OdoVault Security Whitepaper

End-to-End Encryption and Zero-Knowledge Architecture

Version 1.0

Date: February 2026

Executive Summary

OdoVault implements a comprehensive security architecture that ensures user data remains protected at rest on the device and during cloud synchronization. The system employs a defense-in-depth strategy with multiple layers of encryption, combining classical and post-quantum cryptographic algorithms to provide both current security and future-proof protection against quantum computing threats.

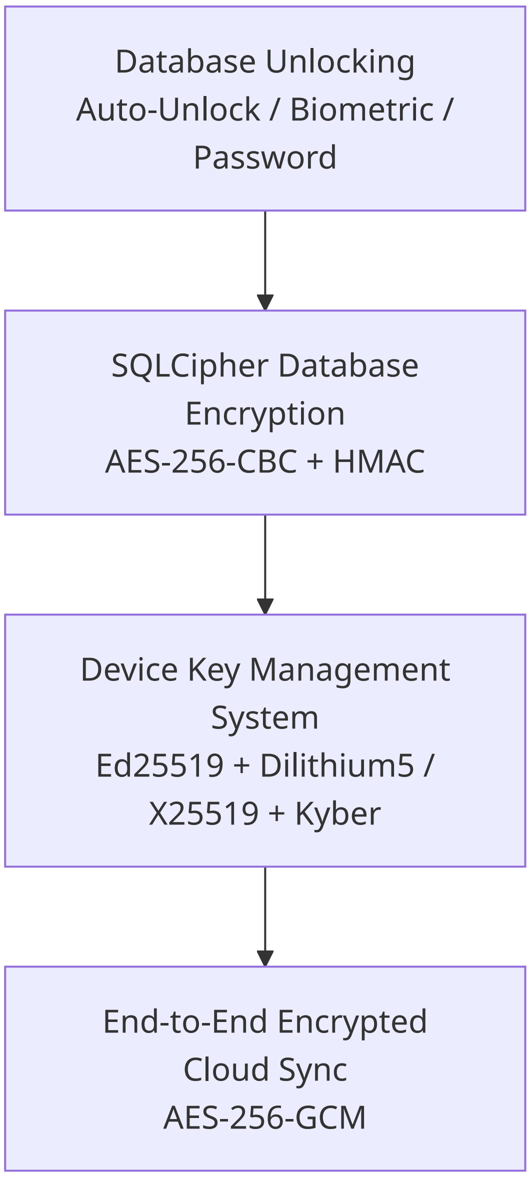
Key security features include:

- **SQLCipher database encryption** with AES-256 for at-rest data protection
- **Default auto-unlock** using a device-stored random password, with optional biometric or manual password modes
- **Hybrid classical/post-quantum cryptography** for device authentication and key exchange
- **Zero-knowledge cloud architecture** where the server never has access to plaintext data
- **Device role authorization (client-assigned)** with server-side gating of snapshot uploads and client-side permission checks on delta replay
- **BIP39 recovery phrases** for secure account recovery
- **Event-sourced synchronization** with end-to-end encrypted deltas
- **Backup verification** that restores snapshots and deltas into a temporary database and compares deterministic fingerprints

Table of Contents

1. Security Architecture Overview

OdoVault's security architecture is built on the principle of **defense in depth**, implementing multiple independent security layers:


Database Unlocking
Auto-Unlock / Biometric / Password
SQLCipher Database Encryption
AES-256-CBC + HMAC
Device Key Management System
Ed25519 + Dilithium5 / X25519 + Kyber
End-to-End Encrypted Cloud Sync
AES-256-GCM

Core Security Principles

1. **Zero-knowledge Architecture**: The server never has access to plaintext data or private keys; it only stores encrypted payloads and public keys
2. **Local-First Security**: All encryption and decryption happens on the device
3. **Quantum-Resistant Cryptography**: Hybrid algorithms protect against future quantum threats
4. **Client-Controlled Key Distribution**: Devices explicitly choose which devices can access their data
5. **Role-Aware Authorization**: Clients assign device roles; the server enforces them for privileged sync actions while clients enforce permissions during delta replay

Threat Model Summary

- Server is honest-but-curious: it stores encrypted blobs and public signing keys only and cannot decrypt user content.
- Network attackers: data is encrypted end-to-end before transport; dual-signature challenges authenticate devices and resist replay.
- Unprivileged device: server blocks snapshot uploads without the proper role; clients reject deltas that violate role-based permissions.
- Lost/removed device: after manual key rotation, future snapshots are encrypted only for trusted devices; historical data on the removed device remains.

Data Residency and Server Role

What lives on the device vs on the server is strictly separated.

- On the device
    - SQLCipher-encrypted database (user data, deltas, sync metadata)
    - Device private keys: Ed25519, X25519, Dilithium5, Kyber1024
    - Symmetric data key (AES-256-GCM) for encrypting user data
    - Device trust state for other devices (trusted/untrusted/expired)
    - Device identifier (hash/fingerprint)
    - Database unlock secret and unlock method indicator in device secure storage for auto-unlock or biometric modes
- On the server
    - Encrypted snapshots and encrypted deltas (never plaintext)
    - Encrypted per-device symmetric keys (payloads target specific devices)
    - Registered public keys for authentication: Ed25519, Dilithium5
    - Device list and minimal metadata for authentication (cryptographic fingerprints of public keys, basic timestamps)
    - Device role assignments (admin/non-admin) for authorization, updated via explicit client requests
    - Session tokens and rate limiting state
- Server role
    - Blind storage and relay of encrypted content
    - Challenge/response for auth using registered public keys
    - Role-based authorization for privileged endpoints like snapshot uploads, based on client-assigned device roles
    - No access to plaintext data or any private keys; symmetric keys are stored only in encrypted form
    - No decision authority over device trust or key distribution

---

2. Device Storage Security

SQLCipher Database Encryption

All user data is stored in a SQLCipher-encrypted SQLite database on the device. SQLCipher provides:

- **Encryption Algorithm**: AES-256 in CBC mode
- **Authentication**: HMAC-SHA512 for tampering detection
- **Key Derivation**: PBKDF2-HMAC-SHA512 with 256,000 iterations
- **Page-Level Encryption**: Each 4KB database page is independently encrypted
- **Initialization Vector**: Random IV per page prevents pattern analysis

---

3. Database Encryption and Access Control

Database Unlocking Methods

OdoVault offers three methods for database access, each with different security trade-offs. The default is auto-unlock using a device-bound secret; users can opt into biometric gating or a manual password.

