture is designed with global privacy regulations in mind. The following table summarizes alignment with key frameworks:

Framework	Relevant Requirement	Reverie's Position
GDPR (EU)	Data minimization; privacy by design; right to erasure.	Content never in plaintext on servers. Erasure deletes ciphertext blobs; without the MK, deleted content is irrecoverable.
CCPA (California)	Right to know; right to delete; no sale of personal info.	Reverie cannot access content to disclose it. Deletion is cryptographically irreversible.
HIPAA	Not directly applicable (Reverie is not a covered entity), but relevant if health-related content is recorded.	End-to-end encryption provides controls consistent with HIPAA technical safeguard requirements.
Apple App Store	App Privacy Label; data-not-linked-to-user requirements.	No plaintext user content is collected. Privacy manifest accurately reflects on-device processing.

10. Future Work

The following security improvements are on Reverie's development roadmap:

- **Replace PBKDF2-HMAC-SHA256 with Argon2id for memory-hard password key derivation, pending stable native Apple platform support.** Argon2id migration:
- **Introduce a key derivation hierarchy where each entry is encrypted under a unique key derived from the Master Key, enabling forward secrecy and granular revocation.** Per-entry key hierarchy:
- **Pad ciphertext blobs to fixed size buckets and add jitter to synchronization timestamps to reduce metadata leakage to backend observers.** Metadata padding:
- **Commission an independent third-party cryptographic audit of the full key management architecture and iOS implementation.** Formal security audit:

11. Responsible Disclosure

Reverie welcomes security research and responsible disclosure of vulnerabilities. If you have identified a security issue in Reverie's implementation or architecture, please contact the security team at the address below. We commit to:

- Acknowledging receipt within 48 hours.
- Providing an initial assessment of severity within 7 days.
- Notifying affected users if a vulnerability results in data exposure.
- Crediting researchers in release notes (unless anonymity is requested).

Security disclosures: security@mindfulslumber.com (PGP key available on request)

Appendix A: Cryptographic Primitive Summary

Primitive	Parameters	Use
AES-GCM	256-bit key, 96-bit nonce, 128-bit tag	All user content encryption (at rest and in transit payloads)
PBKDF2-HMAC-SHA256	600,000 iterations, 32-byte salt, 32-byte output	Vault Password → Password-Derived Key (PDK)
AES-GCM (envelope)	PDK as key, fresh nonce per wrap	Encrypting the Master Key into the envelope
SecRandomCopyBytes	Secure Enclave RNG	Master Key generation, nonce generation, PBKDF2 salt generation
BIP39-style encoding	2048-word custom list, 256-bit input	Human-readable recovery phrase encoding of Master Key
TLS 1.2+	Certificate pinning	Transport security for all client-backend communication

Appendix B: Glossary

Term	Definition
AEAD	Authenticated Encryption with Associated Data. An encryption mode that provides both confidentiality and integrity verification in a single operation.
AES-GCM	Advanced Encryption Standard in Galois/Counter Mode. The symmetric AEAD cipher used for all content encryption in Reverie.
Master Key (MK)	The 256-bit AES key that encrypts all user data. Never transmitted in plaintext.
Master Key Envelope	The Master Key encrypted under the Password-Derived Key. Stored on Reverie's backend.
PBKDF2	Password-Based Key Derivation Function 2. A key stretching algorithm that makes brute-force password attacks computationally expensive.
PDK	Password-Derived Key. The key derived from the Vault Password via PBKDF2. Used to encrypt/decrypt the Master Key Envelope.

Recovery Phrase	A 12-word human-readable encoding of the Master Key. Allows vault recovery when the Vault Password is forgotten.
Secure Enclave	Apple's dedicated security processor. Provides hardware-backed key storage and cryptographic operations isolated from the main CPU.
Vault Password	The user-defined password from which the PDK is derived. Never transmitted to Reverie's servers.
Zero-Knowledge	A design property in which the service provider is technically incapable of accessing user plaintext data.

Document Information

Field	Value
Document Title	Reverie: Security & Privacy Whitepaper
Version	0.1 [DRAFT]
Date	April 2026
Classification	Public
Author	Mindful Slumber Security Team
Review Status	Pending independent audit (see Section 10)
Contact	security@mindfulslumber.com