Cryptnox CLI Manual

Release 1.0.5

June 20, 2026

Contents

1 Cryptnox CLI Overview

`cryptnox-cli` is a command-line interface for managing **Cryptnox Smart cards**, enabling secure seed initialization and cryptographic signing for **Bitcoin** and **Ethereum**.

1.1 Supported Hardware

- **Cryptnox Smart cards**
- **Standard PC/SC Smart card Readers**: either USB NFC reader or a USB smart card reader

Get your card and readers here: shop.cryptnox.com

1.2 Installation

> **Note**
>
> This is only a minimal setup. Additional packages may be required depending on your operating system.

1.2.1 From PyPI

```
pip install cryptnox-cli
```

1.2.2 From source

```
git clone https://github.com/cryptnox/cryptnox-cli.git
cd cryptnox-cli
pip install .
```

This installs the package and makes the `cryptnox` command available (if your Python installation is in your system `PATH`).

1.3 Quick Usage Examples

> **Note**
>
> The examples below are only a subset of available commands. The complete list of commands and detailed usage instructions is described in the *Command Line Interface* section.

1.3.1 1. Dual initialization

1. Factory reset each card:

`cryptnox reset` → enter PUK → verify reset.

2. Initialize each card:

   `cryptnox init` → (optional) set name/email → set **PIN** (4–9 digits) → set or generate **PUK** → verify init.

3. Run dual seed procedure:

   `cryptnox seed dual` — follow prompts: insert Card A (enter PIN), swap to Card B (enter PIN), swap back as requested.

1.3.2 2. Sign and send a Bitcoin transaction

1. Create or obtain a raw unsigned transaction externally.
2. Run the signing & send command:

   ```
   cryptnox btc send <recipient_address> <amount> [-f <fees>]
   ```

1.3.3 3. Change PIN code

1. Run command: `cryptnox change_pin`
2. Enter current PIN → enter new PIN → verify change.
3. Check with `cryptnox info` using new PIN (BTC & ETH accounts displayed).

1.3.4 4. Get extended public key (xpub)

1. Run command: `cryptnox get_xpub`
2. Enter **PIN** → enter **PUK**
3. The card returns the **xpub**

1.4 License

cryptnox-cli is dual-licensed:

- **LGPL-3.0** for open-source projects and proprietary projects that comply with LGPL requirements
- **Commercial license** for projects that require a proprietary license without LGPL obligations (see COMMERCIAL.md for details)

For commercial inquiries, contact: contact@cryptnox.com

2 Command Line Interface

The `cryptnox` command provides a comprehensive CLI for managing Cryptnox cards and performing cryptocurrency operations.

2.1 Installation

After installing the cryptnox-cli package, the `cryptnox` command becomes available in your system:

```
pip install cryptnox-cli
```

2.2 Basic Usage

```
cryptnox [OPTIONS] COMMAND [ARGS]...
```

2.3 Global Options

-v, **--version**
: Show the version and exit.

--verbose
: Turn on logging for detailed output.

-s, **--serial** SERIAL
: Serial number of the card to be used for the command.

--port PORT
: Define port to enable remote feature.

2.4 Commands Overview

2.4.1 Card Management Commands

list

List all connected Cryptnox cards.

```
cryptnox list
```

init

Initialize a Cryptnox card with owner information and PIN/PUK codes.

```
cryptnox init [OPTIONS]
```

Options:

- `-e, --easy_mode`: Initialize card in easy mode (sets PIN and PUK to all zeros)

reset

Reset the card to factory defaults.

```
cryptnox reset
```

Warning: This will erase all data on the card.

info

Display default accounts information for the card.

```
cryptnox info
```

Shows:

- Card serial number and type
- Ethereum and Bitcoin addresses
- Public keys
- Derivation information

cert

Retrieve and display the manufacturer certificate from the card in a human-readable format.

```
cryptnox cert
```

Features:

- Retrieves the full manufacturer certificate from the card
- Displays certificate details (Issuer, Subject, Validity, Public Key, Signature)
- Human-readable format similar to `openssl x509 -text`

2.4.2 Seed Management Commands

seed chip

Generate new root key directly in the card's secure chip.

```
cryptnox seed chip
```

Note: The seed never leaves the card and cannot be backed up.

seed dual

Generate the same seed on two cards for redundancy.

```
cryptnox seed dual
```

Requirements: Two initialized Cryptnox cards

Process:

1. Generate seed on first card
2. Swap cards
3. Load same seed on second card
4. Results in two cards with identical keys

seed recover

Recover a wallet from an existing BIP39 mnemonic phrase (12 or 24 words).

```
cryptnox seed recover
```

Input Required:

- Existing BIP39 mnemonic (12 or 24 words)
- Optional: BIP39 passphrase (13th/25th word)

Use Cases:

- Restoring existing wallet
- Migrating wallet from another device
- Recovering from backup mnemonic

Important: If the original wallet used a BIP39 passphrase, you **must** provide the same passphrase during recovery.

seed upload

Generate new random seed, upload to card, and display BIP39 mnemonic for backup.

```
cryptnox seed upload
```

Features:

- Generates new 32-byte random seed using card's hardware RNG
- Converts to BIP39 mnemonic (12 or 24 words)
- Optional: Add BIP39 passphrase (13th/25th word) for extra security
- Displays mnemonic for manual backup

Output: The generated mnemonic phrase (save it securely!)

Important: If you use a BIP39 passphrase, you **must** remember it. It cannot be recovered.

2.4.3 Security Commands

change_pin

Change the PIN code of the card.

```
cryptnox change_pin
```

Requirements: Current PIN code

change_puk

Change the PUK (PIN Unblocking Key) code of the card.

```
cryptnox change_puk
```

Requirements: Current PUK code

unlock_pin

Unlock a card with blocked PIN using the PUK code and set a new PIN.

```
cryptnox unlock_pin
```

Requirements: PUK code

Note: PIN becomes blocked after multiple failed attempts.

2.4.4 Card Configuration

card_conf

Show or modify card configuration settings.

```
cryptnox card_conf [KEY] [VALUE]
```

Available Settings:

- `pinless`: Enable/disable PIN-less path
- `pin`: Enable/disable PIN authentication

Values: `yes` or `no`

Examples:

```
# Show current configuration
cryptnox card_conf

# Enable PIN-less path
cryptnox card_conf pinless yes

# Disable PIN-less path
cryptnox card_conf pinless no
```

2.4.5 Bitcoin Commands

btc send

Send Bitcoin to an address.

```
cryptnox btc send ADDRESS AMOUNT [OPTIONS]
```

Arguments:

- `ADDRESS`: Bitcoin address (P2PKH, P2SH, or Bech32)
- `AMOUNT`: Amount in BTC

Options:

- `-n, --network {mainnet,testnet}`: Network to use
- `-f, --fees SATOSHIS`: Transaction fees in satoshis per byte

Example:

```
cryptnox btc send 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa 0.001 -n mainnet -f 10
```

btc config

View or modify Bitcoin configuration.

```
cryptnox btc config [KEY] [VALUE]
```

Settings:

- `network`: mainnet or testnet
- `derivation`: Key derivation method

2.4.6 Ethereum Commands

eth send

Send Ether (ETH) to an address.

```
cryptnox eth send ADDRESS AMOUNT [OPTIONS]
```

Arguments:

- `ADDRESS`: Ethereum address (0x...)
- `AMOUNT`: Amount in ETH

Options:

- `-n, --network {mainnet,sepolia,goerli}`: Network to use
- `--price GWEI`: Gas price in Gwei
- `--limit GAS`: Gas limit

Example:

```
cryptnox eth send 0x742d35Cc6634C0532925a3b844Bc9e7595f0bEb 0.1 -n mainnet --
↪price 20
```

eth config

View or modify Ethereum configuration.

```
cryptnox eth config [KEY] [VALUE]
```

Settings:

- `network`: mainnet, sepolia, or other networks
- `derivation`: Key derivation method

- `api_key`: Etherscan/Infura API key
- `endpoint`: RPC endpoint

2.4.7 ERC Token Commands

eth erc20 init

Initialize ERC-20 token contract for transactions.

```
cryptnox eth erc20 init CONTRACT_ADDRESS [OPTIONS]
```

eth erc20 info

Display information about configured ERC-20 tokens.

```
cryptnox eth erc20 info
```

eth erc20 send

Send ERC-20 tokens.

```
cryptnox eth erc20 send TOKEN_ADDRESS RECIPIENT_ADDRESS AMOUNT [OPTIONS]
```

transfer

Transfer ERC-20 or ERC-721 tokens.

```
cryptnox transfer ADDRESS AMOUNT [OPTIONS]
```

Options:

- `--price GWEI`: Gas price
- `--limit GAS`: Gas limit

2.4.8 Configuration Commands

config

List or modify blockchain configurations.

```
cryptnox config [SECTION] [KEY] [VALUE]
```

Examples:

```
# Show all configuration
cryptnox config

# Show Ethereum configuration
cryptnox config eth

# Set Ethereum network
cryptnox config eth network mainnet
```

```
# Set Bitcoin derivation
cryptnox config btc derivation DERIVE
```

2.4.9 History and Information

history

List performed signatures and transactions.

```
cryptnox history [PAGE]
```

Arguments:

- `PAGE`: Page number to display (default: 1)

Note: Shows up to 148 entries, 25 per page.

2.4.10 Advanced Commands

get_xpub

Get extended public key (xpub) for hierarchical deterministic wallets.

```
cryptnox get_xpub [OPTIONS]
```

Options:

- Derivation path
- Key type

get_clearpubkey

Get clear (uncompressed) public key from the card.

```
cryptnox get_clearpubkey [OPTIONS]
```

decrypt

Decrypt data using the card's private key.

```
cryptnox decrypt [OPTIONS]
```

2.4.11 User Key Management

user_key list

List all configured user keys for authentication.

```
cryptnox user_key list
```


user_key add

Add a new user key for authentication (PIV card, Windows Hello).

```
cryptnox user_key add TYPE [DESCRIPTION]
```

Available Types:

- `piv`: PIV-compatible smart card
- `hello`: Windows Hello (biometric)

Example:

```
cryptnox user_key add piv "My PIV Key"
```

user_key delete

Delete a user key.

```
cryptnox user_key delete TYPE
```

2.4.12 Server Mode

server

Start a server or establish connection to a remote server.

```
cryptnox server [OPTIONS]
```

Options:

- `--port PORT`: Server port (default: 5050)
- `--host HOST`: Server host (default: 0.0.0.0)

Use Case: Remote card access over network

2.5 Interactive Mode

Launch interactive CLI mode:

```
cryptnox
```

In interactive mode, you can:

- Execute commands without repeating `cryptnox`
- Use `use` command to switch between multiple cards
- Use `exit` to quit

Example Session:

```
$ cryptnox
Cryptnox CLI 1.0.5

> list
```

```
[Shows available cards]

> info
[Shows card information]

> exit
```

2.6 BIP39 Passphrase Support

The `seed recover` and `seed upload` commands support BIP39 passphrases (also known as the 13th/25th word).