1. Default Auto-Unlock (Keychain/Keystore)

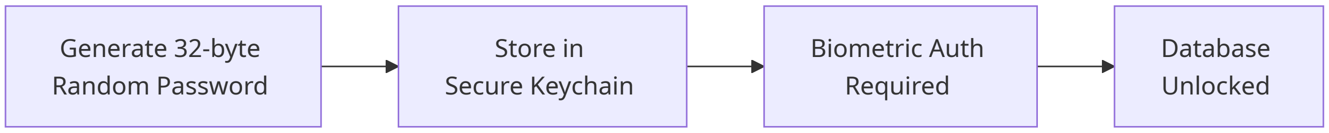
On first launch, the app generates a 32-byte random database password and stores it in the device's secure storage (Keychain/Keystore). The database can be unlocked without user presence, which is required for background sync when the app is not running.

Security Considerations:

- **Device Dependency**: Security depends on the device's secure storage implementation
- **Variable Quality**: Hardware-backed protection varies by device and OS
- **No User Presence**: If the device is unlocked, the app can start without additional verification

2. Biometric Authentication (Convenience-Focused)

When biometric authentication is enabled:



The system uses the same cryptographically secure random password as auto-unlock, but requires biometric user presence when the app starts interactively.

Security Considerations:

- **Device Dependency**: Security depends on the device's secure storage implementation
- **Variable Quality**: Hardware-backed protection varies by device and OS
- **Biometric Weaknesses**: Fingerprints can be lifted, faces can be spoofed
- **Irrevocable**: Biometrics cannot be changed if compromised

- **Coercion Risk**: Biometric traits are harder to keep secret than a long password
- **Convenience Trade-off**: Prioritizes ease of use over maximum security

3. Strong User-Defined Password (Security-Focused)

When password authentication is used:

1. **User Input**: User creates a strong password (recommend 20+ characters with high entropy)
2. **Key Derivation**: Password is processed through PBKDF2 with 256,000 iterations
3. **Memory Management**: Password is cleared from memory after database unlock
4. **Re-authentication**: Required on each app launch
5. **Background Sync**: Disabled when the app is not running because user input is required

Security Advantages with Strong Passwords:

- **No Device Trust Required**: Security doesn't depend on device manufacturer
- **User Control**: Full control over security strength
- **No Hardware Dependencies**: Works identically on all devices
- **Changeable**: Can be updated if compromise is suspected

Background Sync Constraint: Background sync refers to OS-scheduled work when the app is not running. It requires the database to be unlockable without user action. Auto-unlock supports this by design; biometric unlock adds user presence gating for interactive launches but can use the stored secret for background workers. Manual passwords require user input and therefore cannot support background sync.

Security Comparison

Security Aspect	Auto-Unlock (Default)	Biometric	Strong Password	Weak Password
Entropy	256-bit random	256-bit random	100+ bits (if strong)	<40 bits
Storage Location	Device Keychain/Keystore	Device Keychain/Keystore	User memory	User memory
User Presence Required (App Launch)	No	Yes	Yes	Yes
Background Sync (App Not Running)	Yes	Yes (no prompt)	No	No
Device Trust Required	Yes - critical	Yes - critical	No	No
Implementation Variance	High - varies by device	High - varies by device	None	None
Compromise Recovery	Rekey via unlock change	Cannot change biometric	Can change password	Can change password

Security Aspect	Auto-Unlock (Default)	Biometric	Strong Password	Weak Password
Coercion Risk	Device-lock dependent	Higher	Lower	Higher
Physical Attack Surface	Device-level compromise	Fingerprint lifting, face spoofing	Shoulder surfing	Shoulder surfing
Convenience	Very high	High	Low	Medium

---

4. Cryptographic Key Management

Four-Key Hybrid System

Each device maintains four cryptographic keys for different purposes:

Key Type	Algorithm	Purpose	Size
Classical Signing	Ed25519	Authentication & signatures	32-byte public key
Quantum-Resistant Signing	Dilithium5	Future-proof signatures	2592-byte public key
Classical Key Exchange	X25519	Elliptic curve key agreement	32-byte public key
Quantum-Resistant KEM	Kyber1024	Future-proof key encapsulation	1568-byte public key

Key Generation Process

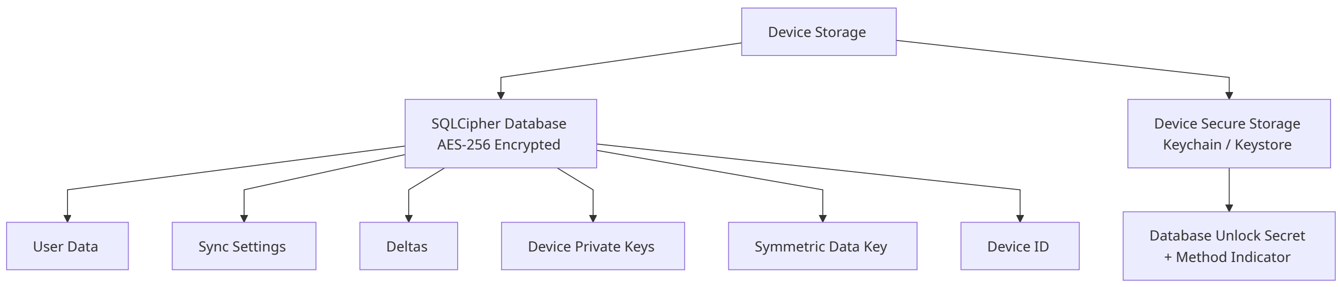
1. **Entropy Source**: Cryptographically secure random number generation
2. **Key Derivation**: Each key type uses appropriate generation algorithms
3. **Storage**: Private keys stored locally, never leave the device
4. **Fingerprinting**: SHA-256 hash of Ed25519 public key serves as device identifier

