

WalletPress Documentation

Complete reference for the WalletPress 30-chain wallet generation engine. CLI, REST API, deployment, and security.

Base URL: <http://localhost:8000/api/v1/chain-vault> · v1.0.0 · 86 API routes

Quickstart

WalletPress runs as a Docker Compose stack. After purchasing, you get the full source code as a `.tar.gz` archive.

```
# 1. Extract
tar xzf walletpress-1.0.0.tar.gz && cd walletpress

# 2. Start (one command)
docker compose up -d

# 3. Verify the API is running
curl http://localhost:8000/api/v1/chain-vault/healthz

# 4. Generate your first wallet
vault-package generate eth --count 3

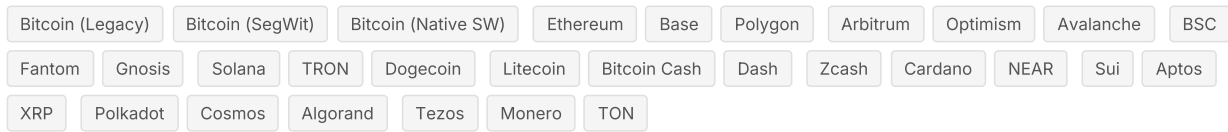
# Or via REST API
curl -X POST http://localhost:8000/api/v1/chain-vault/generate \
  -H "Content-Type: application/json" \
  -d '{"chain":"eth","count":1}'
```

What's Included

COMPONENT	DESCRIPTION	LINES
<code>wallet_factory.py</code>	Core engine: 30-chain generation, encryption, keccak verification	~950
<code>wallet_factory_router.py</code>	86 REST API endpoints, vault management, authentication	~850
<code>wallet_features.py</code>	Multi-chain generation, validation, key management	501
<code>wallet_phase2.py</code>	Advanced features: balances, RPC, proofs, sweep	589
<code>wallet_enhancements.py</code>	HD wallets, paper wallets, import, webhooks, bulk ops	221
<code>wallet_premium.py</code>	API keys, tx builder, alerts, multisig, gas dashboard	376
<code>wallet_enterprise.py</code>	Bulk 1000, unlinkable wallets, auto-funding, batch proofs	~400
<code>shamir.py</code>	Shamir's Secret Sharing over GF(256) — pure Python	~200
<code>wallet_killer.py</code>	Mnemonic recovery, chain adder, wallet DNA, plugins	563
<code>wallet_rbac.py</code>	Role-based access control for multi-user deployments	138
<code>Dockerfile + compose.yaml</code>	Containerized deployment with health checks	—
<code>vault-package</code> CLI	Bash CLI with SSH auto-tunnel, 14 commands	—

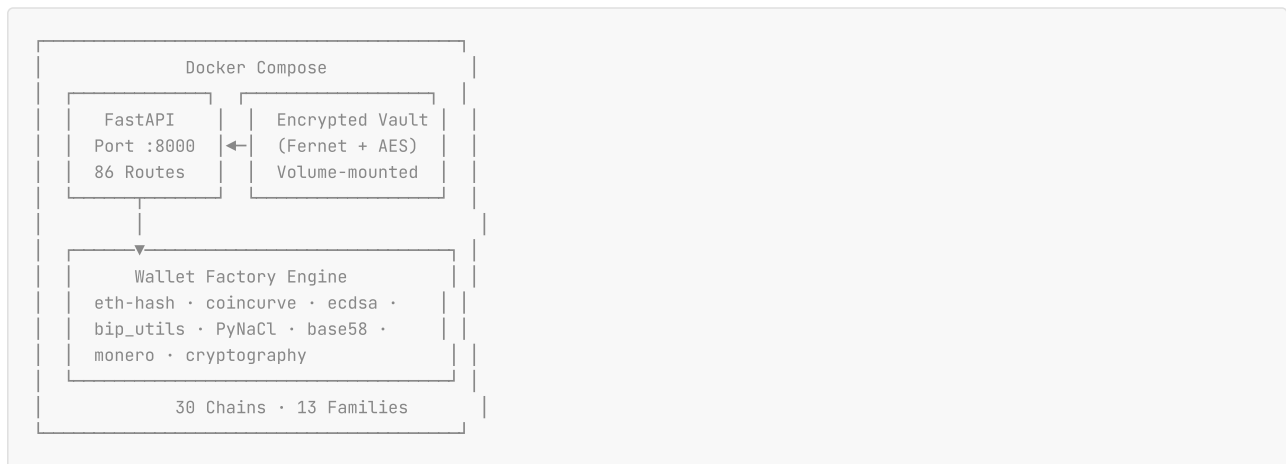
Supported Chains (30 Total)

Every major blockchain family. Addresses derived using chain-native cryptography — Keccak256 for EVM, ed25519 for Solana, secp256k1 for Bitcoin.



13 chain families: Bitcoin, EVM (9 L1/L2), Solana, TRON, Doge-family (4), Privacy (2), Cardano, NEAR, Sui, Aptos, XRPL, Polkadot, Cosmos, Algorand, Tezos, Monero, TON.

