
bip_utils

Emanuele Bellocchia

Apr 15, 2024

CONTENTS

1	Introduction	1
2	Supported coins	3
3	Install the package	7
4	Test and Coverage	9
5	Modules description	11
6	Documentation	13
7	Code examples	15
8	Buy me a coffee	17
9	License	19
10	Modules	21
10.1	bip_utils	21
10.1.1	addr	21
10.1.1.1	P2PKH_addr	21
10.1.1.2	P2SH_addr	23
10.1.1.3	P2TR_addr	25
10.1.1.4	P2WPKH_addr	26
10.1.1.5	ada_byron_addr	27
10.1.1.6	ada_shelley_addr	30
10.1.1.7	addr_dec_utils	32
10.1.1.8	addr_key_validator	34
10.1.1.9	algo_addr	36
10.1.1.10	aptos_addr	37
10.1.1.11	atom_addr	38
10.1.1.12	avax_addr	39
10.1.1.13	bch_addr_converter	40
10.1.1.14	egld_addr	41
10.1.1.15	eos_addr	42
10.1.1.16	ergo_addr	43
10.1.1.17	eth_addr	44
10.1.1.18	fil_addr	45
10.1.1.19	iaddr_decoder	46
10.1.1.20	iaddr_encoder	47

10.1.1.21	icx_addr	47
10.1.1.22	inj_addr	48
10.1.1.23	nano_addr	49
10.1.1.24	near_addr	50
10.1.1.25	neo_addr	51
10.1.1.26	okex_addr	52
10.1.1.27	one_addr	53
10.1.1.28	sol_addr	54
10.1.1.29	substrate_addr	55
10.1.1.30	sui_addr	56
10.1.1.31	trx_addr	57
10.1.1.32	xlm_addr	58
10.1.1.33	xmr_addr	60
10.1.1.34	xrp_addr	62
10.1.1.35	xtz_addr	63
10.1.1.36	zil_addr	64
10.1.2	algorand	65
10.1.2.1	mnemonic	65
10.1.2.1.1	algorand_entropy_generator	65
10.1.2.1.2	algorand_mnemonic	66
10.1.2.1.3	algorand_mnemonic_decoder	66
10.1.2.1.4	algorand_mnemonic_encoder	67
10.1.2.1.5	algorand_mnemonic_generator	67
10.1.2.1.6	algorand_mnemonic_utils	68
10.1.2.1.7	algorand_mnemonic_validator	69
10.1.2.1.8	algorand_seed_generator	69
10.1.3	base58	70
10.1.3.1	base58	70
10.1.3.2	base58_ex	72
10.1.3.3	base58_xmr	72
10.1.4	bech32	73
10.1.4.1	bch_bech32	73
10.1.4.2	bech32	75
10.1.4.3	bech32_base	77
10.1.4.4	bech32_ex	78
10.1.4.5	segwit_bech32	79
10.1.5	bip	80
10.1.5.1	bip32	80
10.1.5.1.1	base	80
10.1.5.1.1.1	bip32_base	80
10.1.5.1.1.2	ibip32_key_derivator	85
10.1.5.1.1.3	ibip32_mst_key_generator	86
10.1.5.1.2	bip32_const	86
10.1.5.1.3	bip32_ex	87
10.1.5.1.4	bip32_key_data	87
10.1.5.1.5	bip32_key_net_ver	93
10.1.5.1.6	bip32_key_ser	94
10.1.5.1.7	bip32_keys	96
10.1.5.1.8	bip32_path	99
10.1.5.1.9	bip32_utils	101
10.1.5.1.10	kholaw	102
10.1.5.1.10.1	bip32_kholaw_ed25519	102
10.1.5.1.10.2	bip32_kholaw_ed25519_key_derivator	103
10.1.5.1.10.3	bip32_kholaw_key_derivator_base	103

10.1.5.1.10.4	bip32_kholaw_mst_key_generator	104
10.1.5.1.11	slip10	104
10.1.5.1.11.1	bip32_slip10_ed25519	104
10.1.5.1.11.2	bip32_slip10_ed25519_blake2b	105
10.1.5.1.11.3	bip32_slip10_key_derivator	106
10.1.5.1.11.4	bip32_slip10_mst_key_generator	108
10.1.5.1.11.5	bip32_slip10_nist256p1	109
10.1.5.1.11.6	bip32_slip10_secp256k1	110
10.1.5.2	bip38	110
10.1.5.2.1	bip38	110
10.1.5.2.2	bip38_addr	112
10.1.5.2.3	bip38_ec	112
10.1.5.2.4	bip38_no_ec	114
10.1.5.3	bip39	116
10.1.5.3.1	bip39_entropy_generator	116
10.1.5.3.2	bip39_mnemonic	117
10.1.5.3.3	bip39_mnemonic_decoder	118
10.1.5.3.4	bip39_mnemonic_encoder	119
10.1.5.3.5	bip39_mnemonic_generator	119
10.1.5.3.6	bip39_mnemonic_utils	120
10.1.5.3.7	bip39_mnemonic_validator	121
10.1.5.3.8	bip39_seed_generator	121
10.1.5.3.9	ibip39_seed_generator	121
10.1.5.4	bip44	122
10.1.5.4.1	bip44	122
10.1.5.5	bip44_base	125
10.1.5.5.1	bip44_base	125
10.1.5.5.2	bip44_base_ex	130
10.1.5.5.3	bip44_keys	130
10.1.5.6	bip49	133
10.1.5.6.1	bip49	133
10.1.5.7	bip84	136
10.1.5.7.1	bip84	136
10.1.5.8	bip86	140
10.1.5.8.1	bip86	140
10.1.5.9	conf	143
10.1.5.9.1	bip44	143
10.1.5.9.1.1	bip44_coins	143
10.1.5.9.1.2	bip44_conf	146
10.1.5.9.1.3	bip44_conf_getter	150
10.1.5.9.2	bip49	152
10.1.5.9.2.1	bip49_coins	152
10.1.5.9.2.2	bip49_conf	153
10.1.5.9.2.3	bip49_conf_getter	154
10.1.5.9.3	bip84	156
10.1.5.9.3.1	bip84_coins	156
10.1.5.9.3.2	bip84_conf	156
10.1.5.9.3.3	bip84_conf_getter	156
10.1.5.9.4	bip86	157
10.1.5.9.4.1	bip86_coins	157
10.1.5.9.4.2	bip86_conf	158
10.1.5.9.4.3	bip86_conf_getter	158
10.1.5.9.5	common	159
10.1.5.9.5.1	bip_bitcoin_cash_conf	159

	10.1.5.9.5.2	bip_coin_conf	159
	10.1.5.9.5.3	bip_coins	162
	10.1.5.9.5.4	bip_conf_const	162
	10.1.5.9.5.5	bip_litecoin_conf	162
10.1.6	brainwallet		163
10.1.6.1	brainwallet		163
10.1.6.2	brainwallet_algo		164
10.1.6.3	brainwallet_algo_getter		166
10.1.6.4	ibrainwallet_algo		167
10.1.7	cardano		167
10.1.7.1	bip32		167
10.1.7.1.1	cardano_byron_legacy_bip32		167
10.1.7.1.2	cardano_byron_legacy_key_derivator		168
10.1.7.1.3	cardano_byron_legacy_mst_key_generator		168
10.1.7.1.4	cardano_icarus_bip32		169
10.1.7.1.5	cardano_icarus_mst_key_generator		169
10.1.7.2	byron		170
10.1.7.2.1	cardano_byron_legacy		170
10.1.7.3	cip1852		172
10.1.7.3.1	cip1852		172
10.1.7.3.2	conf		176
10.1.7.3.2.1	cip1852_coins		176
10.1.7.3.2.2	cip1852_conf		176
10.1.7.3.2.3	cip1852_conf_getter		176
10.1.7.4	mnemonic		177
10.1.7.4.1	cardano_byron_legacy_seed_generator		177
10.1.7.4.2	cardano_icarus_seed_generator		178
10.1.7.5	shelley		178
10.1.7.5.1	cardano_shelley		178
10.1.7.5.2	cardano_shelley_keys		180
10.1.8	coin_conf		182
10.1.8.1	coin_conf		182
10.1.8.2	coins_conf		183
10.1.9	ecc		186
10.1.9.1	common		186
10.1.9.1.1	dummy_point		186
10.1.9.1.2	ikeys		188
10.1.9.1.3	ipoint		192
10.1.9.2	conf		194
10.1.9.3	curve		194
10.1.9.3.1	elliptic_curve		194
10.1.9.3.2	elliptic_curve_getter		196
10.1.9.3.3	elliptic_curve_types		197
10.1.9.4	ecdsa		197
10.1.9.4.1	ecdsa_keys		197
10.1.9.5	ed25519		197
10.1.9.5.1	ed25519		197
10.1.9.5.2	ed25519_const		198
10.1.9.5.3	ed25519_keys		198
10.1.9.5.4	ed25519_point		201
10.1.9.5.5	ed25519_utils		204
10.1.9.5.6	lib		204
10.1.9.5.6.1	ed25519_lib		204
10.1.9.6	ed25519_blake2b		208

10.1.9.6.1	ed25519_blake2b	208
10.1.9.6.2	ed25519_blake2b_const	208
10.1.9.6.3	ed25519_blake2b_keys	209
10.1.9.6.4	ed25519_blake2b_point	211
10.1.9.7	ed25519_kholaw	212
10.1.9.7.1	ed25519_kholaw	212
10.1.9.7.2	ed25519_kholaw_const	212
10.1.9.7.3	ed25519_kholaw_keys	212
10.1.9.7.4	ed25519_kholaw_point	214
10.1.9.8	ed25519_monero	214
10.1.9.8.1	ed25519_monero	214
10.1.9.8.2	ed25519_monero_const	215
10.1.9.8.3	ed25519_monero_keys	215
10.1.9.8.4	ed25519_monero_point	217
10.1.9.9	nist256p1	217
10.1.9.9.1	nist256p1	217
10.1.9.9.2	nist256p1_const	217
10.1.9.9.3	nist256p1_keys	218
10.1.9.9.4	nist256p1_point	220
10.1.9.10	secp256k1	223
10.1.9.10.1	secp256k1	223
10.1.9.10.2	secp256k1_const	223
10.1.9.10.3	secp256k1_keys_coincurve	223
10.1.9.10.4	secp256k1_keys_ecdsa	226
10.1.9.10.5	secp256k1_point_coincurve	229
10.1.9.10.6	secp256k1_point_ecdsa	232
10.1.9.11	sr25519	234
10.1.9.11.1	sr25519	234
10.1.9.11.2	sr25519_const	234
10.1.9.11.3	sr25519_keys	235
10.1.9.11.4	sr25519_point	238
10.1.10	electrum	238
10.1.10.1	electrum_v1	238
10.1.10.2	electrum_v2	240
10.1.10.3	mnemonic_v1	244
10.1.10.3.1	electrum_v1_entropy_generator	244
10.1.10.3.2	electrum_v1_mnemonic	245
10.1.10.3.3	electrum_v1_mnemonic_decoder	246
10.1.10.3.4	electrum_v1_mnemonic_encoder	247
10.1.10.3.5	electrum_v1_mnemonic_generator	247
10.1.10.3.6	electrum_v1_mnemonic_utils	248
10.1.10.3.7	electrum_v1_mnemonic_validator	249
10.1.10.3.8	electrum_v1_seed_generator	249
10.1.10.4	mnemonic_v2	250
10.1.10.4.1	electrum_v2_entropy_generator	250
10.1.10.4.2	electrum_v2_mnemonic	251
10.1.10.4.3	electrum_v2_mnemonic_decoder	252
10.1.10.4.4	electrum_v2_mnemonic_encoder	252
10.1.10.4.5	electrum_v2_mnemonic_generator	253
10.1.10.4.6	electrum_v2_mnemonic_utils	254
10.1.10.4.7	electrum_v2_mnemonic_validator	254
10.1.10.4.8	electrum_v2_seed_generator	254
10.1.11	monero	255
10.1.11.1	conf	255

10.1.11.1.1	monero_coin_conf	255
10.1.11.1.2	monero_coins	256
10.1.11.1.3	monero_conf	257
10.1.11.1.4	monero_conf_getter	257
10.1.11.2	mnemonic	258
10.1.11.2.1	monero_entropy_generator	258
10.1.11.2.2	monero_mnemonic	259
10.1.11.2.3	monero_mnemonic_decoder	260
10.1.11.2.4	monero_mnemonic_encoder	261
10.1.11.2.5	monero_mnemonic_generator	262
10.1.11.2.6	monero_mnemonic_utils	263
10.1.11.2.7	monero_mnemonic_validator	265
10.1.11.2.8	monero_seed_generator	265
10.1.11.3	monero	265
10.1.11.4	monero_ex	268
10.1.11.5	monero_keys	268
10.1.11.6	monero_subaddr	271
10.1.12	slip	272
10.1.12.1	slip173	272
10.1.12.1.1	slip173	272
10.1.12.2	slip32	273
10.1.12.2.1	slip32	273
10.1.12.2.2	slip32_key_net_ver	275
10.1.12.3	slip44	276
10.1.12.3.1	slip44	276
10.1.13	solana	278
10.1.13.1	spl_token	278
10.1.14	ss58	279
10.1.14.1	ss58	279
10.1.14.2	ss58_ex	280
10.1.15	substrate	280
10.1.15.1	conf	280
10.1.15.1.1	substrate_coin_conf	280
10.1.15.1.2	substrate_coins	281
10.1.15.1.3	substrate_conf	282
10.1.15.1.4	substrate_conf_getter	283
10.1.15.2	mnemonic	284
10.1.15.2.1	substrate_bip39_seed_generator	284
10.1.15.3	scale	284
10.1.15.3.1	substrate_scale_enc_base	284
10.1.15.3.2	substrate_scale_enc_bytes	285
10.1.15.3.3	substrate_scale_enc_cuint	285
10.1.15.3.4	substrate_scale_enc_uint	286
10.1.15.4	substrate	288
10.1.15.5	substrate_ex	291
10.1.15.6	substrate_keys	291
10.1.15.7	substrate_path	293
10.1.16	utils	296
10.1.16.1	conf	296
10.1.16.1.1	coin_names	296
10.1.16.2	crypto	297
10.1.16.2.1	aes_ecb	297
10.1.16.2.2	blake2	298
10.1.16.2.3	chacha20_poly1305	300

10.1.16.2.4	crc	301
10.1.16.2.5	hash160	302
10.1.16.2.6	hmac	303
10.1.16.2.7	pbkdf2	304
10.1.16.2.8	ripemd	305
10.1.16.2.9	scrypt	305
10.1.16.2.10	sha2	306
10.1.16.2.11	sha3	308
10.1.16.3	misc	309
10.1.16.3.1	algo	309
10.1.16.3.2	base32	310
10.1.16.3.3	bit	311
10.1.16.3.4	bytes	313
10.1.16.3.5	cbor_indefinite_len_array	315
10.1.16.3.6	data_bytes	316
10.1.16.3.7	integer	319
10.1.16.3.8	string	320
10.1.16.4	mnemonic	320
10.1.16.4.1	entropy_generator	320
10.1.16.4.2	mnemonic	321
10.1.16.4.3	mnemonic_decoder_base	322
10.1.16.4.4	mnemonic_encoder_base	323
10.1.16.4.5	mnemonic_ex	323
10.1.16.4.6	mnemonic_utils	323
10.1.16.4.7	mnemonic_validator	326
10.1.16.5	typing	327
10.1.16.5.1	literal	327
10.1.17	wif	327
10.1.17.1	wif	327

Python Module Index	329
----------------------------	------------

Index	333
--------------	------------

INTRODUCTION

This package allows generating mnemonics, seeds, private/public keys and addresses for different types of cryptocurrencies. In particular:

- Mnemonic and seed generation as defined by [BIP-0039](#)
- Private key encryption/decryption as defined by [BIP-0038](#)
- Keys derivation as defined by:
 - [BIP-0032](#)
 - [SLIP-0010](#)
 - [BIP32-Ed25519 \(Khovratovich/Law\)](#)
- Derivation of a hierarchy of keys as defined by:
 - [BIP-0044](#)
 - [BIP-0049](#) (Bitcoin Segwit)
 - [BIP-0084](#) (Bitcoin Native Segwit)
 - [BIP-0086](#) (Bitcoin Taproot)
 - [CIP-1852](#)
- Mnemonic and seed generation for [Substrate](#) (Polkadot/Kusama ecosystem)
- Keys derivation for [Substrate](#) (Polkadot/Kusama ecosystem, same of Polkadot-JS)
- Keys and addresses generation for Cardano (Byron-Legacy, Byron-Icarus and Shelley, same of Ledger and AdaLite/Yoroi wallets)
- Mnemonic and seed generation for Monero
- Keys and addresses/subaddresses generation for Monero (same of official Monero wallet)
- Mnemonic and seed generation for Algorand (Algorand 25-word mnemonic)
- Mnemonic and seed generation like Electrum wallet (v1 and v2)
- Keys derivation like Electrum wallet (v1 and v2)
- Generation of keys from a passphrase chosen by the user (“brainwallet”)

Other implemented functionalities:

- Parse BIP-0032 derivation paths
- Parse Substrate derivation paths
- Extended key serialization as defined by [SLIP-0032](#)

- Encode/Decode addresses for all the supported coins
- Encode/Decode [WIF](#)
- Encode/Decode [base58](#) and [base58 monero](#)
- Encode/Decode [ss58](#))
- Encode/Decode [bech32](#) and [bech32m](#)
- Encode/Decode [Bitcoin Cash bech32](#)
- Get token account addresses for SPL tokens (i.e. Solana tokens)

Package dependencies:

- [cbor2](#) for CBOR encoding/decoding
- [cremod](#) for CRC computation
- [pycryptodome](#) for cryptographic functions
- [coincurve](#) for secp256k1 curve
- [ecdsa](#) for nist256p1 and secp256k1 curves
- [ed25519-blake2b](#) for ed25519-blake2b curve
- [pynacl](#) for ed25519 curve
- [py-sr25519-bindings](#) for sr25519 curve

Please note that, for the [py-sr25519-bindings](#) library, Rust is required to be installed.

SUPPORTED COINS

Supported BIP coins:

- Akash Network
- Algorand
- Aptos
- Arbitrum
- Avalanche (all the 3 chains)
- Axelar
- Band Protocol
- Binance Chain
- Binance Smart Chain
- Bitcoin (and related test net)
- Bitcoin Cash (and related test net)
- Bitcoin Cash Simple Ledger Protocol (and related test net)
- BitcoinSV (and related test net)
- Cardano (Byron-Legacy, Byron-Icarus and Shelley)
- Celo
- Certik
- Cosmos
- Dash (and related test net)
- Dogecoin (and related test net)
- eCash (and related test net)
- Elrond (MultiversX)
- EOS
- Ergo (and related test net)
- Ethereum
- Ethereum Classic
- Fantom Opera

- Filecoin
- Fetch.ai
- Harmony One (Ethereum and Cosmos addresses)
- Huobi Heco Chain
- IRIS Network
- Kava
- Kusama (based on BIP44 and ed25519 SLIP-0010, like TrustWallet, it won't generate the same addresses of Polkadot-JS)
- Litecoin (and related test net)
- Metis
- Monero (based on BIP44 and secp256k1 or ed25519 SLIP-0010, it won't generate the same addresses of the official wallets, but it supports subaddresses generation)
- Nano
- Near Protocol
- NEO
- OKEx Chain (Ethereum and Cosmos addresses)
- Ontology
- Optimism
- Osmosis
- Pi Network
- Polkadot (based on BIP44 and ed25519 SLIP-0010, like TrustWallet, it won't generate the same addresses of Polkadot-JS)
- Polygon
- Ripple
- Secret Network
- Solana
- Stafi (Cosmos)
- Stellar
- Sui (only ed25519)
- Terra
- Tezos
- Theta Network
- Tron
- VeChain
- Verge
- Zcash (and related test net)
- Zilliqa

Supported Substrate coins:

- Acala
- Bifrost
- Chainx
- Edgeware
- Karura
- Kusama
- Moonbeam
- Moonriver
- Phala Network
- Plasm Network
- Sora
- Stafi
- Polkadot
- Generic Substrate coin

For what regards Monero, it's also possible to generate the same addresses of the official wallets without using BIP44 derivation.

Clearly, for those coins that support Smart Contracts (e.g. Ethereum, Tron, ...), the generated keys and addresses are valid for all the related tokens.

INSTALL THE PACKAGE

For the secp256k1 curve, it's possible to use either the *coincurve* or the *ecdsa* library. *coincurve* is much faster since it's a Python wrapper to the secp256k1 C library, while *ecdsa* is a pure Python implementation. By default *coincurve* will be used, but it's possible to disable it when installing.

To install the package:

- Default installation (*coincurve* will be used for secp256k1)

- Using *pip*, from this directory (local):

```
pip install .
```

- Using *pip*, from PyPI:

```
pip install bip_utils
```

- Alternative installation (*ecdsa* will be used for secp256k1)

- Using *setuptools*:

```
python setup.py install --coincurve=0
```

- Using *pip*, from this directory (local):

```
pip install . --install-option="--coincurve=0"
```

- Using *pip*, from PyPI:

```
pip install bip_utils --install-option="--coincurve=0"
```

NOTES:

- if you are using an Apple M1, please make sure to update *coincurve* to version 17.0.0
- in case of problems when building the *ed25519_blake2b* library, you can try one of the prebuilt wheels [here](#)

TEST AND COVERAGE

Install develop dependencies:

```
pip install -r requirements-dev.txt
```

To run tests:

```
python -m unittest discover
```

To run tests with coverage:

```
coverage run -m unittest discover  
coverage report
```

To run code analysis, just execute the `analyze_code` script.

MODULES DESCRIPTION

- BIP-0039
- Algorand mnemonic
- Electrum mnemonic
- Monero mnemonic
- BIP-0038
- BIP-0032
- BIP-0044
- Brainwallet
- Cardano
- Electrum
- Monero
- Substrate
- Utility libraries

DOCUMENTATION

The library documentation is available at bip-utils.readthedocs.io.

CODE EXAMPLES

For some complete code examples (from mnemonic to keys generation), refer to the [examples](#) folder.

BUY ME A COFFEE

You know, I'm italian and I love drinking coffee (especially while coding). So, if you'd like to buy me one:

- BTC: `bc1qq4r9cglwzd6f2hxxvdkucmdejvr9h8me5hy0k8`
- ERC20/BEP20: `0xf84e4898E5E10bf1fBe9ffA3EEC845e82e364b5B`

Thank you very much for your support.

LICENSE

This software is available under the MIT license.

MODULES

10.1 bip_utils

10.1.1 addr

10.1.1.1 P2PKH_addr

Module for P2PKH address encoding/decoding.

class `P2PKHPubKeyModes`(*value*)

Bases: `Enum`

Enumerative for P2PKH public key modes.

COMPRESSED = 1

UNCOMPRESSED = 2

class `P2PKHAddrDecoder`

Bases: `IAddrDecoder`

P2PKH address decoder class. It allows the Pay-to-Public-Key-Hash address decoding.

static `DecodeAddr`(*addr: str, **kwargs: Any*) → bytes

Decode a P2PKH address to bytes.

Parameters

- **addr** (*str*) – Address string
- **net_ver** (*bytes*) – Expected net address version
- **base58_alph** (`Base58Alphabets`, *optional*) – Base58 alphabet (default: Bitcoin alphabet)

Returns

Public key hash bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class P2PKHAddrEncoderBases: *IAddrEncoder*

P2PKH address encoder class. It allows the Pay-to-Public-Key-Hash address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey]*, ***kwargs: Any*) → str

Encode a public key to P2PKH address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- **net_ver** (*bytes*) – Net address version
- **base58_alph** (*Base58Alphabets*, *optional*) – Base58 alphabet, Bitcoin alphabet by default
- **pub_key_mode** (*P2PKHPubKeyModes*, *optional*) – Public key mode, compressed key by default

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not secp256k1

class BchP2PKHAddrDecoderBases: *IAddrDecoder*

Bitcoin Cash P2PKH address decoder class. It allows the Bitcoin Cash P2PKH decoding.

static DecodeAddr(*addr: str*, ***kwargs: Any*) → bytes

Decode a Bitcoin Cash P2PKH address to bytes.

Parameters

- **addr** (*str*) – Address string
- **hrp** (*str*) – Expected HRP
- **net_ver** (*bytes*) – Expected net address version

Returns

Public key hash bytes

Return type

bytes

Raises**ValueError** – If the address encoding is not valid**class BchP2PKHAddrEncoder**Bases: *IAddrEncoder*

Bitcoin Cash P2PKH address encoder class. It allows the Bitcoin Cash P2PKH encoding.

static EncodeKey(*pub_key*: Union[bytes, IPublicKey], ***kwargs*: Any) → str

Encode a public key to Bitcoin Cash P2PKH address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- **hrp** (*str*) – HRP
- **net_ver** (*bytes*) – Net address version

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not secp256k1

P2PKHAddr

alias of *P2PKHAddrEncoder*

BchP2PKHAddr

alias of *BchP2PKHAddrEncoder*

10.1.1.2 P2SH_addr

Module for P2SH address encoding/decoding.

class P2SHAddrConst

Bases: object

Class container for P2SH constants.

SCRIPT_BYTES: bytes = b'\x00\x14'

class P2SHAddrDecoder

Bases: *IAddrDecoder*

P2SH address decoder class. It allows the Pay-to-Script-Hash address decoding.

static DecodeAddr(*addr*: str, ***kwargs*: Any) → bytes

Decode a P2SH address to bytes.

Parameters

- **addr** (*str*) – Address string
- **net_ver** (*bytes*) – Expected net address version

Returns

Script signature hash bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class P2SHAddrEncoderBases: *IAddrEncoder*

P2SH address encoder class. It allows the Pay-to-Script-Hash address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to P2SH address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- **net_ver** (*bytes*) – Net address version

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not secp256k1

class BchP2SHAddrDecoderBases: *IAddrDecoder*

Bitcoin Cash P2SH address decoder class. It allows the Bitcoin Cash P2SH decoding.

static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode a Bitcoin Cash P2SH address to bytes.

Parameters

- **addr** (*str*) – Address string
- **hrp** (*str*) – Expected HRP
- **net_ver** (*bytes*) – Expected net address version

Returns

Script signature hash bytes

Return type

bytes

Raises**ValueError** – If the address encoding is not valid**class BchP2SHAddrEncoder**Bases: *IAddrEncoder*

Bitcoin Cash P2SH address encoder class. It allows the Bitcoin Cash P2SH encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Bitcoin Cash P2SH address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- **hrp** (*str*) – HRP
- **net_ver** (*bytes*) – Net address version

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not secp256k1

P2SHAddralias of *P2SHAddrEncoder***BchP2SHAddr**alias of *BchP2SHAddrEncoder***10.1.1.3 P2TR_addr**

Module for P2TR address encoding/decoding.

References

<https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki>
[bip-0341.mediawiki](https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki)

<https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki>

class P2TRConst

Bases: object

Class container for P2TR constants.

FIELD_SIZE: int =

115792089237316195423570985008687907853269984665640564039457584007908834671663

TAP_TWEAK_SHA256: bytes = b'\xe8\xf1c\x9c\x9c\xa0P\xe3\xaf\x1b9\xc1C\xc6>B\x9c\xbc\xeb\x15\xd9@\xfb\xb5\xc5\xa1\xf4\xafW\xc5\xe9'

WITNESS_VER: int = 1**class P2TRAddrDecoder**Bases: *IAddrDecoder*

P2WPKH address decoder class. It allows the Pay-to-Witness-Public-Key-Hash address decoding.

static DecodeAddr(addr: str, **kwargs: Any) → bytes

Decode a P2TR address to bytes.

Parameters

- **addr** (str) – Address string
- **hrp** (str) – Expected HRP

Returns

X coordinate of the tweaked public key

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class P2TRAddrEncoder

Bases: *IAddrEncoder*

P2TR address encoder class. It allows the Pay-to-Taproot address encoding.

static EncodeKey(*pub_key*: Union[bytes, *IPublicKey*], ***kwargs*: Any) → str

Encode a public key to P2TR address.

Parameters

- **pub_key** (bytes or *IPublicKey*) – Public key bytes or object
- **hrp** (str) – HRP

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid or cannot be tweaked
- **TypeError** – If the public key is not secp256k1

P2TRAddr

alias of *P2TRAddrEncoder*

10.1.1.4 P2WPKH_addr

Module for P2WPKH address encoding/decoding.

References

<https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki>
<https://github.com/bitcoin/bips/blob/master/bip-0173.mediawiki>

<https://github.com/bitcoin/bips/blob/master/>

class P2WPKHAddrConst

Bases: object

Class container for P2WPKH constants.

WITNESS_VER: int = 0

class P2WPKHAddrDecoder

Bases: *IAddrDecoder*

P2WPKH address decoder class. It allows the Pay-to-Witness-Public-Key-Hash address decoding.

static DecodeAddr(*addr*: str, ***kwargs*: Any) → bytes

Decode a P2WPKH address to bytes.

Parameters

- **addr** (str) – Address string
- **hrp** (str) – Expected HRP

Returns

Public key hash bytes

Return type

bytes

Raises**ValueError** – If the address encoding is not valid**class P2WPKHAddrEncoder**Bases: *IAddrEncoder*

P2WPKH address encoder class. It allows the Pay-to-Witness-Public-Key-Hash address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to P2WPKH address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- **hrp** (*str*) – HRP

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not secp256k1

P2WPKHAddralias of *P2WPKHAddrEncoder***10.1.1.5 ada_byron_addr**

Module for Cardano Byron address encoding/decoding. Both legacy and Icarus addresses are supported.

References

<https://cips.cardano.org/cips/cip19> <https://raw.githubusercontent.com/cardano-foundation/CIPs/master/CIP-0019/CIP-0019-byron-addresses.cddl>

class AdaByronAddrTypes(*value*)Bases: *IntEnum*

Enumerative for Cardano Byron address types.

PUBLIC_KEY = 0**REDEMPTION** = 2**class AdaByronAddrConst**Bases: *object*

Class container for Cardano Byron address constants.

```
CHACHA20_POLY1305_ASSOC_DATA: bytes = b''
```

```
CHACHA20_POLY1305_NONCE: bytes = b'serokellfore'
```

```
PAYLOAD_TAG: int = 24
```

```
class AdaByronAddrDecoder
```

```
    Bases: IAddrDecoder
```

Cardano Byron address decoder class. It allows the Cardano Byron address decoding.

```
    static DecryptHdPath(hd_path_enc_bytes: bytes, hd_path_key_bytes: bytes) → Bip32Path
```

Decrypt an HD path using the specified key.

Parameters

- **hd_path_enc_bytes** (*bytes*) – Encrypted HD path bytes
- **hd_path_key_bytes** (*bytes*) – HD path key bytes

Returns

Bip32Path object

Return type

Bip32Path object

Raises

ValueError – If the decryption fails

```
    static SplitDecodedBytes(dec_bytes: bytes) → Tuple[bytes, bytes]
```

Split the decoded bytes into address root hash and encrypted HD path.

Parameters

dec_bytes (*bytes*) – Decoded bytes

Returns

Address root hash (index 0), encrypted HD path (index 1)

Return type

tuple[bytes, bytes]

```
    static DecodeAddr(addr: str, **kwargs: Any) → bytes
```

Decode a Cardano Byron address (either legacy or Icarus) to bytes. The result can be split with `SplitDecodedBytes` if needed, to get the address root hash and encrypted HD path separately.

Parameters

- **addr** (*str*) – Address string
- **addr_type** (*AdaByronAddrTypes*) – Expected address type (default: public key)

Returns

Address root hash bytes (first 28-byte) and encrypted HD path (following bytes, if present)

Return type

bytes

Raises

- **ValueError** – If the address encoding is not valid
- **TypeError** – If the address type is not a `AdaByronAddrTypes` enum

class AdaByronIcarusAddrEncoderBases: *IAddrEncoder*

Cardano Byron Icarus address encoder class. It allows the Cardano Byron Icarus address encoding (i.e. without the encrypted derivation path, format Ae2...).

static EncodeKey(*pub_key*: Union[bytes, *IPublicKey*], ***kwargs*: Any) → str

Encode a public key to Cardano Byron address.

Parameters

- **pub_key** (bytes or *IPublicKey*) – Public key bytes or object
- **chain_code** (bytes or *Bip32ChainCode* object) – Chain code bytes or object

Returns

Address string

Return type

str

Raises

- *Bip32PathError* – If the path indexes are not valid
- *ValueError* – If the public key, the chain code or the HD path key is not valid
- *TypeError* – If the public key is not ed25519

class AdaByronLegacyAddrEncoderBases: *IAddrEncoder*

Cardano Byron legacy address encoder class. It allows the Cardano Byron legacy address encoding (i.e. containing the encrypted derivation path, format Ddz...).

static EncodeKey(*pub_key*: Union[bytes, *IPublicKey*], ***kwargs*: Any) → str

Encode a public key to Cardano Byron address.

Parameters

- **pub_key** (bytes or *IPublicKey*) – Public key bytes or object
- **chain_code** (bytes or *Bip32ChainCode* object) – Chain code bytes or object
- **hd_path** (str or *Bip32Path* object) – HD path
- **hd_path_key** (bytes) – HD path key bytes, shall be 32-byte long

Returns

Address string

Return type

str

Raises

- *Bip32PathError* – If the path indexes are not valid
- *ValueError* – If the public key, the chain code or the HD path key is not valid
- *TypeError* – If the public key is not ed25519

AdaByronIcarusAddr

alias of *AdaByronIcarusAddrEncoder*

AdaByronLegacyAddr

alias of *AdaByronLegacyAddrEncoder*

10.1.1.6 ada_shelley_addr

Module for Cardano Shelley address encoding/decoding. Reference: <https://cips.cardano.org/cips/cip19>

class AdaShelleyAddrNetworkTags(*value*)

Bases: `IntEnum`

Enumerative for Cardano Shelley network tags.

TESTNET = 0

MAINNET = 1

class AdaShelleyAddrHeaderTypes(*value*)

Bases: `IntEnum`

Enumerative for Cardano Shelley header types.

PAYMENT = 0

REWARD = 14

class AdaShelleyAddrConst

Bases: `object`

Class container for Cardano Shelley address constants.

```
NETWORK_TAG_TO_ADDR_HRP: Dict[AdaShelleyAddrNetworkTags, str] =  
{AdaShelleyAddrNetworkTags.TESTNET: 'addr_test', AdaShelleyAddrNetworkTags.MAINNET:  
'addr'}
```

```
NETWORK_TAG_TO_REWARD_ADDR_HRP: Dict[AdaShelleyAddrNetworkTags, str] =  
{AdaShelleyAddrNetworkTags.TESTNET: 'stake_test', AdaShelleyAddrNetworkTags.MAINNET:  
'stake'}
```

class AdaShelleyAddrDecoder

Bases: *IAddrDecoder*

Cardano Shelley address decoder class. It allows the Cardano Shelley address decoding.

static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode a Cardano Shelley address to bytes.

Parameters

- **addr** (*str*) – Address string
- ****kwargs** – Not used
- **net_tag** (*AdaShelleyAddrNetworkTags*) – Expected network tag (default: main net)

Returns

Public keys hash bytes (public key + public staking key)

Return type

bytes

Raises

- **ValueError** – If the address encoding is not valid
- **TypeError** – If the network tag is not a `AdaShelleyAddrNetworkTags` enum

class `AdaShelleyAddrEncoder`

Bases: `IAddrEncoder`

Cardano Shelley address encoder class. It allows the Cardano Shelley address encoding.

static `EncodeKey(pub_key: Union[bytes, IPublicKey], **kwargs: Any) → str`

Encode a public key to Cardano Shelley address.

Parameters

- **pub_key** (`bytes` or `IPublicKey`) – Public key bytes or object
- **pub_skey** (`bytes` or `IPublicKey`) – Public staking key bytes or object
- **net_tag** (`AdaShelleyAddrNetworkTags`) – Network tag (default: main net)

Returns

Address string

Return type

`str`

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not `ed25519` or the network tag is not a `AdaShelleyAddrNetworkTags` enum

class `AdaShelleyStakingAddrDecoder`

Bases: `IAddrDecoder`

Cardano Shelley staking address decoder class. It allows the Cardano Shelley staking address decoding.

static `DecodeAddr(addr: str, **kwargs: Any) → bytes`

Decode a Cardano Shelley address to bytes.

Parameters

- **addr** (`str`) – Address string
- ****kwargs** – Not used
- **net_tag** (`AdaShelleyAddrNetworkTags`) – Network tag (default: main net)

Returns

Public keys hash bytes (public key + public staking key)

Return type

`bytes`

Raises

- **ValueError** – If the address encoding is not valid
- **TypeError** – If the network tag is not a `AdaShelleyAddrNetworkTags` enum

class `AdaShelleyStakingAddrEncoder`

Bases: `IAddrEncoder`

Cardano Shelley staking address encoder class. It allows the Cardano Shelley staking address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Cardano Shelley staking address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- **net_tag** (*AdaShelleyAddrNetworkTags*) – Network tag (default: main net)

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not ed25519 or the network tag is not a *AdaShelleyAddrNetworkTags* enum

AdaShelleyAddr

alias of *AdaShelleyAddrEncoder*

AdaShelleyStakingAddr

alias of *AdaShelleyStakingAddrEncoder*

AdaShelleyRewardAddrDecoder

alias of *AdaShelleyStakingAddrDecoder*

AdaShelleyRewardAddrEncoder

alias of *AdaShelleyStakingAddrEncoder*

AdaShelleyRewardAddr

alias of *AdaShelleyStakingAddrEncoder*

10.1.1.7 addr_dec_utils

Module with utility functions for address decoding.

class AddrDecUtils

Bases: object

Class container for address decoding utility functions.

static ValidateAndRemovePrefix(*addr: BytesOrStr, prefix: BytesOrStr*) → BytesOrStr

Validate and remove prefix from an address.