Symmetric Key Management

Data encryption uses AES-256-GCM symmetric keys that are:

- Generated when enabling sync
- Encrypted separately for each device using hybrid KEM
- Distributed through the server in encrypted form
- Rotated manually by the user

Key Storage Architecture


Device Storage
SQLCipher Database AES-256 Encrypted
Device Secure Storage Keychain / Keystore
User Data
Sync Settings
Deltas
Device Private Keys
Symmetric Data Key
Device ID
Database Unlock Secret + Method Indicator

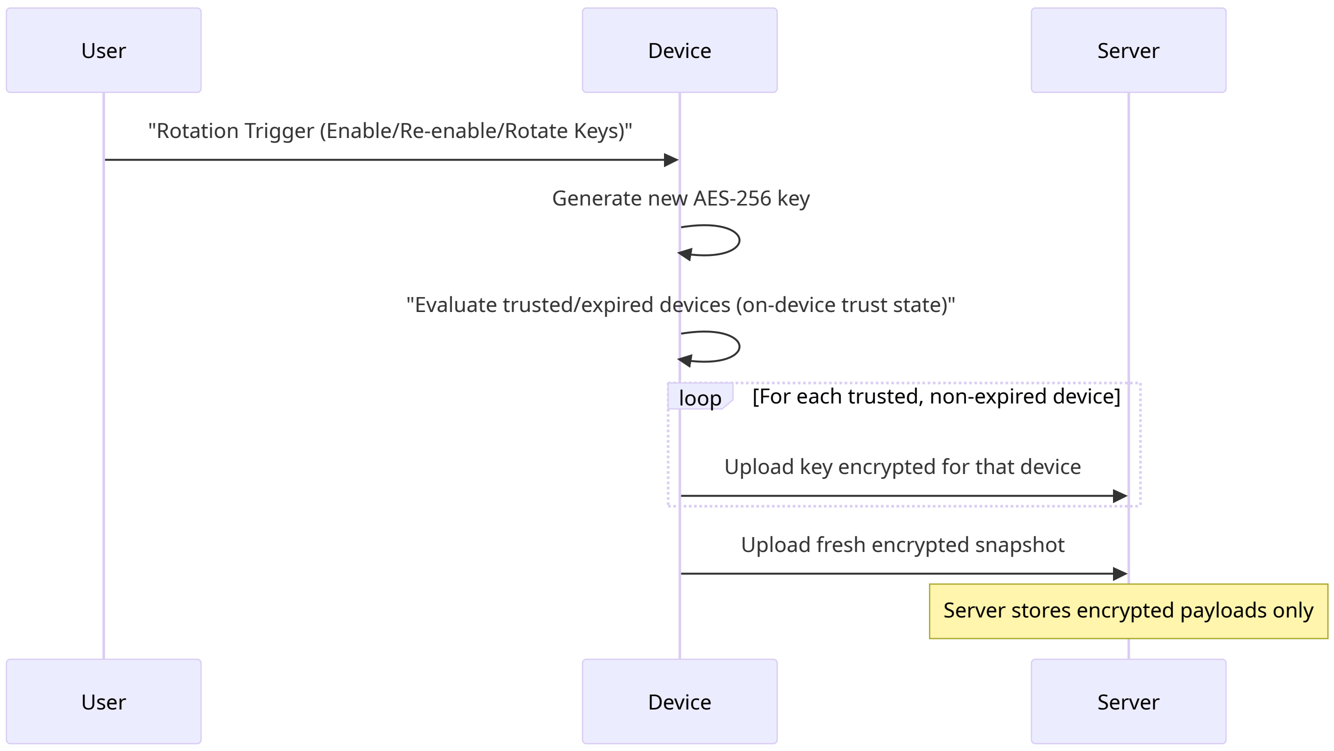
Storage details:

- All application data, device keys, and sync metadata are stored inside the SQLCipher-encrypted database.
- The only material stored outside the database is the database unlock secret and unlock method indicator in OS secure storage (Keychain/Keystore), with optional biometric gating.
- Hardware-backed protection is used where available, otherwise OS-protected secure storage is used.

Key Lifecycle and Rotation

Keys are snapshot-scoped and rotate only on user action (enable/re-enable or manual rotate).

Symmetric Key Rotation and Client Authority


User
Device
Server
"Rotation Trigger (Enable/Re-enable/Rotate Keys)"
Generate new AES-256 key
"Evaluate trusted/expired devices (on-device trust state)"
loop
[For each trusted, non-expired device]
Upload key encrypted for that device
Upload fresh encrypted snapshot
Server stores encrypted payloads only
User
Device
Server

Summary:

- Client filters the local trust list and encrypts the dataset key only for trusted, non-expired devices.
- Server is blind to trust decisions and key contents; it relays encrypted payloads only.
- Manually rotate after removals to revoke future decryption for removed or expired devices.

Rotation can be triggered by enabling sync, re-enabling sync after disabling, or tapping Rotate Keys - the process is identical in all cases.

Device Keys

Device cryptographic keys (Ed25519, X25519, Dilithium5, Kyber1024) are:

- Generated once when device first registers
- Never rotate during device lifetime
- Only replaced by complete device re-registration or recovery