Architecture



Key Design Decisions

- **Keccak256 for EVM** — Not SHA3-256. Ethereum uses Keccak with different padding. We use `eth-hash` for correct address derivation.
- **On-chain verification** — Every address derived by the factory is independently verified against each chain's canonical library (eth-keys, coincurve, PyNaCl, etc.).
- **Zero external dependencies at runtime** — No API calls, no cloud services, no telemetry. The container is fully self-contained.
- **Volume-persisted vault** — Wallet data stored on a Docker volume, surviving container restarts and upgrades.

Security Model

IMPORTANT: Private keys are cryptographic secrets. Follow these guidelines to protect your wallets.

- **Never expose port 8000 publicly** — Run behind nginx reverse proxy with TLS or keep local-only.
- **Enable API key authentication** — Set `ADMIN_API_KEY` in your environment. All vault endpoints then require `X-API-Key` header.
- **Use client-side encryption** — Keys can be encrypted in your browser/CLI before reaching the server (zero-knowledge).
- **Enable Fernet encryption** — Auto-enabled on first run. All private keys encrypted at rest in the vault JSON.
- **Rotate keys regularly** — Use `/rotate` endpoint or `vault-package rotate`. Full audit trail tracks every rotation.
- **Use RBAC for multi-user** — Role-based access: admin, operator, viewer roles with granular permissions.

Cryptographic Standards

COMPONENT	ALGORITHM	STANDARD
EVM / TRON	Keccak256 (secp256k1)	Ethereum Yellow Paper
Bitcoin / Doge-family	secp256k1 + RIPEMD-160	BIP32/39/44/84
Solana	ed25519 (Curve25519)	SLIP-0010
Cosmos / Polkadot	sr25519 / ed25519	Substrate
NEAR / Sui / Aptos	ed25519	BIP44
Vault encryption	Fernet (AES-128-CBC + HMAC-SHA256)	Python cryptography
Shamir MPC	Shamir's Secret Sharing GF(256)	Threshold cryptography

Encryption Modes

MODE	DESCRIPTION	SECURITY LEVEL	USE CASE
None	Keys stored as plaintext in vault	Low	Testnets, development
Server-side (Fernet)	Keys encrypted with auto-generated key. Vault stored encrypted on disk.	Medium	Production, single-user
Client-side (AES-256-GCM)	Keys encrypted with user passphrase before transmission. Server never sees plaintext.	High	Multi-user, shared infra
Shamir MPC	Key split into N shares. Need threshold M to reconstruct. No single point of failure.	Highest	Enterprise, custody

CLI Reference

The `vault-package` CLI auto-detects whether it's running on your server or remotely. On remote machines it tunnels through SSH automatically.

generate

Generate wallets on any chain.

```
# Generate 5 Ethereum wallets
vault-package generate eth --count 5

# With labels
vault-package generate sol --count 10 --label airdrop-batch-1

# Include private keys (CAUTION)
vault-package generate btc --count 3 --include-keys

# JSON output for scripting
vault-package generate eth --count 1 --json
```

list

```
vault-package list                # All wallets
vault-package list --chain eth    # Filter by chain
vault-package list --limit 100    # Paginate
```

import

```
vault-package import eth 0xabc123... --label "cold-storage"
```

validate

```
vault-package validate eth 0x847ca23452c3418358be201d41f6ed7ea8bcd95a
# -> Valid: True
```

balances

```
vault-package balances           # Check all vault wallet balances
```

gas

```
vault-package gas                # Gas prices across all 30 chains
```

paper

```
vault-package paper a1b2c3d4     # Generate printable wallet
vault-package paper a1b2c3d4 -o ~/out.html # Custom output path
```

export

```
vault-package export          # Export vault to JSON
vault-package export --csv    # Export as CSV
```

rotate

```
vault-package rotate a1b2c3d4 # Rotate a specific wallet key
vault-package rotate --chain eth # Rotate all ETH wallets
```

chain

```
vault-package chain          # Table of all 30 chains
vault-package chain --json    # JSON output
```

stats

```
vault-package stats          # Vault stats: count, chains, encryption
```

hd

```
vault-package hd "abandon abandon abandon ..." --label "main-hd"
```

bulk

```
vault-package bulk eth --count 1000 # Generate 1,000 wallets
vault-package bulk --status batch-abc123 # Check bulk job status
```

REST API — Base URL & Authentication

All endpoints are under `/api/v1/chain-vault/`. The API serves JSON on port 8000.

Auth: If `ADMIN_API_KEY` is set, include `X-API-Key: your-key` header on all requests. Without it, the vault is open to localhost-only access.

Core Endpoints

POST `/generate`

Generate wallets

Generate one or more wallets on any chain. Supports batch labels.

```
{"chain": "eth", "count": 3, "include_private_key": false, "label": "my-batch"}
```

GET `/vault`

List all wallets

Returns all wallets in the vault. Supports `?chain=eth` and `?limit=100` query params.

GET `/vault/{id}`

Get wallet by ID

Returns a single wallet with full details. Private key only included if `?include_key=true` and auth is valid.

