Key Manager (Atho)

Last refresh: 2026-04-04.

This note explains how Atho?s key manager stores, loads, and uses keys for wallets, mining rewards, and API authentication.

What it does
- Generates Falcon-512 keypairs for wallet/miner addresses.
- Defaults to mnemonic-backed deterministic key generation (atho-mnemonic-v1) with 24 words (12/24/48 supported).
- Derives hashed public keys (HPK) and Base56 addresses for display/UX.
- Persists keys atomically to Keys/KeyManager_Public_Private_Keys.json (one file per node/instance).
- Supplies keys to transaction signing, mining coinbase outputs, and API auth where needed.
- Imports existing Falcon keys (see below) with hash/base56 verification and a test-sign befor storing.

Storage layout
- Path (by default): Keys/KeyManager_Public_Private_Keys.json
- Structure:
- miner_1, miner_2, ?: labels for each managed key.
- public_key: Falcon public key (stored as normalized hex; canonical policy target is 897 byte legacy 1024-byte compatibility may still be accepted by current policy).
- private_key: Falcon private key components (F, G, f, g).
- mnemonic_phrase, mnemonic_words, mnemonic_text, mnemonic_meta for mnemonic-derived keys.
- Metadata: creation time, role (e.g., miner), network.
- Files are written atomically to avoid partial writes and corruption.

Encrypted-at-rest lockbox mode
- Key file now supports encrypted-at-rest mode with one-password unlock flow.
- Kyber DEK wrap is mandatory in lockbox mode (no Kyber opt-out).
- Lockbox unlock modes:
- default: password-only UX (Kyber secret wrap stored in wallet metadata).
- advanced: password + Kyber unlock text file import on unlock (.txt bundle).
- Encryption stack:
- password KDF: Argon2id
- payload cipher: AES-256-GCM
- DEK wrap: AES-256-GCM
- required PQ wrap: Kyber (kyber1024_ref by default) wrapping the same DEK for hybrid/PQ backu integrity.
- Plaintext fields (kept visible for wallet UX while locked):
- identifier, role, network, defaults, hashed_public_key, address_base56, source/version.
- Encrypted fields:
- raw public_key / cli_public_key
- all private key parts (f/g/F/G)
- mnemonic fields (mnemonic_phrase, mnemonic_words, mnemonic_text, mnemonic_meta).
- Security metadata is stored in _wallet_security (schema athokey-lockbox-v1).
- Advanced-mode Kyber unlock file format:
- schema: athokey-kyber-unlock-v1
- includes strict integrity hash and binding metadata
- importer fails closed on tampered/invalid format.

Kyber + AES-256 together (why both)

Key point: Kyber does not replace AES for payload encryption. They serve different roles.
- AES-256-GCM role:
- Encrypt the actual wallet secret payload (private key parts + mnemonic data).
- Provide confidentiality + integrity tag over the payload.
- Kyber role:
- Wrap/encapsulate the DEK used by AES payload encryption.
- Provide a post-quantum recovery/unlock control plane for key material.

Practical model:
- Generate random DEK.
- Encrypt wallet secret payload with AES-256-GCM(DEK).
- Wrap DEK with password-derived KEK (Argon2id + AES-256-GCM wrap).
- Also wrap the same DEK with Kyber KEM metadata/ciphertext.
- Store encrypted payload + both wrap records in lockbox metadata.

This gives:
- fast encryption performance (AES),
- password UX for daily use,
- PQC-capable DEK recovery path (Kyber),
- and policy flexibility to tighten one plane without rewriting payload format.

How keys are used
- Wallet CLI: signs transactions with the selected key; displays Base56 addresses derived from
HPK.
- Miner: uses the configured key for coinbase rewards and block signing.
- API: keys are separate (see Src/Config/Api_Keys.json), but the key manager feeds addresses f
wallet/miner actions.
- Wallet management API (added): POST /wallet/rename, POST /wallet/delete, POST /wallet/defaul
let the GUI rename, delete, or set a default key (one default per network). They resolve
identifiers by id/base56/HPK and refuse to delete the last key in a network.
- Wallet lockbox API (added):
- GET /wallet/unlock_status
- POST /wallet/unlock
- POST /wallet/lock
- POST /wallet/change_password
- POST /wallet/export_kyber_unlock_file
- POST /wallet/export_encrypted_backup
- POST /wallet/import_encrypted_backup
- GET /wallet/lock_policy
- POST /wallet/lock_policy
- sensitive wallet operations (create/recover/import/export/sign/send/rename/delete/default) r
fail closed with wallet_locked when encrypted and locked.
- unlock endpoint includes brute-force controls (per-client backoff + temporary lockout after
repeated failures).

Safety notes
- The JSON file can run unencrypted (legacy) or encrypted lockbox mode.
- In lockbox mode, private/public raw key material and mnemonic data are encrypted and
unavailable until unlock.