Parameters

- **addr** (*bytes* or *str*) – Address string or bytes
- **prefix** (*bytes* or *str*) – Address prefix

Returns

Address string or bytes with prefix removed

Return type

bytes or str

Raises**ValueError** – If the prefix is not valid**static ValidateLength**(*addr: Union[bytes, str], len_exp: int*) → None

Validate address length.

Parameters

- **addr** (*str*) – Address string or bytes
- **len_exp** (*int*) – Expected address length

Raises**ValueError** – If the length is not valid**static ValidatePubKey**(*pub_key_bytes: bytes, pub_key_cls: Type[IPublicKey]*) → None

Validate address length.

Parameters

- **pub_key_bytes** (*bytes*) – Public key bytes
- **pub_key_cls** (*IPublicKey*) – Public key class type

Raises**ValueError** – If the public key is not valid**static ValidateChecksum**(*payload_bytes: bytes, checksum_bytes_exp: bytes, checksum_fct: Callable[[bytes], bytes]*) → None

Validate address checksum.

Parameters

- **payload_bytes** (*bytes*) – Payload bytes
- **checksum_bytes_exp** (*bytes*) – Expected checksum bytes
- **checksum_fct** (*function*) – Function for computing checksum

Raises**ValueError** – If the computed checksum is not equal to the specified one**static SplitPartsByChecksum**(*addr_bytes: bytes, checksum_len: int*) → Tuple[bytes, bytes]

Split address in two parts, considering the checksum at the end of it.

Parameters

- **addr_bytes** (*bytes*) – Address bytes
- **checksum_len** (*int*) – Checksum length

Returns

Payload bytes (index 0) and checksum bytes (index 1)

Return type

tuple[bytes, bytes]

10.1.1.8 addr_key_validator

Module with utility functions for validating address public keys.

class AddrKeyValidator

Bases: object

Class container for address utility functions.

static ValidateAndGetEd25519Key(*pub_key: Union[bytes, IPublicKey]*) → *IPublicKey*

Validate and get a ed25519 public key.

Parameters

pub_key (*bytes or IPublicKey object*) – Public key bytes or object

Returns

IPublicKey object

Return type

IPublicKey object

Raises

- **TypeError** – If the public key is not ed25519
- **ValueError** – If the public key is not valid

static ValidateAndGetEd25519Blake2bKey(*pub_key: Union[bytes, IPublicKey]*) → *IPublicKey*

Validate and get a ed25519-blake2b public key.

Parameters

pub_key (*bytes or IPublicKey object*) – Public key bytes or object

Returns

IPublicKey object

Return type

IPublicKey object

Raises

- **TypeError** – If the public key is not ed25519-blake2b
- **ValueError** – If the public key is not valid

static ValidateAndGetEd25519MoneroKey(*pub_key: Union[bytes, IPublicKey]*) → *IPublicKey*

Validate and get a ed25519-monero public key.

Parameters

pub_key (*bytes or IPublicKey object*) – Public key bytes or object

Returns

IPublicKey object

Return type

IPublicKey object

Raises

- **TypeError** – If the public key is not ed25519-monero
- **ValueError** – If the public key is not valid

static ValidateAndGetNist256p1Key(*pub_key: Union[bytes, IPublicKey]*) → *IPublicKey*

Validate and get a nist256p1 public key.

Parameters

pub_key (*bytes or IPublicKey object*) – Public key bytes or object

Returns

IPublicKey object

Return type

IPublicKey object

Raises

- **TypeError** – If the public key is not nist256p1
- **ValueError** – If the public key is not valid

static ValidateAndGetSecp256k1Key(*pub_key: Union[bytes, IPublicKey]*) → *IPublicKey*

Validate and get a secp256k1 public key.

Parameters

pub_key (*bytes or IPublicKey object*) – Public key bytes or object

Returns

IPublicKey object

Return type

IPublicKey object

Raises

- **TypeError** – If the public key is not secp256k1
- **ValueError** – If the public key is not valid

static ValidateAndGetSr25519Key(*pub_key: Union[bytes, IPublicKey]*) → *IPublicKey*

Validate and get a sr25519 public key.

Parameters

pub_key (*bytes or IPublicKey object*) – Public key bytes or object

Returns

IPublicKey object

Return type

IPublicKey object

Raises

- **TypeError** – If the public key is not sr25519
- **ValueError** – If the public key is not valid

10.1.1.9 algo_addr

Module for Algorand address encoding/decoding.

class AlgoAddrConst

Bases: object

Class container for Algorand address constants.

CHECKSUM_BYTE_LEN: int = 4

class AlgoAddrDecoder

Bases: *IAddrDecoder*

Algorand address decoder class. It allows the Algorand address decoding.

static DecodeAddr(addr: str, **kwargs: Any) → bytes

Decode an Algorand address to bytes.

Parameters

- **addr** (str) – Address string
- ****kwargs** – Not used

Returns

Public key bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class AlgoAddrEncoder

Bases: *IAddrEncoder*

Algorand address encoder class. It allows the Algorand address encoding.

static EncodeKey(pub_key: Union[bytes, *IPublicKey*], **kwargs: Any) → str

Encode a public key to Algorand address.

Parameters

- **pub_key** (bytes or *IPublicKey*) – Public key bytes or object
- ****kwargs** – Not used

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not ed25519

AlgoAddr

alias of *AlgoAddrEncoder*

10.1.1.10 aptos_addr

Module for Aptos address encoding/decoding.

class AptosAddrConst

Bases: object

Class container for Aptos address constants.

SINGLE_SIG_SUFFIX_BYTE: bytes = b'\x00'

class AptosAddrDecoder

Bases: [IAddrDecoder](#)

Aptos address decoder class. It allows the Aptos address decoding.

static DecodeAddr(addr: str, **kwargs: Any) → bytes

Decode an Aptos address to bytes.

Parameters

- **addr** (str) – Address string
- ****kwargs** – Not used

Returns

Public key bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class AptosAddrEncoder

Bases: [IAddrEncoder](#)

Aptos address encoder class. It allows the Aptos address encoding.

static EncodeKey(pub_key: Union[bytes, [IPublicKey](#)], **kwargs: Any) → str

Encode a public key to Aptos address.

Parameters

- **pub_key** (bytes or [IPublicKey](#)) – Public key bytes or object
- **trim_zeroes** (bool, optional) – True to trim left zeroes from the address string, false otherwise (default)

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not ed25519

AptosAddr

alias of [AptosAddrEncoder](#)

10.1.1.11 atom_addr

Module for Atom address encoding/decoding.

class AtomAddrDecoder

Bases: *IAddrDecoder*

Atom address decoder class. It allows the Atom address decoding.

static DecodeAddr(addr: str, **kwargs: Any) → bytes

Decode an Algorand address to bytes.

Parameters

- **addr** (str) – Address string
- **hrp** (str) – Expected HRP

Returns

Public key hash bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class AtomAddrEncoder

Bases: *IAddrEncoder*

Atom address encoder class. It allows the Atom address encoding.

static EncodeKey(pub_key: Union[bytes, *IPublicKey*], **kwargs: Any) → str

Encode a public key to Atom address.

Parameters

- **pub_key** (bytes or *IPublicKey*) – Public key bytes or object
- **hrp** (str) – HRP

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not secp256k1

AtomAddr

alias of *AtomAddrEncoder*

10.1.1.12 avax_addr

Module for Avax address encoding/decoding.

class AvaxPChainAddrDecoder

Bases: *IAddrDecoder*

Avax P-Chain address decoder class. It allows the Avax P-Chain address decoding.

static DecodeAddr(addr: str, **kwargs: Any) → bytes

Decode an Avax P-Chain address to bytes.

Parameters

- **addr** (str) – Address string
- ****kwargs** – Not used

Returns

Public key hash bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class AvaxPChainAddrEncoder

Bases: *IAddrEncoder*

Avax P-Chain address encoder class. It allows the Avax P-Chain address encoding.

static EncodeKey(pub_key: Union[bytes, *IPublicKey*], **kwargs: Any) → str

Encode a public key to Avax P-Chain address.

Parameters

- **pub_key** (bytes or *IPublicKey*) – Public key bytes or object
- ****kwargs** – Not used

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not secp256k1

class AvaxXChainAddrDecoder

Bases: *IAddrDecoder*

Avax X-Chain address decoder class. It allows the Avax X-Chain address decoding.

static DecodeAddr(addr: str, **kwargs: Any) → bytes

Decode an Avax X-Chain address to bytes.

Parameters

- **addr** (str) – Address string

- ****kwargs** – Not used

Returns

Public key hash bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class AvaxXChainAddrEncoder

Bases: *IAddrEncoder*

Avax X-Chain address encoder class. It allows the Avax X-Chain address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Avax X-Chain address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- ****kwargs** – Not used

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not secp256k1

AvaxPChainAddr

alias of *AvaxPChainAddrEncoder*

AvaxXChainAddr

alias of *AvaxXChainAddrEncoder*

10.1.1.13 bch_addr_converter

Module for converting Bitcoin Cash addresses.

class BchAddrConverter

Bases: object

Bitcoin Cash address converter class. It allows to convert a Bitcoin Cash address by changing its HRP and net version.

static Convert(*address: str, hrp: str, net_ver: Optional[bytes] = None*) → str

Convert a Bitcoin Cash address by changing its HRP and net version.

Parameters

- **address** (*str*) – Bitcoin Cash address
- **hrp** (*str*) – New HRP
- **net_ver** (*bytes, optional*) – New net version (if None, the old one will be used)

Returns

Converted address string

Return type

str

Raises

- ***Bech32ChecksumError*** – If the address checksum is not valid
- **ValueError** – If the address string is not valid

10.1.1.14 egld_addr

Module for Elrond address encoding/decoding.

class EgldAddrDecoder

Bases: *IAddrDecoder*

Elrond address decoder class. It allows the Elrond address decoding.

static DecodeAddr(addr: str, **kwargs: Any) → bytes

Decode an Elrond address to bytes.

Parameters

- **addr** (str) – Address string
- ****kwargs** – Not used

Returns

Public key bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class EgldAddrEncoder

Bases: *IAddrEncoder*

Elrond address encoder class. It allows the Elrond address encoding.

static EncodeKey(pub_key: Union[bytes, *IPublicKey*], **kwargs: Any) → str

Encode a public key to Elrond address.

Parameters

- **pub_key** (bytes or *IPublicKey*) – Public key bytes or object
- ****kwargs** – Not used

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not ed25519

EgldAddr

alias of *EgldAddrEncoder*

10.1.1.15 eos_addr

Module for EOS address encoding/decoding.

class EosAddrConst

Bases: `object`

Class container for EOS address constants.

CHECKSUM_BYTE_LEN: int = 4

class EosAddrDecoder

Bases: *IAddrDecoder*

EOS address decoder class. It allows the EOS address decoding.

static DecodeAddr(addr: str, **kwargs: Any) → bytes

Decode an EOS address to bytes.

Parameters

- **addr** (str) – Address string
- ****kwargs** – Not used

Returns

Public key bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class EosAddrEncoder

Bases: *IAddrEncoder*

EOS address encoder class. It allows the EOS address encoding.

static EncodeKey(pub_key: Union[bytes, *IPublicKey*], **kwargs: Any) → str

Encode a public key to EOS address.

Parameters

- **pub_key** (bytes or *IPublicKey*) – Public key bytes or object
- ****kwargs** – Not used

Returns

Address string

Return type

str

Raised:

ValueError: If the public key is not valid **TypeError:** If the public key is not secp256k1

EosAddr

alias of *EosAddrEncoder*

10.1.1.16 ergo_addr

Module for Ergo address encoding/decoding.

class ErgoAddressTypes(*value*)

Bases: `IntEnum`

Enumerative for Ergo address types.

P2PKH = 1

P2SH = 2

class ErgoNetworkTypes(*value*)

Bases: `IntEnum`

Enumerative for Ergo network types.

MAINNET = 0

TESTNET = 16

class ErgoAddrConst

Bases: `object`

Class container for Ergo address constants.

CHECKSUM_BYTE_LEN: `int` = 4

class ErgoP2PKHAddrDecoder

Bases: `IAddrDecoder`

Ergo P2PKH address decoder class. It allows the Ergo P2PKH address decoding.

static DecodeAddr(*addr*: `str`, ***kwargs*: `Any`) → `bytes`

Decode an Ergo P2PKH address to bytes.

Parameters

- **addr** (`str`) – Address string
- **net_type** (`ErgoNetworkTypes`) – Expected network type (default: main net)

Returns

Public key bytes

Return type

`bytes`

Raises

- **ValueError** – If the address encoding is not valid
- **TypeError** – If the network tag is not a `ErgoNetworkTypes` enum

class ErgoP2PKHAddrEncoder

Bases: `IAddrEncoder`

Ergo P2PKH address encoder class. It allows the Ergo P2PKH address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Ergo P2PKH address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- **net_type** (*ErgoNetworkTypes*) – Network type (default: main net)

Returns

Address string

Return type

str

Raised:

ValueError: If the public key is not valid **TypeError:** If the public key is not secp256k1 or the network tag is not a *ErgoNetworkTypes* enum

ErgoP2PKHAddr

alias of *ErgoP2PKHAddrEncoder*

10.1.1.17 eth_addr

Module for Ethereum address encoding/decoding.

class EthAddrConst

Bases: object

Class container for Ethereum address constants.

START_BYTE: int = 24

ADDR_LEN: int = 40

class EthAddrDecoder

Bases: *IAddrDecoder*

Ethereum address decoder class. It allows the Ethereum address decoding.

static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode an Ethereum address to bytes.

Parameters

- **addr** (*str*) – Address string
- **skip_chksum_enc** (*bool, optional*) – True to skip checksum encoding verification, false otherwise (default)

Returns

Public key hash bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class EthAddrEncoderBases: *IAddrEncoder*

Ethereum address encoder class. It allows the Ethereum address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Ethereum address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- **skip_chksm_enc** (*bool*, *optional*) – True to skip checksum encoding, false otherwise (default)

Returns

Address string

Return type

str

Raised:

ValueError: If the public key is not valid TypeError: If the public key is not secp256k1

EthAddralias of *EthAddrEncoder***10.1.1.18 fil_addr**

Module for Filecoin address encoding/decoding.

class FillAddrTypes(*value*)

Bases: IntEnum

Enumerative for Filecoin address types.

SECP256K1 = 1**BLS** = 3**class FilAddrConst**

Bases: object

Class container for Filecoin address constants.

BASE32_ALPHABET: str = 'abcdefghijklmnopqrstuvwxyz234567'**class FilSecp256k1AddrDecoder**Bases: *IAddrDecoder*

Filecoin address decoder class, based on secp256k1 curve. It allows the Filecoin address decoding.

static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode a Filecoin address to bytes.

Parameters

- **addr** (*str*) – Address string
- ****kwargs** – Not used

Returns

Public key hash bytes

Return type

bytes

Raises**ValueError** – If the address encoding is not valid**class FilSecp256k1AddrEncoder**Bases: *IAddrEncoder*

Filecoin address encoder class, based on secp256k1 curve. It allows the Filecoin address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Filecoin address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- ****kwargs** – Not used

Returns

Address string

Return type

str

Raised:

ValueError: If the public key is not valid TypeError: If the public key is not secp256k1 or the address type is not valid

FilSecp256k1Addralias of *FilSecp256k1AddrEncoder***10.1.1.19 iaddr_decoder**

Module with interface for address encoding classes.

class IAddrDecoder

Bases: ABC

Address decoder interface.

abstract static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode an address to bytes. Depending on the coin, the result can be a public key or a public key hash bytes.

Parameters

- **addr** (*str*) – Address string
- ****kwargs** – Arbitrary arguments depending on the address type

Returns

Public key bytes or public key hash

Return type

bytes

Raises

ValueError – If the address encoding is not valid

10.1.1.20 iaddr_encoder

Module with interface for address encoding classes.

class IAddrEncoder

Bases: ABC

Address encoder interface.

abstract static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode public key to address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- ****kwargs** – Arbitrary arguments depending on the address type

Returns

Address string

Return type

str

Raised:

ValueError: If the public key is not valid TypeError: If the public key is not of the correct type (it depends on the address type)

10.1.1.21 icx_addr

Module for Icon address encoding/decoding.

class IcxAddrConst

Bases: object

Class container for Icon address constants.

KEY_HASH_BYTE_LEN: int = 20

class IcxAddrDecoder

Bases: *IAddrDecoder*

Icon address decoder class. It allows the Icon address decoding.

static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode an Icon address to bytes.

Parameters

- **addr** (*str*) – Address string
- ****kwargs** – Not used

Returns

Public key hash bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class IcxAddrEncoder

Bases: *IAddrEncoder*

Icon address encoder class. It allows the Icon address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Icon address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- ****kwargs** – Not used

Returns

Address string

Return type

str

Raised:

ValueError: If the public key is not valid TypeError: If the public key is not secp256k1

IcxAddr

alias of *IcxAddrEncoder*

10.1.1.22 inj_addr

Module for Injective address encoding/decoding. Reference: <https://docs.injective.network/learn/basic-concepts/accounts>

class InjAddrDecoder

Bases: *IAddrDecoder*

Injective address decoder class. It allows the Injective address decoding.

static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode an Algorand address to bytes.

Parameters

addr (*str*) – Address string

Returns

Public key hash bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class InjAddrEncoder

Bases: *IAddrEncoder*

Injective address encoder class. It allows the Injective address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Injective address.

Parameters

pub_key (*bytes or IPublicKey*) – Public key bytes or object

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not secp256k1

InjAddr

alias of [InjAddrEncoder](#)

10.1.1.23 nano_addr

Module for Nano address encoding/decoding.

class NanoAddrConst

Bases: object

Class container for Nano address constants.

BASE32_ALPHABET: str = '13456789abcdefghijklmnopqrstuvwxyz'

PAYLOAD_PAD_DEC: bytes = b'\x00\x00\x00'

PAYLOAD_PAD_ENC: str = '1111'

class NanoAddrDecoder

Bases: [IAddrDecoder](#)

Nano address decoder class. It allows the Nano address decoding.

static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode a Nano address to bytes.

Parameters

- **addr** (*str*) – Address string
- ****kwargs** – Not used

Returns

Public key bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class NanoAddrEncoderBases: *IAddrEncoder*

Nano address encoder class. It allows the Nano address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Nano address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- ****kwargs** – Not used

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not ed25519-blake2b

NanoAddralias of *NanoAddrEncoder***10.1.1.24 near_addr**

Module for Near Protocol address encoding/decoding.

class NearAddrDecoderBases: *IAddrDecoder*

Near address decoder class. It allows the Near Protocol address decoding.

static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode a Near Protocol address to bytes.

Parameters

- **addr** (*str*) – Address string
- ****kwargs** – Not used

Returns

Public key bytes

Return type

bytes

Raises**ValueError** – If the address encoding is not valid**class NearAddrEncoder**Bases: *IAddrEncoder*

Near address encoder class. It allows the Near Protocol address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Near Protocol address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- ****kwargs** – Not used

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not ed25519

NearAddr

alias of *NearAddrEncoder*

10.1.1.25 neo_addr

Module for Neo address encoding/decoding.

class NeoAddrConst

Bases: object

Class container for NEO address constants.

PREFIX_BYTE: bytes = b'!'

SUFFIX_BYTE: bytes = b'\xac'

class NeoAddrDecoder

Bases: *IAddrDecoder*

Neo address decoder class. It allows the Neo address decoding.

static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode a Neo address to bytes.

Parameters

- **addr** (*str*) – Address string
- **ver** (*bytes*) – Expected version

Returns

Public key hash bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class NeoAddrEncoderBases: *IAddrEncoder*

Neo address encoder class. It allows the Neo address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Neo address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- **ver** (*bytes*) – Version

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not nist256p1

NeoAddralias of *NeoAddrEncoder***10.1.1.26 okex_addr**

Module for OKEx address encoding/decoding.

class OkexAddrDecoderBases: *IAddrDecoder*

OKEx Chain address decoder class. It allows the OKEx Chain address decoding.

static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode a OKEx Chain address to bytes.

Parameters

- **addr** (*str*) – Address string
- ****kwargs** – Not used

Returns

Public key hash bytes

Return type

bytes

Raises**ValueError** – If the address encoding is not valid**class OkexAddrEncoder**Bases: *IAddrEncoder*

OKEx Chain address encoder class. It allows the OKEx Chain address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to OKEx Chain address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- ****kwargs** – Not used

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not secp256k1

OkexAddr

alias of *OkexAddrEncoder*

10.1.1.27 one_addr

Module for Harmony One address encoding/decoding.

class OneAddrDecoder

Bases: *IAddrDecoder*

Harmony One address decoder class. It allows the Harmony One address decoding.

static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode a OKEx Chain address to bytes.

Parameters

- **addr** (*str*) – Address string
- ****kwargs** – Not used

Returns

Public key hash bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class OneAddrEncoder

Bases: *IAddrEncoder*

Harmony One address encoder class. It allows the Harmony One address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Harmony One address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- ****kwargs** – Not used

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not secp256k1

OneAddralias of *OneAddrEncoder***10.1.1.28 sol_addr**

Module for Solana address encoding/decoding.

class SolAddrDecoderBases: *IAddrDecoder*

Solana address decoder class. It allows the Solana address decoding.

static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode a Solana address to bytes.

Parameters

- **addr** (*str*) – Address string
- ****kwargs** – Not used

Returns

Public key bytes

Return type

bytes

Raises**ValueError** – If the address encoding is not valid**class SolAddrEncoder**Bases: *IAddrEncoder*

Solana address encoder class. It allows the Solana address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Solana address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- ****kwargs** – Not used

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not ed25519

SolAddr

alias of *SolAddrEncoder*

10.1.1.29 substrate_addr

Module for Substrate address encoding/decoding.

class SubstrateEd25519AddrDecoder

Bases: *IAddrDecoder*

Substrate address decoder class, based on ed25519 curve. It allows the Substrate address decoding.

static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode a Substrate address to bytes.

Parameters

- **addr** (*str*) – Address string
- **ss58_format** (*int*) – Expected SS58 format

Returns

Public key bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class SubstrateEd25519AddrEncoder

Bases: *IAddrEncoder*

Substrate address encoder class, based on ed25519 curve. It allows the Substrate address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Substrate address.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- **ss58_format** (*int*) – SS58 format

Returns

Address string

Return type

str

Raised:

ValueError: If the public key is not valid

class SubstrateSr25519AddrDecoder

Bases: *IAddrDecoder*

Substrate address decoder class, based on sr25519 curve. It allows the Substrate address decoding.

static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode a Substrate address to bytes.

Parameters

- **addr** (*str*) – Address string
- **ss58_format** (*int*) – Expected SS58 format

Returns

Public key bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class SubstrateSr25519AddrEncoder

Bases: [*IAddrEncoder*](#)

Substrate address encoder class, based on sr25519 curve. It allows the Substrate address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Substrate address.

Parameters

- **pub_key** (*bytes or IPublicKey*) – Public key bytes or object
- **ss58_format** (*int*) – SS58 format

Returns

Address string

Return type

str

Raised:

ValueError: If the public key is not valid

SubstrateEd25519Addr

alias of [*SubstrateEd25519AddrEncoder*](#)

SubstrateSr25519Addr

alias of [*SubstrateSr25519AddrEncoder*](#)

10.1.1.30 sui_addr

Module for Solana address encoding/decoding.

class SuiAddrConst

Bases: object

Class container for Sui address constants.

KEY_TYPE: bytes = b'\x00'

class SuiAddrDecoderBases: *IAddrDecoder*

Sui address decoder class. It allows the Sui address decoding.

static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode a Sui address to bytes.

Parameters

- **addr** (*str*) – Address string
- ****kwargs** – Not used

Returns

Public key hash bytes

Return type

bytes

Raises**ValueError** – If the address encoding is not valid**class SuiAddrEncoder**Bases: *IAddrEncoder*

Sui address encoder class. It allows the Sui address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Sui address.

Parameters

- **pub_key** (*bytes or IPublicKey*) – Public key bytes or object
- ****kwargs** – Not used

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not ed25519

SuiAddralias of *SuiAddrEncoder***10.1.1.31 trx_addr**

Module for Tron address encoding/decoding.

class TrxAddrDecoderBases: *IAddrDecoder*

Tron address decoder class. It allows the Tron address decoding.

static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode a Tron address to bytes.

Parameters

- **addr** (*str*) – Address string
- ****kwargs** – Not used

Returns

Public key hash bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class TrxAddrEncoder

Bases: [*IAddrEncoder*](#)

Tron address encoder class. It allows the Tron address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Tron address.

Parameters

- **pub_key** (*bytes or IPublicKey*) – Public key bytes or object
- ****kwargs** – Not used

Returns

Address string

Return type

str

Raised:

ValueError: If the public key is not valid TypeError: If the public key is not secp256k1

TrxAddr

alias of [*TrxAddrEncoder*](#)

10.1.1.32 xlm_addr

Module for Stellar address encoding/decoding.

class XlmAddrTypes(*value*)

Bases: IntEnum

Enumerative for Stellar address types.

PUB_KEY = 48

PRIV_KEY = 144

class XlmAddrConst

Bases: object

Class container for Stellar address constants.

```
CHECKSUM_BYTE_LEN: int = 2
```

```
class XlmAddrDecoder
```

Bases: [IAddrDecoder](#)

Stellar address decoder class. It allows the Stellar address decoding.

```
static DecodeAddr(addr: str, **kwargs: Any) → bytes
```

Decode a Stellar address to bytes.

Parameters

- **addr** (*str*) – Address string
- **addr_type** ([XlmAddrTypes](#)) – Expected address type (default: public key)

Returns

Public key bytes

Return type

bytes

Raises

- **ValueError** – If the address encoding is not valid
- **TypeError** – If the address type is not a [XlmAddrTypes](#) enum

```
class XlmAddrEncoder
```

Bases: [IAddrEncoder](#)

Stellar address encoder class. It allows the Stellar address encoding.

```
static EncodeKey(pub_key: Union[bytes, IPublicKey], **kwargs: Any) → str
```

Encode a public key to Stellar address.

Parameters

- **pub_key** (*bytes* or [IPublicKey](#)) – Public key bytes or object
- **addr_type** ([XlmAddrTypes](#)) – Address type (default: public key)

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not ed25519 or address type is not a [XlmAddrTypes](#) enum

```
XlmAddr
```

alias of [XlmAddrEncoder](#)

10.1.1.33 xmr_addr

Module for Monero address encoding/decoding.

class **XmrAddrConst**

Bases: `object`

Class container for Monero address constants.

CHECKSUM_BYTE_LEN: `int` = 4

PAYMENT_ID_BYTE_LEN: `int` = 8

class **XmrAddrDecoder**

Bases: `IAddrDecoder`

Monero address decoder class. It allows the Monero address decoding.

static **DecodeAddr**(*addr*: `str`, ***kwargs*: `Any`) → `bytes`

Decode a Monero address to bytes.

Parameters

- **addr** (`str`) – Address string
- **net_ver** (`bytes`) – Expected net version

Returns

Public spend (first) and view (second) keys joined together

Return type

`bytes`

Raises

ValueError – If the address encoding is not valid

class **XmrAddrEncoder**

Bases: `IAddrEncoder`

Monero address encoder class. It allows the Monero address encoding.

static **EncodeKey**(*pub_key*: `Union[bytes, IPublicKey]`, ***kwargs*: `Any`) → `str`

Encode a public key to Monero format.

Parameters

- **pub_key** (`bytes` or `IPublicKey`) – Public spend key bytes or object
- **pub_vkey** (`bytes` or `IPublicKey`) – Public view key bytes or object
- **net_ver** (`bytes`) – Net version
- **payment_id** (`bytes`, *optional*) – Payment ID (only for integrated addresses)

Returns

Address string

Return type

`str`

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not ed25519-monero

class XmrIntegratedAddrDecoderBases: *IAddrDecoder*

Monero integrated address decoder class. It allows the Monero integrated address decoding.

static DecodeAddr(addr: str, **kwargs: Any) → bytes

Decode a Monero address to bytes.

Parameters

- **addr** (str) – Address string
- **net_ver** (bytes) – Expected net version
- **payment_id** (bytes) – Expected payment ID

Returns

Public spend (first) and view (second) keys joined together

Return type

bytes

Raises**ValueError** – If the address encoding is not valid**class XmrIntegratedAddrEncoder**Bases: *IAddrEncoder*

Monero integrated address encoder class. It allows the Monero integrated address encoding.

static EncodeKey(pub_key: Union[bytes, *IPublicKey*], **kwargs: Any) → str

Encode a public key to Monero integrated address.

Parameters

- **pub_key** (bytes or *IPublicKey*) – Public spend key bytes or object
- **pub_vkey** (bytes or *IPublicKey*) – Public view key bytes or object
- **net_ver** (bytes) – Net version
- **payment_id** (bytes) – Payment ID

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not ed25519-monero

XmrAddralias of *XmrAddrEncoder***XmrIntegratedAddr**alias of *XmrIntegratedAddrEncoder*

10.1.1.34 xrp_addr

Module for Ripple address encoding/decoding.

class XrpAddrDecoder

Bases: *IAddrDecoder*

Ripple address decoder class. It allows the Ripple address decoding.

static DecodeAddr(addr: str, **kwargs: Any) → bytes

Decode a Ripple address to bytes.

Parameters

- **addr** (str) – Address string
- ****kwargs** – Not used

Returns

Public key hash bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class XrpAddrEncoder

Bases: *IAddrEncoder*

Ripple address encoder class. It allows the Ripple address encoding.

static EncodeKey(pub_key: Union[bytes, *IPublicKey*], **kwargs: Any) → str

Encode a public key to Ripple address.

Parameters

- **pub_key** (bytes or *IPublicKey*) – Public key bytes or object
- ****kwargs** – Not used

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not secp256k1

XrpAddr

alias of *XrpAddrEncoder*

10.1.1.35 xtz_addr

Module for Tezos address encoding/decoding.

class XtzAddrPrefixes(*value*)

Bases: Enum

Enumerative for Tezos address prefixes.

TZ1 = b'\x06\xa1\x9f'

TZ2 = b'\x06\xa1\xa1'

TZ3 = b'\x06\xa1\xa4'

class XtzAddrDecoder

Bases: *IAddrDecoder*

Tezos address decoder class. It allows the Tezos address decoding.

static DecodeAddr(*addr: str, **kwargs: Any*) → bytes

Decode a Tezos address to bytes.

Parameters

- **addr** (*str*) – Address string
- **prefix** (*XtzAddrPrefixes*) – Expected address prefix

Returns

Public key hash bytes

Return type

bytes

Raises

- **ValueError** – If the address encoding is not valid
- **TypeError** – If the prefix is not a XtzAddrPrefixes enum

class XtzAddrEncoder

Bases: *IAddrEncoder*

Tezos address encoder class. It allows the Tezos address encoding.

static EncodeKey(*pub_key: Union[bytes, IPublicKey], **kwargs: Any*) → str

Encode a public key to Tezos address.

Parameters

- **pub_key** (*bytes or IPublicKey*) – Public key bytes or object
- **prefix** (*XtzAddrPrefixes*) – Address prefix

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not ed25519 or the prefix is not a XtzAddrPrefixes enum

XtzAddr

alias of *XtzAddrEncoder*

10.1.1.36 zil_addr

Module for Zilliqa address encoding/decoding.

class ZilAddrConst

Bases: object

Class container for Zilliqa address constants.

SHA256_BYTE_LEN: int = 20

class ZilAddrDecoder

Bases: *IAddrDecoder*

Zilliqa address decoder class. It allows the Zilliqa address decoding.

static DecodeAddr(addr: str, **kwargs: Any) → bytes

Decode a Zilliqa address to bytes.

Parameters

- **addr** (str) – Address string
- ****kwargs** – Not used

Returns

Public key hash bytes

Return type

bytes

Raises

ValueError – If the address encoding is not valid

class ZilAddrEncoder

Bases: *IAddrEncoder*

Zilliqa address encoder class. It allows the Zilliqa address encoding.

static EncodeKey(pub_key: Union[bytes, *IPublicKey*], **kwargs: Any) → str

Encode a public key to Zilliqa address.

Parameters

- **pub_key** (bytes or *IPublicKey*) – Public key bytes or object
- ****kwargs** – Not used

Returns

Address string

Return type

str

Raises

- **ValueError** – If the public key is not valid
- **TypeError** – If the public key is not secp256k1

ZilAddr

alias of *ZilAddrEncoder*

10.1.2 algorand**10.1.2.1 mnemonic****10.1.2.1.1 algorand_entropy_generator**

Module for Algorand mnemonic entropy generation.

class AlgorandEntropyBitLen(*value*)

Bases: `IntEnum`

Enumerative for Algorand entropy bit lengths.

BIT_LEN_256 = 256

class AlgorandEntropyGeneratorConst

Bases: `object`

Class container for Algorand entropy generator constants.

ENTROPY_BIT_LEN: `List[AlgorandEntropyBitLen]` = [`<AlgorandEntropyBitLen.BIT_LEN_256: 256>`]

class AlgorandEntropyGenerator(*bit_len*: `Union[int, AlgorandEntropyBitLen]` = `AlgorandEntropyBitLen.BIT_LEN_256`)

Bases: *EntropyGenerator*

Algorand entropy generator class. It generates random entropy bytes.

static IsValidEntropyBitLen(*bit_len*: `int`) → `bool`

Get if the specified entropy bit length is valid.

Parameters

bit_len (`int`) – Entropy length in bits

Returns

True if valid, false otherwise

Return type

`bool`

static IsValidEntropyByteLen(*byte_len*: `int`) → `bool`

Get if the specified entropy byte length is valid.

Parameters

byte_len (`int`) – Entropy length in bytes

Returns

True if valid, false otherwise

Return type

`bool`

m_bit_len: `int`

10.1.2.1.2 algorand_mnemonic

Module for Algorand mnemonic.

```
class AlgorandWordsNum(value)
    Bases: IntEnum
    Enumerative for Algorand words number.
    WORDS_NUM_25 = 25

class AlgorandLanguages(value)
    Bases: MnemonicLanguages
    Enumerative for Algorand languages.
    ENGLISH = Bip39Languages.ENGLISH

class AlgorandMnemonicConst
    Bases: object
    Class container for Algorand mnemonic constants.
    MNEMONIC_WORD_NUM: List[AlgorandWordsNum] = [<AlgorandWordsNum.WORDS_NUM_25: 25>]

    CHECKSUM_BYTE_LEN: int = 2

class AlgorandMnemonic(mnemonic_list: List[str])
    Bases: Bip39Mnemonic
    Algorand mnemonic class.
    m_mnemonic_list: List[str]
```

10.1.2.1.3 algorand_mnemonic_decoder

Module for Algorand mnemonic decoding. Reference: <https://github.com/algorand/py-algorand-sdk>

```
class AlgorandMnemonicDecoder(lang: Optional[AlgorandLanguages] = AlgorandLanguages.ENGLISH)
    Bases: MnemonicDecoderBase
    Algorand mnemonic decoder class. It decodes a mnemonic phrase to bytes.

    Decode(mnemonic: Union[str, Mnemonic]) → bytes
        Decode a mnemonic phrase to bytes (no checksum).

    Parameters
        mnemonic (str or Mnemonic object) – Mnemonic

    Returns
        Decoded bytes

    Return type
        bytes

    Raises
        • MnemonicChecksumError – If checksum is not valid
        • ValueError – If mnemonic is not valid
```

```

m_lang: Optional[MnemonicLanguages]

m_words_list: Optional[MnemonicWordsList]

m_words_list_finder_cls: Type[MnemonicWordsListFinderBase]

```

10.1.2.1.4 algorand_mnemonic_encoder

Module for Algorand mnemonic encoding. Reference: <https://github.com/algorand/py-algorand-sdk>

class AlgorandMnemonicEncoder(*lang*: AlgorandLanguages = AlgorandLanguages.ENGLISH)

Bases: *MnemonicEncoderBase*

Algorand mnemonic encoder class. It encodes bytes to the mnemonic phrase.

Encode(*entropy_bytes*: bytes) → *Mnemonic*

Encode bytes to mnemonic phrase.

Parameters

entropy_bytes (bytes) – Entropy bytes

Returns

Encoded mnemonic

Return type

Mnemonic object

Raises

ValueError – If bytes length is not valid

m_words_list: *MnemonicWordsList*

10.1.2.1.5 algorand_mnemonic_generator

Module for Algorand mnemonic generation.

class AlgorandMnemonicGeneratorConst

Bases: object

Class container for Algorand mnemonic generator constants.

```

WORDS_NUM_TO_ENTROPY_LEN: Dict[AlgorandWordsNum, AlgorandEntropyBitLen] =
{AlgorandWordsNum.WORDS_NUM_25: AlgorandEntropyBitLen.BIT_LEN_256}

```

class AlgorandMnemonicGenerator(*lang*: AlgorandLanguages = AlgorandLanguages.ENGLISH)

Bases: object

Algorand mnemonic generator class. It generates 25-words mnemonic in according to Algorand wallets.

m_mnemonic_encoder: *AlgorandMnemonicEncoder*

FromWordsNumber(*words_num*: Union[int, AlgorandWordsNum]) → *Mnemonic*

Generate mnemonic with the specified words number from random entropy. There is no really need of this method, since the words number can only be 25, but it's kept to have the same usage of Bip39/Monero mnemonic generator.

Parameters

words_num (int or AlgorandWordsNum) – Number of words (25)

Returns

Generated mnemonic

Return type

Mnemonic object

Raises

ValueError – If words number is not valid

FromEntropy(*entropy_bytes: bytes*) → *Mnemonic*

Generate mnemonic from the specified entropy bytes.

Parameters

entropy_bytes (*bytes*) – Entropy bytes

Returns

Generated mnemonic

Return type

Mnemonic object

Raises

ValueError – If entropy byte length is not valid

10.1.2.1.6 algorand_mnemonic_utils

Module for Algorand mnemonic utility classes.

class AlgorandMnemonicUtils

Bases: object

Class container for Algorand mnemonic utility functions.

static ComputeChecksum(*data_bytes: bytes*) → bytes

Compute checksum.