2.6.1 What is a BIP39 Passphrase?

A BIP39 passphrase is an optional additional word that enhances security:

- Acts as a “second factor” for your seed
- Creates a completely different wallet from the same mnemonic
- Must be remembered separately (not stored with the mnemonic)
- Cannot be recovered if forgotten

2.6.2 Using BIP39 Passphrase

During seed upload (new wallet):

```
cryptnox seed upload
```

The command will prompt:

```
Do you want to use a BIP39 passphrase? [y/N]
```

If you choose `yes`:

- Enter your desired passphrase
- Confirm the passphrase
- **Remember it!** You’ll need it for recovery

During seed recover (existing wallet):

```
cryptnox seed recover
```

If your wallet was created with a passphrase, you **must** provide the same passphrase when recovering.

Important Notes:

- **!** If you forget your passphrase, your funds are **permanently lost**
- ✓ Same mnemonic + different passphrase = completely different wallet
- ✓ Passphrase can be any UTF-8 string (including spaces and special characters)

2.7 Exit Codes

The CLI returns the following exit codes:

- `0`: Success
- `-1`: Error occurred
- `-2`: Card error or card not found

2.8 Error Handling

When errors occur:

- Detailed error messages are displayed
- Errors are logged to: `~/.local/share/cryptnox-cli/error.log` (Linux/macOS) or `%LOCALAPPDATA%\\cryptnox\\cryptnox-cli\\error.log` (Windows)
- You can report errors to help improve the application

2.9 Examples

Initialize a new card:

```
cryptnox init
```

Generate and upload a seed:

```
cryptnox seed upload
```

Recover from existing mnemonic:

```
cryptnox seed recover
```

Send Bitcoin:

```
cryptnox btc send 1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa 0.001 -n testnet
```

Send Ethereum:

```
cryptnox eth send 0x742d35Cc6634C0532925a3b844Bc9e7595f0bEb 0.1 -n mainnet
```

View card info:

```
cryptnox info
```

Change PIN:

```
cryptnox change_pin
```


2.10 See Also

- *Cryptnox CLI Overview* - Overview of Cryptnox CLI
- `cryptnox_cli.command.seed` - Seed command implementation details
- *cryptnox_cli.command package* - All command implementations
- *API Reference* - Complete API Reference

3 API Reference

3.1 cryptnox_cli package (Cryptnox CLI)

3.1.1 Subpackages

cryptnox_cli.command package

Subpackages

cryptnox_cli.command.card package

Submodules

cryptnox_cli.command.card.info module

cryptnox_cli.command.card.initialize module

Module contents

The `cryptnox_cli.command.card` package provides card management functionality and exports:

class `cryptnox_cli.command.card.`**Info**

Command class for displaying card information. See `cryptnox_cli.command.card.info.Info` for details.

class `cryptnox_cli.command.card.`**Initialize**

Command class for initializing cards. See `cryptnox_cli.command.card.initialize.Initialize` for details.

cryptnox_cli.command.erc_token package

Submodules

cryptnox_cli.command.erc_token.contract module

cryptnox_cli.command.erc_token.info module

cryptnox_cli.command.erc_token.initialize module

Module contents

The `cryptnox_cli.command.erc_token` package provides ERC token management and exports:

class `cryptnox_cli.command.erc_token.`**Info**

Command class for displaying ERC token information. See `cryptnox_cli.command.erc_token.info.Info` for details.

class `cryptnox_cli.command.erc_token.`**Initialize**

Command class for initializing ERC tokens. See `cryptnox_cli.command.erc_token.initialize.Initialize` for details.

cryptnox_cli.command.helper package

Submodules

cryptnox_cli.command.helper.cards module

cryptnox_cli.command.helper.config module

cryptnox_cli.command.helper.helper_methods module

cryptnox_cli.command.helper.notification module

cryptnox_cli.command.helper.security module

cryptnox_cli.command.helper.ui module

Module contents

The `cryptnox_cli.command.helper` package provides utility functions and helper modules for command implementations. This package contains shared functionality used across multiple commands, including user interface components, security helpers, and configuration management.

cryptnox_cli.command.options package

Submodules

cryptnox_cli.command.options.common module

cryptnox_cli.command.options.eth module

cryptnox_cli.command.options.options module

Module contents

The `cryptnox_cli.command.options` package provides command-line argument parsing and exports:

`cryptnox_cli.command.options.`**add**(*subparsers*, *command_class*)

Add command-line options for a command class. See `cryptnox_cli.command.options.options.add()` for details.

cryptnox_cli.command.user_keys package

Subpackages

cryptnox_cli.command.user_keys.hello package

Submodules

cryptnox_cli.command.user_keys.hello.exceptions module

Exceptions Windows Hello can raise.

exception cryptnox_cli.command.user_keys.hello.exceptions.**WindowsHelloExceptions**

Bases: Exception

Base exception for the class exceptions.

exception cryptnox_cli.command.user_keys.hello.exceptions.**NotSupportedException**

Bases: WindowsHelloExceptions

There is no hardware support for Windows Hello.

exception cryptnox_cli.command.user_keys.hello.exceptions.**UnknownErrorException**

Bases: WindowsHelloExceptions

An unknown error occurred.

exception cryptnox_cli.command.user_keys.hello.exceptions.**NotFoundException**

Bases: WindowsHelloExceptions

The credential could not be found.

exception cryptnox_cli.command.user_keys.hello.exceptions.**CanceledException**

Bases: WindowsHelloExceptions

The request was cancelled by the user.

exception cryptnox_cli.command.user_keys.hello.exceptions.**UserPrefersPasswordException**

Bases: WindowsHelloExceptions

The user prefers to enter a password.

exception cryptnox_cli.command.user_keys.hello.exceptions.**SecurityDeviceLockedException**

Bases: WindowsHelloExceptions

The security device was locked.

cryptnox_cli.command.user_keys.hello.hello module

cryptnox_cli.command.user_keys.hello.windows_hello module

Module for signing messages with Windows Hello.

cryptnox_cli.command.user_keys.hello.windows_hello.**delete**(*name: str*) → None

Delete the key from Windows Hello

Parameters

name (str) – name of the slot

cryptnox_cli.command.user_keys.hello.windows_hello.**get_public_key**(*name: str*) → bytearray

Get public key from Windows Hello

Returns

public key

Return type
bytearray

cryptnox_cli.command.user_keys.hello.windows_hello.**sign**(*name: str*, *message_to_sign: str*) → bytearray

Sign given message with Windows Hello

Parameters

- **name** (str) – name of the slot
- **message_to_sign** (str) – message that needs to be signed

Returns
Dictionary with public key and signed message as byte array

Return type
bytearray

Module contents

Module containing Windows Hello user key handling.

cryptnox_cli.command.user_keys.piv package

Submodules

cryptnox_cli.command.user_keys.piv.piv module

cryptnox_cli.command.user_keys.piv.piv_card module

Module contents

Submodules

cryptnox_cli.command.user_keys.authentication module

cryptnox_cli.command.user_keys.user_key_base module

Module containing base class for user key.

exception cryptnox_cli.command.user_keys.user_key_base.**UserKeyException**
Bases: Exception

Base exception for all exceptions for UserKey

exception cryptnox_cli.command.user_keys.user_key_base.**ExitException**
Bases: UserKeyException

The user has exited the request

exception cryptnox_cli.command.user_keys.user_key_base.**NotFoundException**
Bases: UserKeyException

The user key hasn't been found

exception cryptnox_cli.command.user_keys.user_key_base.**NotSupportedException**

Bases: UserKeyException

The user key is not supported

exception cryptnox_cli.command.user_keys.user_key_base.**ProcessingException**

Bases: UserKeyException

There was an issue in processing the user key

class cryptnox_cli.command.user_keys.user_key_base.**UserKey**(*target: str = ''*)

Bases: object

Base class on top of which other user key classes should be implemented

priority: int = 0

__init__(*target: str = ''*)

abstractmethod delete() → None

Delete the key from the user key provider

abstract property description: str

Description of the user key service :rtype: str

Type
return

abstract property name: str

Name of the user key service :rtype: str

Type
return

abstract property public_key: bytes

Public key of the user key service :rtype: bytes

Type
return

abstractmethod sign(*message: bytes*) → bytes

Get signature of message from user key service

Parameters
message (bytes) – Message to sign

Returns
Signature of the given message

Return type
bytes

abstract property slot_index: cryptnox_sdk_py.SlotIndex

Compatible slot index on the card :rtype: cryptnox_sdk_py.SlotIndex

Type
return

property custom_bit: int

Custom bit to use for checking if method is enabled, -1 for not used :rtype: int

Type
return

added(*card*)

classmethod enabled(*card: cryptnox_sdk_py.Card*) → bool

Module contents

The `cryptnox_cli.command.user_keys` package provides authentication functionality and exports:

`cryptnox_cli.command.user_keys.`**add**(*data*)

Add a new authentication method. See `cryptnox_cli.command.user_keys.authentication.add()` for details.

`cryptnox_cli.command.user_keys.`**authenticate**(*data*)

Authenticate using a configured method. See `cryptnox_cli.command.user_keys.authentication.authenticate()` for details.

`cryptnox_cli.command.user_keys.`**delete**(*data*)

Delete an authentication method. See `cryptnox_cli.command.user_keys.authentication.delete()` for details.

`cryptnox_cli.command.user_keys.`**get**(*data*)

Get information about authentication methods. See `cryptnox_cli.command.user_keys.authentication.get()` for details.

Submodules

cryptnox_cli.command.btc module

cryptnox_cli.command.card_configuration module

cryptnox_cli.command.cards module

cryptnox_cli.command.change_pin module

cryptnox_cli.command.change_puk module

cryptnox_cli.command.command module

cryptnox_cli.command.config module

cryptnox_cli.command.decrypt module

cryptnox_cli.command.eth module

cryptnox_cli.command.factory module

cryptnox_cli.command.get_clearpubkey module

cryptnox_cli.command.get_xpub module

cryptnox_cli.command.history module

cryptnox_cli.command.info module

cryptnox_cli.command.initialize module

cryptnox_cli.command.manufacturer_certificate module

cryptnox_cli.command.reset module

cryptnox_cli.command.seed module

cryptnox_cli.command.server module

cryptnox_cli.command.transfer module

cryptnox_cli.command.unknown module

cryptnox_cli.command.unlock_pin module

cryptnox_cli.command.user_key module

Module contents

Module to work with commands given by the user in a command line

The `cryptnox_cli.command` package provides command-line interface commands for interacting with Cryptnox cards. This package contains all command implementations that can be executed from the CLI.

cryptnox_cli.lib package

Subpackages

cryptnox_cli.lib.cryptos package

Subpackages

cryptnox_cli.lib.cryptos.coins package

Submodules

cryptnox_cli.lib.cryptos.coins.base module

Base classes and shared logic for UTXO-based coin implementations.

class cryptnox_cli.lib.cryptos.coins.base.**BaseCoin**(*testnet=False*, ***kwargs*)

Bases: `object`

Base implementation of crypto coin class All child coins must follow same pattern.

