Preview Vault security white paper

Developer: neurasense.io

This document captures the full cryptographic design of Preview Vault, including algorithms, where operations execute, what is stored, and recommended operational practices.

1. System model

Preview Vault separates responsibilities across the client, application server, and object storage. Plaintext is handled only on client devices.

```
Client (Browser/CLI)
  - derive encryption key (PBKDF2)
  - encrypt files + manifest (AES-256-GCM)
  - upload ciphertext to storage
  - decrypt manifests/files locally

API Server
  - identity + membership checks
  - stores project metadata
  - stores encrypted passphrase for specific policies

S3-Compatible Storage
  - encrypted files: projects/<id>/files/<fileId>
  - encrypted manifest: projects/<id>/manifest.enc
  - project metadata: projects/<id>/project.json
```

2. Threat model and goals

- Primary goal: protect file contents and manifest data from the server and storage.
- Assumed trusted: client devices and user passphrases.
- Assumed untrusted: object storage and, in zero-trust mode, the API server.
- Integrity: provided by AES-GCM authentication tags; tampered ciphertext fails to decrypt.
- Out of scope: endpoint compromise, passphrase theft, client malware, and side-channel leakage from access patterns or ciphertext sizes.

3. Cryptographic primitives

- KDF: PBKDF2 with SHA-256, 250,000 iterations, 16-byte per-project salt.
- Symmetric encryption: AES-256-GCM with 12-byte (96-bit) IV per encryption.
- Team shared key exchange: libsodium crypto_box_seal (Curve25519 + XSalsa20-Poly1305, sealed boxes with ephemeral sender keys).
- Server secret wrapping: AES-256-GCM with key derived as SHA-256(PROJECT_KEY_SECRET).
- Sync token hashing: SHA-256 with timing-safe comparison.

4. Key material

- Project passphrase: user-supplied or generated in the client. Never stored in plaintext on the server.
- Salt: 16 random bytes, base64 encoded, stored in projects/project.json.
- Device keypair: crypto_box_keypair stored in localStorage under previewvault:keys for team-shared mode.
- Server secret: PROJECT_KEY_SECRET (environment variable) hashed with SHA-256 to produce a 32-byte AES key for passphrase wrapping.

5. Client encryption workflow

Project creation + upload

```
salt = random(16 bytes)
key = PBKDF2(passphrase, salt, 250000, SHA-256)

for each file:
  iv = random(12 bytes)
  ciphertext = AES-256-GCM(key, iv, file)
  upload ciphertext to storage

manifest = JSON({files, ivs, metadata})
manifest_iv = random(12 bytes)
manifest_ciphertext = AES-256-GCM(key, manifest_iv, manifest)
upload manifest.enc to storage
```

Viewing

```
project.json -> salt
manifest.enc -> decrypt -> file list
file ciphertext -> decrypt on demand
```

6. Manifest and metadata

The encrypted manifest stores per-file metadata. It is encrypted with the same project key as file blobs. File names, paths, and types are therefore encrypted at rest.

```
manifest.enc (JSON)
{
  "iv": "base64(12 bytes)",
  "ciphertext": "base64(AES-GCM output)"
}

project.json (plaintext)
{
  "projectId": "...",
  "projectName": "optional",
  "salt": "base64(16 bytes)",
  "createdAt": "ISO timestamp"
}
```

7. Storage layout

```
projects/<projectId>/
  project.json        (plaintext metadata + salt)
  manifest.enc        (encrypted manifest JSON)
  files/<fileId>      (encrypted file bytes)
```

Encrypted file bytes are written directly to S3-compatible storage. The API issues presigned URLs for browser uploads and accepts direct uploads during GitHub sync.

8. Security policies

Zero-trust (default) - Passphrase never stored. Clients must supply it on each view. - Highest confidentiality against server compromise.

Local device - Passphrase cached in localStorage under previewvault:passphrase for convenience on a single device.

Team shared - Each member has a local device keypair. The passphrase is encrypted with each member public key and

stored as a ProjectKey row in the database.

```
passphrase -> crypto_box_seal(publicKey) -> ProjectKey.encryptedKey
client -> crypto_box_seal_open(privateKey) -> passphrase
```

Server managed - Passphrase encrypted on the server with PROJECT_KEY_SECRET using AES-256-GCM. - Returned only to authenticated project members.

Public link - Server-managed passphrase returned without auth for link-only access. - Use only for non-sensitive assets.

9. GitHub sync pipeline

GitHub Actions can encrypt and upload a repository directory on each push using the preview-sync CLI. The workflow performs all encryption locally in the runner.

```
PREVIEW_PASSPHRASE + salt -> PBKDF2 -> AES-256-GCM
file ciphertext -> POST /api/sync/file
manifest.enc + project.json -> POST /api/sync/commit
```

10. Identity and access

- Auth: NextAuth GitHub OAuth. User IDs are stored in JWT sessions.
- Project roles: OWNER, ADMIN, MEMBER. Server enforces membership for key retrieval and updates.
- Sync tokens: stored as SHA-256 hashes; validated with timing-safe comparisons.

11. Data at rest and in transit

- Files: stored as raw AES-GCM ciphertext bytes. Object names are deterministic by file ID.
- Manifest: stored as JSON with base64 IV and ciphertext.
- Metadata: project.json is plaintext and includes salt and optional project name.
- Transport: all client to server and client to storage communications should use HTTPS.

12. Best practices

- Passphrases: minimum 12 characters, stored in a password manager. Rotate if leaked.
- PROJECT_KEY_SECRET: use a high-entropy secret, rotate periodically, and re-encrypt stored passphrases after rotation.
- Local storage: avoid local-device mode on shared machines.
- Uploads: enforce HTTPS for presigned S3 URLs and run storage behind private networks.
- Backups: back up object storage and database separately; store passphrases out of band.