---

5. Device Authentication System

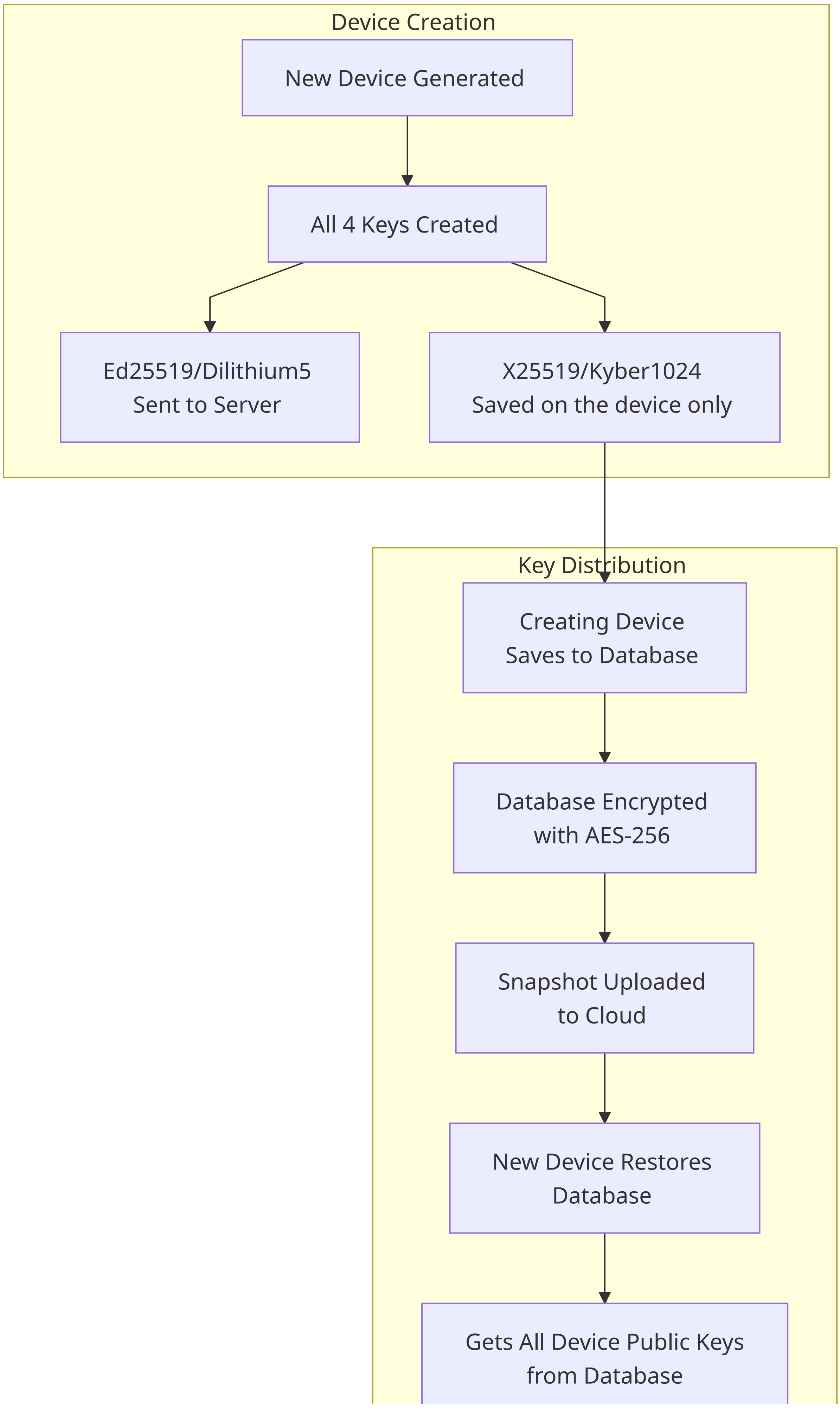
Four-Key Cryptographic System

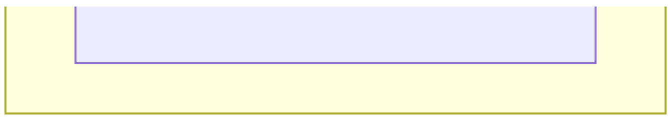
Each device generates and maintains four cryptographic key pairs:

Algorithm	Type	Purpose	Where Stored
Ed25519	Classical	Authentication & Signing	Server (public key) + device
X25519	Classical	Key Exchange (ECDH)	Device only (never sent to server)
Dilithium5	Post-Quantum	Digital Signatures	Server (public key) + device
Kyber1024	Post-Quantum	Key Encapsulation	Device only (never sent to server)

Key Distribution Through Encrypted Database

Critical Security Architecture: The X25519 and Kyber1024 public keys are **never transmitted to the server**. These encryption keys are distributed between devices exclusively through encrypted syncing:

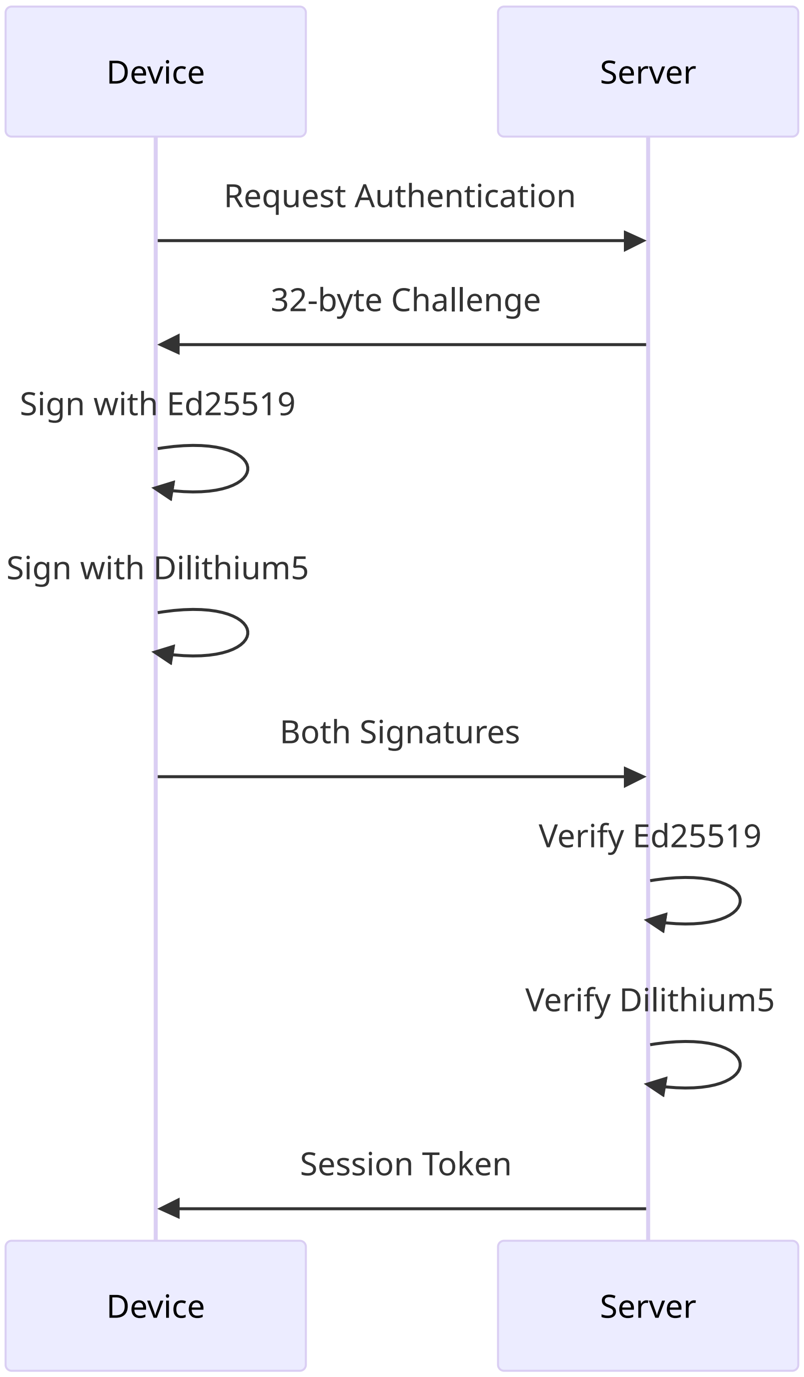

Device Creation
New Device Generated
All 4 Keys Created
Ed25519/Dilithium5
Sent to Server
X25519/Kyber1024
Saved on the device only
Key Distribution
Creating Device
Saves to Database
Database Encrypted
with AES-256
Snapshot Uploaded
to Cloud
New Device Restores
Database
Gets All Device Public Keys
from Database



This architecture provides:

- **No Key Escrow**: Server cannot access private keys or plaintext data
- **Client Authority**: Each client controls its local device trust database
- **Secure Distribution**: Keys shared through encrypted database channel

Dual-Signature Authentication



Challenge-Response Protocol

Summary:

- Server issues a random challenge; device signs it with Ed25519 and Dilithium5.
- Server verifies both signatures against registered public keys and establishes a session on success.
- Challenges are single-use and validated within a short timestamp window to prevent replay.

This dual-signature approach ensures:

- **Current Security**: Ed25519 provides efficient, battle-tested authentication
- **Quantum Resistance**: Dilithium5 protects against future quantum attacks
- **Algorithm Agility**: System can adapt if either algorithm is compromised

Device Roles and Authorization

OdoVault relies on device roles rather than server-side user profiles:

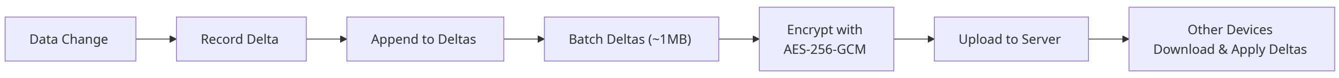
- Client-side user records live only in the encrypted database and sync via deltas for on-device permission enforcement.
- Device roles are assigned by existing clients and sent to the server via explicit client requests. Encrypted deltas propagate role changes to other devices for enforcement; the server stores the role value provided by the client to authorize privileged endpoints (for example, snapshot uploads).
- The server does not decide or modify roles; it enforces the role it has on record.
- Role metadata is minimal and separate from encrypted content; it does not grant the server access to plaintext or keys.

---

6. End-to-End Encryption for Cloud Sync

Deltas

OdoVault uses an event-sourced architecture where all data changes are recorded as deltas and synchronized:



The synchronization process ensures:

- All deltas are captured and transmitted
- Data is encrypted before leaving the device
- Server only stores encrypted delta batches
- Devices must have the current snapshot ID to participate in sync
- AES-256-GCM uses 96-bit random nonces per payload; nonces are included alongside ciphertext

Permission Enforcement on Delta Replay

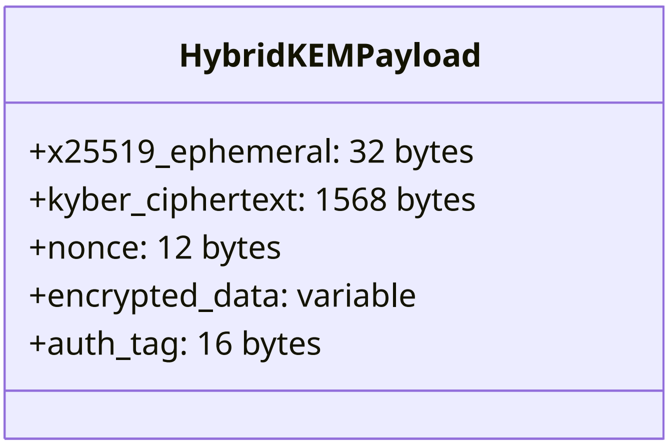
Devices apply role-based permissions when replaying deltas:

- Each delta is associated with its authoring device identity.
- Non-admin devices cannot produce admin-scoped changes; other devices reject deltas that violate permissions.
- Manual database edits by a non-admin user do not propagate because peers refuse unauthorized deltas.

Hybrid Key Encapsulation for Key Distribution

Summary:

- Hybrid KEM combines X25519 ECDH and Kyber1024 encapsulation, mixed with HKDF-SHA256 to derive an AES-256-GCM key used to wrap the dataset key.
- The wrapping key is derived via HKDF-SHA256 from the concatenated X25519 and Kyber shared secrets.