Parameters

data_bytes (*bytes*) – Data bytes

Returns

Computed checksum

Return type

bytes

static ComputeChecksumWordIndex(*data_bytes: bytes*) → int

Compute checksum word index.

Parameters

data_bytes (*bytes*) – Data bytes

Returns

Computed checksum word index

Return type

str

static ConvertBits(*data: Union[bytes, List[int]], from_bits: int, to_bits: int*) → Optional[List[int]]

Perform bit conversion. The function takes the input data (list of integers or byte sequence) and convert every value from the specified number of bits to the specified one. It returns a list of integer where every number is less than $2^{\text{to_bits}}$.

Parameters

- **data** (*list[int] or bytes*) – Data to be converted
- **from_bits** (*int*) – Number of bits to start from
- **to_bits** (*int*) – Number of bits to end with

Returns

List of converted values, None in case of errors

Return type

list[int]

10.1.2.1.7 algorand_mnemonic_validator

Module for Algorand mnemonic validation.

class AlgorandMnemonicValidator(*lang: Optional[AlgorandLanguages] = AlgorandLanguages.ENGLISH*)

Bases: *MnemonicValidator*

Algorand mnemonic validator class. It validates a mnemonic phrase.

m_mnemonic_decoder: *AlgorandMnemonicDecoder*

10.1.2.1.8 algorand_seed_generator

Module for Algorand mnemonic seed generation.

class AlgorandSeedGenerator(*mnemonic: Union[str, Mnemonic], lang: Optional[AlgorandLanguages] = AlgorandLanguages.ENGLISH*)

Bases: object

Algorand seed generator class. It generates the seed from a mnemonic.

m_entropy_bytes: bytes

Generate() → bytes

Generate seed. The seed is simply the entropy bytes in Algorand case. There is no really need of this method, since the seed is always the same, but it's kept in this way to have the same usage of Bip39/Substrate seed generator (i.e. `AlgorandSeedGenerator(mnemonic).Generate()`).

Returns

Generated seed

Return type

bytes

10.1.3 base58

10.1.3.1 base58

Module for base58 decoding/encoding.

class `Base58Alphabets`(*value*)

Bases: Enum

Enumerative for Base58 alphabet.

BITCOIN = 1

RIPPLE = 2

class `Base58Const`

Bases: object

Class container for Base58 constants.

RADIX: int = 58

CHECKSUM_BYTE_LEN: int = 4

ALPHABETS: Dict[`Base58Alphabets`, str] = {<Base58Alphabets.BITCOIN: 1>:
'123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz',
<Base58Alphabets.RIPPLE: 2>:
'rpshnaf39wBUDNEGHJKLM4PQRST7VWXYZ2bcdeCg65jkm8oFqi1tuvAxyz'}

class `Base58Utils`

Bases: object

Class container for Base58 utility functions.

static `ComputeChecksum`(*data_bytes*: bytes) → bytes

Compute Base58 checksum.

Parameters

data_bytes (bytes) – Data bytes

Returns

Computed checksum

Return type

bytes

class `Base58Encoder`

Bases: object

Base58 encoder class. It provides methods for encoding and checksum encoding to Base58 format.

static `Encode`(*data_bytes*: bytes, *alph_idx*: `Base58Alphabets` = `Base58Alphabets.BITCOIN`) → str

Encode bytes into a Base58 string.

Parameters

- **data_bytes** (bytes) – Data bytes
- **alph_idx** (`Base58Alphabets`, optional) – Alphabet index, Bitcoin by default

Returns

Encoded string

Return type

str

Raises**TypeError** – If alphabet index is not a Base58Alphabets enumerative**static CheckEncode**(*data_bytes: bytes, alph_idx: Base58Alphabets = Base58Alphabets.BITCOIN*) → str

Encode bytes into Base58 string with checksum.

Parameters

- **data_bytes** (*bytes*) – Data bytes
- **alph_idx** (*Base58Alphabets, optional*) – Alphabet index, Bitcoin by default

Returns

Encoded string with checksum

Return type

str

Raises**TypeError** – If alphabet index is not a Base58Alphabets enumerative**class Base58Decoder**

Bases: object

Base58 decoder class. It provides methods for decoding and checksum decoding Base58 format.

static Decode(*data_str: str, alph_idx: Base58Alphabets = Base58Alphabets.BITCOIN*) → bytes

Decode bytes from a Base58 string.

Parameters

- **data_str** (*str*) – Data string
- **alph_idx** (*Base58Alphabets, optional*) – Alphabet index, Bitcoin by default

Returns

Decoded bytes

Return type

bytes

Raises**TypeError** – If alphabet index is not a Base58Alphabets enumerative**static CheckDecode**(*data_str: str, alph_idx: Base58Alphabets = Base58Alphabets.BITCOIN*) → bytes

Decode bytes from a Base58 string with checksum.

Parameters

- **data_str** (*str*) – Data string
- **alph_idx** (*Base58Alphabets, optional*) – Alphabet index, Bitcoin by default

Returns

Decoded bytes (checksum removed)

Return type

bytes

Raises

- **ValueError** – If the string is not a valid Base58 format

- **TypeError** – If alphabet index is not a Base58Alphabets enumerative
- **Base58ChecksumError** – If checksum is not valid

10.1.3.2 base58_ex

Module for base58 exceptions.

exception Base58ChecksumError

Bases: Exception

Exception in case of checksum error.

10.1.3.3 base58_xmr

Module for base58-monero decoding/encoding.

class Base58XmrConst

Bases: object

Class container for Base58 Monero constants.

ALPHABET: str = '123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz'

BLOCK_DEC_MAX_BYTE_LEN: int = 8

BLOCK_ENC_MAX_BYTE_LEN: int = 11

BLOCK_ENC_BYTE_LENS: List[int] = [0, 2, 3, 5, 6, 7, 9, 10, 11]

class Base58XmrEncoder

Bases: object

Base58 Monero encoder class. It provides methods for encoding to Base58 format with Monero variation (encoding by blocks of 8-byte).

static Encode(data_bytes: bytes) → str

Encode bytes into a Base58 string with Monero variation.

Parameters

data_bytes (bytes) – Data bytes

Returns

Encoded string

Return type

str

class Base58XmrDecoder

Bases: object

Base58 Monero decoder class. It provides methods for decoding Base58 format with Monero variation (encoding by blocks of 8-byte).

static Decode(data_str: str) → bytes

Decode bytes from a Base58 string with Monero variation.

Parameters

data_str (str) – Data string

Returns

Decoded bytes

Return type

bytes

10.1.4 bech32

10.1.4.1 bch_bech32

Module for BitcoinCash bech32 decoding/encoding. Reference: <https://github.com/bitcoincashorg/bitcoincash.org/blob/master/spec/cashaddr.md>

class BchBech32Const

Bases: object

Class container for Bitcoin Cash Bech32 constants.

SEPARATOR: `str = ':'`**CHECKSUM_STR_LEN:** `int = 8`**class BchBech32Utils**

Bases: object

Class container for Bitcoin Cash utility functions.

static PolyMod(*values: List[int]*) → int

Computes the polynomial modulus.

Parameters**values** (*list[int]*) – List of polynomial coefficients**Returns**

Computed modulus

Return type

int

static HrpExpand(*hrp: str*) → List[int]

Expand the HRP into values for checksum computation.

Parameters**hrp** (*str*) – HRP**Returns**

Expanded HRP values

Return type

list[int]

static ComputeChecksum(*hrp: str, data: List[int]*) → List[int]

Compute the checksum from the specified HRP and data.

Parameters

- **hrp** (*str*) – HRP
- **data** (*list[int]*) – Data part

Returns

Computed checksum

Return type

list[int]

static VerifyChecksum(*hrp: str, data: List[int]*) → bool

Verify the checksum from the specified HRP and converted data characters.

Parameters

- **hrp** (*str*) – HRP
- **data** (*list[int]*) – Data part

Returns

True if valid, false otherwise

Return type

bool

class BchBech32EncoderBases: *Bech32EncoderBase*

Bitcoin Cash Bech32 encoder class. It provides methods for encoding to Bitcoin Cash Bech32 format.

classmethod Encode(*hrp: str, net_ver: bytes, data: bytes*) → str

Encode to Bitcoin Cash Bech32.

Parameters

- **hrp** (*str*) – HRP
- **net_ver** (*bytes*) – Net version
- **data** (*bytes*) – Data

Returns

Encoded address

Return type

str

Raises**ValueError** – If the data is not valid**class BchBech32Decoder**Bases: *Bech32DecoderBase*

Bitcoin Cash Bech32 decoder class. It provides methods for decoding Bitcoin Cash Bech32 format.

classmethod Decode(*hrp: str, addr: str*) → Tuple[bytes, bytes]

Decode from Bitcoin Cash Bech32.

Parameters

- **hrp** (*str*) – Human readable part
- **addr** (*str*) – Address

Returns

Net version (index 0) and data (index 1)

Return type

tuple[bytes, bytes]

Raises

- **ValueError** – If the bech32 string is not valid
- **Bech32ChecksumError** – If the checksum is not valid

10.1.4.2 bech32

Module for bech32/bech32m decoding/encoding.

References

<https://github.com/bitcoin/bips/blob/master/bip-0173.mediawiki> <https://github.com/bitcoin/bips/blob/master/bip-0350.mediawiki> https://github.com/sipa/bech32/blob/master/ref/python/segwit_addr.py

class Bech32Encodings(*value*)

Bases: Enum

Enumerative for Bech32 encoding types.

BECH32 = 1

BECH32M = 2

class Bech32Const

Bases: object

Class container for Bech32 constants.

SEPARATOR: str = '1'

CHECKSUM_STR_LEN: int = 6

ENCODING_CHECKSUM_CONST: Dict[**Bech32Encodings**, int] = {<Bech32Encodings.BECH32: 1>: 1, <Bech32Encodings.BECH32M: 2>: 734539939}

class Bech32Utils

Bases: object

Class container for Bech32 utility functions.

static PolyMod(*values: List[int]*) → int

Computes the polynomial modulus.

Parameters

values (*list[int]*) – List of polynomial coefficients

Returns

Computed modulus

Return type

int

static HrpExpand(*hrp: str*) → List[int]

Expand the HRP into values for checksum computation.

Parameters

hrp (*str*) – HRP

Returns

Expanded HRP values

Return type

list[int]

static ComputeChecksum(*hrp: str, data: List[int], encoding: Bech32Encodings = Bech32Encodings.BECH32*) → List[int]

Compute the checksum from the specified HRP and data.

Parameters

- **hrp** (*str*) – HRP
- **data** (*list[int]*) – Data part
- **encoding** (*Bech32Encodings, optional*) – Encoding type (BECH32 by default)

Returns

Computed checksum

Return type

list[int]

static VerifyChecksum(*hrp: str, data: List[int], encoding: Bech32Encodings = Bech32Encodings.BECH32*) → bool

Verify the checksum from the specified HRP and converted data characters.

Parameters

- **hrp** (*str*) – HRP
- **data** (*list[int]*) – Data part
- **encoding** (*Bech32Encodings, optional*) – Encoding type (BECH32 by default)

Returns

True if valid, false otherwise

Return type

bool

class Bech32Encoder

Bases: *Bech32EncoderBase*

Bech32 encoder class. It provides methods for encoding to Bech32 format.

classmethod Encode(*hrp: str, data: bytes*) → str

Encode to Bech32.

Parameters

- **hrp** (*str*) – HRP
- **data** (*bytes*) – Data

Returns

Encoded address

Return type

str

Raises

ValueError – If the data is not valid

class Bech32DecoderBases: *Bech32DecoderBase*

Bech32 decoder class. It provides methods for decoding Bech32 format.

classmethod Decode(*hrp: str, addr: str*) → bytes

Decode from Bech32.

Parameters

- **hrp** (*str*) – Human readable part
- **addr** (*str*) – Address

Returns

Decoded address

Return type

bytes

Raises

- **ValueError** – If the bech32 string is not valid
- *Bech32ChecksumError* – If the checksum is not valid

10.1.4.3 bech32_base

Module for base bech32 decoding/encoding.

class Bech32BaseConst

Bases: object

Class container for Bech32 constants.

CHARSET: *str* = 'qpzry9x8gf2tvdw0s3jn54khce6mua7l'**class Bech32BaseUtils**

Bases: object

Class container for Bech32 utility functions.

static ConvertToBase32(*data: Union[List[int], bytes]*) → List[int]

Convert data to base32.

Parameters**data** (*list[int]* or *bytes*) – Data to be converted**Returns**

Converted data

Return type

list[int]

Raises**ValueError** – If the string is not valid**static ConvertFromBase32**(*data: Union[List[int], bytes]*) → List[int]

Convert data from base32.

Parameters**data** (*list[int]* or *bytes*) – Data to be converted

Returns

Converted data

Return type

list[int]

Raises**ValueError** – If the string is not valid

static ConvertBits(*data: Union[bytes, List[int]], from_bits: int, to_bits: int, pad: bool = True*) → Optional[List[int]]

Perform bit conversion. The function takes the input data (list of integers or byte sequence) and convert every value from the specified number of bits to the specified one. It returns a list of integer where every number is less than $2^{\text{to_bits}}$.

Parameters

- **data** (*list[int] or bytes*) – Data to be converted
- **from_bits** (*int*) – Number of bits to start from
- **to_bits** (*int*) – Number of bits to end with
- **pad** (*bool, optional*) – True if data must be padded with zeros, false otherwise

Returns

List of converted values, None in case of errors

Return type

list[int]

class Bech32EncoderBase

Bases: ABC

Bech32 encoder base class. It provides methods for encoding to Bech32 format.

class Bech32DecoderBase

Bases: ABC

Bech32 decoder base class. It provides methods for decoding Bech32 format.

10.1.4.4 bech32_ex

Module for bech32 exceptions.

exception Bech32ChecksumError

Bases: Exception

Exception in case of checksum error.

10.1.4.5 segwit_bech32

Module for segwit bech32/bech32m decoding/encoding.

References

<https://github.com/bitcoin/bips/blob/master/bip-0173.mediawiki>
[bip-0350.mediawiki](https://github.com/bitcoin/bips/blob/master/bip-0350.mediawiki)

<https://github.com/bitcoin/bips/blob/master/>

class SegwitBech32Const

Bases: object

Class container for Segwit Bech32 constants.

SEPARATOR: str = '1'

CHECKSUM_STR_LEN: int = 6

WITNESS_PROG_MIN_BYTE_LEN: int = 2

WITNESS_PROG_MAX_BYTE_LEN: int = 40

WITNESS_VER_BECH32: int = 0

WITNESS_VER_MAX_VAL: int = 16

WITNESS_VER_ZERO_DATA_BYTE_LEN: Tuple[int, int] = (20, 32)

class SegwitBech32Encoder

Bases: *Bech32EncoderBase*

Segwit Bech32 encoder class. It provides methods for encoding to Segwit Bech32 format.

classmethod **Encode**(hrp: str, wit_ver: int, wit_prog: bytes) → str

Encode to Segwit Bech32.

Parameters

- **hrp** (str) – HRP
- **wit_ver** (int) – Witness version
- **wit_prog** (bytes) – Witness program

Returns

Encoded address

Return type

str

Raises

ValueError – If the data is not valid

class SegwitBech32Decoder

Bases: *Bech32DecoderBase*

Segwit Bech32 decoder class. It provides methods for decoding Segwit Bech32 format.

classmethod Decode(*hrp: str, addr: str*) → Tuple[int, bytes]

Decode from Segwit Bech32.

Parameters

- **hrp** (*str*) – Human readable part
- **addr** (*str*) – Address

Returns

Witness version (index 0) and witness program (index 1)

Return type

tuple[int, bytes]

Raises

- **Bech32ChecksumError** – If the checksum is not valid
- **ValueError** – If the bech32 string is not valid

10.1.5 bip

10.1.5.1 bip32

10.1.5.1.1 base

10.1.5.1.1.1 bip32_base

Module with BIP32 base class.

class Bip32Base(*priv_key: Optional[Union[bytes, IPrivateKey]], pub_key: Optional[Union[bytes, IPoint, IPublicKey]], key_data: Bip32KeyData, key_net_ver: Bip32KeyNetVersions*)

Bases: ABC

BIP32 base class. It allows master key generation and children keys derivation in according to BIP-0032/SLIP-0010. It shall be derived to implement derivation for a specific elliptic curve.

classmethod FromSeed(*seed_bytes: bytes, key_net_ver: Optional[Bip32KeyNetVersions] = None*) → *Bip32Base*

Create a Bip32 object from the specified seed (e.g. BIP39 seed).

Parameters

- **seed_bytes** (*bytes*) – Seed bytes
- **key_net_ver** (*Bip32KeyNetVersions object, optional*) – Bip32KeyNetVersions object (default: specific class key net version)

Returns

Bip32Base object

Return type

Bip32Base object

Raises

- **ValueError** – If the seed is too short
- **Bip32KeyError** – If the seed is not suitable for master key generation

classmethod FromSeedAndPath(*seed_bytes: bytes, path: Union[str, Bip32Path], key_net_ver: Optional[Bip32KeyNetVersions] = None*) → *Bip32Base*

Create a Bip32 object from the specified seed (e.g. BIP39 seed) and path.

Parameters

- **seed_bytes** (*bytes*) – Seed bytes
- **path** (*str* or *Bip32Path* object) – Path
- **key_net_ver** (*Bip32KeyNetVersions* object, optional) – Bip32KeyNetVersions object (default: specific class key net version)

Returns

Bip32Base object

Return type

Bip32Base object

Raises

- **ValueError** – If the seed length is too short
- **Bip32PathError** – If the path is not valid
- **Bip32KeyError** – If the seed is not suitable for master key generation

classmethod FromExtendedKey(*ex_key_str: str, key_net_ver: Optional[Bip32KeyNetVersions] = None*) → *Bip32Base*

Create a Bip32 object from the specified extended key.

Parameters

- **ex_key_str** (*str*) – Extended key string
- **key_net_ver** (*Bip32KeyNetVersions* object, optional) – Bip32KeyNetVersions object (default: specific class key net version)

Returns

Bip32Base object

Return type

Bip32Base object

Raises

Bip32KeyError – If the key is not valid

classmethod FromPrivateKey(*priv_key: ~typing.Union[bytes, ~bip_utils.ecc.common.ikeys.IPrivateKey], key_data: ~bip_utils.bip.bip32.bip32_key_data.Bip32KeyData = <bip_utils.bip.bip32.bip32_key_data.Bip32KeyData object>, key_net_ver: ~typing.Optional[~bip_utils.bip.bip32.bip32_key_net_ver.Bip32KeyNetVersions] = None*) → *Bip32Base*

Create a Bip32 object from the specified private key and derivation data. If only the private key bytes are specified, the key will be considered a master key with the chain code set to zero, since there is no way to recover the key derivation data.

Parameters

- **priv_key** (*bytes* or *IPrivateKey*) – Private key
- **key_data** (*Bip32KeyData* object, optional) – Key data (default: all zeros)

- **key_net_ver** (*Bip32KeyNetVersions* object, optional) – Bip32KeyNetVersions object (default: specific class key net version)

Returns

Bip32Base object

Return type

Bip32Base object

Raises

Bip32KeyError – If the key is not valid

classmethod FromPublicKey(*pub_key*: ~typing.Union[bytes, ~bip_utils.ecc.common.ipoint.IPoint, ~bip_utils.ecc.common.ikeys.IPublicKey], *key_data*: ~bip_utils.bip.bip32.bip32_key_data.Bip32KeyData = <bip_utils.bip.bip32.bip32_key_data.Bip32KeyData object>, *key_net_ver*: ~typing.Optional[~bip_utils.bip.bip32.bip32_key_net_ver.Bip32KeyNetVersions] = None) → **Bip32Base**

Create a Bip32 object from the specified public key and derivation data. If only the public key bytes are specified, the key will be considered a master key with the chain code set to zero, since there is no way to recover the key derivation data.

Parameters

- **pub_key** (bytes, **IPoint** or **IPublicKey**) – Public key
- **key_data** (*Bip32KeyData* object, optional) – Key data (default: all zeros)
- **key_net_ver** (*Bip32KeyNetVersions* object, optional) – Bip32KeyNetVersions object (default: specific class key net version)

Returns

Bip32Base object

Return type

Bip32Base object

Raises

Bip32KeyError – If the key is not valid

m_priv_key: Optional[**Bip32PrivateKey**]

m_pub_key: **Bip32PublicKey**

ChildKey(*index*: Union[int, Bip32KeyIndex]) → **Bip32Base**

Create and return a child key of the current one with the specified index. The index shall be hardened using HardenIndex method to use the private derivation algorithm.

Parameters

index (int or *Bip32KeyIndex* object) – Index

Returns

Bip32Base object

Return type

Bip32Base object

Raises

Bip32KeyError – If the index results in an invalid key

DerivePath(*path*: Union[str, Bip32Path]) → Bip32Base

Derive children keys from the specified path.

Parameters

path (str or Bip32Path object) – Path

Returns

Bip32Base object

Return type

Bip32Base object

Raises

- **Bip32KeyError** – If the index results in an invalid key
- **Bip32PathError** – If the path is not valid
- **ValueError** – If the path is a master path and the key is a child key

ConvertToPublic() → None

Convert the object into a public one.

IsPublicOnly() → bool

Get if it's public-only.

Returns

True if public-only, false otherwise

Return type

bool

PrivateKey() → Bip32PrivateKey

Return private key object.

Returns

Bip32PrivateKey object

Return type

Bip32PrivateKey object

Raises

Bip32KeyError – If internal key is public-only

PublicKey() → Bip32PublicKey

Return public key object.

Returns

Bip32PublicKey object

Return type

Bip32PublicKey object

KeyNetVersions() → Bip32KeyNetVersions

Get key net versions.

Returns

Bip32KeyNetVersions object

Return type

Bip32KeyNetVersions object

Depth() → *Bip32Depth*

Get current depth.

Returns

Current depth

Return type

Bip32Depth object

Index() → *Bip32KeyIndex*

Get current index.

Returns

Current index

Return type

Bip32KeyIndex object

ChainCode() → *Bip32ChainCode*

Get chain code.

Returns

Chain code

Return type

Bip32ChainCode

FingerPrint() → *Bip32FingerPrint*

Get public key fingerprint.

Returns

Public key fingerprint bytes

Return type

Bip32FingerPrint object

ParentFingerPrint() → *Bip32FingerPrint*

Get parent fingerprint.

Returns

Parent fingerprint bytes

Return type

Bip32FingerPrint object

classmethod Curve() → *EllipticCurve*

Return the elliptic curve.

Returns

EllipticCurve object

Return type

EllipticCurve object

classmethod IsPublicDerivationSupported() → bool

Get if public derivation is supported.

Returns

True if supported, false otherwise.

Return type

bool

abstract static CurveType() → *EllipticCurveTypes*

Return the elliptic curve type.

Returns

Curve type

Return type

EllipticCurveTypes

10.1.5.1.1.2 ibip32_key_derivator

Module for BIP32 SLIP-0010 keys derivation.

class IBip32KeyDerivator

Bases: ABC

Interface for generic BIP32 key derivator.

abstract static IsPublicDerivationSupported() → bool

Get if public derivation is supported.

Returns

True if supported, false otherwise.

Return type

bool

abstract classmethod CkdPriv(*priv_key*: Bip32PrivateKey, *pub_key*: Bip32PublicKey, *index*: Bip32KeyIndex) → Tuple[bytes, bytes]

Derive a child key with the specified index using private derivation.

Parameters

- **priv_key** (*Bip32PrivateKey object*) – Bip32PrivateKey object
- **pub_key** (*Bip32PublicKey object*) – Bip32PublicKey object
- **index** (*Bip32KeyIndex object*) – Key index

Returns

Private key bytes (index 0) and chain code bytes (index 1)

Return type

tuple[bytes, bytes]

Raises

Bip32KeyError – If the index results in an invalid key

abstract classmethod CkdPub(*pub_key*: Bip32PublicKey, *index*: Bip32KeyIndex) → Tuple[Union[bytes, *IPoint*], bytes]

Derive a child key with the specified index using public derivation.

Parameters

- **pub_key** (*Bip32PublicKey object*) – Bip32PublicKey object
- **index** (*Bip32KeyIndex object*) – Key index

Returns

Public key bytes or point (index 0) and chain code bytes (index 1)

Return typetuple[bytes or *IPoint*, bytes]**Raises***Bip32KeyError* – If the index results in an invalid key**10.1.5.1.1.3 ibip32_mst_key_generator**

Module for BIP32 SLIP-0010 keys derivation.

class IBip32MstKeyGenerator

Bases: ABC

Interface for generic BIP32 master key generator.

abstract classmethod **GenerateFromSeed**(*seed_bytes: bytes*) → Tuple[bytes, bytes]

Generate a master key from the specified seed.

Parameters**seed_bytes** (*bytes*) – Seed bytes**Returns**

Private key bytes (index 0) and chain code bytes (index 1)

Return type

tuple[bytes, bytes]

Raises

- *Bip32KeyError* – If the seed is not suitable for master key generation
- *ValueError* – If seed length is not valid

10.1.5.1.2 bip32_const

Module with BIP32 constants.

class Bip32Const

Bases: object

Class container for BIP32 constants.

MAIN_NET_KEY_NET_VERSIONS: *Bip32KeyNetVersions* =
<bip_utils.bip.bip32.bip32_key_net_ver.Bip32KeyNetVersions object>**TEST_NET_KEY_NET_VERSIONS:** *Bip32KeyNetVersions* =
<bip_utils.bip.bip32.bip32_key_net_ver.Bip32KeyNetVersions object>**KHOLAW_KEY_NET_VERSIONS:** *Bip32KeyNetVersions* =
<bip_utils.bip.bip32.bip32_key_net_ver.Bip32KeyNetVersions object>

10.1.5.1.3 bip32_ex

Module with BIP32 exceptions.

exception Bip32KeyError

Bases: Exception

Exception in case of key error.

exception Bip32PathError

Bases: Exception

Exception in case of path error.

10.1.5.1.4 bip32_key_data

Module with helper classes for BIP32 key data.

class Bip32KeyDataConst

Bases: object

Class container for BIP32 key data constants.

CHAINCODE_BYTE_LEN: int = 32

DEPTH_BYTE_LEN: int = 1

FINGERPRINT_BYTE_LEN: int = 4

FINGERPRINT_MASTER_KEY: bytes = b'\x00\x00\x00\x00'

KEY_INDEX_BYTE_LEN: int = 4

KEY_INDEX_MAX_VAL: int = 4294967295

KEY_INDEX_HARDENED_BIT_NUM: int = 31

class Bip32ChainCode(chaincode: bytes =

b'\x00')

Bases: *DataBytes*

BIP32 chaincode class. It represents a BIP32 chaincode.

static FixedLength() → int

Get the fixed length in bytes.

Returns

Length in bytes

Return type

int

m_data_bytes: bytes

class Bip32Fingerprint(fprint: bytes = b'\x00\x00\x00\x00')

Bases: *DataBytes*

BIP32 fingerprint class. It represents a BIP32 fingerprint.

static FixedLength() → int

Get the fixed length in bytes.

Returns

Length in bytes

Return type

int

IsMasterKey() → bool

Get if the fingerprint corresponds to a master key.

Returns

True if it corresponds to a master key, false otherwise

Return type

bool

m_data_bytes: bytes

class Bip32Depth(*depth: int*)

Bases: object

BIP32 depth class. It represents a BIP32 depth.

m_depth: int

static FixedLength() → int

Get the fixed length in bytes.

Returns

Length in bytes

Return type

int

Increase() → *Bip32Depth*

Get a new object with increased depth.

Returns

Bip32Depth object

Return type

Bip32Depth object

ToBytes() → bytes

Get the depth as bytes.

Returns

Depth bytes

Return type

bytes

ToInt() → int

Get the depth as integer.

Returns

Depth index

Return type

int

__int__() → int

Get the depth as integer.

Returns

Depth index

Return type

int

__bytes__() → bytes

Get the depth as bytes.

Returns

Depth bytes

Return type

bytes

__eq__(*other: object*) → bool

Equality operator.

Parameters

other (*int or Bip32Depth object*) – Other object to compare

Returns

True if equal false otherwise

Return type

bool

Raises

TypeError – If the other object is not of the correct type

__gt__(*other: Union[int, Bip32Depth]*) → bool

Greater than operator.

Parameters

other (*int or Bip32Depth object*) – Other value to compare

Returns

True if greater false otherwise

Return type

bool

__lt__(*other: Union[int, Bip32Depth]*) → bool

Lower than operator.

Parameters

other (*int or Bip32Depth object*) – Other value to compare

Returns

True if lower false otherwise

Return type

bool

class Bip32KeyIndex(*idx: int*)

Bases: object

BIP32 key index class. It represents a BIP32 key index.

static HardenIndex(*index: int*) → int

Harden the specified index and return it.

Parameters

index (*int*) – Index

Returns

Hardened index

Return type

int

static UnhardenIndex(*index: int*) → int

Unharden the specified index and return it.

Parameters

index (*int*) – Index

Returns

Unhardened index

Return type

int

static IsHardenedIndex(*index: int*) → bool

Get if the specified index is hardened.

Parameters

index (*int*) – Index

Returns

True if hardened, false otherwise

Return type

bool

classmethod FromBytes(*index_bytes: bytes*) → *Bip32KeyIndex*

Construct class from bytes.

Parameters

index_bytes (*bytes*) – Key index bytes

Returns

Bip32KeyIndex object

Return type

Bip32KeyIndex object

Raises

ValueError – If the index is not valid

m_idx: int

static FixedLength() → int

Get the fixed length in bytes.

Returns

Length in bytes

Return type

int

Harden() → *Bip32KeyIndex*

Get a new Bip32KeyIndex object with the current key index hardened.

Returns

Bip32KeyIndex object

Return type

Bip32KeyIndex object

Unharden() → *Bip32KeyIndex*

Get a new Bip32KeyIndex object with the current key index unhardened.

Returns

Bip32KeyIndex object

Return type

Bip32KeyIndex object

IsHardened() → bool

Get if the key index is hardened.

Returns

True if hardened, false otherwise

Return type

bool

ToBytes(*endianness: typing_extensions.Literal[*little*, *big*] = 'big'*) → bytes

Get the key index as bytes.

Parameters

endianness ("*big*" or "*little*", *optional*) – Endianness (default: big)

Returns

Key bytes

Return type

bytes

ToInt() → int

Get the key index as integer.

Returns

Key index

Return type

int

__int__() → int

Get the key index as integer.

Returns

Key index

Return type

int

__bytes__() → bytes

Get the key index as bytes.

Returns

Key bytes

Return type

Bip32ChainCode object

ParentFingerPrint() → *Bip32FingerPrint*

Get parent fingerprint.

Returns

Parent fingerprint

Return type

Bip32FingerPrint object

10.1.5.1.5 bip32_key_net_ver

Module for BIP32 net version class.

class Bip32KeyNetVersionsConst

Bases: object

Class container for BIP32 key net versions constants.

KEY_NET_VERSION_BYTE_LEN: int = 4**class Bip32KeyNetVersions**(*pub_net_ver: bytes, priv_net_ver: bytes*)

Bases: object

BIP32 key net versions class. It represents a BIP32 key net versions.

m_pub_net_ver: bytes**m_priv_net_ver: bytes****static Length()** → int

Get the key net version length.

Returns

Key net version length

Return type

int

Public() → bytes

Get public net version.

Returns

Public net version

Return type

bytes

Private() → bytes

Get private net version.

Returns

Private net version

Return type

bytes

10.1.5.1.6 bip32_key_ser

Module for BIP32 extended key serialization/deserialization.

class Bip32KeySerConst

Bases: object

Class container for BIP32 key serialize constants.

SERIALIZED_PUB_KEY_BYTE_LEN: int = 78

SERIALIZED_PRIV_KEY_BYTE_LEN: Tuple[int, int] = (78, 110)

class Bip32PrivateKeySerializer

Bases: object

BIP32 private key serializer class. It serializes private keys.

static Serialize(*priv_key: ~bip_utils.ecc.common.ikeys.IPrivateKey, key_data: ~bip_utils.bip.bip32.bip32_key_data.Bip32KeyData, key_net_ver: ~bip_utils.bip.bip32.bip32_key_net_ver.Bip32KeyNetVersions = <bip_utils.bip.bip32.bip32_key_net_ver.Bip32KeyNetVersions object>*) → str

Serialize a private key.

Parameters

- **priv_key** (*IPrivateKey object*) – IPrivateKey object
- **key_data** (*BipKeyData object*) – Key data
- **key_net_ver** (*Bip32KeyNetVersions object, optional*) – Key net versions (BIP32 main net version by default)

Returns

Serialized private key

Return type

str

class Bip32PublicKeySerializer

Bases: object

BIP32 public key serializer class. It serializes public keys.

static Serialize(*pub_key: ~bip_utils.ecc.common.ikeys.IPublicKey, key_data: ~bip_utils.bip.bip32.bip32_key_data.Bip32KeyData, key_net_ver: ~bip_utils.bip.bip32.bip32_key_net_ver.Bip32KeyNetVersions = <bip_utils.bip.bip32.bip32_key_net_ver.Bip32KeyNetVersions object>*) → str

Serialize a public key.

Parameters

- **pub_key** (*IPublicKey object*) – IPublicKey object
- **key_data** (*BipKeyData object*) – Key data
- **key_net_ver** (*Bip32KeyNetVersions object, optional*) – Key net versions (BIP32 main net version by default)

Returns

Serialized public key

Return type

str

class Bip32DeserializedKey(*key_bytes: bytes, key_data: Bip32KeyData, is_public: bool*)

Bases: object

BIP32 deserialized key class. It represents a key deserialized with the Bip32KeyDeserializer.

m_key_bytes: bytes**m_key_data:** *Bip32KeyData***m_is_public:** bool**KeyBytes()** → bytes

Get key bytes.

Returns

Key bytes

Return type

bytes

KeyData() → *Bip32KeyData*

Get key data.

Returns

Bip32KeyData object

Return type

Bip32KeyData object

IsPublic() → bool

Get if public.

Returns

True if the key is public, false otherwise

Return type

bool

class Bip32KeyDeserializer

Bases: object

BIP32 key deserializer class. It deserializes an extended key.

classmethod DeserializeKey(*ser_key_str: str, key_net_ver: ~bip_utils.bip.bip32.bip32_key_net_ver.Bip32KeyNetVersions = <bip_utils.bip.bip32.bip32_key_net_ver.Bip32KeyNetVersions object>*) → *Bip32DeserializedKey*

Deserialize a key.

Parameters

- **ser_key_str** (*str*) – Serialized key string
- **key_net_ver** (*Bip32KeyNetVersions object, optional*) – Key net versions (BIP32 main net version by default)

Returns

Bip32DeserializedKey object

Return type

Bip32DeserializedKey object

Raises**Bip32KeyError** – If the key is not valid**10.1.5.1.7 bip32_keys**

Module for BIP32 keys handling.

class Bip32PublicKey(*pub_key: IPublicKey, key_data: Bip32KeyData, key_net_ver: Bip32KeyNetVersions*)Bases: `_Bip32KeyBase`

BIP32 public key class. It represents a public key used by BIP32 with all the related data (e.g. depth, chain code, etc...).

classmethod FromBytesOrKeyObject(*pub_key: Union[bytes, IPoint, IPublicKey], key_data: Bip32KeyData, key_net_ver: Bip32KeyNetVersions, curve_type: EllipticCurveTypes*) → *Bip32PublicKey*

Get the public key from key bytes or object.

Parameters

- **pub_key** (*bytes*, *IPoint* or *IPublicKey*) – Public key
- **key_data** (*Bip32KeyData* object) – Key data
- **key_net_ver** (*Bip32KeyNetVersions* object) – Key net versions
- **curve_type** (*EllipticCurveTypes*) – Elliptic curve type

Returns

Bip32PublicKey object

Return type

Bip32PublicKey object

Raises**Bip32KeyError** – If the key constructed from the bytes is not valid**classmethod FromBytes**(*key_bytes: bytes, key_data: Bip32KeyData, key_net_ver: Bip32KeyNetVersions, curve_type: EllipticCurveTypes*) → *Bip32PublicKey*

Create from bytes.

Parameters

- **key_bytes** (*bytes*) – Key bytes
- **key_data** (*Bip32KeyData* object) – Key data
- **key_net_ver** (*Bip32KeyNetVersions* object) – Key net versions
- **curve_type** (*EllipticCurveTypes*) – Elliptic curve type

Raises**Bip32KeyError** – If the key constructed from the bytes is not valid**classmethod FromPoint**(*key_point: IPoint, key_data: Bip32KeyData, key_net_ver: Bip32KeyNetVersions*) → *Bip32PublicKey*

Create from point.

Parameters

- **key_point** (*IPoint object*) – Key point
- **key_data** (*Bip32KeyData object*) – Key data
- **key_net_ver** (*Bip32KeyNetVersions object*) – Key net versions

Raises

Bip32KeyError – If the key constructed from the bytes is not valid

m_pub_key: *IPublicKey*

KeyObject() → *IPublicKey*

Return the key object.

Returns

Key object

Return type

IPublicKey object

RawCompressed() → *DataBytes*

Return raw compressed public key.

Returns

DataBytes object

Return type

DataBytes object

RawUncompressed() → *DataBytes*

Return raw uncompressed public key.

Returns

DataBytes object

Return type

DataBytes object

Point() → *IPoint*

Get public key point.

Returns

IPoint object

Return type

IPoint object

FingerPrint() → *Bip32FingerPrint*

Get key fingerprint.

Returns

Key fingerprint bytes

Return type

bytes

KeyIdentifier() → bytes

Get key identifier.

Returns

Key identifier bytes

Return type

bytes

ToExtended() → str

Return key in serialized extended format.