coin_symbol = None

display_name = None

enabled = True

segwit_supported = None

magicbyte = None

script_magicbyte = None

segwit_hrp = None

client_kwargs = {'host': None, 'loop': None, 'max_servers': 5, 'port': 50001, 'server_file': 'bitcoin.json', 'servers': (), 'timeout': 15}

testnet_overrides = {}

hashcode = 1

hd_path = 0

wif_prefix = 128

wif_script_types = {'p2pkh': 0, 'p2sh': 5, 'p2wpkh': 1, 'p2wpkh-p2sh': 2, 'p2wsh': 6, 'p2wsh-p2sh': 7}

xprv_headers = {'p2pkh': 76066276, 'p2wpkh': 78791436, 'p2wpkh-p2sh': 77428856, 'p2wsh': 44726937, 'p2wsh-p2sh': 43364357}

xpub_headers = {'p2pkh': 76067358, 'p2wpkh': 78792518, 'p2wpkh-p2sh': 77429938, 'p2wsh': 44728019, 'p2wsh-p2sh': 43365439}

electrum_xprv_headers = {'p2pkh': 76066276, 'p2wpkh': 78791436, 'p2wpkh-p2sh': 77428856, 'p2wsh': 44726937, 'p2wsh-p2sh': 43364357}

electrum_xpub_headers = {'p2pkh': 76067358, 'p2wpkh': 78792518, 'p2wpkh-p2sh': 77429938, 'p2wsh': 44728019, 'p2wsh-p2sh': 43365439}

__init__(*testnet=False*, ***kwargs*)

is_testnet = False

address_prefixes = ()

secondary_hashcode = None

property rpc_client

Connect to remove server

unspent(**addrs*)

Get unspent transactions for addresses

history(**addrs*, ***kwargs*)

Get transaction history for addresses

fetchtx(*tx*)

Fetch a tx from the blockchain

txinputs(*tx*)
Fetch inputs of a transaction on the blockchain

pushtx(*tx*)
Push/ Broadcast a transaction to the blockchain

privtopub(*privkey*)
Get public key from private key

pubtoaddr(*pubkey*)
Get address from a pubic key

privtoaddr(*privkey*)
Get address from a private key

electrum_address(*masterkey*, *n*, *for_change=0*)
For old electrum seeds

is_address(*addr*)
Check if addr is a valid address for this chain

is_p2sh(*addr*)
Check if addr is a a pay to script address

output_script_to_address(*script*)
Convert an output script to an address

scripttoaddr(*script*)
Convert an input public key hash to an address

p2sh_scriptaddr(*script*)
Convert an output p2sh script to an address

addrtoscript(*addr*)
Convert an output address to a script

pubtop2w(*pub*)
Convert a public key to a pay to witness public key hash address (P2WPKH, required for segwit)

privtop2w(*priv*)
Convert a private key to a pay to witness public key hash address

hash_to_segwit_addr(*hash*)
Convert a hash to the new segwit address format outlined in BIP-0173

privtosegwit(*privkey*)
Convert a private key to the new segwit address format outlined in BIP01743

pubtosegwit(*pubkey*)
Convert a public key to the new segwit address format outlined in BIP01743

script_to_p2wsh(*script*)
Convert a script to the new segwit address format outlined in BIP01743

mk_multsig_address(**args*)

Parameters

args – List of public keys to used to create multisig and M, the number of signatures required to spend

Returns

multisig script

is_segwit(*priv*, *addr*)

Check if addr was generated from priv using segwit script

sign(*txobj*, *i*, *priv*)

Sign a transaction input with index using a private key

signall(*txobj*, *priv*)

Sign all inputs to a transaction using a private key

multisign(*tx*, *i*, *script*, *pk*)

mktx(**args*)

[in0, in1...],[out0, out1...] or in0, in1 ... out0 out1 ...

Make an unsigned transaction from inputs and outputs. Change is not automatically included so any difference in value between inputs and outputs will be given as a miner's fee (transactions with too high fees will normally be blocked by the explorers)

For Bitcoin Cash and other hard forks using SIGHASH_FORKID, ins must be a list of dicts with each containing the outpoint and value of the input.

Inputs originally received with segwit must be a dict in the format: {'outpoint': "tx-hash:index", value:0, "segwit": True}

For other transactions, inputs can be dicts containing only outpoints or strings in the outpoint format. Outpoint format: txhash:index

mksend(**args*, *segwit=False*)

[in0, in1...],[out0, out1...] or in0, in1 ... out0 out1 ...

Make an unsigned transaction from inputs, outputs change address and fee. A change output will be added with change sent to the change address.

For Bitcoin Cash and other hard forks using SIGHASH_FORKID and segwit, ins must be a list of dicts with each containing the outpoint and value of the input.

For other transactions, inputs can be dicts containing only outpoints or strings in the outpoint format. Outpoint format: txhash:index

preparesignedtx(*privkey*, *to*, *value*, *fee=10000*, *change_addr=None*, *segwit=False*, *addr=None*)

Prepare a tx with a specific amount from address belonging to private key to another address, returning change to the from address or change address, if set. Requires private key, target address, value and optionally the change address and fee segwit paramater specifies that the inputs belong to a segwit address addr, if provided, will explicity set the from address, overriding the auto-detection of the

address from the private key.It will also be used, along with the privkey, to automatically detect a segwit transaction for coins which support segwit, overriding the segwit kw

send(*privkey*, *to*, *value*, *fee=10000*, *change_addr=None*, *segwit=False*, *addr=None*)

Send a specific amount from address belonging to private key to another address, returning change to the from address or change address, if set. Requires private key, target address, value and optionally the change address and fee segwit paramater specifies that the inputs belong to a segwit address addr, if provided, will explicity set the from address, overriding the auto-detection of the address from the private key.It will also be used, along with the privkey, to automatically detect a segwit transaction for coins which support segwit, overriding the segwit kw

preparesignedmultitx(*privkey*, **args*, *change_addr=None*, *segwit=False*, *addr=None*)

Prepare transaction with multiple outputs, with change sent back to from addrss Requires private key, address:value pairs and optionally the change address and fee segwit paramater specifies that the inputs belong to a segwit address addr, if provided, will explicity set the from address, overriding the auto-detection of the address from the private key.It will also be used, along with the privkey to automatically detect a segwit transaction for coins which support segwit, overriding the segwit kw

sendmultitx(*privkey*, **args*, *change_addr=None*, *segwit=False*, *addr=None*)

Send transaction with multiple outputs, with change sent back to from addrss Requires private key, address:value pairs and optionally the change address and fee segwit paramater specifies that the inputs belong to a segwit address addr, if provided, will explicity set the from address, overriding the auto-detection of the address from the private key.It will also be used, along with the privkey to automatically detect a segwit transaction for coins which support segwit, overriding the segwit kw

preparetx(*frm*, *to*, *value*, *fee*, *change_addr=None*, *segwit=False*)

Prepare a transaction using from and to addresses, value and a fee, with change sent back to from address

preparemultitx(*frm*, **args*, *change_addr=None*, *segwit=False*)

Prepare transaction with multiple outputs, with change sent to from address Requires from address, to_address:value pairs and fees

block_height(*txhash*)

current_block_height()

block_info(*height*)

inspect(*tx*)

merkle_prove(*txhash*)

Prove that information an explorer returns about a transaction in the blockchain is valid. Only run on a tx with at least 1 confirmation.

encode_privkey(*privkey*, *formt*, *script_type='p2pkh'*)

wallet(*seed*, *passphrase=None*, ***kwargs*)

watch_wallet(*xpub*, ***kwargs*)

p2wpkh_p2sh_wallet(*seed*, *passphrase=None*, ***kwargs*)

watch_p2wpkh_p2sh_wallet(*xpub*, ***kwargs*)

p2wpkh_wallet(*seed*, *passphrase=None*, ***kwargs*)

watch_p2wpkh_wallet(*xpub*, ***kwargs*)

electrum_wallet(*seed*, *passphrase=None*, ***kwargs*)

watch_electrum_wallet(*xpub*, ***kwargs*)

watch_electrum_p2wpkh_wallet(*xpub*, ***kwargs*)

cryptnox_cli.lib.cryptos.coins.bitcoin module

Bitcoin-specific constants, scripts, and address helpers.

class cryptnox_cli.lib.cryptos.coins.bitcoin.**Bitcoin**(*testnet=False*, ***kwargs*)

Bases: BaseCoin

coin_symbol = 'BTC'

display_name = 'Bitcoin'

segwit_supported = True

magicbyte = 0

script_magicbyte = 5

segwit_hrp = 'bc'

client_kwargs = {'server_file': 'bitcoin.json'}

testnet_overrides = {'client_kwargs': {'server_file': 'bitcoin_testnet.json'}, 'coin_symbol': 'BTCTEST', 'display_name': 'Bitcoin Testnet', 'hd_path': 1, 'magicbyte': 111, 'script_magicbyte': 196, 'segwit_hrp': 'tb', 'wif_prefix': 239, 'xprv_headers': {'p2pkh': 70615956, 'p2wpkh': 70615956, 'p2wpkh-p2sh': 71978536, 'p2wsh': 44726937, 'p2wsh-p2sh': 43364357}, 'xpub_headers': {'p2pkh': 70617039, 'p2wpkh': 70617039, 'p2wpkh-p2sh': 71979618, 'p2wsh': 44728019, 'p2wsh-p2sh': 43365439}}

Module contents

Re-exports available coin implementations for convenience.

class cryptnox_cli.lib.cryptos.coins.**Bitcoin**(*testnet=False*, ***kwargs*)

Bases: BaseCoin

coin_symbol = 'BTC'

display_name = 'Bitcoin'

```
segwit_supported = True

magicbyte = 0

script_magicbyte = 5

segwit_hrp = 'bc'

client_kwargs = {'server_file': 'bitcoin.json'}

testnet_overrides = {'client_kwargs': {'server_file':
'bitcoin_testnet.json'}, 'coin_symbol': 'BTCTEST', 'display_name': 'Bitcoin
Testnet', 'hd_path': 1, 'magicbyte': 111, 'script_magicbyte': 196,
'segwit_hrp': 'tb', 'wif_prefix': 239, 'xprv_headers': {'p2pkh': 70615956,
'p2wpkh': 70615956, 'p2wpkh-p2sh': 71978536, 'p2wsh': 44726937, 'p2wsh-p2sh':
43364357}, 'xpub_headers': {'p2pkh': 70617039, 'p2wpkh': 70617039,
'p2wpkh-p2sh': 71979618, 'p2wsh': 44728019, 'p2wsh-p2sh': 43365439}}
```

The `cryptnox_cli.lib.cryptos.coins` package provides cryptocurrency coin implementations and exports:

class `cryptnox_cli.lib.cryptos.coins.`**Bitcoin**

Bitcoin cryptocurrency implementation. See `cryptnox_cli.lib.cryptos.coins.bitcoin.Bitcoin` for details.

Submodules

cryptnox_cli.lib.cryptos.blocks module

Block header encoding/decoding utilities and related primitives.

`cryptnox_cli.lib.cryptos.blocks.`**serialize_header**(*inp*)

`cryptnox_cli.lib.cryptos.blocks.`**deserialize_header**(*inp*)

`cryptnox_cli.lib.cryptos.blocks.`**mk_merkle_proof**(*header*, *hashes*, *index*)

cryptnox_cli.lib.cryptos.composite module

High-level composition utilities combining deterministic keys and transactions.