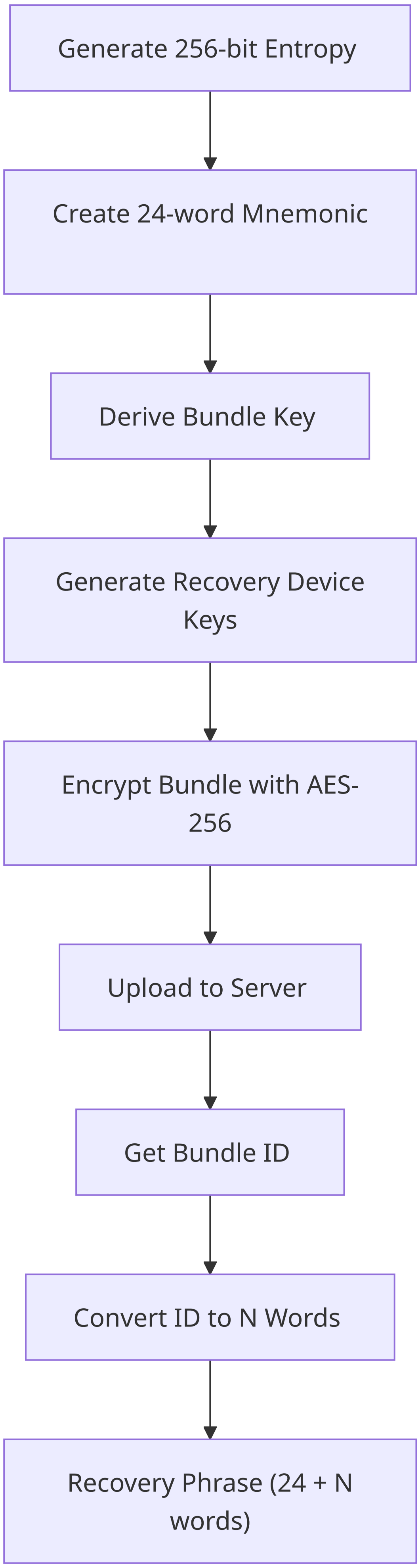

HybridKEMPayload
+x25519_ephemeral: 32 bytes
+kyber_ciphertext: 1568 bytes
+nonce: 12 bytes
+encrypted_data: variable
+auth_tag: 16 bytes

7. Recovery Mechanisms

BIP39 Recovery Phrases

OdoVault uses a recovery phrase composed of a 24-word BIP39 mnemonic plus a variable-length bundle identifier:

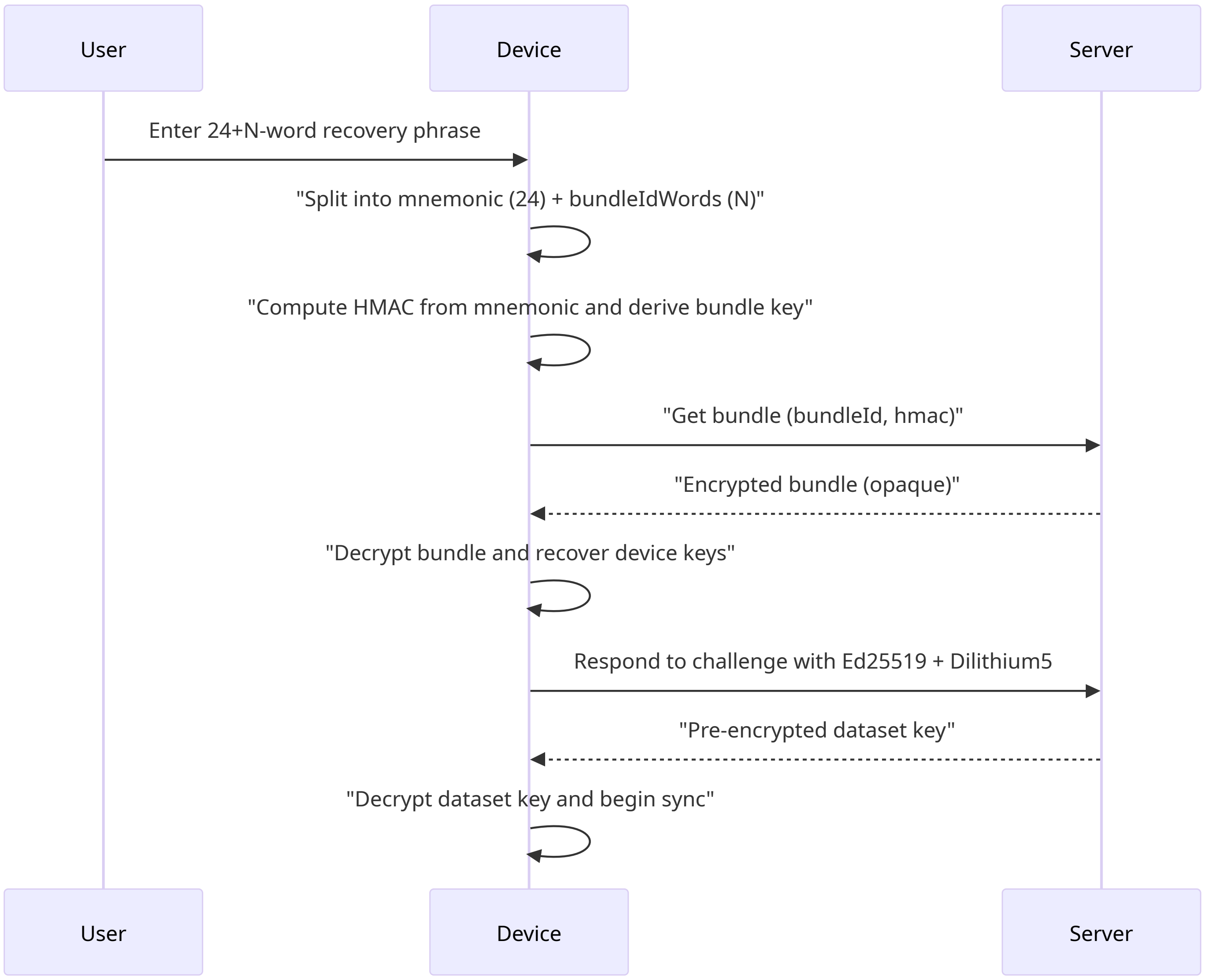
- **24 words**: BIP39 mnemonic encoding 256 bits of entropy (used to derive the bundle encryption key)
- **N words (N ≥ 1)**: Bundle identifier for server lookup (variable number of words depending on the bundle ID)


Generate 256-bit Entropy
Create 24-word Mnemonic
Derive Bundle Key
Generate Recovery Device Keys
Encrypt Bundle with AES-256
Upload to Server
Get Bundle ID
Convert ID to N Words
Recovery Phrase (24 + N words)

Recovery Process

Summary:

- Recover access by entering a 24-word mnemonic plus bundle ID words.
- Device derives the bundle key from the first 24 words, fetches the bundle by (bundleId, hmac), decrypts, proves key possession, and begins sync.
- Proof uses dual signatures (Ed25519 + Dilithium5) to authenticate the recovered device.
- Bundle retrieval is authenticated using HMAC-SHA256 computed with the mnemonic as the key.