Returns

Key in serialized extended format

Return type

str

class Bip32PrivateKey(priv_key: [IPrivateKey](#), key_data: [Bip32KeyData](#), key_net_ver: [Bip32KeyNetVersions](#))Bases: [_Bip32KeyBase](#)

BIP32 private key class. It represents a private key used by BIP32 with all the related data (e.g. depth, chain code, etc...).

classmethod FromBytesOrKeyObject(priv_key: [Union](#)[bytes, [IPrivateKey](#)], key_data: [Bip32KeyData](#), key_net_ver: [Bip32KeyNetVersions](#), curve_type: [EllipticCurveTypes](#)) → [Bip32PrivateKey](#)

Get the public key from key bytes or object.

Parameters

- **priv_key** (bytes or [IPrivateKey](#)) – Private key
- **key_data** ([Bip32KeyData](#) object) – Key data
- **key_net_ver** ([Bip32KeyNetVersions](#) object) – Key net versions
- **curve_type** ([EllipticCurveTypes](#)) – Elliptic curve type

Returns[Bip32PrivateKey](#) object**Return type**[Bip32PrivateKey](#) object**Raises**[Bip32KeyError](#) – If the key constructed from the bytes is not valid**classmethod FromBytes**(key_bytes: bytes, key_data: [Bip32KeyData](#), key_net_ver: [Bip32KeyNetVersions](#), curve_type: [EllipticCurveTypes](#)) → [Bip32PrivateKey](#)

Create from bytes.

Parameters

- **key_bytes** (bytes) – Key bytes
- **key_data** ([Bip32KeyData](#) object) – Key data
- **key_net_ver** ([Bip32KeyNetVersions](#) object) – Key net versions
- **curve_type** ([EllipticCurveTypes](#)) – Elliptic curve type

Raises[Bip32KeyError](#) – If the key constructed from the bytes is not valid**m_priv_key:** [IPrivateKey](#)

KeyObject() → *IPrivateKey*

Return the key object.

Returns

Key object

Return type

IPrivateKey object

Raw() → *DataBytes*

Return raw private key.

Returns

DataBytes object

Return type

DataBytes object

PublicKey() → *Bip32PublicKey*

Get the public key correspondent to the private one.

Returns

Bip32PublicKey object

Return type

Bip32PublicKey object

ToExtended() → str

Return key in serialized extended format.

Returns

Key in serialized extended format

Return type

str

10.1.5.1.8 bip32_path

Module for BIP32 paths parsing and handling.

class Bip32PathConst

Bases: object

Class container for BIP32 path constants.

HARDENED_CHARS: Tuple[str, str, str] = ('"', 'h', 'p')

MASTER_CHAR: str = 'm'

class Bip32Path(elems: Optional[Sequence[Union[int, Bip32KeyIndex]]] = None, is_absolute: bool = True)

Bases: object

BIP32 path class. It represents a BIP-0032 path.

m_elems: List[Bip32KeyIndex]

m_is_absolute: bool

AddElem(*elem*: Union[int, Bip32KeyIndex]) → Bip32Path

Return a new path object with the specified element added.

Parameters

elem (*str* or Bip32KeyIndex) – Path element

Returns

Bip32Path object

Return type

Bip32Path object

Raises

Bip32PathError – If the path element is not valid

IsAbsolute() → bool

Get if absolute path.

Returns

True if absolute path, false otherwise

Return type

bool

Length() → int

Get the number of elements of the path.

Returns

Number of elements

Return type

int

ToList() → List[int]

Get the path as a list of integers.

Returns

Path as a list of integers

Return type

list[int]

ToStr() → str

Get the path as a string.

Returns

Path as a string

Return type

str

__str__() → str

Get the path as a string.

Returns

Path as a string

Return type

str

__getitem__(*idx: int*) → *Bip32KeyIndex*

Get the specified element index.

Parameters

idx (*int*) – Element index

Returns

Bip32KeyIndex object

Return type

Bip32KeyIndex object

__iter__() → Iterator[*Bip32KeyIndex*]

Get the iterator to the current element.

Returns

Iterator to the current element

Return type

Iterator object

class Bip32PathParser

Bases: object

BIP32 path parser class. It parses a BIP-0032 path and returns a Bip32Path object.

static Parse(*path: str*) → *Bip32Path*

Parse a path and return a Bip32Path object.

Parameters

path (*str*) – Path

Returns

Bip32Path object

Return type

Bip32Path object

Raises

Bip32PathError – If the path is not valid

10.1.5.1.9 bip32_utils

Module with BIP32 utility functions.

class Bip32Utils

Bases: object

BIP32 utility class. It contains some helper methods for Bip32 indexes.

Deprecated: only for compatibility, methods were moved to Bip32KeyIndex.

static HardenIndex(*index: int*) → int

Harden the specified index and return it.

Parameters

index (*int*) – Index

Returns

Hardened index

Return type

int

static UnhardenIndex(*index: int*) → int

Unharden the specified index and return it.

Parameters**index** (*int*) – Index**Returns**

Unhardened index

Return type

int

static IsHardenedIndex(*index: int*) → bool

Get if the specified index is hardened.

Parameters**index** (*int*) – Index**Returns**

True if hardened, false otherwise

Return type

bool

10.1.5.1.10 kholaw

10.1.5.1.10.1 bip32_kholaw_ed25519

Module for keys derivation based on ed25519 curve as defined by BIP32 Khovratovich/Law.

```
class Bip32KholawEd25519(priv_key: Optional[Union[bytes, IPrivateKey]], pub_key: Optional[Union[bytes, IPoint, IPublicKey]], key_data: Bip32KeyData, key_net_ver: Bip32KeyNetVersions)
```

Bases: *Bip32Base*

BIP32 Khovratovich/Law ed25519 class. It allows master keys generation and keys derivation using ed25519 curve.

```
static CurveType() → EllipticCurveTypes
```

Return the elliptic curve type.

Returns

Curve type

Return type*EllipticCurveTypes*

```
m_priv_key: Optional[Bip32PrivateKey]
```

```
m_pub_key: Bip32PublicKey
```

Bip32Ed25519Kholawalias of *Bip32KholawEd25519*

10.1.5.1.10.2 bip32_kholaw_ed25519_key_derivator

Module for BIP32 Khovratovich/Law keys derivation. Reference: https://github.com/LedgerHQ/orakolo/blob/master/papers/Ed25519_BIP%20Final.pdf

class Bip32KholawEd25519KeyDerivator

Bases: *Bip32KholawEd25519KeyDerivatorBase*

BIP32 Khovratovich/Law ed25519 key derivator class. It allows keys derivation for ed25519 curves in according to BIP32 Khovratovich/Law.

10.1.5.1.10.3 bip32_kholaw_key_derivator_base

Module for BIP32 Khovratovich/Law keys derivation (base). Reference: https://github.com/LedgerHQ/orakolo/blob/master/papers/Ed25519_BIP%20Final.pdf

class Bip32KholawEd25519KeyDerivatorBase

Bases: *IBip32KeyDerivator*, ABC

BIP32 Khovratovich/Law ed25519 key derivator base class. It allows keys derivation for ed25519 curves in according to BIP32 Khovratovich/Law. It shall be inherited by child classes to customize the derivation algorithm.

static IsPublicDerivationSupported() → bool

Get if public derivation is supported.

Returns

True if supported, false otherwise.

Return type

bool

classmethod CkdPriv(priv_key: Bip32PrivateKey, pub_key: Bip32PublicKey, index: Bip32KeyIndex) → Tuple[bytes, bytes]

Derive a child key with the specified index using private derivation.

Parameters

- **priv_key** (*Bip32PrivateKey object*) – Bip32PrivateKey object
- **pub_key** (*Bip32PublicKey object*) – Bip32PublicKey object
- **index** (*Bip32KeyIndex object*) – Key index

Returns

Private key bytes (index 0) and chain code bytes (index 1)

Return type

tuple[bytes, bytes]

Raises

Bip32KeyError – If the index results in an invalid key

classmethod CkdPub(pub_key: Bip32PublicKey, index: Bip32KeyIndex) → Tuple[Union[bytes, IPoint], bytes]

Derive a child key with the specified index using public derivation.

Parameters

- **pub_key** (*Bip32PublicKey object*) – Bip32PublicKey object

- **index** (*Bip32KeyIndex* object) – Key index

Returns

Public key bytes or point (index 0) and chain code bytes (index 1)

Return type

tuple[bytes or *IPoint*, bytes]

Raises

Bip32KeyError – If the index results in an invalid key

10.1.5.1.10.4 bip32_kholaw_mst_key_generator

Module for BIP32 Khovratovich/Law master key generation. Reference: https://github.com/LedgerHQ/orakolo/blob/master/papers/Ed25519_BIP%20Final.pdf

class Bip32KholawMstKeyGeneratorConst

Bases: object

Class container for BIP32 Khovratovich/Law master key generator constants.

SEED_MIN_BYTE_LEN: int = 16

MASTER_KEY_HMAC_KEY: bytes = b'ed25519 seed'

class Bip32KholawEd25519MstKeyGenerator

Bases: *IBip32MstKeyGenerator*

BIP32 Khovratovich/Law ed25519 master key generator class. It allows master keys generation in according to BIP32 Khovratovich/Law for ed25519 curve.

classmethod GenerateFromSeed(*seed_bytes: bytes*) → Tuple[bytes, bytes]

Generate a master key from the specified seed.

Parameters

seed_bytes (*bytes*) – Seed bytes

Returns

Private key bytes (index 0) and chain code bytes (index 1)

Return type

tuple[bytes, bytes]

Raises

- *Bip32KeyError* – If the seed is not suitable for master key generation
- **ValueError** – If seed length is not valid

10.1.5.1.11 slip10

10.1.5.1.11.1 bip32_slip10_ed25519

Module for derivation scheme based on ed25519 curve as defined by BIP32 SLIP-0010.

class Bip32Slip10Ed25519(*priv_key: Optional[Union[bytes, IPrivateKey]], pub_key: Optional[Union[bytes, IPoint, IPublicKey]], key_data: Bip32KeyData, key_net_ver: Bip32KeyNetVersions*)

Bases: *Bip32Base*

BIP32 SLIP-0010 ed25519 class. It allows master keys generation and keys derivation using ed25519 curve.

static CurveType() → *EllipticCurveTypes*

Return the elliptic curve type.

Returns

Curve type

Return type

EllipticCurveTypes

m_priv_key: *Optional[Bip32PrivateKey]*

m_pub_key: *Bip32PublicKey*

Bip32Ed25519Slip

alias of *Bip32Slip10Ed25519*

10.1.5.1.11.2 bip32_slip10_ed25519_blake2b

Module for derivation scheme based on ed25519-blake2b curve as defined by BIP32 SLIP-0010.

class Bip32Slip10Ed25519Blake2b(*priv_key: Optional[Union[bytes, IPrivateKey]], pub_key: Optional[Union[bytes, IPoint, IPublicKey]], key_data: Bip32KeyData, key_net_ver: Bip32KeyNetVersions*)

Bases: *Bip32Slip10Ed25519*

BIP32 SLIP-0010 ed25519-blake2b class. It allows master keys generation and keys derivation using ed25519-blake2b curve.

static CurveType() → *EllipticCurveTypes*

Return the elliptic curve type.

Returns

Curve type

Return type

EllipticCurveTypes

m_priv_key: *Optional[Bip32PrivateKey]*

m_pub_key: *Bip32PublicKey*

Bip32Ed25519Blake2bSlip

alias of *Bip32Slip10Ed25519Blake2b*

10.1.5.1.11.3 bip32_slip10_key_derivator

Module for BIP32 SLIP-0010 keys derivation.

References

<https://github.com/satoshilabs/slips/blob/master/slip-0010.md> <https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki>

class Bip32Slip10DerivatorConst

Bases: object

Class container for BIP32 SLIP-0010 derivator constants.

PRIV_KEY_PREFIX: bytes = b'\x00'

class Bip32Slip10EcdsaDerivator

Bases: *IBip32KeyDerivator*

BIP32 SLIP-0010 ECDSA key derivator class. It allows keys derivation for ECDSA curves in according to BIP32 SLIP-0010.

static IsPublicDerivationSupported() → bool

Get if public derivation is supported.

Returns

True if supported, false otherwise.

Return type

bool

classmethod CkdPriv(*priv_key*: Bip32PrivateKey, *pub_key*: Bip32PublicKey, *index*: Bip32KeyIndex) → Tuple[bytes, bytes]

Derive a child key with the specified index using private derivation.

Parameters

- **priv_key** (*Bip32PrivateKey object*) – Bip32PrivateKey object
- **pub_key** (*Bip32PublicKey object*) – Bip32PublicKey object
- **index** (*Bip32KeyIndex object*) – Key index

Returns

Private key bytes (index 0) and chain code bytes (index 1)

Return type

tuple[bytes, bytes]

Raises

Bip32KeyError – If the index results in an invalid key

classmethod CkdPub(*pub_key*: Bip32PublicKey, *index*: Bip32KeyIndex) → Tuple[Union[bytes, *IPoint*], bytes]

Derive a child key with the specified index using public derivation.

Parameters

- **pub_key** (*Bip32PublicKey object*) – Bip32PublicKey object
- **index** (*Bip32KeyIndex object*) – Key index

Returns

Public key bytes or point (index 0) and chain code bytes (index 1)

Return type

tuple[bytes or *IPoint*, bytes]

Raises

Bip32KeyError – If the index results in an invalid key

class Bip32Slip10Ed25519Derivator

Bases: *IBip32KeyDerivator*

BIP32 SLIP-0010 ed25519 key derivator class. It allows keys derivation for ed25519 curves in according to BIP32 SLIP-0010.

static IsPublicDerivationSupported() → bool

Get if public derivation is supported.

Returns

True if supported, false otherwise.

Return type

bool

classmethod CkdPriv(*priv_key*: *Bip32PrivateKey*, *pub_key*: *Bip32PublicKey*, *index*: *Bip32KeyIndex*) → Tuple[bytes, bytes]

Derive a child key with the specified index using private derivation.

Parameters

- **priv_key** (*Bip32PrivateKey object*) – Bip32PrivateKey object
- **pub_key** (*Bip32PublicKey object*) – Bip32PublicKey object
- **index** (*Bip32KeyIndex object*) – Key index

Returns

Private key bytes (index 0) and chain code bytes (index 1)

Return type

tuple[bytes, bytes]

Raises

Bip32KeyError – If the index results in an invalid key

classmethod CkdPub(*pub_key*: *Bip32PublicKey*, *index*: *Bip32KeyIndex*) → Tuple[Union[bytes, *IPoint*], bytes]

Derive a child key with the specified index using public derivation.

Parameters

- **pub_key** (*Bip32PublicKey object*) – Bip32PublicKey object
- **index** (*Bip32KeyIndex object*) – Key index

Returns

Public key bytes or point (index 0) and chain code bytes (index 1)

Return type

tuple[bytes or *IPoint*, bytes]

Raises

Bip32KeyError – If the index results in an invalid key

10.1.5.1.11.4 bip32_slip10_mst_key_generator

Module for BIP32 SLIP-0010 master key generation. Reference: <https://github.com/satoshilabs/slips/blob/master/slip-0010.md>

class Bip32Slip10MstKeyGeneratorConst

Bases: object

Class container for BIP32 SLIP-0010 master key generator constants.

SEED_MIN_BYTE_LEN: int = 16

HMAC_KEY_ED25519_BYTES: bytes = b'ed25519 seed'

HMAC_KEY_NIST256P1_BYTES: bytes = b'Nist256p1 seed'

HMAC_KEY_SECP256K1_BYTES: bytes = b'Bitcoin seed'

class Bip32Slip10Ed2519MstKeyGenerator

Bases: *IBip32MstKeyGenerator*

BIP32 SLIP-0010 ed25519 master key generator class. It allows master keys generation in according to BIP32 SLIP-0010 for ed25519 curve.

classmethod GenerateFromSeed(*seed_bytes: bytes*) → Tuple[bytes, bytes]

Generate a master key from the specified seed.

Parameters

seed_bytes (*bytes*) – Seed bytes

Returns

Private key bytes (index 0) and chain code bytes (index 1)

Return type

tuple[bytes, bytes]

Raises

- **Bip32KeyError** – If the seed is not suitable for master key generation
- **ValueError** – If seed length is not valid

class Bip32Slip10Nist256p1MstKeyGenerator

Bases: *IBip32MstKeyGenerator*

BIP32 SLIP-0010 nist256p1 master key generator class. It allows master keys generation in according to BIP32 SLIP-0010 for nist256p1 curve.

classmethod GenerateFromSeed(*seed_bytes: bytes*) → Tuple[bytes, bytes]

Generate a master key from the specified seed.

Parameters

seed_bytes (*bytes*) – Seed bytes

Returns

Private key bytes (index 0) and chain code bytes (index 1)

Return type

tuple[bytes, bytes]

Raises

- **Bip32KeyError** – If the seed is not suitable for master key generation

- **ValueError** – If seed length is not valid

class Bip32Slip10Secp256k1MstKeyGenerator

Bases: *IBip32MstKeyGenerator*

BIP32 SLIP-0010 secp256k1 master key generator class. It allows master keys generation in according to BIP32 SLIP-0010 for secp256k1 curve.

classmethod GenerateFromSeed(*seed_bytes: bytes*) → Tuple[bytes, bytes]

Generate a master key from the specified seed.

Parameters

seed_bytes (*bytes*) – Seed bytes

Returns

Private key bytes (index 0) and chain code bytes (index 1)

Return type

tuple[bytes, bytes]

Raises

- **Bip32KeyError** – If the seed is not suitable for master key generation
- **ValueError** – If seed length is not valid

10.1.5.1.11.5 bip32_slip10_nist256p1

Module for derivation scheme based on nist256p1 curve as defined by BIP32 SLIP-0010.

class Bip32Slip10Nist256p1(*priv_key: Optional[Union[bytes, IPrivateKey]], pub_key: Optional[Union[bytes, IPoint, IPublicKey]], key_data: Bip32KeyData, key_net_ver: Bip32KeyNetVersions*)

Bases: *Bip32Base*

BIP32 SLIP-0010 nist256p1 class. It allows master keys generation and keys derivation using nist256p1 curve.

static CurveType() → *EllipticCurveTypes*

Return the elliptic curve type.

Returns

Curve type

Return type

EllipticCurveTypes

m_priv_key: Optional[*Bip32PrivateKey*]

m_pub_key: *Bip32PublicKey*

Bip32Nist256p1

alias of *Bip32Slip10Nist256p1*

10.1.5.1.11.6 bip32_slip10_secp256k1

Module for derivation scheme based on secp256k1 curve as defined by BIP32 SLIP-0010.

```
class Bip32Slip10Secp256k1(priv_key: Optional[Union[bytes, IPrivateKey]], pub_key: Optional[Union[bytes, IPoint, IPublicKey]], key_data: Bip32KeyData, key_net_ver: Bip32KeyNetVersions)
```

Bases: *Bip32Base*

BIP32 SLIP-0010 secp256k1 v. It allows master keys generation and keys derivation using secp256k1 curve.

```
static CurveType() → EllipticCurveTypes
```

Return the elliptic curve type.

Returns

Curve type

Return type

EllipticCurveTypes

```
m_priv_key: Optional[Bip32PrivateKey]
```

```
m_pub_key: Bip32PublicKey
```

Bip32Secp256k1

alias of *Bip32Slip10Secp256k1*

10.1.5.2 bip38

10.1.5.2.1 bip38

Module for BIP38 encryption/decryption. Reference: <https://github.com/bitcoin/bips/blob/master/bip-0038.mediawiki>

```
class Bip38Encrypter
```

Bases: object

BIP38 encrypter class. It encrypts a private key using the algorithm specified in BIP38.

```
static EncryptNoEc(priv_key: Union[bytes, IPrivateKey], passphrase: str, pub_key_mode: P2PKHPubKeyModes = P2PKHPubKeyModes.COMPRESSED) → str
```

Encrypt the specified private key without EC multiplication.

Parameters

- **priv_key** (*bytes* or *IPrivateKey*) – Private key bytes or object
- **passphrase** (*str*) – Passphrase
- **pub_key_mode** (*Bip38PubKeyModes*, *optional*) – Public key mode

Returns

Encrypted private key

Return type

str

Raises

- **TypeError** – If the private key is not a *Secp256k1PrivateKey*

- **ValueError** – If the private key bytes are not valid

static GeneratePrivateKeyEc(*passphrase: str, pub_key_mode: P2PKHPubKeyModes = P2PKHPubKeyModes.COMPRESSED, lot_num: Optional[int] = None, sequence_num: Optional[int] = None*) → str

Generate a random encrypted private key with EC multiplication, using the specified parameters. This will generate the intermediate passphrase and use it immediately for generating the private key.

Parameters

- **passphrase** (*str*) – Passphrase
- **pub_key_mode** (*Bip38PubKeyModes, optional*) – Public key mode
- **lot_num** (*int, optional*) – Lot number
- **sequence_num** (*int, optional*) – Sequence number

Returns

Encrypted private key

Return type

str

class Bip38Decrypter

Bases: object

BIP38 decrypter class. It decrypts a private key using the algorithm specified in BIP38.

static DecryptNoEc(*priv_key_enc: str, passphrase: str*) → Tuple[bytes, P2PKHPubKeyModes]

Decrypt the specified private key without EC multiplication.

Parameters

- **priv_key_enc** (*str*) – Encrypted private key bytes
- **passphrase** (*str*) – Passphrase

Returns

Decrypted private key (index 0), public key mode (index 1)

Return type

tuple[bytes, Bip38PubKeyModes]

Raises

- **Base58ChecksumError** – If base58 checksum is not valid
- **ValueError** – If the encrypted key is not valid

static DecryptEc(*priv_key_enc: str, passphrase: str*) → Tuple[bytes, P2PKHPubKeyModes]

Decrypt the specified private key with EC multiplication.

Parameters

- **priv_key_enc** (*str*) – Encrypted private key bytes
- **passphrase** (*str*) – Passphrase

Returns

Decrypted private key (index 0), public key mode (index 1)

Return type

tuple[bytes, Bip38PubKeyModes]

Raises

- **Base58ChecksumError** – If base58 checksum is not valid
- **ValueError** – If the encrypted key is not valid

10.1.5.2.2 bip38_addr

Module with BIP38 utility functions.

class Bip38AddrConst

Bases: object

Class container for BIP38 address constants.

ADDR_HASH_LEN: int = 4

class Bip38Addr

Bases: object

Class for BIP38 address computation.

static AddressHash(*pub_key: Union[bytes, IPublicKey]*, *pub_key_mode: P2PKHPubKeyModes*) → bytes

Compute the address hash as specified in BIP38.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key bytes or object
- **pub_key_mode** (*Bip38PubKeyModes*) – Public key mode

Returns

Address hash

Return type

bytes

Raises

- **TypeError** – If the public key is not a *Secp256k1PublicKey*
- **ValueError** – If the public key bytes are not valid

10.1.5.2.3 bip38_ec

Module for BIP38 encryption/decryption. Reference: <https://github.com/bitcoin/bips/blob/master/bip-0038.mediawiki>

class Bip38EcConst

Bases: object

Class container for BIP38 EC constants.

LOT_NUM_MIN_VAL: int = 0

LOT_NUM_MAX_VAL: int = 1048575

SEQ_NUM_MIN_VAL: int = 0

SEQ_NUM_MAX_VAL: int = 4095

OWNER_SALT_WITH_LOT_SEQ_BYTE_LEN: int = 4

```

OWNER_SALT_NO_LOT_SEQ_BYTE_LEN: int = 8

INT_PASS_ENC_BYTE_LEN: int = 49

INT_PASS_MAGIC_WITH_LOT_SEQ = b',\xe9\xb3\xe1\xff9\xe2Q'

INT_PASS_MAGIC_NO_LOT_SEQ = b',\xe9\xb3\xe1\xff9\xe2S'

SEED_B_BYTE_LEN: int = 24

ENC_BYTE_LEN: int = 39

ENC_KEY_PREFIX: bytes = b'\x01C'

FLAG_BIT_COMPRESSED: int = 5

FLAG_BIT_LOT_SEQ: int = 2

SCRYPT_PREFACTOR_KEY_LEN: int = 32

SCRYPT_PREFACTOR_N: int = 16384

SCRYPT_PREFACTOR_P: int = 8

SCRYPT_PREFACTOR_R: int = 8

SCRYPT_HALVES_KEY_LEN: int = 64

SCRYPT_HALVES_N: int = 1024

SCRYPT_HALVES_P: int = 1

SCRYPT_HALVES_R: int = 1

```

class Bip38EcKeysGenerator

Bases: object

BIP38 keys generator class. It generates intermediate codes and private keys using the algorithm specified in BIP38 with EC multiplication.

static GenerateIntermediatePassphrase(*passphrase: str, lot_num: Optional[int] = None, sequence_num: Optional[int] = None*) → str

Generate an intermediate passphrase from the user passphrase as specified in BIP38.

Parameters

- **passphrase** (*str*) – Passphrase
- **lot_num** (*int, optional*) – Lot number
- **sequence_num** (*int, optional*) – Sequence number

Returns

Intermediate passphrase encoded in base58

Return type

str

static GeneratePrivateKey(*int_passphrase: str*, *pub_key_mode: P2PKHPubKeyModes*) → str

Generate a random encrypted private key from the intermediate passphrase.

Parameters

- **int_passphrase** (*str*) – Intermediate passphrase
- **pub_key_mode** (*Bip38PubKeyModes*) – Public key mode

Returns

Encrypted private key

Return type

str

Raises

- **Base58ChecksumError** – If base58 checksum is not valid
- **ValueError** – If the intermediate code is not valid

class Bip38EcDecrypter

Bases: object

BIP38 decrypter class. It decrypts a private key using the algorithm specified in BIP38 with EC multiplication.

static Decrypt(*priv_key_enc: str*, *passphrase: str*) → Tuple[bytes, *P2PKHPubKeyModes*]

Decrypt the specified private key.

Parameters

- **priv_key_enc** (*str*) – Encrypted private key bytes
- **passphrase** (*str*) – Passphrase

Returns

Decrypted private key (index 0), public key mode (index 1)

Return type

tuple[bytes, Bip38PubKeyModes]

Raises

- **Base58ChecksumError** – If base58 checksum is not valid
- **ValueError** – If the encrypted key is not valid

10.1.5.2.4 bip38_no_ec

Module for BIP38 encryption/decryption. Reference: <https://github.com/bitcoin/bips/blob/master/bip-0038.mediawiki>

class Bip38NoEcConst

Bases: object

Class container for BIP38 no EC constants.

ENC_KEY_BYTE_LEN: int = 39

ENC_KEY_PREFIX: bytes = b'\x01B'

FLAGBYTE_COMPRESSED: bytes = b'\xe0'

```
FLAGBYTE_UNCOMPRESSED: bytes = b'\xc0'
```

```
SCRIPT_KEY_LEN: int = 64
```

```
SCRIPT_N: int = 16384
```

```
SCRIPT_P: int = 8
```

```
SCRIPT_R: int = 8
```

```
class Bip38NoEcEncrypter
```

```
Bases: object
```

BIP38 encrypter class. It encrypts a private key using the algorithm specified in BIP38 without EC multiplication.

```
static Encrypt(priv_key: Union[bytes, IPrivateKey], passphrase: str, pub_key_mode:
                P2PKHPubKeyModes) → str
```

Encrypt the specified private key.

Parameters

- **priv_key** (*bytes* or *IPrivateKey*) – Private key bytes or object
- **passphrase** (*str*) – Passphrase
- **pub_key_mode** (*Bip38PubKeyModes*) – Public key mode

Returns

Encrypted private key

Return type

str

Raises

- **TypeError** – If the private key is not a *Secp256k1PrivateKey*
- **ValueError** – If the private key bytes are not valid

```
class Bip38NoEcDecrypter
```

```
Bases: object
```

BIP38 decrypter class. It decrypts a private key using the algorithm specified in BIP38 without EC multiplication.

```
static Decrypt(priv_key_enc: str, passphrase: str) → Tuple[bytes, P2PKHPubKeyModes]
```

Decrypt the specified private key.

Parameters

- **priv_key_enc** (*str*) – Encrypted private key bytes
- **passphrase** (*str*) – Passphrase

Returns

Decrypted private key (index 0), public key mode (index 1)

Return type

tuple[bytes, Bip38PubKeyModes]

Raises

- **Base58ChecksumError** – If base58 checksum is not valid
- **ValueError** – If the encrypted key is not valid

10.1.5.3 bip39

10.1.5.3.1 bip39_entropy_generator

Module for BIP39 mnemonic entropy generation.

class **Bip39EntropyBitLen**(*value*)

Bases: `IntEnum`

Enumerative for BIP39 entropy bit lengths.

BIT_LEN_128 = 128

BIT_LEN_160 = 160

BIT_LEN_192 = 192

BIT_LEN_224 = 224

BIT_LEN_256 = 256

class **Bip39EntropyGeneratorConst**

Bases: `object`

Class container for BIP39 entropy generator constants.

ENTROPY_BIT_LEN: `List[Bip39EntropyBitLen]` = [`<Bip39EntropyBitLen.BIT_LEN_128: 128>`,
`<Bip39EntropyBitLen.BIT_LEN_160: 160>`, `<Bip39EntropyBitLen.BIT_LEN_192: 192>`,
`<Bip39EntropyBitLen.BIT_LEN_224: 224>`, `<Bip39EntropyBitLen.BIT_LEN_256: 256>`]

class **Bip39EntropyGenerator**(*bit_len*: `Union[int, Bip39EntropyBitLen]`)

Bases: `EntropyGenerator`

BIP39 entropy generator class. It generates random entropy bytes with the specified length.

static **IsValidEntropyBitLen**(*bit_len*: `Union[int, Bip39EntropyBitLen]`) → `bool`

Get if the specified entropy bit length is valid.

Parameters

bit_len (*int* or `Bip39EntropyBitLen`) – Entropy length in bits

Returns

True if valid, false otherwise

Return type

`bool`

static **IsValidEntropyByteLen**(*byte_len*: *int*) → `bool`

Get if the specified entropy byte length is valid.

Parameters

byte_len (*int*) – Entropy length in bytes

Returns

True if valid, false otherwise

Return type

`bool`

m_bit_len: *int*

10.1.5.3.2 bip39_mnemonic

Module for BIP39 mnemonic.

```
class Bip39WordsNum(value)
```

Bases: IntEnum

Enumerative for BIP39 words number.

WORDS_NUM_12 = 12

WORDS_NUM_15 = 15

WORDS_NUM_18 = 18

WORDS_NUM_21 = 21

WORDS_NUM_24 = 24

```
class Bip39Languages(value)
```

Bases: *MnemonicLanguages*

Enumerative for BIP39 languages.

CHINESE_SIMPLIFIED = 1

CHINESE_TRADITIONAL = 2

CZECH = 3

ENGLISH = 4

FRENCH = 5

ITALIAN = 6

KOREAN = 7

PORTUGUESE = 8

SPANISH = 9

```
class Bip39MnemonicConst
```

Bases: object

Class container for BIP39 mnemonic constants.

```
MNEMONIC_WORD_NUM: List[Bip39WordsNum] = [<Bip39WordsNum.WORDS_NUM_12: 12>,
<Bip39WordsNum.WORDS_NUM_15: 15>, <Bip39WordsNum.WORDS_NUM_18: 18>,
<Bip39WordsNum.WORDS_NUM_21: 21>, <Bip39WordsNum.WORDS_NUM_24: 24>]
```

```
LANGUAGE_FILES: Dict[MnemonicLanguages, str] = {<Bip39Languages.ENGLISH: 4>:
'wordlist/english.txt', <Bip39Languages.ITALIAN: 6>: 'wordlist/italian.txt',
<Bip39Languages.FRENCH: 5>: 'wordlist/french.txt', <Bip39Languages.SPANISH: 9>:
'wordlist/spanish.txt', <Bip39Languages.PORTUGUESE: 8>: 'wordlist/portuguese.txt',
<Bip39Languages.CZECH: 3>: 'wordlist/czech.txt',
<Bip39Languages.CHINESE_SIMPLIFIED: 1>: 'wordlist/chinese_simplified.txt',
<Bip39Languages.CHINESE_TRADITIONAL: 2>: 'wordlist/chinese_traditional.txt',
<Bip39Languages.KOREAN: 7>: 'wordlist/korean.txt'}
```

```
WORDS_LIST_NUM: int = 2048
```

```
WORD_BIT_LEN: int = 11
```

```
class Bip39Mnemonic(mnemonic_list: List[str])
```

Bases: [Mnemonic](#)

BIP39 mnemonic class. It adds NFKD normalization to mnemonic.

```
m_mnemonic_list: List[str]
```

10.1.5.3.3 bip39_mnemonic_decoder

Module for BIP39 mnemonic decoding. Reference: <https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki>

```
class Bip39MnemonicDecoder(lang: Optional[Bip39Languages] = None)
```

Bases: [MnemonicDecoderBase](#)

BIP39 mnemonic decoder class. It decodes a mnemonic phrase to bytes.

Decode(mnemonic: Union[str, [Mnemonic](#)]) → bytes

Decode a mnemonic phrase to bytes (no checksum).

Parameters

mnemonic (str or *Mnemonic object*) – Mnemonic

Returns

Decoded bytes (no checksum)

Return type

bytes

Raises

- [MnemonicChecksumError](#) – If checksum is not valid
- [ValueError](#) – If mnemonic is not valid

DecodeWithChecksum(mnemonic: Union[str, [Mnemonic](#)]) → bytes

Decode a mnemonic phrase to bytes (with checksum).

Parameters

mnemonic (str or *Mnemonic object*) – Mnemonic

Returns

Decoded bytes (with checksum)

Return type

bytes

Raises

- [MnemonicChecksumError](#) – If checksum is not valid
- [ValueError](#) – If mnemonic is not valid

```
m_lang: Optional[MnemonicLanguages]
```

```
m_words_list: Optional[MnemonicWordsList]
```

```
m_words_list_finder_cls: Type[MnemonicWordsListFinderBase]
```


10.1.5.3.4 bip39_mnemonic_encoder

Module for BIP39 mnemonic encoding. Reference: <https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki>

class Bip39MnemonicEncoder(*lang*: `Bip39Languages` = `Bip39Languages.ENGLISH`)

Bases: `MnemonicEncoderBase`

BIP39 mnemonic encoder class. It encodes bytes to the mnemonic phrase.

Encode(*entropy_bytes*: `bytes`) → `Mnemonic`

Encode bytes to mnemonic phrase.

Parameters

entropy_bytes (`bytes`) – Entropy bytes (accepted lengths in bits: 128, 160, 192, 224, 256)

Returns

Encoded mnemonic

Return type

Mnemonic object

Raises

ValueError – If entropy is not valid

m_words_list: `MnemonicWordsList`

10.1.5.3.5 bip39_mnemonic_generator

Module for BIP39 mnemonic generation.

class Bip39MnemonicGenerator(*lang*: `Bip39Languages` = `Bip39Languages.ENGLISH`)

Bases: `object`

BIP39 mnemonic generator class. It generates mnemonics in according to BIP39. Mnemonic can be generated randomly from words number or from a specified entropy.

m_mnemonic_encoder: `Bip39MnemonicEncoder`

FromWordsNumber(*words_num*: `Union[int, Bip39WordsNum]`) → `Mnemonic`

Generate mnemonic with the specified words number from random entropy.

Parameters

words_num (`int` or `Bip39WordsNum`) – Number of words (12, 15, 18, 21, 24)

Returns

Generated mnemonic

Return type

Mnemonic object

Raises

ValueError – If words number is not valid

FromEntropy(*entropy_bytes*: `bytes`) → `Mnemonic`

Generate mnemonic from the specified entropy bytes.

Parameters

entropy_bytes (`bytes`) – Entropy bytes (accepted lengths in bits: 128, 160, 192, 224, 256)

Returns

Generated mnemonic

Return type

Mnemonic object

Raises**ValueError** – If entropy byte length is not valid

10.1.5.3.6 bip39_mnemonic_utils

Module for BIP39 mnemonic utility classes.

class Bip39WordsListGetterBases: *MnemonicWordsListGetterBase*

BIP39 words list getter class. It allows to get words list by language so that they are loaded from file only once per language.

GetByLanguage(*lang*: *MnemonicLanguages*) → *MnemonicWordsList*

Get words list by language. Words list of a specific language are loaded from file only the first time they are requested.

Parameters**lang** (*MnemonicLanguages*) – Language**Returns**

MnemonicWordsList object

Return type

MnemonicWordsList object

Raises

- **TypeError** – If the language is not a Bip39Languages enum
- **ValueError** – If loaded words list is not valid

m_words_lists: Dict[*MnemonicLanguages*, *MnemonicWordsList*]**class Bip39WordsListFinder**Bases: *MnemonicWordsListFinderBase*

BIP39 words list finder class. It automatically finds the correct words list from a mnemonic.

classmethod FindLanguage(*mnemonic*: *Mnemonic*) → Tuple[*MnemonicWordsList*, *MnemonicLanguages*]

Automatically find the language of the specified mnemonic and get the correct MnemonicWordsList class for it.

Parameters**mnemonic** (*Mnemonic object*) – Mnemonic object**Returns**

MnemonicWordsList object (index 0), mnemonic language (index 1)

Return typetuple[*MnemonicWordsList*, *MnemonicLanguages*]**Raises****ValueError** – If the mnemonic language cannot be found

10.1.5.3.7 bip39_mnemonic_validator

Module for BIP39 mnemonic validation.

class Bip39MnemonicValidator(*lang: Optional[Bip39Languages] = None*)

Bases: *MnemonicValidator*

BIP39 mnemonic validator class. It validates a mnemonic phrase.

m_mnemonic_decoder: *MnemonicDecoderBase*

10.1.5.3.8 bip39_seed_generator

Module for BIP39 mnemonic seed generation. Reference: <https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki>

class Bip39SeedGeneratorConst

Bases: *object*

Class container for BIP39 seed generator constants.