`cryptnox_cli.lib.cryptos.composite.`**bip32_hdm_script**(**args*)

`cryptnox_cli.lib.cryptos.composite.`**bip32_hdm_addr**(**args*)

`cryptnox_cli.lib.cryptos.composite.`**setup_coinvault_tx**(*tx*, *script*)

`cryptnox_cli.lib.cryptos.composite.`**sign_coinvault_tx**(*tx*, *priv*)

cryptnox_cli.lib.cryptos.constants module

Core constants and magic values used across cryptocurrency modules.

cryptnox_cli.lib.cryptos.deterministic module

Deterministic key and path derivation utilities (BIP32-like logic).

cryptnox_cli.lib.cryptos.deterministic.**electrum_stretch**(*seed*)

cryptnox_cli.lib.cryptos.deterministic.**electrum_mpk**(*seed*)

cryptnox_cli.lib.cryptos.deterministic.**electrum_privkey**(*seed*, *n*, *for_change=0*)

cryptnox_cli.lib.cryptos.deterministic.**electrum_pubkey**(*masterkey*, *n*, *for_change=0*)

cryptnox_cli.lib.cryptos.deterministic.**electrum_address**(*masterkey*, *n*, *for_change=0*, *magicbyte=0*)

cryptnox_cli.lib.cryptos.deterministic.**crack_electrum_wallet**(*mpk*, *pk*, *n*, *for_change=0*)

cryptnox_cli.lib.cryptos.deterministic.**raw_bip32_ckd**(*rawtuple*, *i*, *prefixes=DEFAULT*)

cryptnox_cli.lib.cryptos.deterministic.**bip32_serialize**(*rawtuple*, *prefixes=DEFAULT*)

cryptnox_cli.lib.cryptos.deterministic.**bip32_deserialize**(*data*, *prefixes=DEFAULT*)

cryptnox_cli.lib.cryptos.deterministic.**is_xprv**(*text*, *prefixes=DEFAULT*)

cryptnox_cli.lib.cryptos.deterministic.**is_xpub**(*text*, *prefixes=DEFAULT*)

cryptnox_cli.lib.cryptos.deterministic.**raw_bip32_privtopub**(*rawtuple*, *prefixes=DEFAULT*)

cryptnox_cli.lib.cryptos.deterministic.**bip32_privtopub**(*data*, *prefixes=DEFAULT*)

cryptnox_cli.lib.cryptos.deterministic.**bip32_ckd**(*key*, *path*, *prefixes=DEFAULT*, *public=False*)

cryptnox_cli.lib.cryptos.deterministic.**bip32_master_key**(*seed*, *prefixes=DEFAULT*)

cryptnox_cli.lib.cryptos.deterministic.**bip32_bin_extract_key**(*data*, *prefixes=DEFAULT*)

cryptnox_cli.lib.cryptos.deterministic.**bip32_extract_key**(*data*, *prefixes=DEFAULT*)

cryptnox_cli.lib.cryptos.deterministic.**bip32_derive_key**(*key*, *path*, *prefixes=DEFAULT*, ***kwargs*)

cryptnox_cli.lib.cryptos.deterministic.**raw_crack_bip32_privkey**(*parent_pub*, *priv*, *prefixes=DEFAULT*)

cryptnox_cli.lib.cryptos.deterministic.**crack_bip32_privkey**(*parent_pub*, *priv*, *prefixes=DEFAULT*)

cryptnox_cli.lib.cryptos.deterministic.**coinvault_pub_to_bip32**(**args*, *prefixes=DEFAULT*)

cryptnox_cli.lib.cryptos.deterministic.**coinvault_priv_to_bip32**(**args*, *prefixes=DEFAULT*)

cryptnox_cli.lib.cryptos.deterministic.**bip32_descend**(**args*, *prefixes=DEFAULT*)

Descend masterkey and return privkey

cryptnox_cli.lib.cryptos.deterministic.**parse_bip32_path**(*path*)

Takes bip32 path, “m/0’/2H” or “m/0H/1/2H/2/1000000000.pub”, returns list of ints

cryptnox_cli.lib.cryptos.keystore module

HD wallet structures, serialization, derivation, and key store helpers.

class cryptnox_cli.lib.cryptos.keystore.**KeyStore**(*coin*, *addresses=()*)

Bases: object

__init__(*coin*, *addresses=()*)

has_seed()

is_watching_only()

can_import()

get_tx_derivations(*tx*)

can_sign(*tx*)

class cryptnox_cli.lib.cryptos.keystore.**Software_KeyStore**(*coin*, *addresses=()*)

Bases: KeyStore

may_have_password()

class cryptnox_cli.lib.cryptos.keystore.**Imported_KeyStore**(*d*, *coin*)

Bases: Software_KeyStore

__init__(*d*, *coin*)

is_deterministic()

can_change_password()

get_master_public_key()

dump()

can_import()

check_password(*password=None*)

import_privkey(*sec*, *password=None*)

delete_imported_key(*key*)

get_private_key(*pubkey*, *password=None*)

get_pubkey_derivation(*x_pubkey*)

update_password(*old_password*, *new_password*)

class cryptnox_cli.lib.cryptos.keystore.**Deterministic_KeyStore**(*d*, *coin*)

Bases: Software_KeyStore

__init__(*d*, *coin*)

is_deterministic()

dump()

has_seed()

is_watching_only()

can_change_password()

add_seed(*seed*)

get_seed(*password*)

get_passphrase(*password*)

class cryptnox_cli.lib.cryptos.keystore.**Xpub**

Bases: object

__init__()

get_master_public_key()

derive_pubkey(*for_change*, *n*)

classmethod get_pubkey_from_xpub(*xpub*, *sequence*, *bip39_prefixes*)

class cryptnox_cli.lib.cryptos.keystore.**BIP32_KeyStore**(*d*, *coin*)

Bases: Deterministic_KeyStore, Xpub

__init__(*d*, *coin*)

format_seed(*seed*)

dump()

get_master_private_key(*password=None*)

check_password(*password=None*)

update_password(*old_password*, *new_password*)

is_watching_only()

add_xprv(*xprv*)

add_xpub(*xpub*, *xtype*, *electrum=False*)

add_xprv_from_seed(*bip32_seed*, *xtype*, *derivation*, *electrum=False*)

get_private_key(*sequence*, *password*)

class cryptnox_cli.lib.cryptos.keystore.**Hardware_KeyStore**(*d*, *coin*)

Bases: KeyStore, Xpub

max_change_outputs = 1

__init__(*d*, *coin*)

set_label(*label*)

may_have_password()

is_deterministic()

dump()

unpaired()
A device paired with the wallet was diconnected. This can be called in any thread context.

paired()
A device paired with the wallet was (re-)connected. This can be called in any thread context.

can_export()

is_watching_only()
The wallet is not watching-only; the user will be prompted for pin and passphrase as appropriate when needed.

can_change_password()

cryptnox_cli.lib.cryptos.keystore.**bip39_to_seed**(*mnemonic*, *passphrase*)

cryptnox_cli.lib.cryptos.keystore.**bip39_is_checksum_valid**(*mnemonic*)

cryptnox_cli.lib.cryptos.keystore.**from_bip39_seed**(*seed*, *passphrase*, *derivation*, *coin*)

cryptnox_cli.lib.cryptos.keystore.**standard_from_bip39_seed**(*seed*, *passphrase*, *coin*)

cryptnox_cli.lib.cryptos.keystore.**p2wpkh_from_bip39_seed**(*seed*, *passphrase*, *coin*)

cryptnox_cli.lib.cryptos.keystore.**p2wpkh_p2sh_from_bip39_seed**(*seed*, *passphrase*, *coin*)

cryptnox_cli.lib.cryptos.keystore.**xtype_from_derivation**(*derivation*)
Returns the script type to be used for this derivation.

cryptnox_cli.lib.cryptos.keystore.**is_xpubkey**(*x_pubkey*)

cryptnox_cli.lib.cryptos.keystore.**parse_xpubkey**(*x_pubkey*)

cryptnox_cli.lib.cryptos.keystore.**xpubkey_to_address**(*x_pubkey*, *coin*)

cryptnox_cli.lib.cryptos.keystore.**xpubkey_to_pubkey**(*x_pubkey*, *coin*)

cryptnox_cli.lib.cryptos.keystore.**register_keystore**(*hw_type*, *constructor*)

cryptnox_cli.lib.cryptos.keystore.**hardware_keystore**(*d*)

cryptnox_cli.lib.cryptos.keystore.**is_address_list**(*text*, *coin*)

cryptnox_cli.lib.cryptos.keystore.**get_private_keys**(*text*)

cryptnox_cli.lib.cryptos.keystore.**is_private_key_list**(*text*)

cryptnox_cli.lib.cryptos.keystore.**is_private**(*x*)

cryptnox_cli.lib.cryptos.keystore.**is_master_key**(*x*)

cryptnox_cli.lib.cryptos.keystore.**is_private_key**(*x*)

cryptnox_cli.lib.cryptos.keystore.**is_bip32_key**(*x*)

cryptnox_cli.lib.cryptos.keystore.**from_electrum_seed**(*seed*, *passphrase*, *is_p2sh*, *coin*)

cryptnox_cli.lib.cryptos.keystore.**from_private_key_list**(*text*, *coin*)

cryptnox_cli.lib.cryptos.keystore.**from_xpub**(*xpub*, *coin*, *xtype*, *electrum=False*)

cryptnox_cli.lib.cryptos.keystore.**from_xprv**(*xprv*, *coin*)

cryptnox_cli.lib.cryptos.keystore.**from_master_key**(*text*, *coin*)

cryptnox_cli.lib.cryptos.main module

Core cryptographic and encoding primitives used by higher-level modules.

This module is based on code from pybitcointools: https://github.com/primal100/pybitcointools/blob/master/cryptos/main.py

cryptnox_cli.lib.cryptos.main.**change_curve**(*p*, *n*, *a*, *b*, *gx*, *gy*)

cryptnox_cli.lib.cryptos.main.**getG**()

cryptnox_cli.lib.cryptos.main.**inv**(*a*, *n*)

cryptnox_cli.lib.cryptos.main.**access**(*obj*, *prop*)

cryptnox_cli.lib.cryptos.main.**multiaccess**(*obj*, *prop*)

cryptnox_cli.lib.cryptos.main.**slice**(*obj*, *start=0*, *end=2**200*)

cryptnox_cli.lib.cryptos.main.**count**(*obj*)

cryptnox_cli.lib.cryptos.main.**sum**(*obj*)

cryptnox_cli.lib.cryptos.main.**isinf**(*p*)

cryptnox_cli.lib.cryptos.main.**to_jacobian**(*p*)

cryptnox_cli.lib.cryptos.main.**jacobian_double**(*p*)

cryptnox_cli.lib.cryptos.main.**jacobian_add**(*p*, *q*)

cryptnox_cli.lib.cryptos.main.**from_jacobian**(*p*)

cryptnox_cli.lib.cryptos.main.**jacobian_multiply**(*a*, *n*)

cryptnox_cli.lib.cryptos.main.**fast_multiply**(*a*, *n*)

cryptnox_cli.lib.cryptos.main.**fast_add**(*a*, *b*)

cryptnox_cli.lib.cryptos.main.**get_pubkey_format**(*pub*)

cryptnox_cli.lib.cryptos.main.**encode_pubkey**(*pub*, *formt*)

cryptnox_cli.lib.cryptos.main.**decode_pubkey**(*pub*, *formt=None*)

cryptnox_cli.lib.cryptos.main.**get_privkey_format**(*priv*)

cryptnox_cli.lib.cryptos.main.**encode_privkey**(*priv*, *formt*, *vbyte=128*)

cryptnox_cli.lib.cryptos.main.**decode_privkey**(*priv*, *formt=None*)