- Mnemonic passphrases are never stored in key JSON or export files; keep passphrases separately.
- Wallet encryption passwords now require non-empty input (no fixed minimum length enforced by backend).
- Wallet exports are .txt and are marked read-only on save.
- Each node/container should have its own Keys directory; don?t share across independent nodes unless you intend to reuse the same addresses.
- For production, add OS-level permissions (e.g., chmod 600 Keys/*.json) and consider an encrypted keystore if you extend the system. A simple AES-GCM+scrypt proof-of-concept CLI exis at Src/Accounts/ecript.py for manual encryption/decryption tests.
- Key file writes are lock-protected (<key_file>.lock) to reduce concurrent-write corruption risk.
- Saves use backup-on-write and keep rolling backups in Keys/backups/ before replacing the active key JSON.
- During plaintext -> encrypted migration, KeyManager no longer creates a new plaintext pre_write backup and now purges any existing plaintext-secret backups under Keys/backups/.
- Mnemonic sidecar files (Keys/mnemonic_recovery/...) are automatically removed once wallet lockbox encryption is enabled.
- Startup/disk reload now applies strict schema validation and fails closed on malformed key files instead of silently proceeding with bad structure.
- Wallet backup/import paths for encrypted snapshots are validated under the keys root and use the same atomic/backup controls.
- Unlock auto-relock policy:
- default unlock window is 30 seconds (_wallet_policy.unlock_ttl_seconds),
- 0 disables auto-relock timer,
- lock status includes unlock_expires_in_seconds.
- API unlock session context now persists across calls while key-file fingerprint is unchanged (prevents accidental relock-on-every-request).
- wallet_unlock_status no longer exposes absolute key-file paths by default.

Addresses and HPK
- Public keys are hashed (SHA3-384) ? HPK ? encoded to Base56 for display.
- Base56 avoids ambiguous characters and is used consistently across CLI and wallet displays.
- Transaction witness wire format uses canonical base64 for pubkey/signature; KeyManager still operates on internal hex/bytes for Falcon signing and verification.

Defaults and generation
- At load, the manager ensures at least miner_1 and miner_2 exist on mainnet, testnet, and regnet; defaults are set to miner_1 if missing.
- Canonical miner slots are miner_1 and miner_2; additional miner identifiers (miner_3+) are still supported and preserved.
- Only CLI-format keys are kept. Legacy/non-CLI records are regenerated as CLI keys.
- New key generation defaults to mnemonic-backed derivation with 24 words unless explicitly overridden.
- CLI key-creation/import/recovery flow now prompts (interactive TTY only) to enable encryptio before proceeding when wallet is still unencrypted.
- Changing the default key updates all roles for that network and is enforced by the API and GUI.

Recovery audit trail
- Mnemonic recovery operations are written to:
- logs/.../wallet/wallet_recovery_audit.log

- Audit records include timestamp, network, identifier, canonical source, and canonical flag s
recovery decisions are traceable.

Importing existing keys (wallet/CLI)
You can import a Falcon keypair instead of pasting long values into the terminal:

Supported file formats:
- Wallet/GUI import: Atho wallet export .txt (current structured text export format is
supported; legacy warning-header export format is still accepted for backward compatibility).
- CLI helper (change_default_key.py import): still accepts JSON with fields public_key, f, g,
G, optional hpk/address_base56 (HPK/Base56 are recomputed when present).

Import safeguards:
- Rejects/ignores duplicates by HPK or Base56; identifier collisions are auto-renamed (import_
import_2, ?) when HPK/Base56 are unique.
- Recomputes HPK and ensures provided HPK/Base56 matches.
- Runs a test sign before persisting.
- Requires at least one key per network; deletion is blocked only if it would remove the last
key.
- GUI export enforces read-only .txt; imports are no longer hard-coupled to one literal
warning-header line.

Exporting keys (wallet) for offline storage
- From the wallet tab, use the export/download icon to save a single key to a .txt file. The
file:
- Is forced to .txt and marked read-only on save.
- Contains private/public key material, HPK/Base56, and mnemonic phrase/word list sections whe
the key is mnemonic-derived.
- Explicitly warns that mnemonic passphrase is not stored.
- Offline workflow:
- Export the desired key(s) to .txt and store them securely (offline/backup).
- When re-importing later, use the wallet import button (or import_export_file in KeyManager)
load the .txt.
- Imports reject duplicate HPK/Base56; identifier clashes are auto-renamed to import_#.
- Defaults are reassigned as needed; deletion is blocked only if it would leave zero keys on
that network.
- For extra protection, you can manually encrypt exported files using the AES-GCM + scrypt
helper at Src/Accounts/ecript.py before storing them offline.

GUI actions backed by the key manager
- Wallet tab actions (copy, rename, delete, set default/star) call the wallet API endpoints
above.
- Hover tooltips and click animations are provided for clarity; the star indicates the current
default.