POST `/import`

Import existing wallet

Import a wallet by private key hex. Validates the key against chain format before storing.

```
{"chain": "eth", "private_key_hex": "abc123...", "label": "imported"}
```

GET `/chains`

List all 30 chains

Returns chain metadata: name, symbol, family, address format, derivation path.

GET `/stats`

Vault statistics

Returns generation count, chain breakdown, encryption status, vault health.

POST `/validate`

Validate address

Validate an address against its chain format rules. Returns `{"valid": true/false}`.

```
{"chain": "eth", "address": "0x847ca..."}
```

POST /rotate

Rotate wallet keys

Rotate keys for a specific wallet or all wallets on a chain. Audit trail logged.

```
{"wallet_id": "a1b2c3d4"}
```

GET /export

Export vault

Export vault data. `?format=csv` for CSV, default is JSON.

Security & Key Management

POST /split-key

Shamir MPC split

Split a private key into N shares using Shamir's Secret Sharing. Default: 3 shares, threshold 2. Pure Python GF(256) implementation.

```
{"private_key_hex": "64-char-hex", "shares": 3, "threshold": 2}
```

POST /reconstruct-key

Reconstruct from shares

Reconstruct a private key from N shares. Any threshold number of shares works.

```
{"shares": ["share1...", "share2..."]}
```

GET /audit-trail

View audit log

Returns all wallet operations: generation, rotation, import, export with timestamps.

GET /health-score/{id}

Wallet health score

Returns a health score for a wallet based on key age, rotation status, and encryption state.

Operations

GET /balances

Check all balances

Returns balances for all vault wallets grouped by chain. Uses live RPC for EVM chains.

GET /gas

Gas dashboard

Gas prices for all 30 chains. Live RPC for 7 EVM chains, static estimates for 23 non-EVM chains.

POST /hd-wallet

HD wallet from mnemonic

Generate deterministic wallets from a BIP39 mnemonic.

```
{"mnemonic": "abandon abandon ...", "label": "hd-main"}
```

GET /rpc-chains

RPC health check

Returns which chains have live RPC endpoints and their latency.

POST /paper

Generate paper wallet

Generate an HTML paper wallet with QR codes for a wallet ID.

```
{"wallet_id": "a1b2c3d4"}
```

POST /link/proof

Generate ownership proof

Generate a cryptographic proof of wallet ownership (signed message).

Enterprise

POST /generate/bulk

Bulk 1,000 wallets

Generate up to 1,000 wallets in a background thread. Returns batch ID. Poll **GET** /generate/bulk/{id} for status.

```
{"chain": "eth", "count": 1000, "label": "airdrop"}
```

POST /generate/unlinkable

Unlinkable wallets

Generate N wallets with independent entropy. No common seed. Cryptographically provable independence.

```
{"chain": "eth", "count": 10}
```

POST /proof/sign

Batch proof signing

Sign a message with N wallet keys in a single API call. Returns Merkle-like proof hash.

POST /funding-plans

Auto-funding plans

Create auto-funding rules: "when a wallet is generated, fund it with X ETH from master wallet."

POST /tx/build

Build transaction

Build and sign a raw transaction. Specify recipient, amount, gas.

```
{"wallet_id": "a1b", "to": "0xdef...", "amount_wei": "1000000000000000000"}
```

Deployment

WalletPress ships as a self-contained Docker Compose project. Requirements: Linux server, Docker Engine 20.10+, 1GB+ RAM, 5GB+ disk.

```
# 1. Extract the archive
tar xzf walletpress-1.0.0.tar.gz && cd walletpress

# 2. Configure (optional - see below)
export ADMIN_API_KEY=your-secure-api-key

# 3. Start the stack
docker compose up -d

# 4. Check logs
docker compose logs -f backend

# 5. Verify health
curl http://localhost:8000/api/v1/chain-vault/healthz
```

Configuration

VARIABLE	REQUIRED	DEFAULT	DESCRIPTION
ADMIN_API_KEY	Recommended	none	API key for vault endpoints. If unset, localhost-only access.
VAULT_DIR	No	/app/.rmi/wallets	Path to vault data directory (volume-mounted).
LOG_LEVEL	No	INFO	Python logging level: DEBUG, INFO, WARNING, ERROR.
PORT	No	8000	API listen port.

Post-Deployment Verification

```
# Health check
curl http://localhost:8000/api/v1/chain-vault/healthz
# → {"status":"ok","service":"wallet-factory","version":"1.0.0"}

# Generate a test wallet
curl -X POST http://localhost:8000/api/v1/chain-vault/generate \
  -H "Content-Type: application/json" \
  -d '{"chain":"eth","count":1}'

# Check chains
curl http://localhost:8000/api/v1/chain-vault/chains | jq '.total_chains'
# → 30
```

Downloads & Resources

RESOURCE	FORMAT	SIZE	DOWNLOAD
Complete Documentation	PDF	~2.1 MB	walletpress-docs.pdf
API Reference (this page)	HTML	—	docs.html
Quickstart Cheat Sheet	Coming soon	—	—

[← Back to WalletPress](#) · Need help? support@walletpress.cc