cryptnox_cli.lib.cryptos.main.**add_pubkeys**(*p1*, *p2*)

cryptnox_cli.lib.cryptos.main.**add_privkeys**(*p1*, *p2*)

cryptnox_cli.lib.cryptos.main.**mul_privkeys**(*p1*, *p2*)

cryptnox_cli.lib.cryptos.main.**multiply**(*pubkey*, *privkey*)

cryptnox_cli.lib.cryptos.main.**divide**(*pubkey*, *privkey*)

cryptnox_cli.lib.cryptos.main.**compress**(*pubkey*)

cryptnox_cli.lib.cryptos.main.**decompress**(*pubkey*)

cryptnox_cli.lib.cryptos.main.**privkey_to_pubkey**(*privkey*)

cryptnox_cli.lib.cryptos.main.**privtopub**(*privkey*)

cryptnox_cli.lib.cryptos.main.**privkey_to_address**(*priv*, *magicbyte=0*)

cryptnox_cli.lib.cryptos.main.**privtoaddr**(*priv*, *magicbyte=0*)

cryptnox_cli.lib.cryptos.main.**neg_pubkey**(*pubkey*)

cryptnox_cli.lib.cryptos.main.**neg_privkey**(*privkey*)

cryptnox_cli.lib.cryptos.main.**subtract_pubkeys**(*p1*, *p2*)

cryptnox_cli.lib.cryptos.main.**subtract_privkeys**(*p1*, *p2*)

cryptnox_cli.lib.cryptos.main.**bin_hash160**(*string*)

cryptnox_cli.lib.cryptos.main.**hash160**(*string*)

cryptnox_cli.lib.cryptos.main.**hex_to_hash160**(*s_hex*)

cryptnox_cli.lib.cryptos.main.**bin_sha256**(*string*)

cryptnox_cli.lib.cryptos.main.**sha256**(*string*)

cryptnox_cli.lib.cryptos.main.**bin_ripemd160**(*string*)

cryptnox_cli.lib.cryptos.main.**ripemd160**(*string*)

cryptnox_cli.lib.cryptos.main.**bin_dbl_sha256**(*s*)

cryptnox_cli.lib.cryptos.main.**dbl_sha256**(*string*)

cryptnox_cli.lib.cryptos.main.**bin_slowsha**(*string*)

cryptnox_cli.lib.cryptos.main.**slowsha**(*string*)

cryptnox_cli.lib.cryptos.main.**hash_to_int**(*x*)

cryptnox_cli.lib.cryptos.main.**num_to_var_int**(*x*)

cryptnox_cli.lib.cryptos.main.**electrum_sig_hash**(*message*)

cryptnox_cli.lib.cryptos.main.**random_key**()

cryptnox_cli.lib.cryptos.main.**random_electrum_seed**()

cryptnox_cli.lib.cryptos.main.**b58check_to_bin**(*inp*)

cryptnox_cli.lib.cryptos.main.**get_version_byte**(*inp*)

cryptnox_cli.lib.cryptos.main.**hex_to_b58check**(*inp*, *magicbyte=0*)

cryptnox_cli.lib.cryptos.main.**b58check_to_hex**(*inp*)

cryptnox_cli.lib.cryptos.main.**pubkey_to_hash**(*pubkey*)

cryptnox_cli.lib.cryptos.main.**pubkey_to_hash_hex**(*pubkey*)

cryptnox_cli.lib.cryptos.main.**pubkey_to_address**(*pubkey*, *magicbyte=0*)

cryptnox_cli.lib.cryptos.main.**pubtoaddr**(*pubkey*, *magicbyte=0*)

cryptnox_cli.lib.cryptos.main.**is_privkey**(*priv*)

cryptnox_cli.lib.cryptos.main.**is_pubkey**(*pubkey*)

cryptnox_cli.lib.cryptos.main.**encode_sig**(*v*, *r*, *s*)

cryptnox_cli.lib.cryptos.main.**decode_sig**(*sig*)

cryptnox_cli.lib.cryptos.main.**deterministic_generate_k**(*msghash*, *priv*)

cryptnox_cli.lib.cryptos.main.**ecdsa_raw_sign**(*msghash*, *priv*)

cryptnox_cli.lib.cryptos.main.**ecdsa_sign**(*msg*, *priv*, *coin*)

cryptnox_cli.lib.cryptos.main.**ecdsa_raw_verify**(*msghash*, *vrs*, *pub*)

cryptnox_cli.lib.cryptos.main.**ecdsa_verify_addr**(*msg*, *sig*, *addr*, *coin*)

cryptnox_cli.lib.cryptos.main.**ecdsa_verify**(*msg*, *sig*, *pub*, *coin*)

cryptnox_cli.lib.cryptos.main.**ecdsa_raw_recover**(*msghash*, *vrs*)

cryptnox_cli.lib.cryptos.main.**ecdsa_recover**(*msg*, *sig*)

cryptnox_cli.lib.cryptos.main.**add**(*p1*, *p2*)

cryptnox_cli.lib.cryptos.main.**subtract**(*p1*, *p2*)

cryptnox_cli.lib.cryptos.main.**magicbyte_to_prefix**(*magicbyte*)

cryptnox_cli.lib.cryptos.mnemonic module

Mnemonic phrase generation and seed handling (BIP39-style operations).

This module is based on code from pybtctools: https://github.com/danvergara/pybtctools/blob/master/bitcoin/mnemonic.py

cryptnox_cli.lib.cryptos.mnemonic.**is_CJK**(*c*)

cryptnox_cli.lib.cryptos.mnemonic.**normalize_text**(*seed*)

cryptnox_cli.lib.cryptos.mnemonic.**eint_to_bytes**(*entint*, *entbits*)

cryptnox_cli.lib.cryptos.mnemonic.**mnemonic_int_to_words**(*mint*, *mint_num_words*, *wordlist=wordlist_english*)

cryptnox_cli.lib.cryptos.mnemonic.**entropy_cs**(*entbytes*)

cryptnox_cli.lib.cryptos.mnemonic.**entropy_to_words**(*entbytes*, *wordlist=wordlist_english*)

cryptnox_cli.lib.cryptos.mnemonic.**words_bisect**(*word*, *wordlist=wordlist_english*)

cryptnox_cli.lib.cryptos.mnemonic.**words_split**(*wordstr*, *wordlist=wordlist_english*)

cryptnox_cli.lib.cryptos.mnemonic.**words_to_mnemonic_int**(*words*, *wordlist=wordlist_english*)

cryptnox_cli.lib.cryptos.mnemonic.**words_verify**(*words*, *wordlist=wordlist_english*)

cryptnox_cli.lib.cryptos.mnemonic.**bip39_normalize_passphrase**(*passphrase*)

cryptnox_cli.lib.cryptos.mnemonic.**bip39_is_checksum_valid**(*mnemonic*)

cryptnox_cli.lib.cryptos.mnemonic.**mnemonic_to_seed**(*mnemonic_phrase*, *passphrase=''*, *passphrase_prefix=b'mnemonic'*)

cryptnox_cli.lib.cryptos.mnemonic.**bip39_mnemonic_to_seed**(*mnemonic_phrase*, *passphrase=''*)

cryptnox_cli.lib.cryptos.mnemonic.**electrum_mnemonic_to_seed**(*mnemonic_phrase*, *passphrase=''*)

cryptnox_cli.lib.cryptos.mnemonic.**is_old_seed**(*seed*)

cryptnox_cli.lib.cryptos.mnemonic.**seed_prefix**(*seed_type*)

cryptnox_cli.lib.cryptos.mnemonic.**seed_type**(*x*)

cryptnox_cli.lib.cryptos.mnemonic.**is_seed**(*x*)

cryptnox_cli.lib.cryptos.mnemonic.**words_mine**(*prefix*, *entbits*, *satisfunction*, *wordlist=wordlist_english*, *randombits=random.getrandbits*)

cryptnox_cli.lib.cryptos.pbkdf2 module

PBKDF2 key derivation function implementation and helpers.

cryptnox_cli.lib.cryptos.pbkdf2.**isunicode**(*s*)

cryptnox_cli.lib.cryptos.pbkdf2.**isbytes**(*s*)

cryptnox_cli.lib.cryptos.pbkdf2.**isinteger**(*n*)

cryptnox_cli.lib.cryptos.pbkdf2.**callable**(*obj*)

cryptnox_cli.lib.cryptos.pbkdf2.**b**(*s*)

cryptnox_cli.lib.cryptos.pbkdf2.**binxor**(*a*, *b*)

cryptnox_cli.lib.cryptos.pbkdf2.**b64encode**(*data*, *chars='+/'*)

cryptnox_cli.lib.cryptos.pbkdf2.**b2a_hex**(*s*)

class cryptnox_cli.lib.cryptos.pbkdf2.**PBKDF2**(*passphrase*, *salt*, *iterations=1000*, *digestmodule=SHA1*, *macmodule=HMAC*)

Bases: object

__init__(*passphrase*, *salt*, *iterations=1000*, *digestmodule=SHA1*, *macmodule=HMAC*)

static crypt(*word*, *salt=None*, *iterations=None*)

PBKDF2-based unix crypt(3) replacement.

The number of iterations specified in the salt overrides the 'iterations' parameter.

The effective hash length is 192 bits.

read(*bytes*)

Read the specified number of key bytes.

hexread(*octets*)

Read the specified number of octets. Return them as hexadecimal.

Note that len(obj.hexread(n)) == 2*n.

close()

Close the stream.

cryptnox_cli.lib.cryptos.pbkdf2.**crypt**(*word*, *salt=None*, *iterations=None*)

PBKDF2-based unix crypt(3) replacement.

The number of iterations specified in the salt overrides the 'iterations' parameter.

The effective hash length is 192 bits.

cryptnox_cli.lib.cryptos.ripemd module

RIPEMD-160 hash function implementation in pure Python.

class cryptnox_cli.lib.cryptos.ripemd.**RIPEMD160**(*arg=None*)

Bases: object

Return a new RIPEMD160 object. An optional string argument may be provided; if present, this string will be automatically hashed.