SEED_SALT_MOD: *str = 'mnemonic'*

SEED_PBKDF2_ROUNDS: *int = 2048*

class Bip39SeedGenerator(*mnemonic: Union[str, Mnemonic], lang: Optional[Bip39Languages] = None*)

Bases: *IBip39SeedGenerator*

BIP39 seed generator class. It generates the seed from a mnemonic in according to BIP39.

m_mnemonic: *Mnemonic*

Generate(*passphrase: str = ""*) → *bytes*

Generate the seed using the specified passphrase.

Parameters

passphrase (*str, optional*) – Passphrase, empty if not specified

Returns

Generated seed

Return type

bytes

10.1.5.3.9 ibip39_seed_generator

Module with interface for BIP39 seed generation classes.

class IBip39SeedGenerator(*mnemonic: Union[str, Mnemonic], lang: Optional[Bip39Languages]*)

Bases: *ABC*

BIP39 seed generator interface.

abstract Generate(*passphrase: str*) → *bytes*

Generate the seed using the specified passphrase.

Parameters

passphrase (*str, optional*) – Passphrase, empty if not specified

Returns

Generated seed

Return type

bytes

10.1.5.4 bip44**10.1.5.4.1 bip44**

Module for BIP44 keys derivation. Reference: <https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki>

class Bip44Const

Bases: object

Class container for BIP44 constants.

SPEC_NAME: str = 'BIP-0044'**PURPOSE:** int = 2147483692**class Bip44**(*bip32_obj*: Bip32Base, *coin_conf*: BipCoinConf)

Bases: Bip44Base

BIP44 class. It allows master key generation and children keys derivation in according to BIP-0044.

classmethod FromSeed(*seed_bytes*: bytes, *coin_type*: BipCoins) → Bip44Base

Create a Bip44Base object from the specified seed (e.g. BIP39 seed).

Parameters

- **seed_bytes** (bytes) – Seed bytes
- **coin_type** (BipCoins) – Coin type, shall be a Bip44Coins enum

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip44Coins enum
- **ValueError** – If the seed is too short
- **Bip32KeyError** – If the seed is not suitable for master key generation

classmethod FromExtendedKey(*ex_key_str*: str, *coin_type*: BipCoins) → Bip44Base

Create a Bip44Base object from the specified extended key.

Parameters

- **ex_key_str** (str) – Extended key string
- **coin_type** (BipCoins) – Coin type, shall be a Bip44Coins enum

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip44Coins enum
- **Bip32KeyError** – If the extended key is not valid

classmethod FromPrivateKey(*priv_key*: ~typing.Union[bytes, ~bip_utils.ecc.common.ikeys.IPrivateKey],
coin_type: ~bip_utils.bip.conf.common.bip_coins.BipCoins, *key_data*:
~bip_utils.bip.bip32.bip32_key_data.Bip32KeyData =
<bip_utils.bip.bip32.bip32_key_data.Bip32KeyData object>) →
Bip44Base

Create a Bip44Base object from the specified private key and derivation data. If only the private key bytes are specified, the key will be considered a master key with the chain code set to zero, since there is no way to recover the key derivation data.

Parameters

- **priv_key** (bytes or IPrivateKey) – Private key
- **coin_type** (BipCoins) – Coin type, shall be a Bip44Coins enum
- **key_data** (Bip32KeyData object, optional) – Key data (default: all zeros)

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip44Coins enum
- **Bip32KeyError** – If the key is not valid

classmethod FromPublicKey(*pub_key*: ~typing.Union[bytes, ~bip_utils.ecc.common.ikeys.IPublicKey],
coin_type: ~bip_utils.bip.conf.common.bip_coins.BipCoins, *key_data*:
~bip_utils.bip.bip32.bip32_key_data.Bip32KeyData =
<bip_utils.bip.bip32.bip32_key_data.Bip32KeyData object>) → *Bip44Base*

Create a Bip44Base object from the specified public key and derivation data. If only the public key bytes are specified, the key will be considered an account key with the chain code set to zero, since there is no way to recover the key derivation data.

Parameters

- **pub_key** (bytes or IPublicKey) – Public key
- **coin_type** (BipCoins) – Coin type, shall be a Bip44Coins enum
- **key_data** (Bip32KeyData object, optional) – Key data (default: all zeros with account depth)

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip44Coins enum
- **Bip32KeyError** – If the key is not valid

Purpose() → *Bip44Base*

Derive a child key from the purpose and return a new Bip44Base object.

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- *Bip44DepthError* – If current depth is not suitable for deriving keys
- *Bip32KeyError* – If the derivation results in an invalid key

Coin() → *Bip44Base*

Derive a child key from the coin type specified at construction and return a new Bip44Base object.

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- *Bip44DepthError* – If current depth is not suitable for deriving keys
- *Bip32KeyError* – If the derivation results in an invalid key

Account(*acc_idx: int*) → *Bip44Base*

Derive a child key from the specified account index and return a new Bip44Base object.

Parameters

acc_idx (*int*) – Account index

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- *Bip44DepthError* – If current depth is not suitable for deriving keys
- *Bip32KeyError* – If the derivation results in an invalid key

Change(*change_type: Bip44Changes*) → *Bip44Base*

Derive a child key from the specified change type and return a new Bip44Base object.

Parameters

change_type (*Bip44Changes*) – Change type, must a Bip44Changes enum

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If change type is not a Bip44Changes enum
- *Bip44DepthError* – If current depth is not suitable for deriving keys

- **Bip32KeyError** – If the derivation results in an invalid key

AddressIndex(*addr_idx: int*) → *Bip44Base*

Derive a child key from the specified address index and return a new Bip44Base object.

Parameters

addr_idx(*int*) – Address index

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

static SpecName() → str

Get specification name.

Returns

Specification name

Return type

str

m_bip32_obj: *Bip32Base*

m_coin_conf: *BipCoinConf*

10.1.5.5 bip44_base

10.1.5.5.1 bip44_base

Module with BIP44 base class.

class Bip44Changes(*value*)

Bases: IntEnum

Enumerative for BIP44 changes.

CHAIN_EXT = 0

CHAIN_INT = 1

class Bip44Levels(*value*)

Bases: IntEnum

Enumerative for BIP44 levels.

MASTER = 0

PURPOSE = 1

COIN = 2

ACCOUNT = 3

CHANGE = 4

ADDRESS_INDEX = 5

class Bip44Base(*bip32_obj*: Bip32Base, *coin_conf*: BipCoinConf)

Bases: ABC

BIP44 base class. It allows coin, account, chain and address keys generation in according to BIP44 or its extensions. The class is meant to be derived by classes implementing BIP44 or its extensions.

m_bip32_obj: *Bip32Base*

m_coin_conf: *BipCoinConf*

PublicKey() → *Bip44PublicKey*

Return the public key.

Returns

Bip44PublicKey object

Return type

Bip44PublicKey object

PrivateKey() → *Bip44PrivateKey*

Return the private key.

Returns

Bip44PrivateKey object

Return type

Bip44PrivateKey object

Raises

Bip32KeyError – If the Bip32 object is public-only

Bip32Object() → *Bip32Base*

Return the BIP32 object.

Returns

Bip32Base object

Return type

Bip32Base object

CoinConf() → *BipCoinConf*

Get coin configuration.

Returns

BipCoinConf object

Return type

BipCoinConf object

IsPublicOnly() → bool

Get if it's public-only.

Returns

True if public-only, false otherwise

Return type

bool

Level() → *Bip44Levels*

Return the current level.

Returns

Current level

Return type

Bip44Levels

IsLevel(*level*: *Bip44Levels*) → bool

Return if the current level is the specified one.

Parameters

level (*Bip44Levels*) – Level to be checked

Returns

True if it's the specified level, false otherwise

Return type

bool

Raises

TypeError – If the level index is not a *Bip44Levels* enum

DeriveDefaultPath() → *Bip44Base*

Derive the default coin path and return a new *Bip44Base* object.

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- *Bip44DepthError* – If the current depth is not suitable for deriving keys
- *Bip32KeyError* – If the derivation results in an invalid key

abstract classmethod FromSeed(*seed_bytes*: bytes, *coin_type*: *BipCoins*) → *Bip44Base*

Create a *Bip44Base* object from the specified seed (e.g. BIP39 seed). The test net flag is automatically set when the coin is derived. However, if you want to get the correct master or purpose keys, you have to specify here if it's a test net.

Parameters

- **seed_bytes** (bytes) – Seed bytes
- **coin_type** (*BipCoins*) – Coin type (the type depends on the specific child class)

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not of the correct type
- **ValueError** – If the seed is too short
- *Bip32KeyError* – If the seed is not suitable for master key generation

abstractmethod FromExtendedKey(*ex_key_str*: str, *coin_type*: BipCoins) → Bip44Base

Create a Bip44Base object from the specified extended key.

Parameters

- **ex_key_str** (*str*) – Extended key string
- **coin_type** (BipCoins) – Coin type (the type depends on the specific child class)

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not of the correct type
- **Bip32KeyError** – If the extended key is not valid

abstractmethod FromPrivateKey(*priv_key*: Union[bytes, IPrivateKey], *coin_type*: BipCoins, *key_data*: Bip32KeyData) → Bip44Base

Create a Bip44Base object from the specified private key and derivation data. If only the private key bytes are specified, the key will be considered a master key with the chain code set to zero, since there is no way to recover the key derivation data.

Parameters

- **priv_key** (*bytes* or IPrivateKey) – Private key
- **coin_type** (BipCoins) – Coin type, shall be a Bip44Coins enum
- **key_data** (Bip32KeyData object) – Key data

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip44Coins enum
- **Bip32KeyError** – If the key is not valid

abstractmethod FromPublicKey(*pub_key*: Union[bytes, IPublicKey], *coin_type*: BipCoins, *key_data*: Bip32KeyData) → Bip44Base

Create a Bip44Base object from the specified public key and derivation data. If only the public key bytes are specified, the key will be considered an account key with the chain code set to zero, since there is no way to recover the key derivation data.

Parameters

- **pub_key** (*bytes* or IPublicKey) – Public key
- **coin_type** (BipCoins) – Coin type, shall be a Bip44Coins enum
- **key_data** (Bip32KeyData object) – Key data

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip44Coins enum
- **Bip32KeyError** – If the key is not valid

abstract Purpose() → *Bip44Base*

Derive a child key from the purpose and return a new Bip44Base object.

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

abstract Coin() → *Bip44Base*

Derive a child key from the coin type specified at construction and return a new Bip44Base object.

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

abstract Account(acc_idx: int) → *Bip44Base*

Derive a child key from the specified account index and return a new Bip44Base object.

Parameters

acc_idx (*int*) – Account index

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

abstract Change(change_type: Bip44Changes) → *Bip44Base*

Derive a child key from the specified change type and return a new Bip44Base object.

Parameters

change_type (*Bip44Changes*) – Change type, must a Bip44Changes enum

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If change type is not a Bip44Changes enum
- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

abstract AddressIndex(*addr_idx: int*) → *Bip44Base*

Derive a child key from the specified address index and return a new Bip44Base object.

Parameters

addr_idx (*int*) – Address index

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

abstract static SpecName() → str

Get specification name.

Returns

Specification name

Return type

str

10.1.5.5.2 bip44_base_ex

Module for BIP44 exceptions.

exception Bip44DepthError

Bases: Exception

Exception in case of derivation from wrong depth.

10.1.5.5.3 bip44_keys

Module for BIP44 keys handling.

class Bip44PublicKey(*pub_key: Bip32PublicKey, coin_conf: BipCoinConf*)

Bases: object

BIP44 public key class. It contains Bip32PublicKey and add the possibility to compute the address from the coin type.

m_pub_key: *Bip32PublicKey*

m_coin_conf: *BipCoinConf*

Bip32Key() → *Bip32PublicKey*

Return the BIP32 key object.

Returns

BIP32 key object

Return type

Bip32PublicKey object

ToExtended() → str

Return key in serialized extended format.

Returns

Key in serialized extended format

Return type

str

ChainCode() → *Bip32ChainCode*

Return the chain code.

Returns

Bip32ChainCode object

Return type

Bip32ChainCode object

RawCompressed() → *DataBytes*

Return raw compressed public key.

Returns

DataBytes object

Return type

DataBytes object

RawUncompressed() → *DataBytes*

Return raw uncompressed public key.

Returns

DataBytes object

Return type

DataBytes object

ToAddress() → str

Return the address correspondent to the public key.

Returns

Address string

Return type

str

class Bip44PrivateKey(*priv_key*: *Bip32PrivateKey*, *coin_conf*: *BipCoinConf*)

Bases: object

BIP44 private key class. It contains Bip32PrivateKey and add the possibility to compute the WIF from the coin type.

m_priv_key: *Bip32PrivateKey*

m_coin_conf: *BipCoinConf*

Bip32Key() → *Bip32PrivateKey*

Return the BIP32 key object.

Returns

BIP32 key object

Return type

Bip32PublicKey object

ToExtended() → str

Return key in serialized extended format.

Returns

Key in serialized extended format

Return type

str

ChainCode() → *Bip32ChainCode*

Return the chain code.

Returns

Bip32ChainCode object

Return type

Bip32ChainCode object

Raw() → *DataBytes*

Return raw compressed public key.

Returns

DataBytes object

Return type

DataBytes object

PublicKey() → *Bip44PublicKey*

Get the public key correspondent to the private one.

Returns

Bip44PublicKey object

Return type

Bip44PublicKey object

ToWif(*pub_key_mode: P2PKHPubKeyModes = P2PKHPubKeyModes.COMPRESSED*) → str

Return key in WIF format.

Parameters

pub_key_mode (*WifPubKeyModes*) – Specify if the private key corresponds to a compressed public key

Returns

Key in WIF format

Return type

str

10.1.5.6 bip49

10.1.5.6.1 bip49

Module for BIP49 keys derivation. Reference: <https://github.com/bitcoin/bips/blob/master/bip-0049.mediawiki>

class Bip49Const

Bases: object

Class container for BIP49 constants.

SPEC_NAME: str = 'BIP-0049'

PURPOSE: int = 2147483697

class Bip49(*bip32_obj*: Bip32Base, *coin_conf*: BipCoinConf)

Bases: Bip44Base

BIP49 class. It allows master key generation and children keys derivation in according to BIP-0049.

classmethod FromSeed(*seed_bytes*: bytes, *coin_type*: BipCoins) → Bip44Base

Create a Bip44Base object from the specified seed (e.g. BIP39 seed).

Parameters

- **seed_bytes** (bytes) – Seed bytes
- **coin_type** (BipCoins) – Coin type, shall be a Bip49Coins enum

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip49Coins enum
- **ValueError** – If the seed is too short
- **Bip32KeyError** – If the seed is not suitable for master key generation

classmethod FromExtendedKey(*ex_key_str*: str, *coin_type*: BipCoins) → Bip44Base

Create a Bip44Base object from the specified extended key.

Parameters

- **ex_key_str** (str) – Extended key string
- **coin_type** (BipCoins) – Coin type, shall be a Bip49Coins enum

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip49Coins enum
- **Bip32KeyError** – If the extended key is not valid

```
classmethod FromPrivateKey(priv_key: ~typing.Union[bytes, ~bip_utils.ecc.common.ikeys.IPrivateKey],
                           coin_type: ~bip_utils.bip.conf.common.bip_coins.BipCoins, key_data:
                           ~bip_utils.bip.bip32.bip32_key_data.Bip32KeyData =
                           <bip_utils.bip.bip32.bip32_key_data.Bip32KeyData object>) →
                           Bip44Base
```

Create a Bip44Base object from the specified private key and derivation data. If only the private key bytes are specified, the key will be considered a master key with the chain code set to zero, since there is no way to recover the key derivation data.

Parameters

- **priv_key** (*bytes* or *IPrivateKey*) – Private key
- **coin_type** (*BipCoins*) – Coin type, shall be a Bip49Coins enum
- **key_data** (*Bip32KeyData object*, *optional*) – Key data (default: all zeros)

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip49Coins enum
- **Bip32KeyError** – If the key is not valid

```
classmethod FromPublicKey(pub_key: ~typing.Union[bytes, ~bip_utils.ecc.common.ikeys.IPublicKey],
                           coin_type: ~bip_utils.bip.conf.common.bip_coins.BipCoins, key_data:
                           ~bip_utils.bip.bip32.bip32_key_data.Bip32KeyData =
                           <bip_utils.bip.bip32.bip32_key_data.Bip32KeyData object>) → Bip44Base
```

Create a Bip44Base object from the specified public key and derivation data. If only the public key bytes are specified, the key will be considered an account key with the chain code set to zero, since there is no way to recover the key derivation data.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key
- **coin_type** (*BipCoins*) – Coin type, shall be a Bip44Coins enum
- **key_data** (*Bip32KeyData object*, *optional*) – Key data (default: all zeros with account depth)

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip44Coins enum
- **Bip32KeyError** – If the key is not valid

Purpose() → *Bip44Base*

Derive a child key from the purpose and return a new Bip44Base object.

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

Coin() → *Bip44Base*

Derive a child key from the coin type specified at construction and return a new Bip44Base object.

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

Account(*acc_idx: int*) → *Bip44Base*

Derive a child key from the specified account index and return a new Bip44Base object.

Parameters**acc_idx** (*int*) – Account index**Returns**

Bip44Base object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

Change(*change_type: Bip44Changes*) → *Bip44Base*

Derive a child key from the specified change type and return a new Bip44Base object.

Parameters**change_type** (*Bip44Changes*) – Change type, must a Bip44Changes enum**Returns**

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If change type is not a Bip44Changes enum
- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

AddressIndex(*addr_idx: int*) → *Bip44Base*

Derive a child key from the specified address index and return a new Bip44Base object.

Parameters

addr_idx (*int*) – Address index

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

static SpecName() → str

Get specification name.

Returns

Specification name

Return type

str

m_bip32_obj: *Bip32Base*

m_coin_conf: *BipCoinConf*

10.1.5.7 bip84

10.1.5.7.1 bip84

Module for BIP84 keys derivation. Reference: <https://github.com/bitcoin/bips/blob/master/bip-0084.mediawiki>

class Bip84Const

Bases: object

Class container for BIP84 constants.

SPEC_NAME: str = 'BIP-0084'

PURPOSE: int = 2147483732

class Bip84(*bip32_obj: Bip32Base, coin_conf: BipCoinConf*)

Bases: *Bip44Base*

BIP84 class. It allows master key generation and children keys derivation in according to BIP-0084.

classmethod FromSeed(*seed_bytes: bytes, coin_type: BipCoins*) → *Bip44Base*

Create a Bip44Base object from the specified seed (e.g. BIP39 seed).

Parameters

- **seed_bytes** (*bytes*) – Seed bytes
- **coin_type** (*BipCoins*) – Coin type, shall be a Bip84Coins enum

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip84Coins enum
- **ValueError** – If the seed is too short
- **Bip32KeyError** – If the seed is not suitable for master key generation

classmethod FromExtendedKey(*ex_key_str*: str, *coin_type*: BipCoins) → Bip44Base

Create a Bip44Base object from the specified extended key.

Parameters

- **ex_key_str** (str) – Extended key string
- **coin_type** (BipCoins) – Coin type, shall be a Bip84Coins enum

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip84Coins enum
- **Bip32KeyError** – If the extended key is not valid

classmethod FromPrivateKey(*priv_key*: ~typing.Union[bytes, ~bip_utils.ecc.common.ikeys.IPrivateKey],
coin_type: ~bip_utils.bip.conf.common.bip_coins.BipCoins, *key_data*:
~bip_utils.bip.bip32.bip32_key_data.Bip32KeyData =
<bip_utils.bip.bip32.bip32_key_data.Bip32KeyData object>) →
Bip44Base

Create a Bip44Base object from the specified private key and derivation data. If only the private key bytes are specified, the key will be considered a master key with the chain code set to zero, since there is no way to recover the key derivation data.

Parameters

- **priv_key** (bytes or IPrivateKey) – Private key
- **coin_type** (BipCoins) – Coin type, shall be a Bip84Coins enum
- **key_data** (Bip32KeyData object, optional) – Key data (default: all zeros)

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip84Coins enum
- **Bip32KeyError** – If the key is not valid

classmethod FromPublicKey(*pub_key*: ~typing.Union[bytes, ~bip_utils.ecc.common.ikeys.IPublicKey],
coin_type: ~bip_utils.bip.conf.common.bip_coins.BipCoins, *key_data*:
~bip_utils.bip.bip32.bip32_key_data.Bip32KeyData =
<bip_utils.bip.bip32.bip32_key_data.Bip32KeyData object>) → Bip44Base

Create a Bip44Base object from the specified public key and derivation data. If only the public key bytes are specified, the key will be considered an account key with the chain code set to zero, since there is no way to recover the key derivation data.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key
- **coin_type** (*BipCoins*) – Coin type, shall be a Bip44Coins enum
- **key_data** (*Bip32KeyData object, optional*) – Key data (default: all zeros with account depth)

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip44Coins enum
- **Bip32KeyError** – If the key is not valid

Purpose() → *Bip44Base*

Derive a child key from the purpose and return a new Bip44Base object.

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

Coin() → *Bip44Base*

Derive a child key from the coin type specified at construction and return a new Bip44Base object.

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

Account(*acc_idx: int*) → *Bip44Base*

Derive a child key from the specified account index and return a new Bip44Base object.

Parameters

acc_idx (*int*) – Account index

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys

- ***Bip32KeyError*** – If the derivation results in an invalid key

Change(*change_type*: *Bip44Changes*) → *Bip44Base*

Derive a child key from the specified change type and return a new *Bip44Base* object.

Parameters

change_type (*Bip44Changes*) – Change type, must a *Bip44Changes* enum

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- ***TypeError*** – If change type is not a *Bip44Changes* enum
- ***Bip44DepthError*** – If current depth is not suitable for deriving keys
- ***Bip32KeyError*** – If the derivation results in an invalid key

AddressIndex(*addr_idx*: *int*) → *Bip44Base*

Derive a child key from the specified address index and return a new *Bip44Base* object.

Parameters

addr_idx (*int*) – Address index

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- ***Bip44DepthError*** – If current depth is not suitable for deriving keys
- ***Bip32KeyError*** – If the derivation results in an invalid key

static SpecName() → *str*

Get specification name.

Returns

Specification name

Return type

str

m_bip32_obj: *Bip32Base*

m_coin_conf: *BipCoinConf*

10.1.5.8 bip86

10.1.5.8.1 bip86

Module for BIP86 keys derivation. Reference: <https://github.com/bitcoin/bips/blob/master/bip-0086.mediawiki>

class Bip86Const

Bases: object

Class container for BIP86 constants.

SPEC_NAME: str = 'BIP-0086'

PURPOSE: int = 2147483734

class Bip86(*bip32_obj*: Bip32Base, *coin_conf*: BipCoinConf)

Bases: Bip44Base

BIP86 class. It allows master key generation and children keys derivation in according to BIP-0086.

classmethod FromSeed(*seed_bytes*: bytes, *coin_type*: BipCoins) → Bip44Base

Create a Bip44Base object from the specified seed (e.g. BIP39 seed).

Parameters

- **seed_bytes** (bytes) – Seed bytes
- **coin_type** (BipCoins) – Coin type, shall be a Bip86Coins enum

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip86Coins enum
- **ValueError** – If the seed is too short
- **Bip32KeyError** – If the seed is not suitable for master key generation

classmethod FromExtendedKey(*ex_key_str*: str, *coin_type*: BipCoins) → Bip44Base

Create a Bip44Base object from the specified extended key.

Parameters

- **ex_key_str** (str) – Extended key string
- **coin_type** (BipCoins) – Coin type, shall be a Bip86Coins enum

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip86Coins enum
- **Bip32KeyError** – If the extended key is not valid

```
classmethod FromPrivateKey(priv_key: ~typing.Union[bytes, ~bip_utils.ecc.common.ikeys.IPrivateKey],
                           coin_type: ~bip_utils.bip.conf.common.bip_coins.BipCoins, key_data:
                           ~bip_utils.bip.bip32.bip32_key_data.Bip32KeyData =
                           <bip_utils.bip.bip32.bip32_key_data.Bip32KeyData object>) →
                           Bip44Base
```

Create a Bip44Base object from the specified private key and derivation data. If only the private key bytes are specified, the key will be considered a master key with the chain code set to zero, since there is no way to recover the key derivation data.

Parameters

- **priv_key** (*bytes* or *IPrivateKey*) – Private key
- **coin_type** (*BipCoins*) – Coin type, shall be a Bip86Coins enum
- **key_data** (*Bip32KeyData object*, *optional*) – Key data (default: all zeros)

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip86Coins enum
- **Bip32KeyError** – If the key is not valid

```
classmethod FromPublicKey(pub_key: ~typing.Union[bytes, ~bip_utils.ecc.common.ikeys.IPublicKey],
                           coin_type: ~bip_utils.bip.conf.common.bip_coins.BipCoins, key_data:
                           ~bip_utils.bip.bip32.bip32_key_data.Bip32KeyData =
                           <bip_utils.bip.bip32.bip32_key_data.Bip32KeyData object>) → Bip44Base
```

Create a Bip44Base object from the specified public key and derivation data. If only the public key bytes are specified, the key will be considered a master key with the chain code set to zero, since there is no way to recover the key derivation data.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key
- **coin_type** (*BipCoins*) – Coin type, shall be a Bip44Coins enum
- **key_data** (*Bip32KeyData object*, *optional*) – Key data (default: all zeros with account depth)

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Bip44Coins enum
- **Bip32KeyError** – If the key is not valid

Purpose() → *Bip44Base*

Derive a child key from the purpose and return a new Bip44Base object.

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

Coin() → *Bip44Base*

Derive a child key from the coin type specified at construction and return a new Bip44Base object.

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

Account(*acc_idx: int*) → *Bip44Base*

Derive a child key from the specified account index and return a new Bip44Base object.

Parameters**acc_idx** (*int*) – Account index**Returns**

Bip44Base object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

Change(*change_type: Bip44Changes*) → *Bip44Base*

Derive a child key from the specified change type and return a new Bip44Base object.

Parameters**change_type** (*Bip44Changes*) – Change type, must a Bip44Changes enum**Returns**

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If change type is not a Bip44Changes enum
- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

AddressIndex(*addr_idx: int*) → *Bip44Base*

Derive a child key from the specified address index and return a new Bip44Base object.

Parameters**addr_idx** (*int*) – Address index**Returns**

Bip44Base object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

static SpecName() → str

Get specification name.

Returns

Specification name

Return type

str

m_bip32_obj: *Bip32Base***m_coin_conf:** *BipCoinConf***10.1.5.9 conf****10.1.5.9.1 bip44****10.1.5.9.1.1 bip44_coins**

Module for BIP44 coins enum.

class Bip44Coins(*value*)Bases: *BipCoins*

Enumerative for supported BIP44 coins.

AKASH_NETWORK = 1**ALGORAND** = 2**APTOS** = 3**ARBITRUM** = 4**AVAX_C_CHAIN** = 5**AVAX_P_CHAIN** = 6**AVAX_X_CHAIN** = 7**AXELAR** = 8**BAND_PROTOCOL** = 9**BINANCE_CHAIN** = 10

```
BINANCE_SMART_CHAIN = 11
BITCOIN = 12
BITCOIN_CASH = 13
BITCOIN_CASH_SLP = 14
BITCOIN_SV = 15
CARDANO_BYRON_ICARUS = 16
CARDANO_BYRON_LEDGER = 17
CELO = 18
CERTIK = 19
CHIHUAHUA = 20
COSMOS = 21
DASH = 22
DOGECOIN = 23
ECASH = 24
ELROND = 25
EOS = 26
ERGO = 27
ETHEREUM = 28
ETHEREUM_CLASSIC = 29
FANTOM_OPERA = 30
FETCH_AI = 31
FETCH_AI_ETH = 32
FILECOIN = 33
HARMONY_ONE_ATOM = 34
HARMONY_ONE_ETH = 35
HARMONY_ONE_METAMASK = 36
HUOBI_CHAIN = 37
ICON = 38
INJECTIVE = 39
IRIS_NET = 40
KAVA = 41
```

KUSAMA_ED25519_SLIP = 42
LITECOIN = 43
METIS = 44
MONERO_ED25519_SLIP = 45
MONERO_SECP256K1 = 46
MULTIVERSX = 47
NANO = 48
NEAR_PROTOCOL = 49
NEO = 50
NINE_CHRONICLES_GOLD = 51
OKEX_CHAIN_ATOM = 52
OKEX_CHAIN_ATOM_OLD = 53
OKEX_CHAIN_ETH = 54
ONTOLOGY = 55
OPTIMISM = 56
OSMOSIS = 57
PI_NETWORK = 58
POLKADOT_ED25519_SLIP = 59
POLYGON = 60
RIPPLE = 61
SECRET_NETWORK_OLD = 62
SECRET_NETWORK_NEW = 63
SOLANA = 64
STAFI = 65
STELLAR = 66
SUI = 67
TERRA = 68
TEZOS = 69
THETA = 70
TRON = 71
VECHAIN = 72

```
VERGE = 73
ZCASH = 74
ZILLIQA = 75
BITCOIN_CASH_TESTNET = 76
BITCOIN_CASH_SLP_TESTNET = 77
BITCOIN_SV_TESTNET = 78
BITCOIN_REGTEST = 79
BITCOIN_TESTNET = 80
DASH_TESTNET = 81
DOGECOIN_TESTNET = 82
ECASH_TESTNET = 83
ERGO_TESTNET = 84
LITECOIN_TESTNET = 85
ZCASH_TESTNET = 86
```

10.1.5.9.1.2 bip44_conf

Module for BIP44 coins configuration.

class Bip44Conf

Bases: object

Class container for BIP44 configuration.

AkashNetwork: *BipCoinConf* = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

Algorand: *BipCoinConf* = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

Aptos: *BipCoinConf* = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

Arbitrum: *BipCoinConf* = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

AvaxCChain: *BipCoinConf* = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

AvaxPChain: *BipCoinConf* = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

AvaxXChain: *BipCoinConf* = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

Axelar: *BipCoinConf* = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

```

BandProtocol: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

BinanceChain: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

BinanceSmartChain: BipCoinConf =
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

BitcoinMainNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

BitcoinRegTest: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

BitcoinTestNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

BitcoinCashMainNet: BipBitcoinCashConf =
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>

BitcoinCashTestNet: BipBitcoinCashConf =
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>

BitcoinCashSlpMainNet: BipBitcoinCashConf =
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>

BitcoinCashSlpTestNet: BipBitcoinCashConf =
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>

BitcoinSvMainNet: BipCoinConf =
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

BitcoinSvTestNet: BipCoinConf =
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

CardanoByronIcarus: BipCoinConf =
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

CardanoByronLedger: BipCoinConf =
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

Celo: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

Certik: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

Chihuahua: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

Cosmos: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

DashMainNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

DashTestNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

```

```
DogecoinMainNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

DogecoinTestNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

EcashMainNet: BipBitcoinCashConf =
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>

EcashTestNet: BipBitcoinCashConf =
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>

Elrond: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

Eos: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

ErgoMainNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

ErgoTestNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

Ethereum: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

EthereumClassic: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

FantomOpera: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

FetchAi: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

FetchAiEth: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

Filecoin: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

HarmonyOneMetamask: BipCoinConf =
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

HarmonyOneEth: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

HarmonyOneAtom: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

HuobiChain: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

Icon: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

Injective: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

IrisNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>
```

```
Kava: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

KusamaEd25519Slip: BipCoinConf =
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

LitecoinMainNet: BipLitecoinConf =
<bip_utils.bip.conf.common.bip_litecoin_conf.BipLitecoinConf object>

LitecoinTestNet: BipLitecoinConf =
<bip_utils.bip.conf.common.bip_litecoin_conf.BipLitecoinConf object>

Metis: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

MoneroEd25519Slip: BipCoinConf =
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

MoneroSecp256k1: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

Nano: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

NearProtocol: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

Neo: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

NineChroniclesGold: BipCoinConf =
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

OkexChainEth: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

OkexChainAtom: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

OkexChainAtomOld: BipCoinConf =
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

Ontology: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

Optimism: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

Osmosis: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

PiNetwork: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

PolkadotEd25519Slip: BipCoinConf =
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

Polygon: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

Ripple: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

SecretNetworkOld: BipCoinConf =
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>
```

```
SecretNetworkNew: BipCoinConf =  
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>  
  
Solana: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>  
Stafi: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>  
Stellar: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>  
Sui: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>  
Terra: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>  
Tezos: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>  
Theta: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>  
Tron: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>  
VeChain: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>  
Verge: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>  
  
ZcashMainNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf  
object>  
  
ZcashTestNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf  
object>  
  
Zilliqa: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>
```

10.1.5.9.1.3 bip44_conf_getter

Module for getting BIP44 coins configuration.

class Bip44ConfGetterConst

Bases: object

Class container for BIP44 configuration getter constants.


```

COIN_TO_CONF: Dict[BipCoins, BipCoinConf] = {<Bip44Coins.AKASH_NETWORK: 1>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>, <Bip44Coins.ALGORAND:
2>: <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip44Coins.APTOS: 3>: <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>, <Bip44Coins.ARBITRUM: 4>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip44Coins.AVAX_C_CHAIN: 5>: <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>, <Bip44Coins.AVAX_P_CHAIN: 6>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip44Coins.AVAX_X_CHAIN: 7>: <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>, <Bip44Coins.AXELAR: 8>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip44Coins.BAND_PROTOCOL: 9>: <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>, <Bip44Coins.BINANCE_CHAIN: 10>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip44Coins.BINANCE_SMART_CHAIN: 11>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>, <Bip44Coins.BITCOIN:
12>: <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip44Coins.BITCOIN_REGTEST: 79>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip44Coins.BITCOIN_TESTNET: 80>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip44Coins.BITCOIN_CASH: 13>:
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>,
<Bip44Coins.BITCOIN_CASH_TESTNET: 76>:
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>,
<Bip44Coins.BITCOIN_CASH_SLP: 14>:
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>,
<Bip44Coins.BITCOIN_CASH_SLP_TESTNET: 77>:
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>,
<Bip44Coins.BITCOIN_SV: 15>: <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>, <Bip44Coins.BITCOIN_SV_TESTNET: 78>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip44Coins.CARDANO_BYRON_ICARUS: 16>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip44Coins.CARDANO_BYRON_LEDGER: 17>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>, <Bip44Coins.CELO: 18>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>, <Bip44Coins.CERTIK:
19>: <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip44Coins.CHIHUAHUA: 20>: <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>, <Bip44Coins.COSMOS: 21>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>, <Bip44Coins.DASH: 22>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip44Coins.DASH_TESTNET: 81>: <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>, <Bip44Coins.DOGE_COIN: 23>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip44Coins.DOGE_COIN_TESTNET: 82>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>, <Bip44Coins.ECASH:
24>: <bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>,
<Bip44Coins.ECASH_TESTNET: 83>:
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>,
<Bip44Coins.ELROND: 25>: <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>, <Bip44Coins.EOS: 26>: <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>, <Bip44Coins.ERGO: 27>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip44Coins.ERGO_TESTNET: 84>: <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>, <Bip44Coins.ETHEREUM: 28>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip44Coins.ETHEREUM_CLASSIC: 29>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,

```

class Bip44ConfGetter

Bases: object

BIP44 configuration getter class. It allows to get the BIP44 configuration of a specific coin.

static **GetConfig**(*coin_type*: BipCoins) → BipCoinConf

Get coin configuration.

Parameters

coin_type (BipCoins) – Coin type

Returns

Coin configuration

Return type

BipCoinConf

Raises

TypeError – If coin type is not of a Bip44Coins enumerative

10.1.5.9.2 bip49

10.1.5.9.2.1 bip49_coins

Module for BIP49 coins enum.

class Bip49Coins(*value*)

Bases: *BipCoins*

Enumerative for supported BIP49 coins.

BITCOIN = 1

BITCOIN_CASH = 2

BITCOIN_CASH_SLP = 3

BITCOIN_SV = 4

DASH = 5

DOGECOIN = 6

ECASH = 7

LITECOIN = 8

ZCASH = 9

BITCOIN_CASH_TESTNET = 10

BITCOIN_CASH_SLP_TESTNET = 11

BITCOIN_SV_TESTNET = 12

BITCOIN_REGTEST = 13

```

BITCOIN_TESTNET = 14

DASH_TESTNET = 15

DOGECOIN_TESTNET = 16

ECASH_TESTNET = 17

LITECOIN_TESTNET = 18

ZCASH_TESTNET = 19

```

10.1.5.9.2.2 bip49_conf

Module for BIP49 coins configuration.

class Bip49Conf

Bases: object

Class container for BIP49 configuration.