User
Device
Server
Enter 24+N-word recovery phrase
"Split into mnemonic (24) + bundleIdWords (N)"
"Compute HMAC from mnemonic and derive bundle key"
"Get bundle (bundleId, hmac)"
"Encrypted bundle (opaque)"
"Decrypt bundle and recover device keys"
Respond to challenge with Ed25519 + Dilithium5
"Pre-encrypted dataset key"
"Decrypt dataset key and begin sync"
User
Device
Server

Security Properties

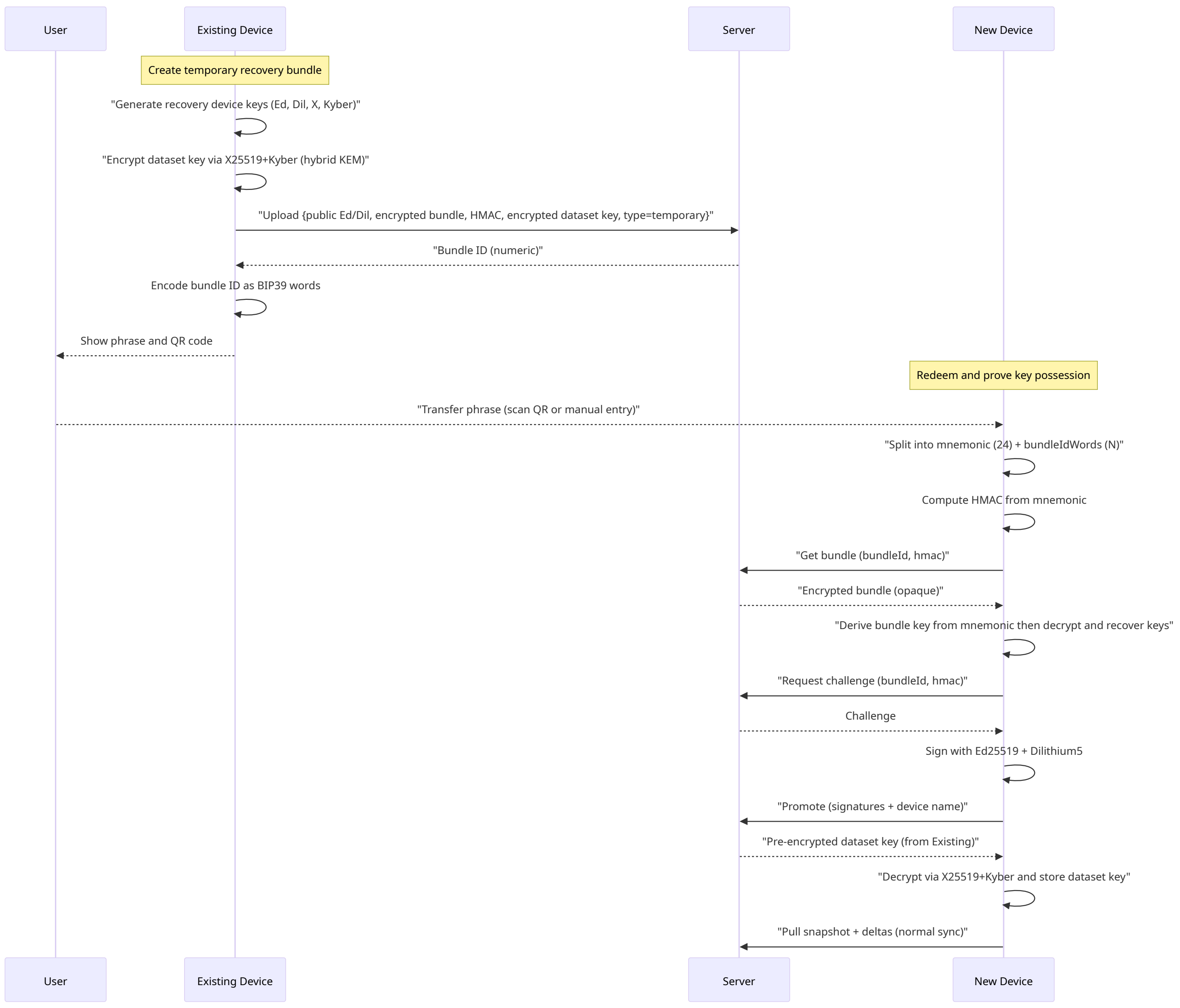
- **Server-Blind**: Server stores encrypted bundle but cannot decrypt
- **One-Time Display**: Phrase shown once during creation

Adding a New Device (Temporary Recovery Bundle)

Adding a new device uses the same cryptographic path as recovery. The existing, authenticated device creates a "temporary recovery device" and a one-time recovery phrase; the new device redeems it and promotes itself to a permanent device by proving key possession. Until promotion, the device is temporary and automatically expires.

Security model:

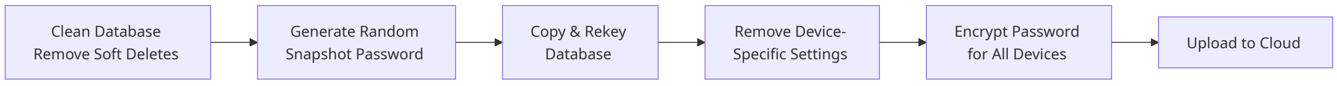
- Same cryptography as recovery: 24-word BIP39 mnemonic + bundle ID words; AES-256-GCM bundle; hybrid X25519+Kyber key wrapping.
- Zero-knowledge preserved: server stores an opaque encrypted bundle and a pre-encrypted dataset key; it never sees private keys or plaintext dataset keys.
- Single-use and time-bound: temporary devices carry an expires_at; if not promoted, they lapse with no access to data.
- Client-controlled distribution: the existing device encrypts the dataset key for the temporary device locally; the server cannot add devices or forge key distribution.
- Out-of-band transfer: user scans a QR code or manually enters the phrase on the new device.