__init__(*arg=None*)

update(*arg*)

digest()

hexdigest()

copy()

cryptnox_cli.lib.cryptos.ripemd.**new**(*arg=None*)

Return a new RIPEMD160 object. An optional string argument may be provided; if present, this string will be automatically hashed.

class cryptnox_cli.lib.cryptos.ripemd.**RMDContext**

Bases: object

__init__()

copy()

cryptnox_cli.lib.cryptos.ripemd.**ROL**(*n*, *x*)

cryptnox_cli.lib.cryptos.ripemd.**F0**(*x*, *y*, *z*)

cryptnox_cli.lib.cryptos.ripemd.**F1**(*x*, *y*, *z*)

cryptnox_cli.lib.cryptos.ripemd.**F2**(*x*, *y*, *z*)

cryptnox_cli.lib.cryptos.ripemd.**F3**(*x*, *y*, *z*)

cryptnox_cli.lib.cryptos.ripemd.**F4**(*x*, *y*, *z*)

cryptnox_cli.lib.cryptos.ripemd.**R**(*a*, *b*, *c*, *d*, *e*, *Fj*, *Kj*, *sj*, *rj*, *X*)

cryptnox_cli.lib.cryptos.ripemd.**RMD160Transform**(*state*, *block*)

cryptnox_cli.lib.cryptos.ripemd.**RMD160Update**(*ctx*, *inp*, *inplen*)

cryptnox_cli.lib.cryptos.ripemd.**RMD160Final**(*ctx*)

cryptnox_cli.lib.cryptos.segwit_addr module

SegWit address encoding/decoding utilities (bech32/bech32m).

cryptnox_cli.lib.cryptos.segwit_addr.**bech32_polymod**(*values*)

Internal function that computes the Bech32 checksum.

cryptnox_cli.lib.cryptos.segwit_addr.**bech32_hrp_expand**(*hrp*)

Expand the HRP into values for checksum computation.

cryptnox_cli.lib.cryptos.segwit_addr.**bech32_verify_checksum**(*hrp*, *data*)

Verify a checksum given HRP and converted data characters.

cryptnox_cli.lib.cryptos.segwit_addr.**bech32_create_checksum**(*hrp*, *data*)

Compute the checksum values given HRP and data.

cryptnox_cli.lib.cryptos.segwit_addr.**bech32_encode**(*hrp*, *data*)

Compute a Bech32 string given HRP and data values.

cryptnox_cli.lib.cryptos.segwit_addr.**bech32_decode**(*bech*)

Validate a Bech32 string, and determine HRP and data.

cryptnox_cli.lib.cryptos.segwit_addr.**convertbits**(*data*, *frombits*, *tobits*, *pad=True*)

General power-of-2 base conversion.

cryptnox_cli.lib.cryptos.segwit_addr.**decode**(*hrp*, *addr*)

Decode a segwit address.

cryptnox_cli.lib.cryptos.segwit_addr.**encode**(*hrp*, *witver*, *witprog*)

Encode a segwit address.

cryptnox_cli.lib.cryptos.specials module

Low-level helpers, type adapters, and encoding utilities shared across modules.

cryptnox_cli.lib.cryptos.specials.**bin_dbl_sha256**(*s*)

cryptnox_cli.lib.cryptos.specials.**lpad**(*msg*, *symbol*, *length*)

cryptnox_cli.lib.cryptos.specials.**get_code_string**(*base*)

cryptnox_cli.lib.cryptos.specials.**changebase**(*string*, *frm*, *to*, *minlen=0*)

cryptnox_cli.lib.cryptos.specials.**bin_to_b58check**(*inp*, *magicbyte=0*)

cryptnox_cli.lib.cryptos.specials.**bytes_to_hex_string**(*b*)

cryptnox_cli.lib.cryptos.specials.**safe_from_hex**(*s*)

cryptnox_cli.lib.cryptos.specials.**from_int_representation_to_bytes**(*a*)

cryptnox_cli.lib.cryptos.specials.**from_int_to_byte**(*a*)

cryptnox_cli.lib.cryptos.specials.**from_byte_to_int**(*a*)

cryptnox_cli.lib.cryptos.specials.**from_string_to_bytes**(*a*)

cryptnox_cli.lib.cryptos.specials.**safe_hexlify**(*a*)

cryptnox_cli.lib.cryptos.specials.**encode**(*val*, *base*, *minlen=0*)

cryptnox_cli.lib.cryptos.specials.**decode**(*string*, *base*)

cryptnox_cli.lib.cryptos.specials.**random_string**(*x*)

cryptnox_cli.lib.cryptos.stealth module

Stealth address generation and parsing utilities.

cryptnox_cli.lib.cryptos.stealth.**shared_secret_sender**(*scan_pubkey*, *ephem_privkey*)

cryptnox_cli.lib.cryptos.stealth.**shared_secret_receiver**(*ephem_pubkey*, *scan_privkey*)

cryptnox_cli.lib.cryptos.stealth.**uncover_pay_pubkey_sender**(*scan_pubkey*, *spend_pubkey*, *ephem_privkey*)

cryptnox_cli.lib.cryptos.stealth.**uncover_pay_pubkey_receiver**(*scan_privkey*, *spend_pubkey*, *ephem_pubkey*)

cryptnox_cli.lib.cryptos.stealth.**uncover_pay_privkey**(*scan_privkey*, *spend_privkey*, *ephem_pubkey*)

cryptnox_cli.lib.cryptos.stealth.**pubkeys_to_basic_stealth_address**(*scan_pubkey*, *spend_pubkey*, *magic_byte=42*)

cryptnox_cli.lib.cryptos.stealth.**basic_stealth_address_to_pubkeys**(*stealth_address*)

cryptnox_cli.lib.cryptos.stealth.**mk_stealth_metadata_script**(*ephem_pubkey*, *nonce*)

cryptnox_cli.lib.cryptos.stealth.**mk_stealth_tx_outputs**(*stealth_addr*, *value*, *ephem_privkey*, *nonce*, *network='btc'*)

cryptnox_cli.lib.cryptos.stealth.**ephem_pubkey_from_tx_script**(*stealth_tx_script*)

cryptnox_cli.lib.cryptos.transaction module

Transaction building, serialization, signing, and parsing utilities.

cryptnox_cli.lib.cryptos.transaction.**json_is_base**(*obj*, *base*)

cryptnox_cli.lib.cryptos.transaction.**json_changebase**(*obj*, *changer*)

cryptnox_cli.lib.cryptos.transaction.**encode_1_byte**(*val*)

cryptnox_cli.lib.cryptos.transaction.**encode_4_bytes**(*val*)

cryptnox_cli.lib.cryptos.transaction.**encode_8_bytes**(*val*)

cryptnox_cli.lib.cryptos.transaction.**list_to_bytes**(*vals*)

cryptnox_cli.lib.cryptos.transaction.**dbl_sha256_list**(*vals*)

cryptnox_cli.lib.cryptos.transaction.**is_segwit**(*tx*)

cryptnox_cli.lib.cryptos.transaction.**deserialize**(*tx*)

cryptnox_cli.lib.cryptos.transaction.**serialize**(*txobj*, *include_witness=True*)

cryptnox_cli.lib.cryptos.transaction.**uahf_digest**(*txobj*, *i*)

cryptnox_cli.lib.cryptos.transaction.**signature_form**(*tx*, *i*, *script*, *hashcode=SIGHASH_ALL*)

cryptnox_cli.lib.cryptos.transaction.**der_encode_sig**(*v*, *r*, *s*)

cryptnox_cli.lib.cryptos.transaction.**der_decode_sig**(*sig*)

cryptnox_cli.lib.cryptos.transaction.**is_bip66**(*sig*)

Checks hex DER sig for BIP66 consistency

cryptnox_cli.lib.cryptos.transaction.**txhash**(*tx*, *hashcode=None*, *wtxid=True*)

cryptnox_cli.lib.cryptos.transaction.**public_txhash**(*tx*, *hashcode=None*)

cryptnox_cli.lib.cryptos.transaction.**bin_txhash**(*tx*, *hashcode=None*)

cryptnox_cli.lib.cryptos.transaction.**ecdsa_tx_sign**(*tx*, *priv*, *hashcode=SIGHASH_ALL*)

cryptnox_cli.lib.cryptos.transaction.**ecdsa_tx_verify**(*tx*, *sig*, *pub*, *hashcode=SIGHASH_ALL*)

cryptnox_cli.lib.cryptos.transaction.**ecdsa_tx_recover**(*tx*, *sig*, *hashcode=SIGHASH_ALL*)

cryptnox_cli.lib.cryptos.transaction.**mk_pubkey_script**(*addr*)

Used in converting p2pkh address to input or output script

cryptnox_cli.lib.cryptos.transaction.**mk_scripthash_script**(*addr*)

Used in converting p2sh address to output script

cryptnox_cli.lib.cryptos.transaction.**output_script_to_address**(*script*, *magicbyte=0*)

cryptnox_cli.lib.cryptos.transaction.**mk_p2w_scripthash_script**(*witver*, *witprog*)

Used in converting a decoded pay to witness script hash address to output script

cryptnox_cli.lib.cryptos.transaction.**mk_p2wpkh_redeemscript**(*pubkey*)

Used in converting public key to p2wpkh script

cryptnox_cli.lib.cryptos.transaction.**mk_p2wpkh_script**(*pubkey*)

Used in converting public key to p2wpkh script

cryptnox_cli.lib.cryptos.transaction.**mk_p2wpkh_scriptcode**(*pubkey*)

Used in signing for tx inputs

cryptnox_cli.lib.cryptos.transaction.**p2wpkh_nested_script**(*pubkey*)

cryptnox_cli.lib.cryptos.transaction.**deserialize_script**(*script*)

cryptnox_cli.lib.cryptos.transaction.**serialize_script_unit**(*unit*)

cryptnox_cli.lib.cryptos.transaction.**serialize_script**(*script*)

cryptnox_cli.lib.cryptos.transaction.**mk_multisig_script**(**args*)

Parameters

args – List of public keys to used to create multisig and M, the number of signatures required to spend

Returns

multisig script

cryptnox_cli.lib.cryptos.transaction.**verify_tx_input**(*tx*, *i*, *script*, *sig*, *pub*)

cryptnox_cli.lib.cryptos.transaction.**multisign**(*tx*, *i*, *script*, *pk*, *hashcode=SIGHASH_ALL*)

cryptnox_cli.lib.cryptos.transaction.**apply_multisignatures**(*txobj*, *i*, *script*, **args*)

cryptnox_cli.lib.cryptos.transaction.**is_inp**(*arg*)

cryptnox_cli.lib.cryptos.transaction.**select**(*unspent*, *value*)

cryptnox_cli.lib.cryptos.wallet module

Simple wallet abstractions for addresses, balances, and signing flows.

This module is based on code from pybtctools: https://github.com/primal100/pybitcointools/blob/master/cryptos/wallet.py

class cryptnox_cli.lib.cryptos.wallet.**HDWallet**(*keystore*, *num_addresses=0*, *last_receiving_index=0*, *last_change_index=0*)

Bases: object

__init__(*keystore*, *num_addresses=0*, *last_receiving_index=0*, *last_change_index=0*)

privkey(*address*, *formt='wif_compressed'*, *password=None*)

export_privkeys(*password=None*)

pubkey_receiving(*index*)

pubkey_change(*index*)

pubtoaddr(*pubkey*)

receiving_address(*index*)

change_address(*index*)

property receiving_addresses

property change_addresses

new_receiving_address_range(*num*)

new_change_address_range(*num*)

new_receiving_addresses(*num=10*)

new_change_addresses(*num=10*)

new_receiving_address()

new_change_address()

balance()

unspent()

select()

history()

sign(*tx*, *password=None*)

mksend(*outs*)

sign_message(*message*, *address*, *password=None*)

is_mine(*address*)

is_change(*address*)

account(*address*, *password=None*)

details(*password=None*)

cryptnox_cli.lib.cryptos.wallet_utils module

General wallet helper functions (formatting, hashing, conversions).