BitcoinMainNet: *BipCoinConf* = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

BitcoinRegTest: *BipCoinConf* = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

BitcoinTestNet: *BipCoinConf* = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

BitcoinCashMainNet: *BipBitcoinCashConf* =
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>

BitcoinCashTestNet: *BipBitcoinCashConf* =
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>

BitcoinCashSlpMainNet: *BipBitcoinCashConf* =
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>

BitcoinCashSlpTestNet: *BipBitcoinCashConf* =
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>

BitcoinSvMainNet: *BipCoinConf* =
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

BitcoinSvTestNet: *BipCoinConf* =
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

DashMainNet: *BipCoinConf* = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

DashTestNet: *BipCoinConf* = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

DogecoinMainNet: *BipCoinConf* = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

```
DogecoinTestNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

EcashMainNet: BipBitcoinCashConf =
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>

EcashTestNet: BipBitcoinCashConf =
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>

LitecoinMainNet: BipLitecoinConf =
<bip_utils.bip.conf.common.bip_litecoin_conf.BipLitecoinConf object>

LitecoinTestNet: BipLitecoinConf =
<bip_utils.bip.conf.common.bip_litecoin_conf.BipLitecoinConf object>

ZcashMainNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>

ZcashTestNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>
```

10.1.5.9.2.3 bip49_conf_getter

Module for getting BIP49 coins configuration.

class Bip49ConfGetterConst

Bases: object

Class container for BIP49 configuration getter constants.

```

COIN_TO_CONF: Dict[BipCoins, BipCoinConf] = {<Bip49Coins.BITCOIN: 1>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip49Coins.BITCOIN_REGTEST: 13>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip49Coins.BITCOIN_TESTNET: 14>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip49Coins.BITCOIN_CASH: 2>:
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>,
<Bip49Coins.BITCOIN_CASH_TESTNET: 10>:
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>,
<Bip49Coins.BITCOIN_CASH_SLP: 3>:
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>,
<Bip49Coins.BITCOIN_CASH_SLP_TESTNET: 11>:
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>,
<Bip49Coins.BITCOIN_SV: 4>: <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>, <Bip49Coins.BITCOIN_SV_TESTNET: 12>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>, <Bip49Coins.DASH: 5>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip49Coins.DASH_TESTNET: 15>: <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>, <Bip49Coins.DOGE_COIN: 6>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip49Coins.DOGE_COIN_TESTNET: 16>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>, <Bip49Coins.ECASH: 7>:
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>,
<Bip49Coins.ECASH_TESTNET: 17>:
<bip_utils.bip.conf.common.bip_bitcoin_cash_conf.BipBitcoinCashConf object>,
<Bip49Coins.LITECOIN: 8>:
<bip_utils.bip.conf.common.bip_litecoin_conf.BipLitecoinConf object>,
<Bip49Coins.LITECOIN_TESTNET: 18>:
<bip_utils.bip.conf.common.bip_litecoin_conf.BipLitecoinConf object>,
<Bip49Coins.ZCASH: 9>: <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
object>, <Bip49Coins.ZCASH_TESTNET: 19>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>}

```

class Bip49ConfGetter

Bases: object

BIP49 configuration getter class. It allows to get the BIP49 configuration of a specific coin.

static **GetConfig**(*coin_type*: BipCoins) → BipCoinConf

Get coin configuration.

Parameters

coin_type (BipCoins) – Coin type

Returns

Coin configuration

Return type

BipCoinConf

Raises

TypeError – If coin type is not of a Bip49Coins enumerative

10.1.5.9.3 bip84

10.1.5.9.3.1 bip84_coins

Module for BIP84 coins enum.

```
class Bip84Coins(value)
    Bases: BipCoins
    Enumerative for supported BIP84 coins.
    BITCOIN = 1
    LITECOIN = 2
    BITCOIN_REGTEST = 3
    BITCOIN_TESTNET = 4
    LITECOIN_TESTNET = 5
```

10.1.5.9.3.2 bip84_conf

Module for BIP84 coins configuration.

```
class Bip84Conf
    Bases: object
    Class container for BIP84 configuration.
    BitcoinMainNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
    object>
    BitcoinRegTest: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
    object>
    BitcoinTestNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
    object>
    LitecoinMainNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
    object>
    LitecoinTestNet: BipCoinConf = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf
    object>
```

10.1.5.9.3.3 bip84_conf_getter

Module for getting BIP84 coins configuration.

```
class Bip84ConfGetterConst
    Bases: object
    Class container for BIP84 configuration getter constants.
```

```

COIN_TO_CONF: Dict[BipCoins, BipCoinConf] = {<Bip84Coins.BITCOIN: 1>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip84Coins.BITCOIN_REGTEST: 3>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip84Coins.BITCOIN_TESTNET: 4>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>, <Bip84Coins.LITECOIN:
2>: <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip84Coins.LITECOIN_TESTNET: 5>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>}

```

class Bip84ConfGetter

Bases: object

BIP84 configuration getter class. It allows to get the BIP84 configuration of a specific coin.

static **GetConfig**(*coin_type*: BipCoins) → BipCoinConf

Get coin configuration.

Parameters

coin_type (BipCoins) – Coin type

Returns

Coin configuration

Return type

BipCoinConf

Raises

TypeError – If coin type is not of a Bip84Coins enumerative

10.1.5.9.4 bip86

10.1.5.9.4.1 bip86_coins

Module for BIP86 coins enum.

class Bip86Coins(*value*)

Bases: *BipCoins*

Enumerative for supported BIP86 coins.

BITCOIN = 1

BITCOIN_REGTEST = 2

BITCOIN_TESTNET = 3

10.1.5.9.4.2 bip86_conf

Module for BIP86 coins configuration.

class Bip86Conf

Bases: object

Class container for BIP86 configuration.

BitcoinMainNet: *BipCoinConf* = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

BitcoinRegTest: *BipCoinConf* = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

BitcoinTestNet: *BipCoinConf* = <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>

10.1.5.9.4.3 bip86_conf_getter

Module for getting BIP86 coins configuration.

class Bip86ConfGetterConst

Bases: object

Class container for BIP86 configuration getter constants.

COIN_TO_CONF: Dict[*BipCoins*, *BipCoinConf*] = {<Bip86Coins.BITCOIN: 1>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip86Coins.BITCOIN_REGTEST: 2>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
<Bip86Coins.BITCOIN_TESTNET: 3>:
<bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>}

class Bip86ConfGetter

Bases: object

BIP86 configuration getter class. It allows to get the BIP86 configuration of a specific coin.

static GetConfig(*coin_type*: *BipCoins*) → *BipCoinConf*

Get coin configuration.

Parameters

coin_type (*BipCoins*) – Coin type

Returns

Coin configuration

Return type

BipCoinConf

Raises

TypeError – If coin type is not of a Bip86Coins enumerative

10.1.5.9.5 common

10.1.5.9.5.1 bip_bitcoin_cash_conf

Module with helper class for Bitcoin Cash configuration handling.

```
class BipBitcoinCashConf(coin_names: CoinNames, coin_idx: int, is_testnet: bool, def_path: str, key_net_ver: Bip32KeyNetVersions, wif_net_ver: bytes, bip32_cls: Type[Bip32Base], addr_cls: Type[IAddrEncoder], addr_cls_legacy: Type[IAddrEncoder], addr_params: Dict[str, Any])
```

Bases: *BipCoinConf*

Bitcoin Cash configuration class. It allows to return different addresses depending on the configuration.

m_addr_cls_legacy: *Type[IAddrEncoder]*

m_use_legacy_addr: *bool*

UseLegacyAddress(*value: bool*) → *None*

Select if use the legacy address.

Parameters

value (*bool*) – True for using legacy address, false for using the standard one

AddrClass() → *Type[IAddrEncoder]*

Get the address type. It overrides the method in BipCoinConf.

Returns

Address class

Return type

IAddrEncoder class

AddrParams() → *Dict[str, Any]*

Get the address parameters. It overrides the method in BipCoinConf.

Returns

Address parameters

Return type

dict

10.1.5.9.5.2 bip_coin_conf

Module with helper class for generic BIP coins configuration handling.

```
class BipCoinFctCallsConf(*args: str)
```

Bases: *object*

Bip coin function calls configuration class.

m_fct_names: *Tuple[str, ...]*

ResolveCalls(*pub_key: Bip32PublicKey*) → *Any*

Resolve function calls and get the result.

Parameters

pub_key (*Bip32PublicKey object*) – Bip32PublicKey object

Returns

Result

Return type

Any

```
class BipCoinConf(coin_names: CoinNames, coin_idx: int, is_testnet: bool, def_path: str, key_net_ver:
    Bip32KeyNetVersions, wif_net_ver: Optional[bytes], bip32_cls: Type[Bip32Base],
    addr_cls: Type[IAddrEncoder], addr_params: Dict[str, Any])
```

Bases: object

Bip coin configuration class.

m_coin_names: *CoinNames*

m_coin_idx: int

m_is_testnet: bool

m_def_path: str

m_key_net_ver: *Bip32KeyNetVersions*

m_wif_net_ver: Optional[bytes]

m_bip32_cls: Type[*Bip32Base*]

m_addr_params: Dict[str, Any]

m_any_addr_params_fct_call: bool

m_addr_cls: Type[*IAddrEncoder*]

CoinNames() → *CoinNames*

Get coin names.

Returns

CoinNames object

Return type

CoinNames object

CoinIndex() → int

Get coin index.

Returns

Coin index

Return type

int

IsTestNet() → bool

Get if test net.

Returns

True if test net, false otherwise

Return type

bool

DefaultPath() → str

Get the default derivation path.

Returns

Default derivation path

Return type

str

KeyNetVersions() → *Bip32KeyNetVersions*

Get key net versions.

Returns

Bip32KeyNetVersions object

Return type

Bip32KeyNetVersions object

WifNetVersion() → Optional[bytes]

Get WIF net version.

Returns

WIF net version bytes None: If WIF is not supported

Return type

bytes

Bip32Class() → Type[*Bip32Base*]

Get the Bip32 class.

Returns

Bip32Base class

Return type

Bip32Base class

AddrParams() → Dict[str, Any]

Get the address parameters.

Returns

Address parameters

Return type

dict

AddrParamsWithResolvedCalls(*pub_key*: Bip32PublicKey) → Dict[str, Any]

Get the address parameters with resolved function calls.

Parameters

pub_key (*Bip32PublicKey object*) – Bip32PublicKey object

Returns

Address parameters

Return type

dict

AddrClass() → Type[*IAddrEncoder*]

Get the address class.

Returns

Address class

Return type

IAddrEncoder class

10.1.5.9.5.3 bip_coins

Module for generic BIP coins enum.

class BipCoins(*value*)

Bases: Enum

Base enum for bip coins.

10.1.5.9.5.4 bip_conf_const

Module for generic BIP configuration constants.

10.1.5.9.5.5 bip_litecoin_conf

Module with helper class for Litecoin configuration handling.

class BipLitecoinConf(*coin_names: CoinNames, coin_idx: int, is_testnet: bool, def_path: str, key_net_ver: Bip32KeyNetVersions, alt_key_net_ver: Bip32KeyNetVersions, wif_net_ver: bytes, bip32_cls: Type[Bip32Base], addr_cls: Type[IAddrEncoder], addr_params: Dict[str, Any]*)Bases: *BipCoinConf*

Litecoin configuration class. It allows to return different addresses and key net versions depending on the configuration.

m_alt_key_net_ver: *Bip32KeyNetVersions***m_use_alt_key_net_ver:** *bool***m_use_depr_addr:** *bool***UseAlternateKeyNetVersions**(*value: bool*) → None

Select if use the alternate key net version.

Parameters**value** (*bool*) – True for using alternate key net version, false for using the standard one**UseDeprecatedAddress**(*value: bool*) → None

Select if use the deprecated address.

Parameters**value** (*bool*) – True for using deprecated address, false for using the standard one**KeyNetVersions**() → *Bip32KeyNetVersions*

Get key net versions. It overrides the method in BipCoinConf. Litecoin overrides the method because it can have 2 different key net versions.

Returns*Bip32KeyNetVersions* object**Return type***Bip32KeyNetVersions* object

AddrParams() → Dict[str, Any]

Get the address parameters. It overrides the method in BipCoinConf.

Returns

Address parameters

Return type

dict

10.1.6 brainwallet

10.1.6.1 brainwallet

Module for keys generation using a brainwallet (i.e. passphrase chosen by the user).

class Brainwallet(*bip44_obj*: Bip44Base)

Bases: object

Brainwallet class. It allows to generate a key pair from a passphrase chosen by the user for different coins and with different algorithms.

classmethod Generate(*passphrase*: str, *coin_type*: Bip44Coins, *algo_type*: BrainwalletAlgos, ***algo_params*: Any) → Brainwallet

Generate a brainwallet from the specified passphrase and coin with the specified algorithm.

Parameters

- **passphrase** (*str*) – Passphrase
- **coin_type** (*BrainwalletCoins*) – Coin type
- **algo_type** (*BrainwalletAlgos*) – Algorithm type
- ****algo_params** – Algorithm parameters, if any

Returns

Algorithm class

Return type

Brainwallet object

Raises

TypeError – If algorithm type is not of a BrainwalletAlgos enumerative or coin type is not of a BrainwalletCoins enumerative

classmethod GenerateWithCustomAlgo(*passphrase*: str, *coin_type*: Bip44Coins, *algo_cls*: Type[IBrainwalletAlgo], ***algo_params*: Any) → Brainwallet

Generate a brainwallet from the specified passphrase and coin with a custom algorithm.

Parameters

- **passphrase** (*str*) – Passphrase
- **coin_type** (*BrainwalletCoins*) – Coin type
- **algo_cls** (*IBrainwalletAlgo class*) – Algorithm class
- ****algo_params** – Algorithm parameters, if any

Returns

Algorithm class

Return type

Brainwallet object

Raises**TypeError** – If algorithm type is not of a BrainwalletAlgos enumerative or coin type is not of a BrainwalletCoins enumerative**bip44_obj:** *Bip44Base***PublicKey()** → *Bip44PublicKey*

Return the public key.

Returns

Bip44PublicKey object

Return type

Bip44PublicKey object

PrivateKey() → *Bip44PrivateKey*

Return the private key.

Returns

Bip44PrivateKey object

Return type

Bip44PrivateKey object

10.1.6.2 brainwallet_algo

Module for implementing algorithms for brainwallet generation.

class BrainwalletAlgos(*value*)

Bases: Enum

Enum for brainwallet algorithms.

SHA256 = 1**DOUBLE_SHA256** = 2**PBKDF2_HMAC_SHA512** = 3**SCRYPT** = 4**class BrainwalletAlgoConst**

Bases: object

Class container for brainwallet algorithm constants.

PBKDF2_HMAC_SHA512_KEY_LEN: int = 32**PBKDF2_HMAC_SHA512_DEF_ITR_NUM**: int = 2097152**SCRYPT_KEY_LEN**: int = 32**SCRYPT_DEF_N**: int = 131072**SCRYPT_DEF_P**: int = 8**SCRYPT_DEF_R**: int = 8

class BrainwalletAlgoSha256Bases: *IBrainwalletAlgo*

Compute the private key from passphrase using SHA256 algorithm.

static ComputePrivateKey(*passphrase: str, **kwargs: Any*) → bytes

Compute the private key from the specified passphrase.

Parameters

- **passphrase** (*str*) – Passphrase
- ****kwargs** – Not used

Returns

Private key bytes

Return type

bytes

class BrainwalletAlgoDoubleSha256Bases: *IBrainwalletAlgo*

Compute the private key from passphrase using double SHA256 algorithm.

static ComputePrivateKey(*passphrase: str, **kwargs: Any*) → bytes

Compute the private key from the specified passphrase.

Parameters

- **passphrase** (*str*) – Passphrase
- ****kwargs** – Not used

Returns

Private key bytes

Return type

bytes

class BrainwalletAlgoPbkdf2HmacSha512Bases: *IBrainwalletAlgo*

Compute the private key from passphrase using PBKDF2 HMAC-SHA512 algorithm.

static ComputePrivateKey(*passphrase: str, **kwargs: Any*) → bytes

Compute the private key from the specified passphrase.

Parameters

- **passphrase** (*str*) – Passphrase
- **salt** (*str*) – Salt for PBKDF2 algorithm (default: empty)
- **itr_num** (*int*) – Number of iteration for PBKDF2 algorithm (default: 2097152)

Returns

Private key bytes

Return type

bytes

class BrainwalletAlgoScriptBases: *IBrainwalletAlgo*

Compute the private key from passphrase using Script algorithm.

static ComputePrivateKey(*passphrase: str, **kwargs: Any*) → bytes

Compute the private key from the specified passphrase.

Parameters

- **passphrase** (*str*) – Passphrase
- **salt** (*str*) – Salt for Script algorithm (default: empty)
- **n** (*int*) – CPU/Memory cost parameter for Script algorithm (default: 131072)
- **r** (*int*) – Block size parameter for Script algorithm (default: 8)
- **p** (*int*) – Parallelization parameter for Script algorithm (default: 8)

Returns

Private key bytes

Return type

bytes

10.1.6.3 brainwallet_algo_getter

Module for getting brainwallet algorithms.

class BrainwalletAlgoGetterConst

Bases: object

Class container for brainwallet algorithm getter constants.

```
ENUM_TO_ALGO: Dict[BrainwalletAlgos, Type[IBrainwalletAlgo]] =
{<BrainwalletAlgos.SHA256: 1>: <class
' bip_utils.brainwallet.brainwallet_algo.BrainwalletAlgoSha256'>,
<BrainwalletAlgos.DOUBLE_SHA256: 2>: <class
' bip_utils.brainwallet.brainwallet_algo.BrainwalletAlgoDoubleSha256'>,
<BrainwalletAlgos.PBKDF2_HMAC_SHA512: 3>: <class
' bip_utils.brainwallet.brainwallet_algo.BrainwalletAlgoPbkdf2HmacSha512'>,
<BrainwalletAlgos.SCRYPT: 4>: <class
' bip_utils.brainwallet.brainwallet_algo.BrainwalletAlgoScript'>}
```

class BrainwalletAlgoGetter

Bases: object

Brainwallet algorithm getter class. It allows to get the a specific brainwallet algorithm.

static GetAlgo(*algo_type: BrainwalletAlgos*) → Type[*IBrainwalletAlgo*]

Get algorithm class.

Parameters**algo_type** (*BrainwalletAlgos*) – Algorithm type**Returns**

Algorithm class

Return type

IBrainwalletAlgo class

Raises**TypeError** – If algorithm type is not of a BrainwalletAlgos enumerative**10.1.6.4 ibrainwallet_algo**

Module for implementing algorithms for brainwallet generation.

class IBrainwalletAlgo

Bases: ABC

Interface for an algorithm that computes a private key for a brainwallet. It can be inherited to implement custom algorithms.

abstract static ComputePrivateKey(*passphrase: str, **kwargs: Any*) → bytes

Compute the private key from the specified passphrase.

Parameters

- **passphrase** (*str*) – Passphrase
- ****kwargs** – Arbitrary arguments depending on the algorithm

Returns

Private key bytes

Return type

bytes

10.1.7 cardano**10.1.7.1 bip32****10.1.7.1.1 cardano_byron_legacy_bip32**

Module for keys derivation based for Cardano Byron (legacy).

class CardanoByronLegacyBip32(*priv_key: Optional[Union[bytes, IPrivateKey]], pub_key: Optional[Union[bytes, IPoint, IPublicKey]], key_data: Bip32KeyData, key_net_ver: Bip32KeyNetVersions*)

Bases: *Bip32Base*

Cardano Byron legacy BIP32 class. It allows master keys generation and keys derivation for Cardano-Byron (legacy, used by old Daedalus). Derivation based on BIP32 ed25519 Khovratovich/Law with a different algorithm for master key generation and keys derivation.

static CurveType() → *EllipticCurveTypes*

Return the elliptic curve type.

Returns

Curve type

Return type*EllipticCurveTypes***m_priv_key:** *Optional[Bip32PrivateKey]***m_pub_key:** *Bip32PublicKey*

10.1.7.1.2 cardano_byron_legacy_key_derivator

Module for Cardano Byron legacy BIP32 keys derivation.

References

<https://input-output-hk.github.io/cardano-wallet/concepts/master-key-generation-byron.md> <https://cips.cardano.org/cips/cip3/>

class CardanoByronLegacyKeyDerivator

Bases: *Bip32KholawEd25519KeyDerivatorBase*

Cardano Byron legacy key derivator class. It allows keys derivation for Cardano-Byron (legacy, used by old versions of Daedalus). Derivation based on BIP32 ed25519 Khovratovich/Law with some differences on keys computation.

10.1.7.1.3 cardano_byron_legacy_mst_key_generator

Module for Cardano Byron legacy master key generation.

References

<https://input-output-hk.github.io/cardano-wallet/concepts/master-key-generation-byron.md> <https://cips.cardano.org/cips/cip3/>

class CardanoByronLegacyMstKeyGeneratorConst

Bases: object

Class container for Cardano Byron legacy BIP32 constants.

HMAC_MESSAGE_FORMAT: bytes = b'Root Seed Chain %d'

SEED_BYTE_LEN: int = 32

class CardanoByronLegacyMstKeyGenerator

Bases: *IBip32MstKeyGenerator*

Cardano Byron legacy master key generator class. It allows master keys generation in according to Cardano Byron (legacy, used by old versions of Daedalus).

classmethod GenerateFromSeed(seed_bytes: bytes) → Tuple[bytes, bytes]

Generate a master key from the specified seed.

Parameters

seed_bytes (bytes) – Seed bytes

Returns

Private key bytes (index 0) and chain code bytes (index 1)

Return type

tuple[bytes, bytes]

Raises

- **Bip32KeyError** – If the seed is not suitable for master key generation
- **ValueError** – If seed length is not valid

10.1.7.1.4 cardano_icarus_bip32

Module for keys derivation based for Cardano Icarus.

```
class CardanoIcarusBip32(priv_key: Optional[Union[bytes, IPrivateKey]], pub_key: Optional[Union[bytes,
    IPoint, IPublicKey]], key_data: Bip32KeyData, key_net_ver:
    Bip32KeyNetVersions)
```

Bases: *Bip32KholawEd25519*

Cardano Icarus BIP32 class. It allows master keys generation and keys derivation for Cardano Icarus. Derivation based on BIP32 ed25519 Khovratovich/Law with a different algorithm for master key generation.

m_priv_key: Optional[*Bip32PrivateKey*]

m_pub_key: *Bip32PublicKey*

10.1.7.1.5 cardano_icarus_mst_key_generator

Module for Cardano Icarus master key generation.

References

<https://input-output-hk.github.io/cardano-wallet/concepts/master-key-generation> <https://cips.cardano.org/cips/cip3/icarus.md>

```
class CardanoIcarusMasterKeyGeneratorConst
```

Bases: object

Class container for Cardano Icarus master key generator constants.

PBKDF2_PASSWORD: str = ''

PBKDF2_ROUNDS: int = 4096

PBKDF2_OUT_BYTE_LEN: int = 96

```
class CardanoIcarusMstKeyGenerator
```

Bases: *IBip32MstKeyGenerator*

Cardano Icarus master key generator class. It allows master keys generation in according to Cardano Icarus.

```
classmethod GenerateFromSeed(seed_bytes: bytes) → Tuple[bytes, bytes]
```

Generate a master key from the specified seed.

Parameters

seed_bytes (*bytes*) – Seed bytes

Returns

Private key bytes (index 0) and chain code bytes (index 1)

Return type

tuple[bytes, bytes]

Raises

- *Bip32KeyError* – If the seed is not suitable for master key generation
- *ValueError* – If seed length is not valid

10.1.7.2 byron

10.1.7.2.1 cardano_byron_legacy

Module for Cardano Byron legacy keys derivation.

class `CardanoByronLegacyConst`

Bases: `object`

Class container for Cardano Byron legacy constants.

HD_PATH_KEY_PBKDF2_SALT: `str` = 'address-hashing'

HD_PATH_KEY_PBKDF2_ROUNDS: `int` = 500

HD_PATH_KEY_PBKDF2_OUT_BYTE_LEN: `int` = 32

class `CardanoByronLegacy`(*bip32_obj*: `Bip32Base`)

Bases: `object`

Cardano Byron legacy class. It allows master key generation, children keys derivation and addresses computation like the old Daedalus wallet. Addresses are in the Ddz... format, which contains the encrypted derivation path.

classmethod `FromSeed`(*seed_bytes*: `bytes`) → `CardanoByronLegacy`

Construct class from seed bytes.

Parameters

seed_bytes (`bytes`) – Seed bytes

Returns

`CardanoByronLegacy` object

Return type

`CardanoByronLegacy` object

m_bip32_obj: `Bip32Base`

Bip32Object() → `Bip32Base`

Return the BIP32 object.

Returns

`Bip32Base` object

Return type

`Bip32Base` object

HdPathKey() → `bytes`

Get the key used for HD path decryption/encryption.

Returns

Key bytes

Return type

`bytes`

HdPathFromAddress(*address*: `str`) → `Bip32Path`

Get the HD path from an address by decrypting it. The address shall be derived from the current object master key (i.e. `self.m_bip32_obj`) in order to successfully decrypt the path.

Parameters

address (`str`) – Address string

Returns

Bip32Path object

Return type

Bip32Path object

Raises**ValueError** – If the address encoding is not valid or the path cannot be decrypted**MasterPrivateKey()** → *Bip32PrivateKey*

Get the master private key.

Returns

Bip32PrivateKey object

Return type

Bip32PrivateKey object

MasterPublicKey() → *Bip32PublicKey*

Get the master public key.

Returns

Bip32PublicKey object

Return type

Bip32PublicKey object

GetPrivateKey(*first_idx: Union[int, Bip32KeyIndex]*, *second_idx: Union[int, Bip32KeyIndex]*) → *Bip32PrivateKey*

Get the private key with the specified indexes. Derivation path: m/first_idx'/second_idx' The indexes will be automatically hardened if not (e.g. 0, 1' -> 0', 1').

Parameters

- **first_idx** (*int or Bip32KeyIndex object*) – First index
- **second_idx** (*int or Bip32KeyIndex object*) – Second index

Returns

IPrivateKey object

Return type

IPrivateKey object

Raises

- **Bip32KeyError** – If the derivation results in an invalid key
- **Bip32PathError** – If the path indexes are not valid

GetPublicKey(*first_idx: Union[int, Bip32KeyIndex]*, *second_idx: Union[int, Bip32KeyIndex]*) → *Bip32PublicKey*

Get the public key with the specified indexes. Derivation path: m/first_idx'/second_idx' The indexes will be automatically hardened if not (e.g. 0, 1' -> 0', 1').

Parameters

- **first_idx** (*int or Bip32KeyIndex object*) – First index
- **second_idx** (*int or Bip32KeyIndex object*) – Second index

Returns

IPublicKey object

Return type

IPublicKey object

Raises

- **Bip32KeyError** – If the derivation results in an invalid key
- **Bip32PathError** – If the path indexes are not valid

GetAddress(*first_idx: Union[int, Bip32KeyIndex]*, *second_idx: Union[int, Bip32KeyIndex]*) → str

Get the address with the specified indexes. Derivation path: m/first_idx'/second_idx' The indexes will be automatically hardened if not (e.g. 0, 1' -> 0', 1').

Parameters

- **first_idx** (*int or Bip32KeyIndex object*) – First index
- **second_idx** (*int or Bip32KeyIndex object*) – Second index

Returns

Address

Return type

str

Raises

- **Bip32KeyError** – If the derivation results in an invalid key
- **Bip32PathError** – If the path indexes are not valid

10.1.7.3 cip1852

10.1.7.3.1 cip1852

Module for CIP-1852 keys derivation. Reference: <https://cips.cardano.org/cips/cip1852>

class Cip1852Const

Bases: object

Class container for CIP-1852 constants.

SPEC_NAME: str = 'CIP-1852'**PURPOSE: int = 2147485500****class Cip1852**(*bip32_obj: Bip32Base*, *coin_conf: BipCoinConf*)Bases: **Bip44Base**

CIP-1852 class. It allows master key generation and children keys derivation in according to CIP-1852.

classmethod FromSeed(*seed_bytes: bytes*, *coin_type: BipCoins*) → **Bip44Base**

Create a Bip44Base object from the specified seed (e.g. BIP39 seed).

Parameters

- **seed_bytes** (*bytes*) – Seed bytes
- **coin_type** (**BipCoins**) – Coin type, shall be a Cip1852Coins enum

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Cip1852Coins enum
- **ValueError** – If the seed is too short
- **Bip32KeyError** – If the seed is not suitable for master key generation

classmethod FromExtendedKey(*ex_key_str*: str, *coin_type*: BipCoins) → Bip44Base

Create a Bip44Base object from the specified extended key.

Parameters

- **ex_key_str** (str) – Extended key string
- **coin_type** (BipCoins) – Coin type, shall be a Cip1852Coins enum

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Cip1852Coins enum
- **Bip32KeyError** – If the extended key is not valid

classmethod FromPrivateKey(*priv_key*: ~typing.Union[bytes, ~bip_utils.ecc.common.ikeys.IPrivateKey],
coin_type: ~bip_utils.bip.conf.common.bip_coins.BipCoins, *key_data*:
~bip_utils.bip.bip32.bip32_key_data.Bip32KeyData =
<bip_utils.bip.bip32.bip32_key_data.Bip32KeyData object>) →
Bip44Base

Create a Bip44Base object from the specified private key and derivation data. If only the private key bytes are specified, the key will be considered a master key with the chain code set to zero, since there is no way to recover the key derivation data.

Parameters

- **priv_key** (bytes or IPrivateKey) – Private key
- **coin_type** (BipCoins) – Coin type, shall be a Cip1852Coins enum
- **key_data** (Bip32KeyData object, optional) – Key data (default: all zeros)

Returns

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If coin type is not a Cip1852Coins enum
- **Bip32KeyError** – If the key is not valid

classmethod FromPublicKey(*pub_key*: ~typing.Union[bytes, ~bip_utils.ecc.common.ikeys.IPublicKey],
coin_type: ~bip_utils.bip.conf.common.bip_coins.BipCoins, *key_data*:
~bip_utils.bip.bip32.bip32_key_data.Bip32KeyData =
<bip_utils.bip.bip32.bip32_key_data.Bip32KeyData object>) → Bip44Base

Create a `Bip44Base` object from the specified public key and derivation data. If only the public key bytes are specified, the key will be considered an account key with the chain code set to zero, since there is no way to recover the key derivation data.

Parameters

- **pub_key** (*bytes or `IPublicKey`*) – Public key
- **coin_type** (`BipCoins`) – Coin type, shall be a `Cip1852Coins` enum
- **key_data** (*`Bip32KeyData` object, optional*) – Key data (default: all zeros with account depth)

Returns

`Bip44Base` object

Return type

`Bip44Base` object

Raises

- **`TypeError`** – If coin type is not a `Cip1852Coins` enum
- **`Bip32KeyError`** – If the key is not valid

Purpose() → *`Bip44Base`*

Derive a child key from the purpose and return a new `Bip44Base` object.

Returns

`Bip44Base` object

Return type

`Bip44Base` object

Raises

- **`Bip44DepthError`** – If current depth is not suitable for deriving keys
- **`Bip32KeyError`** – If the derivation results in an invalid key

Coin() → *`Bip44Base`*

Derive a child key from the coin type specified at construction and return a new `Bip44Base` object.

Returns

`Bip44Base` object

Return type

`Bip44Base` object

Raises

- **`Bip44DepthError`** – If current depth is not suitable for deriving keys
- **`Bip32KeyError`** – If the derivation results in an invalid key

Account(*acc_idx: int*) → *`Bip44Base`*

Derive a child key from the specified account index and return a new `Bip44Base` object.

Parameters

acc_idx (*int*) – Account index

Returns

`Bip44Base` object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

Change(*change_type*: Bip44Changes) → Bip44Base

Derive a child key from the specified change type and return a new Bip44Base object.

Parameters**change_type** (Bip44Changes) – Change type, must a Bip44Changes enum**Returns**

Bip44Base object

Return type

Bip44Base object

Raises

- **TypeError** – If change type is not a Bip44Changes enum
- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

AddressIndex(*addr_idx*: int) → Bip44Base

Derive a child key from the specified address index and return a new Bip44Base object.

Parameters**addr_idx** (int) – Address index**Returns**

Bip44Base object

Return type

Bip44Base object

Raises

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

static SpecName() → str

Get specification name.

Returns

Specification name

Return type

str

m_bip32_obj: Bip32Base**m_coin_conf**: BipCoinConf

10.1.7.3.2 conf

10.1.7.3.2.1 cip1852_coins

Module for CIP-1852 coins enum.

```
class Cip1852Coins(value)
    Bases: BipCoins
    Enumerative for supported CIP-1852 coins.
    CARDANO_ICARUS = 1
    CARDANO_LEDGER = 2
    CARDANO_ICARUS_TESTNET = 3
    CARDANO_LEDGER_TESTNET = 4
```

10.1.7.3.2.2 cip1852_conf

Module for CIP-1852 coins configuration.

```
class Cip1852Conf
    Bases: object
    Class container for CIP-1852 configuration.
    CardanoIcarusMainNet: BipCoinConf =
    <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>
    CardanoIcarusTestNet: BipCoinConf =
    <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>
    CardanoLedgerMainNet: BipCoinConf =
    <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>
    CardanoLedgerTestNet: BipCoinConf =
    <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>
```

10.1.7.3.2.3 cip1852_conf_getter

Module for getting CIP-1852 coins configuration.

```
class Cip1852ConfGetterConst
    Bases: object
    Class container for CIP-1852 configuration getter constants.
    COIN_TO_CONF: Dict[BipCoins, BipCoinConf] = {<Cip1852Coins.CARDANO_ICARUS: 1>:
    <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
    <Cip1852Coins.CARDANO_LEDGER: 2>:
    <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
    <Cip1852Coins.CARDANO_ICARUS_TESTNET: 3>:
    <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>,
    <Cip1852Coins.CARDANO_LEDGER_TESTNET: 4>:
    <bip_utils.bip.conf.common.bip_coin_conf.BipCoinConf object>}
```

class Cip1852ConfGetter

Bases: object

CIP-1852 configuration getter class. It allows to get the CIP-1852 configuration of a specific coin.

static **GetConfig**(*coin_type*: BipCoins) → BipCoinConf

Get coin configuration.

Parameters

coin_type (BipCoins) – Coin type

Returns

Coin configuration

Return type

BipCoinConf

Raises

TypeError – If coin type is not of a Cip1852Coins enumerative

10.1.7.4 mnemonic**10.1.7.4.1 cardano_byron_legacy_seed_generator**

Module for Cardano Byron legacy mnemonic seed generation (old Daedalus version).

class CardanoByronLegacySeedGenerator(*mnemonic*: Union[str, Mnemonic], *lang*: Optional[Bip39Languages] = None)

Bases: object

Cardano Byron legacy seed generator class. It generates seeds from a BIP39 mnemonic for Cardano Byron (legacy).

m_ser_seed_bytes: bytes

Generate() → bytes

Generate seed. The seed is simply the entropy bytes in Cardano case. There is no really need of this method, since the seed is always the same, but it's kept in this way to have the same usage of Bip39/Substrate seed generator (i.e. CardanoSeedGenerator(mnemonic).Generate()).

Returns

Generated seed

Return type

bytes

10.1.7.4.2 cardano_icarus_seed_generator

Module for Cardano Icarus mnemonic seed generation.

class CardanoIcarusSeedGenerator(*mnemonic: Union[str, Mnemonic], lang: Optional[Bip39Languages] = None*)

Bases: object

Cardano Icarus seed generator class. It generates seeds from a BIP39 mnemonic for Cardano Icarus.

m_entropy_bytes: bytes

Generate() → bytes

Generate seed. The seed is simply the entropy bytes in Cardano case. There is no really need of this method, since the seed is always the same, but it's kept in this way to have the same usage of Bip39/Substrate seed generator (i.e. CardanoSeedGenerator(mnemonic).Generate()).

Returns

Generated seed

Return type

bytes

10.1.7.5 shelley

10.1.7.5.1 cardano_shelley

Module for Cardano Shelley keys derivation. Reference: <https://cips.cardano.org/cips/cip11>

class CardanoShelley(*bip_obj: Bip44Base, bip_sk_obj: Bip44Base*)

Bases: object

Cardano Shelley class. It allows keys derivation and addresses computation (including the staking one) in according to Cardano Shelley.

classmethod FromCip18520bject(*bip_obj: Bip44Base*) → *CardanoShelley*

Create a CardanoShelley object from the specified Cip1852 object.

Parameters

bip_obj (*Bip44Base object*) – Bip44Base object

Returns

CardanoShelley object

Return type

CardanoShelley object

Raises

- **ValueError** – If the seed is too short
- **Bip32KeyError** – If the seed is not suitable for master key generation

m_bip_obj: *Bip44Base*

m_bip_sk_obj: *Bip44Base*

PublicKeys() → *CardanoShelleyPublicKeys*

Return the public keys.

Returns

CardanoShelleyPublicKeys object

Return type

CardanoShelleyPublicKeys object

PrivateKeys() → *CardanoShelleyPrivateKeys*

Return the private keys.

Returns

CardanoShelleyPrivateKeys object

Return type

CardanoShelleyPrivateKeys object

Raises

Bip32KeyError – If the Bip32 object is public-only

RewardObject() → *Bip44Base*

Alias for StakingObject.

Returns

Bip44Base object

Return type

Bip44Base object

StakingObject() → *Bip44Base*

Return the staking object.

Returns

Bip44Base object

Return type

Bip44Base object

IsPublicOnly() → bool

Get if it's public-only.

Returns

True if public-only, false otherwise

Return type

bool

Change(*change_type*: *Bip44Changes*) → *CardanoShelley*

Derive a child key from the specified change type and return a new CardanoShelley object.

Parameters

change_type (*Bip44Changes*) – Change type, must a Bip44Changes enum

Returns

CardanoShelley object

Return type

CardanoShelley object

Raises

- **TypeError** – If change type is not a Bip44Changes enum

- **Bip44DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

AddressIndex(*addr_idx: int*) → *CardanoShelley*

Derive a child key from the specified address index and return a new CardanoShelley object.

Parameters

addr_idx (*int*) – Address index

Returns

CardanoShelley object

Return type

CardanoShelley object

Raises

- **Cip1852DepthError** – If current depth is not suitable for deriving keys
- **Bip32KeyError** – If the derivation results in an invalid key

10.1.7.5.2 cardano_shelley_keys

Module for Cardano Shelley keys handling.

```
class CardanoShelleyPublicKeys(pub_addr_key: Bip32PublicKey, pub_sk_key: Bip32PublicKey, coin_conf:  
                               BipCoinConf)
```

Bases: object

Cardano Shelley public key class. It contains 2 CIP-1852 public keys (address + staking) and allows to get the Cardano Shelley address from them.

m_pub_addr_key: *Bip32PublicKey*

m_pub_sk_key: *Bip32PublicKey*

m_coin_conf: *BipCoinConf*

AddressKey() → *Bip32PublicKey*

Get the address public key.

Returns

Bip32PublicKey object

Return type

Bip32PublicKey object

RewardKey() → *Bip32PublicKey*

Alias for StakingKey.

Returns

Bip32PublicKey object

Return type

Bip32PublicKey object

StakingKey() → *Bip32PublicKey*

Get the staking address public key.