User
Existing Device
Server
New Device
Create temporary recovery bundle
"Generate recovery device keys (Ed, Dil, X, Kyber)"
"Encrypt dataset key via X25519+Kyber (hybrid KEM)"
"Upload {public Ed/Dil, encrypted bundle, HMAC, encrypted dataset key, type=temporary}"
"Bundle ID (numeric)"
Encode bundle ID as BIP39 words
Show phrase and QR code
Redeem and prove key possession
"Transfer phrase (scan QR or manual entry)"
"Split into mnemonic (24) + bundleIdWords (N)"
Compute HMAC from mnemonic
"Get bundle (bundleId, hmac)"
"Encrypted bundle (opaque)"
"Derive bundle key from mnemonic then decrypt and recover keys"
"Request challenge (bundleId, hmac)"
Challenge
Sign with Ed25519 + Dilithium5
"Promote (signatures + device name)"
"Pre-encrypted dataset key (from Existing)"
"Decrypt via X25519+Kyber and store dataset key"
"Pull snapshot + deltas (normal sync)"
User
Existing Device
Server
New Device

8. Snapshot System and Data Migration

Database Snapshots

Snapshots provide clean database states for synchronization:


Clean Database Remove Soft Deletes
Generate Random Snapshot Password
Copy & Rekey Database
Remove Device-Specific Settings
Encrypt Password for All Devices
Upload to Cloud

Snapshot uploads are authorized server-side based on device role. Devices without the admin role cannot upload new snapshots, preventing unprivileged devices from resetting shared state.

Hot-Swap Process

Summary:

- Download encrypted snapshot and password, decrypt locally, swap the database atomically, restore device-specific settings, and resume operations.

This enables cross-device migration, conflict resolution, and clean recovery from corruption.

---

9. Backup Verification and Restore Validation

Backup verification exists because the server is zero-knowledge and cannot validate backups. A backup is only a backup if it can be verified. End-to-end sync is complex and must survive asynchronous, out-of-order delta replays along with schema updates and migrations, so OdoVault treats verification as a security requirement rather than a convenience.

Verification Flow (Implementation)

1. **Quiesce sync**: background sync is paused, and a full sync is performed to ensure the cloud backup is current.
2. **Reject pending changes**: verification aborts if local outgoing changes are still pending.
3. **Fingerprint live database**: compute a full database fingerprint (schema + table hashes).
4. **Create verification device**: generate a temporary recovery bundle, open a separate verification database with a random password, and join the account with that device.
5. **Restore snapshot and deltas**: restore the latest snapshot and apply deltas using the standard restore path, rekeying into the verification database.
6. **Fingerprint restored database**: compute the fingerprint of the restored database and compare to the live fingerprint.
7. **Cleanup and persist status**: remove the temporary device, delete the verification database files, resume sync, and store the verification result.

Fingerprinting Details

The fingerprinting process hashes **full row content** across tables to detect missing data or incorrect replay. If the restored fingerprint does not match the live fingerprint, verification fails and the user can choose to upload a fresh snapshot.

Verification Staleness

Verification status is marked stale after significant sync activity or schema changes so users are encouraged to re-verify. The current implementation flags staleness after a threshold of processed sync chunks (incoming or outgoing) or when the schema version changes.

---

10. Cryptography and Privacy

Cryptographic Algorithms

- **AES-256**: SQLCipher at-rest encryption and AES-256-GCM for sync payloads
- **SHA-256 / HMAC-SHA512**: hashing and tamper detection
- **PBKDF2-HMAC-SHA512**: 256,000 iterations for password derivation
- **HKDF-SHA256**: key derivation
- **Ed25519**: device authentication and signatures
- **X25519**: key exchange
- **Dilithium5**: post-quantum signatures
- **Kyber1024**: post-quantum key encapsulation
- **BIP39**: mnemonic recovery phrases

Privacy Posture

- **Data Minimization**: Server stores only encrypted data; we cannot access user content
- **User Control**: Users manage and can delete their encrypted data via the app
- **Data Locality**: All encryption/decryption happens on the device
- **Consent**: Explicit opt-in for cloud synchronization
- **Payments & Subscriptions**: Payments are processed by third parties (Stripe, Apple App Store, Google Play). We receive subscription status and non-identifying receipt references to validate access. We do not handle or store card numbers, bank details, or billing addresses; any such data remains with the payment provider

---

Conclusion

OdoVault's security architecture provides comprehensive protection for user data through multiple layers of encryption and authentication. The combination of:

- **Local SQLCipher encryption** for at-rest protection
- **Default auto-unlock** with optional biometric or manual password controls

- **Hybrid post-quantum cryptography** for future-proof security
- **Zero-knowledge cloud architecture** for privacy preservation
- **Role-aware authorization** to protect snapshots and delta replay
- **BIP39 recovery phrases** for account recovery

ensures that users maintain complete control over their data while benefiting from secure multi-device synchronization.

The system is designed to withstand both current and future threats, including the advent of quantum computing, while maintaining usability and performance. Through careful implementation of industry-standard cryptographic primitives and a defense-in-depth approach, OdoVault provides enterprise-grade security for personal vehicle data.

Appendix: Cryptographic Parameters

Symmetric Encryption

- **AES-256-GCM**: 256-bit keys, 96-bit nonces, 128-bit authentication tags

Key Derivation

- **PBKDF2**: 256,000 iterations with SHA-512
- **HKDF**: SHA-256 based key derivation

Classical Cryptography

- **Ed25519**: 256-bit private keys, 512-bit signatures
- **X25519**: 256-bit keys, elliptic curve Diffie-Hellman

Post-Quantum Cryptography

- **Dilithium5**: NIST-selected lattice-based signatures
- **Kyber1024**: NIST-selected lattice-based KEM

This security whitepaper is a living document and will be updated as the security architecture evolves. For security concerns or vulnerability reports, please contact the OdoVault security team through responsible disclosure channels.