cryptnox_cli.lib.cryptos.wallet_utils.**hmac_sha_512**(*x*, *y*)

cryptnox_cli.lib.cryptos.wallet_utils.**rev_hex**(*s*)

cryptnox_cli.lib.cryptos.wallet_utils.**int_to_hex**(*i*, *length=1*)

exception cryptnox_cli.lib.cryptos.wallet_utils.**InvalidPassword**
Bases: Exception

exception cryptnox_cli.lib.cryptos.wallet_utils.**InvalidPasswordException**
Bases: Exception

exception cryptnox_cli.lib.cryptos.wallet_utils.**InvalidPadding**
Bases: Exception

cryptnox_cli.lib.cryptos.wallet_utils.**assert_bytes**(**args*)
porting helper, assert args type

cryptnox_cli.lib.cryptos.wallet_utils.**append_PKCS7_padding**(*data*)

cryptnox_cli.lib.cryptos.wallet_utils.**strip_PKCS7_padding**(*data*)

cryptnox_cli.lib.cryptos.wallet_utils.**aes_encrypt_with_iv**(*key*, *iv*, *data*)

cryptnox_cli.lib.cryptos.wallet_utils.**aes_decrypt_with_iv**(*key*, *iv*, *data*)

cryptnox_cli.lib.cryptos.wallet_utils.**EncodeAES**(*secret*, *s*)

cryptnox_cli.lib.cryptos.wallet_utils.**DecodeAES**(*secret*, *e*)

cryptnox_cli.lib.cryptos.wallet_utils.**pw_encode**(*s*, *password*)

cryptnox_cli.lib.cryptos.wallet_utils.**pw_decode**(*s*, *password*)

cryptnox_cli.lib.cryptos.wallet_utils.**is_new_seed**(*x*, *prefix=version.SEED_PREFIX*)

cryptnox_cli.lib.cryptos.wallet_utils.**seed_type**(*x*)

cryptnox_cli.lib.cryptos.wallet_utils.**is_seed**(*x*)

cryptnox_cli.lib.cryptos.wallet_utils.**inv_dict**(*d*)

cryptnox_cli.lib.cryptos.wallet_utils.**is_minikey**(*text*)

cryptnox_cli.lib.cryptos.wallet_utils.**minikey_to_private_key**(*text*)

cryptnox_cli.lib.cryptos.wallet_utils.**get_pubkeys_from_secret**(*secret*)

cryptnox_cli.lib.cryptos.wallet_utils.**xprv_header**(*xtype*)

cryptnox_cli.lib.cryptos.wallet_utils.**xpub_header**(*xtype*)

cryptnox_cli.lib.cryptos.wallet_utils.**number_of_significant_digits**(*number: float*) → int

Module contents

Cryptocurrency utilities package - aggregates and re-exports submodules for easier imports.

The `cryptnox_cli.lib.cryptos` package provides cryptocurrency utilities and re-exports functionality from:

- `cryptnox_cli.lib.cryptos.blocks` - Block-related utilities
- `cryptnox_cli.lib.cryptos.composite` - Composite key handling
- `cryptnox_cli.lib.cryptos.deterministic` - Deterministic wallet functions
- `cryptnox_cli.lib.cryptos.main` - Main crypto operations
- `cryptnox_cli.lib.cryptos.mnemonic` - Mnemonic phrase handling
- `cryptnox_cli.lib.cryptos.specials` - Special cryptocurrency operations
- `cryptnox_cli.lib.cryptos.stealth` - Stealth address support
- `cryptnox_cli.lib.cryptos.transaction` - Transaction handling
- `cryptnox_cli.lib.cryptos.coins` - Coin implementations
- `cryptnox_cli.lib.cryptos.keystore` - Key storage utilities
- `cryptnox_cli.lib.cryptos.wallet` - Wallet functionality

All public members from these modules are available directly from `cryptnox_cli.lib.cryptos`.

Module contents

The `cryptnox_cli.lib` package provides library functionality for cryptocurrency operations. The main subpackage is `cryptnox_cli.lib.cryptos` which contains comprehensive cryptocurrency utilities.

cryptnox_cli.wallet package

Subpackages

cryptnox_cli.wallet.eth package

Submodules

cryptnox_cli.wallet.eth.api module

A basic Ethereum wallet library

cryptnox_cli.wallet.eth.api.**address**(*public_key: str*) → str

cryptnox_cli.wallet.eth.api.**checksum_address**(*public_key: str*) → str

class cryptnox_cli.wallet.eth.api.**Api**(*endpoint: str*, *network:* EthNetwork | *str*, *api_key: str*)

Bases: object

PATH = "m/44'/60'/0'/0/0"

SYMBOL = 'eth'

__init__(*endpoint: str*, *network:* EthNetwork | *str*, *api_key: str*)

property block_number

contract(*address=''*, *abi=''*)

get_transaction_count(*address: str*, *blocks: str = None*) → int

get_balance(*address: str*) → int

property gas_price

property network

transaction_hash(*transaction: Dict[str, Any]*, *vrs: bool = False*)

push(*transaction*, *signature*, *public_key*)

class cryptnox_cli.wallet.eth.api.**EthValidator**(*endpoint: str = 'publicnode'*, *network: str = 'sepolia'*, *price: int = 8*, *limit: int = 2500*, *derivation: str = 'DERIVE'*, *api_key=''*)

Bases: object

Class defining Ethereum validators

__init__(*endpoint: str = 'publicnode'*, *network: str = 'sepolia'*, *price: int = 8*, *limit: int = 2500*, *derivation: str = 'DERIVE'*, *api_key=''*)

endpoint

Validator to validate endpoints for the Ethereum network

network

Class for validating if value is part of the enum

price
Class for validating if value is integer

limit
Class for validating if value is integer

derivation
Class for validating if value is part of the enum

api_key

validate()

cryptnox_cli.wallet.eth.endpoint module

Module for endpoints that can be used for working with the Ethereum network

class cryptnox_cli.wallet.eth.endpoint.**EndpointValidator**(*valid_values: str = None*)

Bases: Validator

Validator to validate endpoints for the Ethereum network

__init__(*valid_values: str = None*)

validate(*value*)

Parameters
value – Evaluated value

Returns
None

Raises
ValidationError – Validation criteria not satisfied

class cryptnox_cli.wallet.eth.endpoint.**Endpoint**(*network:* EthNetwork, *api_key: str = ''*)

Bases: object

Abstract base class for interface for endpoint implementations

__init__(*network:* EthNetwork, *api_key: str = ''*)

abstract property available_networks: List[EthNetwork]
Ethereum networks handled by the endpoint implementation :rtype: List[enums.EthNetwork]

Type
return

abstract property domain: str
Domain that is used by the endpoint implementation :rtype: str

Type
return

abstract property name: str
Name of the endpoint implementation :rtype: str

Type
return

abstract property provider: str
Full URL that can be used as Web3 provider :rtype: str

Type
return

class cryptnox_cli.wallet.eth.endpoint.**InfuraEndpoint**(*network:* EthNetwork, *api_key: str = ''*)

Bases: Endpoint

Implementation of the Infura endpoint

name = 'infura'

available_networks = ['MAINNET', 'KOVAN', 'GOERLI', 'RINKEBY', 'SEPOLIA']

__init__(*network:* EthNetwork, *api_key: str = ''*)

property domain: str
Domain that is used by the endpoint implementation :rtype: str

Type
return

property provider: str
Full URL that can be used as Web3 provider :rtype: str

Type
return

class cryptnox_cli.wallet.eth.endpoint.**CryptnoxEndpoint**(*network:* EthNetwork, *api_key: str = ''*)

Bases: Endpoint

Implementation of the Cryptnox endpoint

name = 'cryptnox'

available_networks = ['MAINNET', 'SEPOLIA']

property domain: str
Domain that is used by the endpoint implementation :rtype: str

Type
return

property provider: str
Full URL that can be used as Web3 provider :rtype: str

Type
return

class cryptnox_cli.wallet.eth.endpoint.**PublicNodeEndpoint**(*network:* EthNetwork, *api_key: str = ''*)

Bases: Endpoint

Implementation of the PublicNode public RPC endpoint (free, no API key required)

name = 'publicnode'

available_networks = ['MAINNET', 'SEPOLIA']

property domain: str

Domain that is used by the endpoint implementation :rtype: str

Type
return

property provider: str

Full URL that can be used as Web3 provider :rtype: str

Type
return

class cryptnox_cli.wallet.eth.endpoint.**DirectEndpoint**(*url: str*, *network:* EthNetwork)

Bases: object

__init__(*url: str*, *network:* EthNetwork)

property provider: str

cryptnox_cli.wallet.eth.endpoint.**factory**(*endpoint: str*, *network:* EthNetwork, *api_key: str = ''*) → *Endpoint*

Factory method for Endpoint instances

Parameters

- **endpoint** – Name of the endpoint to use
- **network** – Ethereum network to use
- **api_key** – API key to use on the endpoint

Type
str

Type
enums.EthNetwork

Type
str

Returns
Return an Endpoint instance that can be used to get the urls

Return type
Endpoint

Raises
ValueError – In case endpoint with endpoint name wasn't found

Module contents

Ethereum wallet utilities - aggregates and re-exports API components for easier imports.

The `cryptnox_cli.wallet.eth` package provides Ethereum wallet utilities and exports:

`cryptnox_cli.wallet.eth.`**`address`**`(`*`public_key`*`)`

Generate an Ethereum address from a public key. See `cryptnox_cli.wallet.eth.api.address()` for details.

`class` `cryptnox_cli.wallet.eth.`**`Api`**

Ethereum API interface. See `cryptnox_cli.wallet.eth.api.Api` for details.

`cryptnox_cli.wallet.eth.`**`checksum_address`**`(`*`address_str`*`)`

Convert an address to checksummed format. See `cryptnox_cli.wallet.eth.api.checksum_address()` for details.

`class` `cryptnox_cli.wallet.eth.`**`EthValidator`**

Ethereum address and transaction validator. See `cryptnox_cli.wallet.eth.api.EthValidator` for details.

Submodules

cryptnox_cli.wallet.btc module

A basic BTC wallet library

`class` `cryptnox_cli.wallet.btc.`**`BtcNetworks`**`(`*`*values`*`)`

Bases: `Enum`

Class defining possible Bitcoin networks

`MAINNET = 'mainnet'`

`TESTNET = 'testnet'`

`TESTNET4 = 'testnet4'`

`class` `cryptnox_cli.wallet.btc.`**`BlockCypherApi`**

Bases: `object`

`__init__``(`*`api_key`*`, `*`network`*`)`

`get_data``(`*`endpoint: str`*`, `*`params: Dict = None`*`, `*`data: bytes = None`*`)` → None

Return type
None

Parameters

- **`endpoint`** – str
- **`params`** – dict
- **`data`** – bytes

check_api_resp() → None

Return type
None

get_utx_os(*addr: str*, *n_conf*) → List

Parameters

- **addr** – str
- **n_conf** – int (0 or 1)

Returns
List

push_tx(*tx_hex: str*) → Dict

Parameters
tx_hex

Returns
str

get_key(*key_char: str*) → Dict
:param key_char:str :return: Dict

class cryptnox_cli.wallet.btc.**BlkHubApi**

Bases: object

__init__(*network*, *api_key: str = ''*, *endpoint: str = ''*)

static get_api(*network: str*) → str
Get API url for given network

Parameters
network (str)

Returns
API url

Return type
str

get_data(*endpoint: str*, *params: Dict = None*, *data: bytes = None*) → None

Parameters

- **endpoint** – str
- **params** – dict
- **data** – bytes

Returns
None

check_api_resp() → None

Return type
None

get_fee_estimates(*blocks=6*) → int

get_utx_os(*addr: str*, *_n_conf: int*) → List

:param addr:str :param int _n_conf: 0 or 1 :return: list

push_tx(*tx_hex: str*) → List

Parameters
tx_hex – str

Returns
List

get_key(*key_char: str*) → List

Parameters
key_char – str

Returns
list

cryptnox_cli.wallet.btc.**test_addr**(*btc_addr: str*)