Returns

Bip32PublicKey object

Return type

Bip32PublicKey object

ToRewardAddress() → str

Alias for ToStakingAddress.

Returns

Reward address string

Return type

str

Raises**ValueError** – If the public key is not correspondent to an address index level**ToStakingAddress()** → str

Return the staking address correspondent to the public key.

Returns

Staking address string

Return type

str

Raises**ValueError** – If the public key is not correspondent to an address index level**ToAddress()** → str

Return the address correspondent to the public key.

Returns

Address string

Return type

str

Raises**ValueError** – If the public key is not correspondent to an address index level

```
class CardanoShelleyPrivateKeys(priv_addr_key: Bip32PrivateKey, priv_sk_key: Bip32PrivateKey,
                                coin_conf: BipCoinConf)
```

Bases: object

Cardano Shelley private key class. It contains 2 BIP32 private keys (address + staking).

m_priv_addr_key: *Bip32PrivateKey***m_priv_sk_key:** *Bip32PrivateKey***m_coin_conf:** *BipCoinConf***AddressKey()** → *Bip32PrivateKey*

Get the address private key.

Returns

Bip32PrivateKey object

Return type

Bip32PrivateKey object

RewardKey() → *Bip32PrivateKey*

Alias for StakingKey.

Returns

Bip32PrivateKey object

Return type

Bip32PrivateKey object

StakingKey() → *Bip32PrivateKey*

Get the staking address private key.

Returns

Bip32PrivateKey object

Return type

Bip32PrivateKey object

PublicKeys() → *CardanoShelleyPublicKeys*

Get the public keys correspondent to the private ones.

Returns

CardanoShelleyPublicKeys object

Return type

CardanoShelleyPublicKeys object

10.1.8 coin_conf

10.1.8.1 coin_conf

Module with helper class for generic coins configuration handling.

class CoinConf(*coin_name*: *CoinNames*, *params*: *Dict[str, Any]*)

Bases: object

Coin configuration class.

m_coin_name: *CoinNames*

m_params: *Dict[str, Any]*

CoinNames() → *CoinNames*

Get coin names.

Returns

CoinNames object

Return type

CoinNames object

ParamByKey(*key*: *str*) → *Any*

Get the parameter by key.

Parameters

key (*str*) – Parameter key

Returns

Parameter value

Return type

Any

10.1.8.2 coins_conf

Module with generic coins configuration for all other modules.

class CoinsConf

Bases: object

Class container for coins configuration.

Acala: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

AkashNetwork: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

Algorand: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

Aptos: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

Arbitrum: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

AvaxCChain: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

AvaxPChain: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

AvaxXChain: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

Axelar: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

BandProtocol: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

Bifrost: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

BinanceChain: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

BinanceSmartChain: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

BitcoinMainNet: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

BitcoinTestNet: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

BitcoinRegTest: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

BitcoinCashMainNet: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

BitcoinCashTestNet: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

BitcoinCashSlpMainNet: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

BitcoinCashSlpTestNet: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

BitcoinSvMainNet: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

BitcoinSvTestNet: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

CardanoMainNet: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

CardanoTestNet: *CoinConf* = <bip_utils.coin_conf.coin_conf.CoinConf object>

```
Celo: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Certik: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
ChainX: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Chihuahua: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Cosmos: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
DashMainNet: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
DashTestNet: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
DogecoinMainNet: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
DogecoinTestNet: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
EcashMainNet: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
EcashTestNet: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Edgware: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Elrond: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Eos: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
ErgoMainNet: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
ErgoTestNet: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Ethereum: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
EthereumClassic: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
FantomOpera: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
FetchAi: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Filecoin: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
GenericSubstrate: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
HarmonyOne: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
HuobiChain: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Icon: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Injective: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
IrisNet: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Karura: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Kava: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Kusama: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
LitecoinMainNet: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
```

```

LitecoinTestNet: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Metis: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
MoneroMainNet: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
MoneroStageNet: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
MoneroTestNet: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Moonbeam: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Moonriver: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Nano: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
NearProtocol: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Neo: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
NineChroniclesGold: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
OkexChain: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Ontology: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Optimism: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Osmosis: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Phala: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
PiNetwork: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Plasm: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Polkadot: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Polygon: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Ripple: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
SecretNetwork: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Solana: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Sora: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Stafi: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Stellar: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Sui: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Terra: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Tezos: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Theta: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Tron: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>

```

```
VeChain: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Verge: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
ZcashMainNet: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
ZcashTestNet: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
Zilliqa: CoinConf = <bip_utils.coin_conf.coin_conf.CoinConf object>
```

10.1.9 ecc

10.1.9.1 common

10.1.9.1.1 dummy_point

Module with helper class for representing a dummy point.

```
class DummyPointConst
    Bases: object
    Class container for dummy point constants.
    POINT_COORD_BYTE_LEN: int = 32

class DummyPoint(point_obj: Any)
    Bases: IPoint, ABC
    Dummy point class.
    classmethod FromBytes(point_bytes: bytes) → IPoint
        Construct class from point bytes.
        Parameters
            point_bytes (bytes) – Point bytes
        Returns
            IPoint object
        Return type
            IPoint
    classmethod FromCoordinates(x: int, y: int) → IPoint
        Construct class from point coordinates.
        Parameters
            • x (int) – X coordinate of the point
            • y (int) – Y coordinate of the point
        Returns
            IPoint object
        Return type
            IPoint
    m_x: int
    m_y: int
```

static CoordinateLength() → int

Get the coordinate length.

Returns

Coordinate key length

Return type

int

UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

X() → int

Get point X coordinate.

Returns

Point X coordinate

Return type

int

Y() → int

Get point Y coordinate.

Returns

Point Y coordinate

Return type

int

Raw() → *DataBytes*

Return the point raw bytes.

Returns

DataBytes object

Return type

DataBytes object

RawEncoded() → *DataBytes*

Return the encoded point raw bytes.

Returns

DataBytes object

Return type

DataBytes object

RawDecoded() → *DataBytes*

Return the decoded point raw bytes.

Returns

DataBytes object

Return type

DataBytes object

__add__(*point: IPoint*) → *IPoint*

Add point to another point.

Parameters

point (*IPoint object*) – IPoint object

Returns

IPoint object

Return type

IPoint object

__radd__(*point: IPoint*) → *IPoint*

Add point to another point.

Parameters

point (*IPoint object*) – IPoint object

Returns

IPoint object

Return type

IPoint object

__mul__(*scalar: int*) → *IPoint*

Multiply point by a scalar.

Parameters

scalar (*int*) – scalar

Returns

IPoint object

Return type

IPoint object

__rmul__(*scalar: int*) → *IPoint*

Multiply point by a scalar.

Parameters

scalar (*int*) – scalar

Returns

IPoint object

Return type

IPoint object

10.1.9.1.2 ikeys

Module with interfaces for public/private keys classes.

class IPublicKey

Bases: ABC

Interface for a generic elliptic curve public key. Verify method is missing because not needed.

abstract classmethod FromBytes(*key_bytes: bytes*) → *IPublicKey*

Construct class from key bytes.

Parameters**key_bytes** (*bytes*) – Key bytes**Returns**

IPublicKey object

Return type*IPublicKey***Raises****ValueError** – If key bytes are not valid**abstract classmethod FromPoint**(*key_point: IPoint*) → *IPublicKey*

Construct class from key point.

Parameters**key_point** (*IPoint object*) – Key point**Returns**

IPublicKey object

Return type*IPublicKey***Raises****ValueError** – If key point is not valid**abstract static CurveType**() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type*EllipticCurveTypes***classmethod IsValidBytes**(*key_bytes: bytes*) → bool

Return if the specified bytes represents a valid public key.

Parameters**key_bytes** (*bytes*) – Key bytes**Returns**

True if valid, false otherwise

Return type

bool

classmethod IsValidPoint(*key_point: IPoint*) → bool

Return if the specified point represents a valid public key.

Parameters**key_point** (*IPoint object*) – Key point**Returns**

True if valid, false otherwise

Return type

bool

abstract static CompressedLength() → int

Get the compressed key length.

Returns

Compressed key length

Return type

int

abstract static UncompressedLength() → int

Get the uncompressed key length.

Returns

Uncompressed key length

Return type

int

abstract UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

abstract RawCompressed() → *DataBytes*

Return raw compressed public key.

Returns

DataBytes object

Return type

DataBytes object

abstract RawUncompressed() → *DataBytes*

Return raw uncompressed public key.

Returns

DataBytes object

Return type

DataBytes object

abstract Point() → *IPoint*

Return the public key point.

Returns

IPoint object

Return type

IPoint object

class IPrivateKey

Bases: ABC

Interface for a generic elliptic curve private key. Sign method is missing because not needed.

abstract classmethod FromBytes(*key_bytes: bytes*) → *IPrivateKey*

Construct class from key bytes.

Parameters**key_bytes** (*bytes*) – Key bytes

Returns

IPrivateKey object

Return type

IPrivateKey

Raises

ValueError – If key bytes are not valid

abstract static CurveType() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type

EllipticCurveTypes

classmethod IsValidBytes(key_bytes: bytes) → bool

Return if the specified bytes represent a valid private key.

Parameters

key_bytes (bytes) – key bytes

Returns

True if valid, false otherwise

Return type

bool

abstract static Length() → int

Get the key length.

Returns

Key length

Return type

int

abstract UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

abstract Raw() → *DataBytes*

Return raw private key.

Returns

DataBytes object

Return type

DataBytes object

abstract PublicKey() → *IPublicKey*

Get the public key correspondent to the private one.

Returns

IPublicKey object

Return type

IPublicKey object

10.1.9.1.3 ipoint

Module with interfaces for point classes.

class IPoint

Bases: ABC

Interface for a generic elliptic curve point.

abstract classmethod FromBytes(*point_bytes: bytes*) → *IPoint*

Construct class from point bytes.

Parameters

point_bytes (*bytes*) – Point bytes

Returns

IPoint object

Return type

IPoint

abstract classmethod FromCoordinates(*x: int, y: int*) → *IPoint*

Construct class from point coordinates.

Parameters

- **x** (*int*) – X coordinate of the point
- **y** (*int*) – Y coordinate of the point

Returns

IPoint object

Return type

IPoint

abstract static CurveType() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type

EllipticCurveTypes

abstract static CoordinateLength() → int

Get the coordinate length.

Returns

Coordinate key length

Return type

int

abstract UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

abstract X() → int

Return X coordinate of the point.

Returns

X coordinate of the point

Return type

int

abstract Y() → int

Return Y coordinate of the point.

Returns

Y coordinate of the point

Return type

int

abstract Raw() → *DataBytes*

Return the point raw bytes.

Returns

DataBytes object

Return type

DataBytes object

abstract RawEncoded() → *DataBytes*

Return the encoded point raw bytes.

Returns

DataBytes object

Return type

DataBytes object

abstract RawDecoded() → *DataBytes*

Return the decoded point raw bytes.

Returns

DataBytes object

Return type

DataBytes object

abstract __add__(point: *IPoint*) → *IPoint*

Add point to another point.

Parameters**point** (*IPoint* object) – *IPoint* object**Returns***IPoint* object**Return type***IPoint* object

abstract `__radd__`(*point*: *IPoint*) → *IPoint*

Add point to another point.

Parameters

point (*IPoint* *object*) – *IPoint* object

Returns

IPoint object

Return type

IPoint object

abstract `__mul__`(*scalar*: *int*) → *IPoint*

Multiply point by a scalar.

Parameters

scalar (*int*) – scalar

Returns

IPoint object

Return type

IPoint object

abstract `__rmul__`(*scalar*: *int*) → *IPoint*

Multiply point by a scalar.

Parameters

scalar (*int*) – scalar

Returns

IPoint object

Return type

IPoint object

10.1.9.2 conf

Module for ECC configuration.

class `EccConf`

Bases: `object`

ECC configuration class.

USE_COINCURVE: `bool` = `True`

10.1.9.3 curve

10.1.9.3.1 elliptic_curve

Module with helper class for elliptic curves.

class `EllipticCurve`(*name*: *str*, *order*: *int*, *generator*: *IPoint*, *point_cls*: *Type*[*IPoint*], *pub_key_cls*: *Type*[*IPublicKey*], *priv_key_cls*: *Type*[*IPrivateKey*])

Bases: `object`

Class for a generic elliptic curve. This is not meant to be complete but just the minimum required to abstract the bip module from the specific ECC library.

```

m_name:  str
m_order:  int
m_generator:  IPoint
m_point_cls:  Type[IPoint]
m_pub_key_cls:  Type[IPublicKey]
m_priv_key_cls:  Type[IPrivateKey]

Name() → str
    Return the curve name.

    Returns
        Curve name

    Return type
        str

Order() → int
    Return the curve order.

    Returns
        Curve order

    Return type
        int

Generator() → IPoint
    Get the curve generator point.

    Returns
        IPoint object

    Return type
        IPoint object

PointClass() → Type[IPoint]
    Return the point class.

    Returns
        Point class

    Return type
        IPoint class

PublicKeyClass() → Type[IPublicKey]
    Return the public key class.

    Returns
        Public key class

    Return type
        IPublicKey class

PrivateKeyClass() → Type[IPrivateKey]
    Return the private key class.

    Returns
        Private key class

```

Return type
PrivateKey class

10.1.9.3.2 elliptic_curve_getter

Module for getting elliptic curves classes.

class EllipticCurveGetterConst

Bases: object

Class container for elliptic curve getter constants.

```
TYPE_TO_INSTANCE: Dict[EllipticCurveTypes, EllipticCurve] =
{<EllipticCurveTypes.ED25519: 1>:
<bip_utils.ecc.curve.elliptic_curve.EllipticCurve object>,
<EllipticCurveTypes.ED25519_BLAKE2B: 2>:
<bip_utils.ecc.curve.elliptic_curve.EllipticCurve object>,
<EllipticCurveTypes.ED25519_KHOLAW: 3>:
<bip_utils.ecc.curve.elliptic_curve.EllipticCurve object>,
<EllipticCurveTypes.ED25519_MONERO: 4>:
<bip_utils.ecc.curve.elliptic_curve.EllipticCurve object>,
<EllipticCurveTypes.NIST256P1: 5>:
<bip_utils.ecc.curve.elliptic_curve.EllipticCurve object>,
<EllipticCurveTypes.SECP256K1: 6>:
<bip_utils.ecc.curve.elliptic_curve.EllipticCurve object>,
<EllipticCurveTypes.SR25519: 7>: <bip_utils.ecc.curve.elliptic_curve.EllipticCurve
object>}
```

class EllipticCurveGetter

Bases: object

Elliptic curve getter class. It allows to get the elliptic curve class from its type.

static FromType(*curve_type*: *EllipticCurveTypes*) → *EllipticCurve*

Get the elliptic curve class from its type.

Parameters

curve_type (*EllipticCurveTypes*) – Curve type

Returns

EllipticCurve object

Return type

EllipticCurve object

Raises

TypeError – If curve type is not a EllipticCurveTypes enum

10.1.9.3.3 elliptic_curve_types

Module for elliptic curves enum.

class EllipticCurveTypes(*value*)

Bases: Enum

Enumerative for elliptic curve types.

ED25519 = 1

ED25519_BLAKE2B = 2

ED25519_KHOLAW = 3

ED25519_MONERO = 4

NIST256P1 = 5

SECP256K1 = 6

SR25519 = 7

10.1.9.4 ecdsa

10.1.9.4.1 ecdsa_keys

Module with some ECDSA keys constants.

class EcdsaKeysConst

Bases: object

Class container for ECDSA keys constants.

POINT_COORD_BYTE_LEN: int = 32

PRIV_KEY_BYTE_LEN: int = 32

PUB_KEY_UNCOMPRESSED_PREFIX: bytes = b'\x04'

PUB_KEY_COMPRESSED_BYTE_LEN: int = 33

PUB_KEY_UNCOMPRESSED_BYTE_LEN: int = 65

10.1.9.5 ed25519

10.1.9.5.1 ed25519

Module with ed25519 curve.

10.1.9.5.2 ed25519_const

Module with ed25519 constants.

class **Ed25519Const**

Bases: object

Class container for Ed25519 constants.

NAME: str = 'Ed25519'

CURVE_ORDER: int =

7237005577332262213973186563042994240857116359379907606001950938285454250989

GENERATOR: *IPoint* = <bip_utils.ecc.ed25519.ed25519_point.Ed25519Point object>

10.1.9.5.3 ed25519_keys

Module for ed25519 keys.

class **Ed25519KeysConst**

Bases: object

Class container for ed25519 keys constants.

PUB_KEY_PREFIX: bytes = b'\x00'

PUB_KEY_BYTE_LEN: int = 32

PRIV_KEY_BYTE_LEN: int = 32

class **Ed25519PublicKey**(key_obj: *VerifyKey*)

Bases: *IPublicKey*

Ed25519 public key class.

classmethod **FromBytes**(key_bytes: bytes) → *IPublicKey*

Construct class from key bytes.

Parameters

key_bytes (bytes) – Key bytes

Returns

IPublicKey object

Return type

IPublicKey

Raises

ValueError – If key bytes are not valid

classmethod **FromPoint**(key_point: *IPoint*) → *IPublicKey*

Construct class from key point.

Parameters

key_point (*IPoint* object) – Key point

Returns

IPublicKey object

Return type*IPublicKey***Raises****ValueError** – If key point is not valid**m_ver_key:** **VerifyKey****static CurveType()** → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type*EllipticCurveTypes***static CompressedLength()** → int

Get the compressed key length.

Returns

Compressed key length

Return type

int

static UncompressedLength() → int

Get the uncompressed key length.

Returns

Uncompressed key length

Return type

int

UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

RawCompressed() → *DataBytes*

Return raw compressed public key.

Returns

DataBytes object

Return type

DataBytes object

RawUncompressed() → *DataBytes*

Return raw uncompressed public key.

Returns

DataBytes object

Return type

DataBytes object

Point() → *IPoint*

Get public key point.

Returns

IPoint object

Return type

IPoint object

class Ed25519PrivateKey(key_obj: *SigningKey*)

Bases: *IPrivateKey*

Ed25519 private key class.

classmethod FromBytes(key_bytes: *bytes*) → *IPrivateKey*

Construct class from key bytes.

Parameters

key_bytes (*bytes*) – Key bytes

Returns

IPrivateKey object

Return type

IPrivateKey

Raises

ValueError – If key bytes are not valid

m_sign_key: *SigningKey*

static CurveType() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type

EllipticCurveTypes

static Length() → int

Get the key length.

Returns

Key length

Return type

int

UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

Raw() → *DataBytes*

Return raw private key.

Returns

DataBytes object

Return type

DataBytes object

PublicKey() → *IPublicKey*

Get the public key correspondent to the private one.

Returns

IPublicKey object

Return type

IPublicKey object

10.1.9.5.4 ed25519_point

Module for ed25519 point.

class Ed25519PointConst

Bases: object

Class container for ed25519 point constants.

POINT_COORD_BYTE_LEN: int = 32**class Ed25519Point**(*point_bytes: bytes*)Bases: *IPoint*

Ed25519 point class.

classmethod FromBytes(*point_bytes: bytes*) → *IPoint*

Construct class from point bytes.

Parameters**point_bytes** (*bytes*) – Point bytes**Returns**

IPoint object

Return type*IPoint***classmethod FromCoordinates**(*x: int, y: int*) → *IPoint*

Construct class from point coordinates.

Parameters

- **x** (*int*) – X coordinate of the point
- **y** (*int*) – Y coordinate of the point

Returns

IPoint object

Return type*IPoint***m_enc_bytes:** bytes**m_is_generator:** bool**m_x:** Optional[int]

m_y: Optional[int]

static CurveType() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type

EllipticCurveTypes

static CoordinateLength() → int

Get the coordinate length.

Returns

Coordinate key length

Return type

int

UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

X() → int

Get point X coordinate.

Returns

Point X coordinate

Return type

int

Y() → int

Get point Y coordinate.

Returns

Point Y coordinate

Return type

int

Raw() → *DataBytes*

Return the point encoded to raw bytes.

Returns

DataBytes object

Return type

DataBytes object

RawEncoded() → *DataBytes*

Return the encoded point raw bytes.

Returns

DataBytes object

Return type

DataBytes object

RawDecoded() → *DataBytes*

Return the decoded point raw bytes.

Returns

DataBytes object

Return type

DataBytes object

__add__(*point: IPoint*) → *IPoint*

Add point to another point.

Parameters**point** (*IPoint object*) – IPoint object**Returns**

IPoint object

Return type

IPoint object

__radd__(*point: IPoint*) → *IPoint*

Add point to another point.

Parameters**point** (*IPoint object*) – IPoint object**Returns**

IPoint object

Return type

IPoint object

__mul__(*scalar: int*) → *IPoint*

Multiply point by a scalar.

Parameters**scalar** (*int*) – scalar**Returns**

IPoint object

Return type

IPoint object

__rmul__(*scalar: int*) → *IPoint*

Multiply point by a scalar.

Parameters**scalar** (*int*) – scalar**Returns**

IPoint object

Return type

IPoint object

10.1.9.5.5 ed25519_utils

Module for ed25519 utility functions.

class Ed25519Utils

Bases: object

Class container for ed25519 utility functions.

static IntDecode(*int_bytes: bytes*) → int

Decode int from bytes.

Parameters

int_bytes (*bytes*) – Integer bytes

Returns

Decoded integer

Return type

int

static IntEncode(*int_val: int*) → bytes

Encode int to bytes.

Parameters

int_val (*int*) – Integer value

Returns

Encoded integer

Return type

bytes

static ScalarReduce(*scalar: Union[bytes, int]*) → bytes

Convert the specified bytes to integer and return its lowest 32-bytes modulo ed25519-order.

Parameters

scalar (*bytes or int*) – Scalar

Returns

Lowest 32-bytes modulo ed25519-order

Return type

bytes

10.1.9.5.6 lib

10.1.9.5.6.1 ed25519_lib

Helper library for ed25519 point encoding/decoding, which cannot be done with pynacl APIs. Encode/Decode operations copied from: <https://github.com/warner/python-pure25519/blob/master/pure25519/basic.py>

int_decode(*int_bytes: bytes*) → int

Decode int from bytes.

Parameters

int_bytes (*bytes*) – Integer bytes

Returns

Decoded integer

Return type

int

int_encode(*int_val: int*) → bytes

Encode int to bytes.

Parameters**int_val** (*int*) – Integer value**Returns**

Encoded integer

Return type

bytes

point_is_decoded_bytes(*point_bytes: bytes*) → bool

Get if point bytes are in decoded format.

Parameters**point_bytes** (*bytes*) – Point bytes**Returns**

True if in decoded format, false otherwise

Return type

bool

point_is_encoded_bytes(*point_bytes: bytes*) → bool

Get if point bytes are in encoded format.

Parameters**point_bytes** (*bytes*) – Point bytes**Returns**

True if in encoded format, false otherwise

Return type

bool

point_is_valid_bytes(*point_bytes: bytes*) → bool

Get if point bytes are valid.

Parameters**point_bytes** (*bytes*) – Point bytes**Returns**

True if valid, false otherwise

Return type

bool

point_bytes_to_coord(*point_bytes: bytes*) → Tuple[int, int]

Convert point bytes to coordinates.

Parameters**point_bytes** (*bytes*) – Point bytes**Returns**

Point coordinates

Return type

tuple[int, int]

Raises

ValueError – If point bytes are not valid

point_coord_to_bytes(*point_coord*: *Tuple[int, int]*) → bytes

Convert point coordinates to bytes.

Parameters

point_coord (*tuple[int, int]*) – Point coordinates

Returns

Point bytes

Return type

bytes

point_decode_no_check(*point_bytes*: bytes) → *Tuple[int, int]*

Decode point bytes to coordinates without checking if it lies on the ed25519 curve.

Parameters

point_bytes (bytes) – Point bytes

Returns

Point coordinates

Return type

tuple[int, int]

Raises

ValueError – If point bytes are not valid

point_decode(*point_bytes*: bytes) → *Tuple[int, int]*

Decode point bytes to coordinates by checking if it lies on the ed25519 curve.

Parameters

point_bytes (bytes) – Point bytes

Returns

Point coordinates

Return type

tuple[int, int]

Raises

ValueError – If the point bytes are not valid or the decoded point doesn't lie on the curve

point_encode(*point_coord*: *Tuple[int, int]*) → bytes

Encode point coordinates to bytes.

Parameters

point_coord (*tuple[int, int]*) – Point coordinates

Returns

Point bytes

Return type

bytes

point_is_generator(*point*: *Union[bytes, Tuple[int, int]]*) → bool

Get if the point is the generator of the ed25519 curve.

Parameters

point (bytes or *tuple[int, int]*) – Point

Returns

True if generator, false otherwise

Return type

bool

Raises

ValueError – If point bytes are not valid

point_is_on_curve(*point*: Union[bytes, Tuple[int, int]]) → bool

Get if the point lies on the ed25519 curve. This method is used because nacl.bindings.crypto_core_ed25519_is_valid_point performs more strict checks, which results in points (i.e. public keys) that are considered not valid even if they are accepted by wallets.

Parameters

point (bytes or tuple[int, int]) – Point

Returns

True if it lies on the curve, false otherwise

Return type

bool

Raises

ValueError – If point bytes are not valid

point_add(*point_1*: Union[bytes, Tuple[int, int]], *point_2*: Union[bytes, Tuple[int, int]]) → bytes

Add two points on the ed25519 curve.

Parameters

- **point_1** (bytes or tuple[int, int]) – Point 1
- **point_2** (bytes or tuple[int, int]) – Point 2

Returns

New point resulting from the addition

Return type

bytes

point_scalar_mul(*scalar*: Union[bytes, int], *point*: Union[bytes, Tuple[int, int]]) → bytes

Multiply a point on the ed25519 curve with a scalar.

Parameters

- **scalar** (bytes or int) – Scalar
- **point** (bytes or tuple[int, int]) – Point

Returns

New point resulting from the multiplication

Return type

bytes

point_scalar_mul_base(*scalar*: Union[bytes, int]) → bytes

Multiply the base (i.e. generator) point of the ed25519 curve with a scalar.

Parameters

scalar (bytes or int) – Scalar

Returns

New point resulting from the multiplication

Return type

bytes

scalar_reduce(*scalar: Union[bytes, int]*) → bytes

Convert the specified bytes to integer and return its lowest 32-bytes modulo ed25519 curve order.

Parameters**scalar** (*bytes or int*) – Scalar**Returns**

Lowest 32-bytes modulo ed25519-order

Return type

bytes

scalar_is_valid(*scalar: Union[bytes, int]*) → bool

Get if the specified scalar is valid (i.e. less than the ed25519 curve order).

Parameters**scalar** (*bytes or int*) – Scalar**Returns**

True if lower, false otherwise

Return type

bool

10.1.9.6 ed25519_blake2b**10.1.9.6.1 ed25519_blake2b**

Module for ed25519-blake2b curve.

10.1.9.6.2 ed25519_blake2b_const

Module for ed25519-blake2b constants.

class Ed25519Blake2bConst

Bases: object

Class container for Ed25519-Blake2b constants.

NAME: str = 'Ed25519-Blake2b'**CURVE_ORDER:** int =

7237005577332262213973186563042994240857116359379907606001950938285454250989

GENERATOR: *IPoint* = <bip_utils.ecc.ed25519.ed25519_point.Ed25519Point object>

10.1.9.6.3 ed25519_blake2b_keys

Module for ed25519-blake2b keys.

class **Ed25519Blake2bPublicKey**(*key_obj: VerifyingKey*)

Bases: *IPublicKey*

Ed25519-Blake2b public key class.

classmethod **FromBytes**(*key_bytes: bytes*) → *IPublicKey*

Construct class from key bytes.

Parameters

key_bytes (*bytes*) – Key bytes

Returns

IPublicKey object

Return type

IPublicKey

Raises

ValueError – If key bytes are not valid

classmethod **FromPoint**(*key_point: IPoint*) → *IPublicKey*

Construct class from key point.

Parameters

key_point (*IPoint object*) – Key point

Returns

IPublicKey object

Return type

IPublicKey

Raises

ValueError – If key point is not valid

m_ver_key: *VerifyingKey*

static **CurveType**() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type

EllipticCurveTypes

static **CompressedLength**() → int

Get the compressed key length.

Returns

Compressed key length

Return type

int

static **UncompressedLength**() → int

Get the uncompressed key length.

Returns

Uncompressed key length

Return type

int

UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

RawCompressed() → *DataBytes*

Return raw compressed public key.

Returns

DataBytes object

Return type

DataBytes object

RawUncompressed() → *DataBytes*

Return raw uncompressed public key.

Returns

DataBytes object

Return type

DataBytes object

Point() → *IPoint*

Get public key point.

Returns

IPoint object

Return type

IPoint object

class Ed25519Blake2bPrivateKey(key_obj: *SigningKey*)Bases: *IPrivateKey*

Ed25519-Blake2b private key class.

classmethod FromBytes(key_bytes: bytes) → *IPrivateKey*

Construct class from key bytes.

Parameters**key_bytes** (bytes) – Key bytes**Returns**

IPrivateKey object

Return type*IPrivateKey***Raises****ValueError** – If key bytes are not valid

m_sign_key: **SigningKey**

static CurveType() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type

EllipticCurveTypes

static Length() → int

Get the key length.

Returns

Key length

Return type

int

UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

Raw() → *DataBytes*

Return raw private key.

Returns

DataBytes object

Return type

DataBytes object

PublicKey() → *IPublicKey*

Get the public key correspondent to the private one.

Returns

IPublicKey object

Return type

IPublicKey object

10.1.9.6.4 ed25519_blake2b_point

Module for ed25519-blake2b point.

class Ed25519Blake2bPoint(*point_bytes: bytes*)

Bases: *Ed25519Point*

Ed25519-Blake2b point class.

static CurveType() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type

EllipticCurveTypes

m_is_generator: bool

m_enc_bytes: bytes

m_x: Optional[int]

m_y: Optional[int]

10.1.9.7 ed25519_kholaw

10.1.9.7.1 ed25519_kholaw

Module with ed25519-kholaw curve.

10.1.9.7.2 ed25519_kholaw_const

Module with ed25519-kholaw constants.

class Ed25519KholawConst

Bases: object

Class container for Ed25519-Kholaw constants.

NAME: str = 'Ed25519-Kholaw'

CURVE_ORDER: int =

7237005577332262213973186563042994240857116359379907606001950938285454250989

GENERATOR: *IPoint* = <bip_utils.ecc.ed25519.ed25519_point.Ed25519Point object>

10.1.9.7.3 ed25519_kholaw_keys

Module for ed25519-kholaw keys. With respect to ed25519, the private key has a length of 64-byte (left 32-byte of the ed25519 private key and a right 32-byte extension part).

class Ed25519KholawKeysConst

Bases: object

Class container for ed25519-kholaw keys constants.

PRIV_KEY_BYTE_LEN: int = 64

class Ed25519KholawPublicKey(key_obj: VerifyKey)

Bases: *Ed25519PublicKey*

Ed25519-Kholaw public key class.

static CurveType() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type*EllipticCurveTypes***Point()** → *IPoint*

Get public key point.

Returns

IPoint object

Return type

IPoint object

m_ver_key: **VerifyKey****class Ed25519KholawPrivateKey**(*key_obj: IPrivateKey, key_ex_bytes: bytes*)Bases: *IPrivateKey*

Ed25519-Kholaw private key class.

classmethod FromBytes(*key_bytes: bytes*) → *IPrivateKey*

Construct class from key bytes.

Parameters**key_bytes** (*bytes*) – Key bytes**Returns**

IPrivateKey object

Return type*IPrivateKey***Raises****ValueError** – If key bytes are not valid**m_sign_key:** *Ed25519PrivateKey***m_ext_key:** **bytes****static CurveType**() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type*EllipticCurveTypes***static Length**() → **int**

Get the key length.

Returns

Key length

Return type**int****UnderlyingObject**() → **Any**

Get the underlying object.

Returns

Underlying object

Return type

Any

Raw() → *DataBytes*

Return raw private key.

Returns

DataBytes object

Return type

DataBytes object

PublicKey() → *IPublicKey*

Get the public key correspondent to the private one.

Returns

IPublicKey object

Return type

IPublicKey object

10.1.9.7.4 ed25519_kholaw_point

Module for ed25519-kholaw point.

class Ed25519KholawPoint(*point_bytes: bytes*)Bases: *Ed25519Point*

Ed25519-Kholaw point class.

static CurveType() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type*EllipticCurveTypes***m_is_generator:** bool**m_enc_bytes:** bytes**m_x:** Optional[int]**m_y:** Optional[int]**10.1.9.8 ed25519_monero****10.1.9.8.1 ed25519_monero**

Module with ed25519-monero curve.

10.1.9.8.2 ed25519_monero_const

Module with ed25519-monero constants.

class Ed25519MoneroConst

Bases: object

Class container for Ed25519-Monero constants.

NAME: str = 'Ed25519-Monero'

CURVE_ORDER: int =

7237005577332262213973186563042994240857116359379907606001950938285454250989

GENERATOR: *IPoint* =

<bip_utils.ecc.ed25519_monero.ed25519_monero_point.Ed25519MoneroPoint object>

10.1.9.8.3 ed25519_monero_keys

Module for ed25519-monero keys.

class Ed25519MoneroPublicKey(key_obj: VerifyKey)

Bases: *Ed25519PublicKey*

Ed25519-Monero public key class.

static CurveType() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type

EllipticCurveTypes

static CompressedLength() → int

Get the compressed key length.

Returns

Compressed key length

Return type

int

static UncompressedLength() → int

Get the uncompressed key length.

Returns

Uncompressed key length

Return type

int

RawCompressed() → *DataBytes*

Return raw compressed public key.

Returns

DataBytes object

Return type

DataBytes object

Point() → *IPoint*

Get public key point.

Returns

IPoint object

Return type

IPoint object

m_ver_key: **VerifyKey****class Ed25519MoneroPrivateKey**(*key_obj: SigningKey*)Bases: *Ed25519PrivateKey*

Ed25519-Monero private key class.

classmethod FromBytes(*key_bytes: bytes*) → *IPrivateKey*

Construct class from key bytes.

Parameters**key_bytes** (*bytes*) – Key bytes**Returns**

IPrivateKey object

Return type*IPrivateKey***Raises****ValueError** – If key bytes are not valid**static CurveType**() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type*EllipticCurveTypes***PublicKey**() → *IPublicKey*

Get the public key correspondent to the private one.

Returns

IPublicKey object

Return type

IPublicKey object

m_sign_key: **SigningKey**

10.1.9.8.4 ed25519_monero_point

Module for ed25519-monero point.

```
class Ed25519MoneroPoint(point_bytes: bytes)
    Bases: Ed25519Point
    Ed25519-Monero point class.
    static CurveType() → EllipticCurveTypes
        Get the elliptic curve type.
        Returns
            Elliptic curve type
        Return type
            EllipticCurveTypes
    m_is_generator: bool
    m_enc_bytes: bytes
    m_x: Optional[int]
    m_y: Optional[int]
```

10.1.9.9 nist256p1

10.1.9.9.1 nist256p1

Module with nist256p1 curve.

10.1.9.9.2 nist256p1_const

Module with nist256p1 constants.

```
class Nist256p1Const
    Bases: object
    Class container for Nist256p1 constants.
    NAME: str = 'Nist256p1'
    CURVE_ORDER: int =
    115792089210356248762697446949407573529996955224135760342422259061068512044369
    GENERATOR: IPoint = <bip_utils.ecc.nist256p1.nist256p1_point.Nist256p1Point object>
```

10.1.9.9.3 nist256p1_keys

Module for nist256p1 keys.

class Nist256p1PublicKey(key_obj: *VerifyingKey*)

Bases: *IPublicKey*

Nist256p1 public key class.

classmethod FromBytes(key_bytes: *bytes*) → *IPublicKey*

Construct class from key bytes.

Parameters

key_bytes (*bytes*) – Key bytes

Returns

IPublicKey object

Return type

IPublicKey

Raises

ValueError – If key bytes are not valid

classmethod FromPoint(key_point: *IPoint*) → *IPublicKey*

Construct class from key point.

Parameters

key_point (*IPoint* object) – Key point

Returns

IPublicKey object

Return type

IPublicKey

Raises

ValueError – If key point is not valid

m_ver_key: *VerifyingKey*

static CurveType() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type

EllipticCurveTypes

static CompressedLength() → int

Get the compressed key length.

Returns

Compressed key length

Return type

int

static UncompressedLength() → int

Get the uncompressed key length.

Returns

Uncompressed key length

Return type

int

UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

RawCompressed() → *DataBytes*

Return raw compressed public key.

Returns

DataBytes object

Return type

DataBytes object

RawUncompressed() → *DataBytes*

Return raw uncompressed public key.

Returns

DataBytes object

Return type

DataBytes object

Point() → *IPoint*

Get public key point.

Returns

IPoint object

Return type

IPoint object

class Nist256p1PrivateKey(key_obj: *SigningKey*)Bases: *IPrivateKey*

Nist256p1 private key class.

classmethod FromBytes(key_bytes: bytes) → *IPrivateKey*

Construct class from key bytes.

Parameters**key_bytes** (bytes) – Key bytes**Returns**

IPrivateKey object

Return type*IPrivateKey***Raises****ValueError** – If key bytes are not valid

m_sign_key: **SigningKey**

static CurveType() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type

EllipticCurveTypes

static Length() → int

Get the key length.

Returns

Key length

Return type

int

UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

Raw() → *DataBytes*

Return raw private key.

Returns

DataBytes object

Return type

DataBytes object

PublicKey() → *IPublicKey*

Get the public key correspondent to the private one.

Returns

IPublicKey object

Return type

IPublicKey object

10.1.9.9.4 nist256p1_point

Module for nist256p1 point.

class Nist256p1Point(*point_obj: PointJacobi*)

Bases: *IPoint*

Nist256p1 point class.

classmethod FromBytes(*point_bytes: bytes*) → *IPoint*

Construct class from point bytes.