Parameters
btc_addr – str

Returns

class cryptnox_cli.wallet.btc.**BTCwallet**

Bases: object

Parameters

- **pubkey** – str
- **coin_type** – str
- **api**
- **connection**

PATH = "m/44'/0'/0'/0/0"

__init__(*pubkey: str*, *coin_type: str*, *api*, *card*) → None

Parameters

- **pubkey** – str
- **coin_type** – str
- **api**
- **connection**

get_utx_os(*n_conf: int = 0*)

Parameters
n_conf – int (0 or 1)

Returns

get_balance() → float

Returns
float

get_fee_estimate()

prepare(*to_addr: str*, *payment_value: float*, *fee: float*, *utx_os: List = None*) → float | int

Parameters

- **to_addr** – str
- **payment_value** – float
- **fee** – float
- **utx_os** – pre-fetched UTXOs (fetched in parallel with fee estimate)

Returns
Union[float, int]

send(*to_addr: str*, *payment_value: float*, *signature: List[bytes]*) → str

Parameters

- **to_addr** – str
- **payment_value** – float

Returns
str

static balance_fm_utxos(*utxos*) → float

Parameters
utxos

Returns
float

static select_utxos(*amount: float*, *utxos*) → List

Parameters

- **amount** – float
- **utxos**

Returns
float

class cryptnox_cli.wallet.btc.**BtcValidator**(*network: str = 'testnet'*, *fees: int = 2000*, *derivation: str = 'DERIVE'*, *api_key: str = ''*, *endpoint: str = ''*)

Bases: object

Class defining Bitcoin validators

__init__(*network: str = 'testnet'*, *fees: int = 2000*, *derivation: str = 'DERIVE'*, *api_key: str = ''*, *endpoint: str = ''*)

network
Class for validating if value is part of the enum

fees
Class for validating if value is integer

derivation
Class for validating if value is part of the enum

api_key

endpoint

cryptnox_cli.wallet.validators module

Module for defining validators

exception cryptnox_cli.wallet.validators.**ValidationError**
Bases: Exception

Exception for indicating that validation criteria are not met

class cryptnox_cli.wallet.validators.**Validator**(*valid_values: str = None*)
Bases: ABC

Validator class defining setters and getters

__init__(*valid_values: str = None*)

abstractmethod validate(*value*)

Parameters
value – Evaluated value

Returns
None

Raises
ValidationError – Validation criteria not satisfied

class cryptnox_cli.wallet.validators.**AnyValidator**(*valid_values: str = None*)
Bases: Validator

validate(*value*)

Parameters
value – Evaluated value

Returns
None

Raises
ValidationError – Validation criteria not satisfied

class cryptnox_cli.wallet.validators.**IntValidator**(*max_value: int = None*, *min_value: int = None*)
Bases: Validator

Class for validating if value is integer

__init__(*max_value: int = None*, *min_value: int = None*)

validate(*value*) → int

Parameters
value – Evaluated value

Returns
None

Raises
ValidationError – Validation criteria not satisfied

class cryptnox_cli.wallet.validators.**EnumValidator**(*current_enum*)

Bases: Validator

Class for validating if value is part of the enum

__init__(*current_enum*)

validate(*value*) → None

Parameters
value – Evaluated value

Returns
None

Raises
ValidationError – Validation criteria not satisfied

class cryptnox_cli.wallet.validators.**UrlValidator**(*valid_values: str = None*)

Bases: Validator

Class for validating URLs

validate(*value: str*) → str

Parameters
value – Evaluated value

Returns
None

Raises
ValidationError – Validation criteria not satisfied

cryptnox_cli.wallet.validators.**is_int**(*value: Any*) → bool

Module contents

The cryptnox_cli.wallet package provides wallet functionality for various cryptocurrencies. It includes Bitcoin wallet operations, Ethereum wallet utilities, and validation functions.

3.1.2 Submodules

3.1.3 cryptnox_cli.config module

3.1.4 cryptnox_cli.enums module

Module for defining enumerations used throughout the CryptnoxPro application.

class cryptnox_cli.enums.**EthNetwork**(**values*)

Bases: Enum

Class defining possible Ethereum networks

MAINNET = 1

KOVAN = 42

GOERLI = 5

RINKEBY = 4

SEPOLIA = 11155111

class cryptnox_cli.enums.**SolanaNetwork**(**values*)

Bases: Enum

Class defining possible Solana networks

MAINNET = 1

DEVNET = 2

TESTNET = 3

class cryptnox_cli.enums.**Command**(**values*)

Bases: Enum

BTC = 'btc'

CARD_CONFIGURATION = 'card_conf'

CHANGE_PIN = 'change_pin'

CHANGE_PUK = 'change_puk'

CONFIG = 'config'

ETH = 'eth'

HISTORY = 'history'

INFO = 'info'

INITIALIZE = 'init'

CARD = 'list'

SERVER = 'server'

RESET = 'reset'

```
SEED = 'seed'

SOLANA = 'solana'

UNLOCK_PIN = 'unlock_pin'

USER_KEY = 'user_key'

TRANSFER = 'transfer'

CERTIFICATE = 'cert'
```

3.1.5 cryptnox_cli.constants module

Centralized constants for the Cryptnox CLI.

3.1.6 cryptnox_cli.interactive_cli module

3.1.7 cryptnox_cli.main module

3.1.8 Module contents

The main `cryptnox_cli` package (Cryptnox CLI) exports:

`cryptnox_cli.`**`__version__: str = "1.0.5"`**

Current version of the Cryptnox CLI package.

4 License

This project is available under the terms of the GNU LESSER GENERAL PUBLIC LICENSE (LGPL) v3.

```
                   GNU LESSER GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.


  This version of the GNU Lesser General Public License incorporates
the terms and conditions of version 3 of the GNU General Public
License, supplemented by the additional permissions listed below.

  0. Additional Definitions.

  As used herein, "this License" refers to version 3 of the GNU Lesser
General Public License, and the "GNU GPL" refers to version 3 of the GNU
General Public License.

  "The Library" refers to a covered work governed by this License,
other than an Application or a Combined Work as defined below.

  An "Application" is any work that makes use of an interface provided
by the Library, but which is not otherwise based on the Library.
Defining a subclass of a class defined by the Library is deemed a mode
of using an interface provided by the Library.

  A "Combined Work" is a work produced by combining or linking an
Application with the Library.  The particular version of the Library
with which the Combined Work was made is also called the "Linked
Version".

  The "Minimal Corresponding Source" for a Combined Work means the
Corresponding Source for the Combined Work, excluding any source code
for portions of the Combined Work that, considered in isolation, are
based on the Application, and not on the Linked Version.

  The "Corresponding Application Code" for a Combined Work means the
object code and/or source code for the Application, including any data
and utility programs needed for reproducing the Combined Work from the
Application, but excluding the System Libraries of the Combined Work.

  1. Exception to Section 3 of the GNU GPL.
```

```
  You may convey a covered work under sections 3 and 4 of this License
without being bound by section 3 of the GNU GPL.

  2. Conveying Modified Versions.

  If you modify a copy of the Library, and, in your modifications, a
facility refers to a function or data to be supplied by an Application
that uses the facility (other than as an argument passed when the
facility is invoked), then you may convey a copy of the modified
version:

   a) under this License, provided that you make a good faith effort to
   ensure that, in the event an Application does not supply the
   function or data, the facility still operates, and performs
   whatever part of its purpose remains meaningful, or

   b) under the GNU GPL, with none of the additional permissions of
   this License applicable to that copy.

  3. Object Code Incorporating Material from Library Header Files.

  The object code form of an Application may incorporate material from
a header file that is part of the Library.  You may convey such object
code under terms of your choice, provided that, if the incorporated
material is not limited to numerical parameters, data structure
layouts and accessors, or small macros, inline functions and templates
(ten or fewer lines in length), you do both of the following:

   a) Give prominent notice with each copy of the object code that the
   Library is used in it and that the Library and its use are
   covered by this License.

   b) Accompany the object code with a copy of the GNU GPL and this license
   document.

  4. Combined Works.

  You may convey a Combined Work under terms of your choice that,
taken together, effectively do not restrict modification of the
portions of the Library contained in the Combined Work and reverse
engineering for debugging such modifications, if you also do each of
the following:

   a) Give prominent notice with each copy of the Combined Work that
   the Library is used in it and that the Library and its use are
   covered by this License.

   b) Accompany the Combined Work with a copy of the GNU GPL and this license
   document.
```

```
   c) For a Combined Work that displays copyright notices during
   execution, include the copyright notice for the Library among
   these notices, as well as a reference directing the user to the
   copies of the GNU GPL and this license document.

   d) Do one of the following:

       0) Convey the Minimal Corresponding Source under the terms of this
       License, and the Corresponding Application Code in a form
       suitable for, and under terms that permit, the user to
       recombine or relink the Application with a modified version of
       the Linked Version to produce a modified Combined Work, in the
       manner specified by section 6 of the GNU GPL for conveying
       Corresponding Source.

       1) Use a suitable shared library mechanism for linking with the
       Library.  A suitable mechanism is one that (a) uses at run time
       a copy of the Library already present on the user's computer
       system, and (b) will operate properly with a modified version
       of the Library that is interface-compatible with the Linked
       Version.

   e) Provide Installation Information, but only if you would otherwise
   be required to provide such information under section 6 of the
   GNU GPL, and only to the extent that such information is
   necessary to install and execute a modified version of the
   Combined Work produced by recombining or relinking the
   Application with a modified version of the Linked Version. (If
   you use option 4d0, the Installation Information must accompany
   the Minimal Corresponding Source and Corresponding Application
   Code. If you use option 4d1, you must provide the Installation
   Information in the manner specified by section 6 of the GNU GPL
   for conveying Corresponding Source.)

  5. Combined Libraries.

  You may place library facilities that are a work based on the
Library side by side in a single library together with other library
facilities that are not Applications and are not covered by this
License, and convey such a combined library under terms of your
choice, if you do both of the following:

   a) Accompany the combined library with a copy of the same work based
   on the Library, uncombined with any other library facilities,
   conveyed under the terms of this License.

   b) Give prominent notice with the combined library that part of it
   is a work based on the Library, and explaining where to find the
   accompanying uncombined form of the same work.
```

```
  6. Revised Versions of the GNU Lesser General Public License.

  The Free Software Foundation may publish revised and/or new versions
of the GNU Lesser General Public License from time to time. Such new
versions will be similar in spirit to the present version, but may
differ in detail to address new problems or concerns.

  Each version is given a distinguishing version number. If the
Library as you received it specifies that a certain numbered version
of the GNU Lesser General Public License "or any later version"
applies to it, you have the option of following the terms and
conditions either of that published version or of any later version
published by the Free Software Foundation. If the Library as you
received it does not specify a version number of the GNU Lesser
General Public License, you may choose any version of the GNU Lesser
General Public License ever published by the Free Software Foundation.

  If the Library as you received it specifies that a proxy can decide
whether future versions of the GNU Lesser General Public License shall
apply, that proxy's public statement of acceptance of any version is
permanent authorization for you to choose that version for the
Library.
```