Parameters

point_bytes (*bytes*) – Point bytes

Returns

IPoint object

Return type*IPoint***classmethod FromCoordinates**(*x: int, y: int*) → *IPoint*

Construct class from point coordinates.

Parameters

- **x** (*int*) – X coordinate of the point
- **y** (*int*) – Y coordinate of the point

Returns

IPoint object

Return type*IPoint***m_point: PointJacobi****static CurveType**() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type*EllipticCurveTypes***static CoordinateLength**() → int

Get the coordinate length.

Returns

Coordinate key length

Return type

int

UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

X() → int

Get point X coordinate.

Returns

Point X coordinate

Return type

int

Y() → int

Get point Y coordinate.

Returns

Point Y coordinate

Return type

int

Raw() → *DataBytes*

Return the point raw bytes.

Returns

DataBytes object

Return type

DataBytes object

RawEncoded() → *DataBytes*

Return the encoded point raw bytes.

Returns

DataBytes object

Return type

DataBytes object

RawDecoded() → *DataBytes*

Return the decoded point raw bytes.

Returns

DataBytes object

Return type

DataBytes object

__add__(point: *IPoint*) → *IPoint*

Add point to another point.

Parameters**point** (*IPoint object*) – IPoint object**Returns**

IPoint object

Return type

IPoint object

__radd__(point: *IPoint*) → *IPoint*

Add point to another point.

Parameters**point** (*IPoint object*) – IPoint object**Returns**

IPoint object

Return type

IPoint object

__mul__(scalar: *int*) → *IPoint*

Multiply point by a scalar.

Parameters**scalar** (*int*) – scalar

Returns

IPoint object

Return type

IPoint object

__rmul__(*scalar: int*) → *IPoint*

Multiply point by a scalar.

Parameters**scalar** (*int*) – scalar**Returns**

IPoint object

Return type

IPoint object

10.1.9.10 secp256k1**10.1.9.10.1 secp256k1**

Module with secp256k1 curve.

10.1.9.10.2 secp256k1_const

Module with secp256k1 constants.

class Secp256k1Const

Bases: object

Class container for Secp256k1 constants.

NAME: str = 'Secp256k1'**CURVE_ORDER:** int =

115792089237316195423570985008687907852837564279074904382605163141518161494337

GENERATOR: *IPoint* =

<bip_utils.ecc.secp256k1.secp256k1_point_coincurve.Secp256k1PointCoincurve object>

10.1.9.10.3 secp256k1_keys_coincurve

Module for secp256k1 keys based on coincurve library.

class Secp256k1PublicKeyCoincurve(*key_obj: PublicKey*)Bases: *IPublicKey*

Secp256k1 public key class.

classmethod FromBytes(*key_bytes: bytes*) → *IPublicKey*

Construct class from key bytes.

Parameters**key_bytes** (*bytes*) – Key bytes

Returns

IPublicKey object

Return type*IPublicKey***Raises****ValueError** – If key bytes are not valid**classmethod FromPoint**(key_point: *IPoint*) → *IPublicKey*

Construct class from key point.

Parameters**key_point** (*IPoint* object) – Key point**Returns**

IPublicKey object

Return type*IPublicKey***Raises****ValueError** – If key point is not valid**m_ver_key:** **PublicKey****static CurveType**() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type*EllipticCurveTypes***static CompressedLength**() → int

Get the compressed key length.

Returns

Compressed key length

Return type

int

static UncompressedLength() → int

Get the uncompressed key length.

Returns

Uncompressed key length

Return type

int

UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

RawCompressed() → *DataBytes*

Return raw compressed public key.

Returns

DataBytes object

Return type

DataBytes object

RawUncompressed() → *DataBytes*

Return raw uncompressed public key.

Returns

DataBytes object

Return type

DataBytes object

Point() → *IPoint*

Get public key point.

Returns

IPoint object

Return type

IPoint object

class Secp256k1PrivateKeyCoincurve(key_obj: *PrivateKey*)

Bases: *IPrivateKey*

Secp256k1 private key class.

classmethod FromBytes(key_bytes: *bytes*) → *IPrivateKey*

Construct class from key bytes.

Parameters

key_bytes (*bytes*) – Key bytes

Returns

IPrivateKey object

Return type

IPrivateKey

Raises

ValueError – If key bytes are not valid

m_sign_key: *PrivateKey*

static CurveType() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type

EllipticCurveTypes

static Length() → int

Get the key length.

Returns

Key length

Return type

int

UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

Raw() → *DataBytes*

Return raw private key.

Returns

DataBytes object

Return type

DataBytes object

PublicKey() → *IPublicKey*

Get the public key correspondent to the private one.

Returns

IPublicKey object

Return type

IPublicKey object

10.1.9.10.4 secp256k1_keys_ecdsa

Module for secp256k1 keys based on ecdsa library.

class **Secp256k1PublicKeyEcdsa**(*key_obj: VerifyingKey*)Bases: *IPublicKey*

Secp256k1 public key class.

classmethod **FromBytes**(*key_bytes: bytes*) → *IPublicKey*

Construct class from key bytes.

Parameters**key_bytes** (*bytes*) – Key bytes**Returns**

IPublicKey object

Return type*IPublicKey***Raises****ValueError** – If key bytes are not valid**classmethod** **FromPoint**(*key_point: IPoint*) → *IPublicKey*

Construct class from key point.

Parameters**key_point** (*IPoint object*) – Key point

Returns

IPublicKey object

Return type*IPublicKey***Raises****ValueError** – If key point is not valid**m_ver_key:** **VerifyingKey****static CurveType()** → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type*EllipticCurveTypes***static CompressedLength()** → int

Get the compressed key length.

Returns

Compressed key length

Return type

int

static UncompressedLength() → int

Get the uncompressed key length.

Returns

Uncompressed key length

Return type

int

UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

RawCompressed() → *DataBytes*

Return raw compressed public key.

Returns

DataBytes object

Return type

DataBytes object

RawUncompressed() → *DataBytes*

Return raw uncompressed public key.

Returns

DataBytes object

Return type

DataBytes object

Point() → *IPoint*

Get public key point.

Returns

IPoint object

Return type

IPoint object

class Secp256k1PrivateKeyEcdsa(key_obj: *SigningKey*)Bases: *IPrivateKey*

Secp256k1 private key class.

classmethod FromBytes(key_bytes: *bytes*) → *IPrivateKey*

Construct class from key bytes.

Parameters**key_bytes** (*bytes*) – Key bytes**Returns**

IPrivateKey object

Return type*IPrivateKey***Raises****ValueError** – If key bytes are not valid**m_sign_key**alias of *SigningKey***static CurveType**() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type*EllipticCurveTypes***static Length**() → int

Get the key length.

Returns

Key length

Return type

int

UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

Raw() → *DataBytes*

Return raw private key.

Returns

DataBytes object

Return type

DataBytes object

PublicKey() → *IPublicKey*

Get the public key correspondent to the private one.

Returns

IPublicKey object

Return type

IPublicKey object

10.1.9.10.5 secp256k1_point_coincurve

Module for secp256k1 point based on coincurve library.

class **Secp256k1PointCoincurve**(*point_obj: PublicKey*)

Bases: *IPoint*

Secp256k1 point class. In coincurve library, all the point functions (e.g. add, multiply) are coded inside the PublicKey class. For this reason, a PublicKey is used as underlying object.

classmethod **FromBytes**(*point_bytes: bytes*) → *IPoint*

Construct class from point bytes.

Parameters

point_bytes (*bytes*) – Point bytes

Returns

IPoint object

Return type

IPoint

classmethod **FromCoordinates**(*x: int, y: int*) → *IPoint*

Construct class from point coordinates.

Parameters

- **x** (*int*) – X coordinate of the point
- **y** (*int*) – Y coordinate of the point

Returns

IPoint object

Return type

IPoint

m_pub_key: **PublicKey**

static **CurveType**() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type*EllipticCurveTypes***static CoordinateLength()** → int

Get the coordinate length.

Returns

Coordinate key length

Return type

int

UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

X() → int

Get point X coordinate.

Returns

Point X coordinate

Return type

int

Y() → int

Get point Y coordinate.

Returns

Point Y coordinate

Return type

int

Raw() → *DataBytes*

Return the point raw bytes.

Returns

DataBytes object

Return type

DataBytes object

RawEncoded() → *DataBytes*

Return the encoded point raw bytes.

Returns

DataBytes object

Return type

DataBytes object

RawDecoded() → *DataBytes*

Return the decoded point raw bytes.

Returns

DataBytes object

Return type

DataBytes object

__add__(*point: IPoint*) → *IPoint*

Add point to another point.

Parameters

point (*IPoint object*) – IPoint object

Returns

IPoint object

Return type

IPoint object

__radd__(*point: IPoint*) → *IPoint*

Add point to another point.

Parameters

point (*IPoint object*) – IPoint object

Returns

IPoint object

Return type

IPoint object

__mul__(*scalar: int*) → *IPoint*

Multiply point by a scalar.

Parameters

scalar (*int*) – scalar

Returns

IPoint object

Return type

IPoint object

__rmul__(*scalar: int*) → *IPoint*

Multiply point by a scalar.

Parameters

scalar (*int*) – scalar

Returns

IPoint object

Return type

IPoint object

10.1.9.10.6 secp256k1_point_ecdsa

Module for secp256k1 point based on ecdsa library.

class **Secp256k1PointEcdsa**(*point_obj: PointJacobi*)

Bases: *IPoint*

Secp256k1 point class.

classmethod **FromBytes**(*point_bytes: bytes*) → *IPoint*

Construct class from point bytes.

Parameters

point_bytes (*bytes*) – Point bytes

Returns

IPoint object

Return type

IPoint

classmethod **FromCoordinates**(*x: int, y: int*) → *IPoint*

Construct class from point coordinates.

Parameters

- **x** (*int*) – X coordinate of the point
- **y** (*int*) – Y coordinate of the point

Returns

IPoint object

Return type

IPoint

m_point: *PointJacobi*

static **CurveType**() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type

EllipticCurveTypes

static **CoordinateLength**() → *int*

Get the coordinate length.

Returns

Coordinate key length

Return type

int

UnderlyingObject() → *Any*

Get the underlying object.

Returns

Underlying object

Return type

Any

X() → int

Get point X coordinate.

Returns

Point X coordinate

Return type

int

Y() → int

Get point Y coordinate.

Returns

Point Y coordinate

Return type

int

Raw() → *DataBytes*

Return the point raw bytes.

Returns

DataBytes object

Return type

DataBytes object

RawEncoded() → *DataBytes*

Return the encoded point raw bytes.

Returns

DataBytes object

Return type

DataBytes object

RawDecoded() → *DataBytes*

Return the decoded point raw bytes.

Returns

DataBytes object

Return type

DataBytes object

__add__(*point: IPoint*) → *IPoint*

Add point to another point.

Parameters**point** (*IPoint object*) – IPoint object**Returns**

IPoint object

Return type

IPoint object

__radd__(*point: IPoint*) → *IPoint*

Add point to another point.

Parameters

point (*IPoint object*) – IPoint object

Returns

IPoint object

Return type

IPoint object

__mul__(*scalar: int*) → *IPoint*

Multiply point by a scalar.

Parameters

scalar (*int*) – scalar

Returns

IPoint object

Return type

IPoint object

__rmul__(*scalar: int*) → *IPoint*

Multiply point by a scalar.

Parameters

scalar (*int*) – scalar

Returns

IPoint object

Return type

IPoint object

10.1.9.11 sr25519

10.1.9.11.1 sr25519

Module with sr25519 curve.

10.1.9.11.2 sr25519_const

Module with sr25519 constants.

class Sr25519Const

Bases: object

Class container for Sr25519 constants.

NAME: str = 'Sr25519'

CURVE_ORDER: int = 0

GENERATOR: *IPoint* = <bip_utils.ecc.sr25519.sr25519_point.Sr25519Point object>

10.1.9.11.3 sr25519_keys

Module for sr25519 keys.

class **Sr25519KeysConst**

Bases: object

Class container for ed25519 keys constants.

PUB_KEY_BYTE_LEN: int = 32

PRIV_KEY_BYTE_LEN: int = 64

class **Sr25519PublicKey**(*key_bytes: bytes*)

Bases: *IPublicKey*

Sr25519 public key class.

classmethod **FromBytes**(*key_bytes: bytes*) → *IPublicKey*

Construct class from key bytes.

Parameters

key_bytes (*bytes*) – Key bytes

Returns

IPublicKey object

Return type

IPublicKey

Raises

ValueError – If key bytes are not valid

classmethod **FromPoint**(*key_point: IPoint*) → *IPublicKey*

Construct class from key point.

Parameters

key_point (*IPoint object*) – Key point

Returns

IPublicKey object

Return type

IPublicKey

Raises

ValueError – If key point is not valid

m_ver_key: bytes

static **CurveType**() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type

EllipticCurveTypes

static CompressedLength() → int

Get the compressed key length.

Returns

Compressed key length

Return type

int

static UncompressedLength() → int

Get the uncompressed key length.

Returns

Uncompressed key length

Return type

int

UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

RawCompressed() → *DataBytes*

Return raw compressed public key.

Returns

DataBytes object

Return type

DataBytes object

RawUncompressed() → *DataBytes*

Return raw uncompressed public key.

Returns

DataBytes object

Return type

DataBytes object

Point() → *IPoint*

Get public key point.

Returns

IPoint object

Return type

IPoint object

class Sr25519PrivateKey(key_bytes: bytes)

Bases: *IPrivateKey*

Sr25519 private key class.

classmethod FromBytes(key_bytes: bytes) → *IPrivateKey*

Construct class from key bytes.

Parameters

key_bytes (*bytes*) – Key bytes

Returns

IPrivateKey object

Return type

IPrivateKey

Raises

ValueError – If key bytes are not valid

m_sign_key: **bytes**

static CurveType() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type

EllipticCurveTypes

static Length() → int

Get the key length.

Returns

Key length

Return type

int

UnderlyingObject() → Any

Get the underlying object.

Returns

Underlying object

Return type

Any

Raw() → *DataBytes*

Return raw private key.

Returns

DataBytes object

Return type

DataBytes object

PublicKey() → *IPublicKey*

Get the public key correspondent to the private one.

Returns

IPublicKey object

Return type

IPublicKey object

10.1.9.11.4 sr25519_point

Module for sr25519 point.

class **Sr25519Point**(*point_obj: Any*)

Bases: *DummyPoint*

Sr25519 point class. Dummy class since not needed.

static **CurveType**() → *EllipticCurveTypes*

Get the elliptic curve type.

Returns

Elliptic curve type

Return type

EllipticCurveTypes

m_x: **int**

m_y: **int**

10.1.10 electrum

10.1.10.1 electrum_v1

Module containing utility classes for Electrum v1 keys derivation, since it uses its own algorithm.

class **ElectrumV1**(*priv_key: Optional[PrivateKey], pub_key: Optional[PublicKey]*)

Bases: *object*

Electrum v1 class. It derives keys like the Electrum wallet with old (v1) mnemonic.

classmethod **FromSeed**(*seed_bytes: bytes*) → *ElectrumV1*

Construct class from seed bytes.

Parameters

seed_bytes (*bytes*) – Seed bytes

Returns

ElectrumV1 object

Return type

ElectrumV1 object

classmethod **FromPrivateKey**(*priv_key: Union[bytes, PrivateKey]*) → *ElectrumV1*

Construct class from private key.

Parameters

priv_key (*bytes or PrivateKey*) – Private key

Returns

ElectrumV1 object

Return type

ElectrumV1 object

Raises

TypeError – if the private key is not a *Secp256k1PrivateKey*

classmethod FromPublicKey(*pub_key: Union[bytes, IPublicKey]*) → *ElectrumV1*

Construct class from public key.

Parameters

pub_key (*bytes* or *IPublicKey*) – Public key

Returns

ElectrumV1 object

Return type

ElectrumV1 object

Raises

TypeError – if the public key is not a Secp256k1PublicKey

m_priv_key: *Optional[IPrivateKey]*

m_pub_key: *IPublicKey*

IsPublicOnly() → bool

Get if it's public-only.

Returns

True if public-only, false otherwise

Return type

bool

MasterPrivateKey() → *IPrivateKey*

Get the master private key.

Returns

IPrivateKey object

Return type

IPrivateKey object

MasterPublicKey() → *IPublicKey*

Get the master public key.

Returns

IPublicKey object

Return type

IPublicKey object

GetPrivateKey(*change_idx: int, addr_idx: int*) → *IPrivateKey*

Get the private key with the specified change and address indexes. Derivation path (not BIP32 derivation):
m/change_idx/addr_idx

Parameters

- **change_idx** (*int*) – Change index
- **addr_idx** (*int*) – Address index

Returns

IPrivateKey object

Return type

IPrivateKey object

Raises

ValueError – If one of the index is not valid

GetPublicKey(*change_idx: int, addr_idx: int*) → *IPublicKey*

Get the public key with the specified change and address indexes. Derivation path (not BIP32 derivation):
m/change_idx/addr_idx

Parameters

- **change_idx** (*int*) – Change index
- **addr_idx** (*int*) – Address index

Returns

IPublicKey object

Return type

IPublicKey object

Raises

ValueError – If one of the index is not valid

GetAddress(*change_idx: int, addr_idx: int*) → str

Get the address with the specified change and address indexes. Derivation path (not BIP32 derivation):
m/change_idx/addr_idx

Parameters

- **change_idx** (*int*) – Change index
- **addr_idx** (*int*) – Address index

Returns

Address

Return type

str

Raises

ValueError – If one of the index is not valid

10.1.10.2 electrum_v2

Module containing utility classes for Electrum v2 keys derivation, since it uses its own paths.

class ElectrumV2Base(*bip32_obj: Bip32Base*)

Bases: ABC

Electrum v2 base class.

classmethod FromSeed(*seed_bytes: bytes*) → *ElectrumV2Base*

Construct class from seed bytes.

Parameters

seed_bytes (*bytes*) – Seed bytes

Returns

ElectrumV2Base object

Return type

ElectrumV2Base object

m_bip32_obj: *Bip32Base*

Bip32Object() → *Bip32Base*

Return the BIP32 object.

Returns

Bip32Base object

Return type

Bip32Base object

IsPublicOnly() → bool

Get if it's public-only.

Returns

True if public-only, false otherwise

Return type

bool

MasterPrivateKey() → *Bip32PrivateKey*

Get the master private key.

Returns

Bip32PrivateKey object

Return type

Bip32PrivateKey object

MasterPublicKey() → *Bip32PublicKey*

Get the master public key.

Returns

Bip32PublicKey object

Return type

Bip32PublicKey object

abstract GetPrivateKey(*change_idx: Union[int, Bip32KeyIndex], addr_idx: Union[int, Bip32KeyIndex]*) → *Bip32PrivateKey*

Get the private key with the specified change and address indexes.

Parameters

- **change_idx** (*int* or *Bip32KeyIndex* object) – Change index
- **addr_idx** (*int* or *Bip32KeyIndex* object) – Address index

Returns

Bip32PrivateKey object

Return type

Bip32PrivateKey object

Raises

- **Bip32KeyError** – If the derivation results in an invalid key or the object is public-only
- **Bip32PathError** – If the path indexes are not valid

abstract GetPublicKey(*change_idx: Union[int, Bip32KeyIndex], addr_idx: Union[int, Bip32KeyIndex]*) → *Bip32PublicKey*

Get the public key with the specified change and address indexes.

Parameters

- **change_idx** (*int or Bip32KeyIndex object*) – Change index
- **addr_idx** (*int or Bip32KeyIndex object*) – Address index

Returns

Bip32PublicKey object

Return type

Bip32PublicKey object

Raises

- **Bip32KeyError** – If the derivation results in an invalid key
- **Bip32PathError** – If the path indexes are not valid

abstract **GetAddress**(*change_idx: Union[int, Bip32KeyIndex], addr_idx: Union[int, Bip32KeyIndex]*) → str

Get the address with the specified change and address indexes.

Parameters

- **change_idx** (*int or Bip32KeyIndex object*) – Change index
- **addr_idx** (*int or Bip32KeyIndex object*) – Address index

Returns

Address

Return type

str

Raises

- **Bip32KeyError** – If the derivation results in an invalid key
- **Bip32PathError** – If the path indexes are not valid

class **ElectrumV2Standard**(*bip32_obj: Bip32Base*)

Bases: *ElectrumV2Base*

Electrum v2 standard class. It derives keys like the Electrum wallet with standard mnemonic.

GetPrivateKey(*change_idx: Union[int, Bip32KeyIndex], addr_idx: Union[int, Bip32KeyIndex]*) → *Bip32PrivateKey*

Get the private key with the specified change and address indexes. Derivation path: m/change_idx/addr_idx

Parameters

- **change_idx** (*int or Bip32KeyIndex object*) – Change index
- **addr_idx** (*int or Bip32KeyIndex object*) – Address index

Returns

Bip32PrivateKey object

Return type

Bip32PrivateKey object

Raises

- **Bip32KeyError** – If the derivation results in an invalid key or the object is public-only
- **Bip32PathError** – If the path indexes are not valid

GetPublicKey(*change_idx*: Union[int, Bip32KeyIndex], *addr_idx*: Union[int, Bip32KeyIndex]) → Bip32PublicKey

Get the public key with the specified change and address indexes. Derivation path: m/change_idx/addr_idx

Parameters

- **change_idx** (int or Bip32KeyIndex object) – Change index
- **addr_idx** (int or Bip32KeyIndex object) – Address index

Returns

Bip32PublicKey object

Return type

Bip32PublicKey object

Raises

- **Bip32KeyError** – If the derivation results in an invalid key
- **Bip32PathError** – If the path indexes are not valid

GetAddress(*change_idx*: Union[int, Bip32KeyIndex], *addr_idx*: Union[int, Bip32KeyIndex]) → str

Get the address with the specified change and address indexes. Derivation path: m/change_idx/addr_idx

Parameters

- **change_idx** (int or Bip32KeyIndex object) – Change index
- **addr_idx** (int or Bip32KeyIndex object) – Address index

Returns

Address

Return type

str

Raises

- **Bip32KeyError** – If the derivation results in an invalid key
- **Bip32PathError** – If the path indexes are not valid

m_bip32_obj: Bip32Base

class ElectrumV2Segwit(bip32: Bip32Base)

Bases: ElectrumV2Base

Electrum v2 segwit class. It derives keys like the Electrum wallet with segwit mnemonic.

m_bip32_acc: Bip32Base

GetPrivateKey(*change_idx*: Union[int, Bip32KeyIndex], *addr_idx*: Union[int, Bip32KeyIndex]) → Bip32PrivateKey

Get the private key with the specified change and address indexes. Derivation path: m/0'/change_idx/addr_idx

Parameters

- **change_idx** (int or Bip32KeyIndex object) – Change index
- **addr_idx** (int or Bip32KeyIndex object) – Address index

Returns

Bip32PrivateKey object

Return type

Bip32PrivateKey object

Raises

- **Bip32KeyError** – If the derivation results in an invalid key or the object is public-only
- **Bip32PathError** – If the path indexes are not valid

GetPublicKey(*change_idx*: Union[int, Bip32KeyIndex], *addr_idx*: Union[int, Bip32KeyIndex]) → Bip32PublicKey

Get the public key with the specified change and address indexes. Derivation path: m/0'/change_idx/addr_idx

Parameters

- **change_idx** (int or Bip32KeyIndex object) – Change index
- **addr_idx** (int or Bip32KeyIndex object) – Address index

Returns

Bip32PublicKey object

Return type

Bip32PublicKey object

Raises

- **Bip32KeyError** – If the derivation results in an invalid key
- **Bip32PathError** – If the path indexes are not valid

GetAddress(*change_idx*: Union[int, Bip32KeyIndex], *addr_idx*: Union[int, Bip32KeyIndex]) → str

Get the address with the specified change and address indexes. Derivation path: m/0'/change_idx/addr_idx

Parameters

- **change_idx** (int or Bip32KeyIndex object) – Change index
- **addr_idx** (int or Bip32KeyIndex object) – Address index

Returns

Address

Return type

str

Raises

- **Bip32KeyError** – If the derivation results in an invalid key
- **Bip32PathError** – If the path indexes are not valid

10.1.10.3 mnemonic_v1

10.1.10.3.1 electrum_v1_entropy_generator

Module for Electrum v1 mnemonic entropy generation.

class **ElectrumV1EntropyBitLen**(*value*)

Bases: IntEnum

Enumerative for Electrum entropy bit lengths (v1).

```
BIT_LEN_128 = 128
```

```
class ElectrumV1EntropyGeneratorConst
```

```
Bases: object
```

```
Class container for Electrum entropy generator constants (v1).
```

```
ENTROPY_BIT_LEN: List[ElectrumV1EntropyBitLen] =
[<ElectrumV1EntropyBitLen.BIT_LEN_128: 128>]
```

```
class ElectrumV1EntropyGenerator(bit_len: Union[int, ElectrumV1EntropyBitLen] =
ElectrumV1EntropyBitLen.BIT_LEN_128)
```

```
Bases: EntropyGenerator
```

```
Electrum entropy generator class (v1). It generates random entropy bytes.
```

```
static IsValidEntropyBitLen(bit_len: int) → bool
```

```
Get if the specified entropy bit length is valid.
```

```
Parameters
```

```
bit_len (int) – Entropy length in bits
```

```
Returns
```

```
True if valid, false otherwise
```

```
Return type
```

```
bool
```

```
static IsValidEntropyByteLen(byte_len: int) → bool
```

```
Get if the specified entropy byte length is valid.
```

```
Parameters
```

```
byte_len (int) – Entropy length in bytes
```

```
Returns
```

```
True if valid, false otherwise
```

```
Return type
```

```
bool
```

```
m_bit_len: int
```

10.1.10.3.2 electrum_v1_mnemonic

```
Module for Electrum v1 mnemonic.
```

```
class ElectrumV1WordsNum(value)
```

```
Bases: IntEnum
```

```
Enumerative for Electrum words number (v1).
```

```
WORDS_NUM_12 = 12
```

```
class ElectrumV1Languages(value)
```

```
Bases: MnemonicLanguages
```

```
Enumerative for Electrum languages (v1).
```

```
ENGLISH = 1
```

```
class ElectrumV1MnemonicConst
```

Bases: object

Class container for Electrum v1 mnemonic constants.

```
MNEMONIC_WORD_NUM: List[ElectrumV1WordsNum] = [<ElectrumV1WordsNum.WORDS_NUM_12:
12>]
```

```
LANGUAGE_FILES: Dict[MnemonicLanguages, str] = {ElectrumV1Languages.ENGLISH:
'wordlist/english.txt'}
```

```
WORDS_LIST_NUM: int = 1626
```

```
class ElectrumV1Mnemonic(mnemonic_list: List[str])
```

Bases: *Bip39Mnemonic*

Electrum v1 mnemonic class.

```
m_mnemonic_list: List[str]
```

10.1.10.3.3 electrum_v1_mnemonic_decoder

Module for Electrum v1 mnemonic decoding. Reference: <https://github.com/spesmilo/electrum>

```
class ElectrumV1MnemonicDecoder(lang: Optional[ElectrumV1Languages] =
ElectrumV1Languages.ENGLISH)
```

Bases: *MnemonicDecoderBase*

Electrum v1 mnemonic decoder class. It decodes a mnemonic phrase to bytes.

```
Decode(mnemonic: Union[str, Mnemonic]) → bytes
```

Decode a mnemonic phrase to bytes.

Parameters

mnemonic (*str* or *Mnemonic* object) – Mnemonic

Returns

Decoded bytes

Return type

bytes

Raises

ValueError – If mnemonic is not valid

```
m_lang: Optional[MnemonicLanguages]
```

```
m_words_list: Optional[MnemonicWordsList]
```

```
m_words_list_finder_cls: Type[MnemonicWordsListFinderBase]
```


10.1.10.3.4 electrum_v1_mnemonic_encoder

Module for Electrum v1 mnemonic encoding. Reference: <https://github.com/spesmilo/electrum>

class ElectrumV1MnemonicEncoder(*lang*: [ElectrumV1Languages](#) = *ElectrumV1Languages.ENGLISH*)

Bases: [MnemonicEncoderBase](#)

Electrum v1 mnemonic encoder class. It encodes bytes to the mnemonic phrase.

Encode(*entropy_bytes*: *bytes*) → *Mnemonic*

Encode bytes to mnemonic phrase.

Parameters

entropy_bytes (*bytes*) – Entropy bytes (accepted lengths in bits: 128)

Returns

Encoded mnemonic

Return type

Mnemonic object

Raises

ValueError – If bytes length is not valid

m_words_list: [MnemonicWordsList](#)

10.1.10.3.5 electrum_v1_mnemonic_generator

Module for Electrum v1 mnemonic generation.

class ElectrumV1MnemonicGeneratorConst

Bases: object

Class container for Electrum v1 mnemonic generator constants.

WORDS_NUM_TO_ENTROPY_LEN: Dict[[ElectrumV1WordsNum](#), [ElectrumV1EntropyBitLen](#)] =
{[ElectrumV1WordsNum.WORDS_NUM_12](#): [ElectrumV1EntropyBitLen.BIT_LEN_128](#)}

class ElectrumV1MnemonicGenerator(*lang*: [ElectrumV1Languages](#) = *ElectrumV1Languages.ENGLISH*)

Bases: object

Electrum v1 mnemonic generator class. It generates 12-words mnemonic in according to v1 Electrum mnemonic.

m_mnemonic_encoder: [ElectrumV1MnemonicEncoder](#)

FromWordsNumber(*words_num*: Union[int, [ElectrumV1WordsNum](#)]) → *Mnemonic*

Generate mnemonic with the specified words number from random entropy. There is no really need of this method, since the words number can only be 12, but it's kept to have the same usage of Bip39/Monero mnemonic generator.

Parameters

words_num (*int* or [ElectrumV1WordsNum](#)) – Number of words (12)

Returns

Generated mnemonic

Return type

Mnemonic object

Raises

ValueError – If words number is not valid

FromEntropy(*entropy_bytes: bytes*) → *Mnemonic*

Generate mnemonic from the specified entropy bytes.

Parameters

entropy_bytes (*bytes*) – Entropy bytes

Returns

Generated mnemonic

Return type

Mnemonic object

Raises

ValueError – If entropy byte length is not valid

10.1.10.3.6 electrum_v1_mnemonic_utils

Module for Electrum v1 mnemonic utility classes.

class ElectrumV1WordsListGetter

Bases: *MnemonicWordsListGetterBase*

Electrum words list getter class (v1). It allows to get words list by language so that they are loaded from file only once per language.

GetByLanguage(*lang: MnemonicLanguages*) → *MnemonicWordsList*

Get words list by language. Words list of a specific language are loaded from file only the first time they are requested.

Parameters

lang (*MnemonicLanguages*) – Language

Returns

MnemonicWordsList object

Return type

MnemonicWordsList object

Raises

- **TypeError** – If the language is not a Bip39Languages enum
- **ValueError** – If loaded words list is not valid

m_words_lists: Dict[*MnemonicLanguages*, *MnemonicWordsList*]

class ElectrumV1WordsListFinder

Bases: *MnemonicWordsListFinderBase*

Electrum words list finder class (v1). It automatically finds the correct words list from a mnemonic.

classmethod FindLanguage(*mnemonic: Mnemonic*) → Tuple[*MnemonicWordsList*, *MnemonicLanguages*]

Automatically find the language of the specified mnemonic and get the correct MnemonicWordsList class for it.

Parameters

mnemonic (*Mnemonic object*) – Mnemonic object

Returns

MnemonicWordsList object (index 0), mnemonic language (index 1)

Return type

tuple[*MnemonicWordsList*, *MnemonicLanguages*]

Raises

ValueError – If the mnemonic language cannot be found

10.1.10.3.7 electrum_v1_mnemonic_validator

Module for Electrum v1 mnemonic validation.

```
class ElectrumV1MnemonicValidator(lang: Optional[ElectrumV1Languages] =  
                                ElectrumV1Languages.ENGLISH)
```

Bases: *MnemonicValidator*

Electrum v1 mnemonic validator class. It validates a mnemonic phrase.

m_mnemonic_decoder: *ElectrumV1MnemonicDecoder*

10.1.10.3.8 electrum_v1_seed_generator

Module for Electrum v1 mnemonic seed generation.

```
class ElectrumV1SeedGeneratorConst
```

Bases: object

Class container for Electrum v1 seed generator constants.

HASH_ITR_NUM: int = 100000

```
class ElectrumV1SeedGenerator(mnemonic: Union[str, Mnemonic], lang: Optional[ElectrumV1Languages] =  
                             ElectrumV1Languages.ENGLISH)
```

Bases: object

Electrum seed generator class (v1). It generates the seed from a mnemonic.

m_seed: bytes

Generate() → bytes

Generate seed. There is no really need of this method, since the seed is always the same, but it's kept in this way to have the same usage of Bip39/Substrate seed generator (i.e. `ElectrumV1SeedGenerator(mnemonic).Generate()`).

Returns

Generated seed

Return type

bytes

10.1.10.4 mnemonic_v2

10.1.10.4.1 electrum_v2_entropy_generator

Module for Electrum v2 mnemonic entropy generation.

class `ElectrumV2EntropyBitLen`(*value*)

Bases: `IntEnum`

Enumerative for Electrum entropy bit lengths (v2).

BIT_LEN_132 = 132

BIT_LEN_264 = 264

class `ElectrumV2EntropyGeneratorConst`

Bases: `object`

Class container for Electrum entropy generator constants (v2).

ENTROPY_BIT_LEN: `List[ElectrumV2EntropyBitLen]` =
[<ElectrumV2EntropyBitLen.BIT_LEN_132: 132>, <ElectrumV2EntropyBitLen.BIT_LEN_264: 264>]

class `ElectrumV2EntropyGenerator`(*bit_len*: `Union[int, ElectrumV2EntropyBitLen]`)

Bases: `EntropyGenerator`

Electrum entropy generator class (v2). It generates random entropy bytes.

static `IsValidEntropyBitLen`(*bit_len*: `int`) → `bool`

Get if the specified entropy bit length is valid.

Parameters

bit_len (`int`) – Entropy length in bits

Returns

True if valid, false otherwise

Return type

`bool`

static `IsValidEntropyByteLen`(*byte_len*: `int`) → `bool`

Get if the specified entropy byte length is valid.

Parameters

byte_len (`int`) – Entropy length in bytes

Returns

True if valid, false otherwise

Return type

`bool`

static `AreEntropyBitsEnough`(*entropy*: `Union[bytes, int]`) → `bool`

Get if the entropy bits are enough to generate a valid mnemonic.

Parameters

entropy (`bytes` or `int`) – Entropy

Returns

True if enough, false otherwise

Return type

bool

m_bit_len: int

10.1.10.4.2 electrum_v2_mnemonic

Module for Electrum v2 mnemonic.

class ElectrumV2WordsNum(*value*)

Bases: IntEnum

Enumerative for Electrum words number (v2).

WORDS_NUM_12 = 12

WORDS_NUM_24 = 24

class ElectrumV2Languages(*value*)

Bases: *MnemonicLanguages*

Enumerative for Electrum languages (v2).

CHINESE_SIMPLIFIED = Bip39Languages.CHINESE_SIMPLIFIED

ENGLISH = Bip39Languages.ENGLISH

PORTUGUESE = Bip39Languages.PORTUGUESE

SPANISH = Bip39Languages.SPANISH

class ElectrumV2MnemonicTypes(*value*)

Bases: Enum

Enumerative for Electrum v2 mnemonic types.

STANDARD = 1

SEGWIT = 2

STANDARD_2FA = 3

SEGWIT_2FA = 4

class ElectrumV2MnemonicConst

Bases: object

Class container for Electrum v2 mnemonic constants.

MNEMONIC_WORD_NUM: List[*ElectrumV2WordsNum*] = [<ElectrumV2WordsNum.WORDS_NUM_12: 12>, <ElectrumV2WordsNum.WORDS_NUM_24: 24>]

TYPE_TO_PREFIX: Dict[*ElectrumV2MnemonicTypes*, str] =
{<ElectrumV2MnemonicTypes.STANDARD: 1>: '01', <ElectrumV2MnemonicTypes.SEGWIT: 2>:
'100', <ElectrumV2MnemonicTypes.STANDARD_2FA: 3>: '101',
<ElectrumV2MnemonicTypes.SEGWIT_2FA: 4>: '102'}

WORD_BIT_LEN: int = 11

```
class ElectrumV2Mnemonic(mnemonic_list: List[str])
    Bases: Bip39Mnemonic
    Electrum mnemonic class.
    m_mnemonic_list: List[str]
```

10.1.10.4.3 electrum_v2_mnemonic_decoder

Module for Electrum v2 mnemonic decoding. Reference: <https://github.com/electrum/py-electrum-sdk>

```
class ElectrumV2MnemonicDecoder(mnemonic_type: Optional[ElectrumV2MnemonicTypes] = None, lang:
    Optional[ElectrumV2Languages] = None)
```

Bases: *MnemonicDecoderBase*

Electrum v2 mnemonic decoder class. It decodes a mnemonic phrase to bytes.

```
m_mnemonic_type: Optional[ElectrumV2MnemonicTypes]
```

```
Decode(mnemonic: Union[str, Mnemonic]) → bytes
```

Decode a mnemonic phrase to bytes (no checksum).

Parameters

mnemonic (*str or Mnemonic object*) – Mnemonic

Returns

Decoded bytes

Return type

bytes

Raises

- *MnemonicChecksumError* – If checksum is not valid
- *ValueError* – If mnemonic is not valid

10.1.10.4.4 electrum_v2_mnemonic_encoder

Module for Electrum v2 mnemonic encoding. Reference: <https://github.com/spesmilo/electrum>

```
class ElectrumV2MnemonicEncoder(mnemonic_type: ElectrumV2MnemonicTypes, lang:
    ElectrumV2Languages = ElectrumV2Languages.ENGLISH)
```

Bases: *MnemonicEncoderBase*

Electrum v2 mnemonic encoder class. It encodes bytes to the mnemonic phrase.

```
m_mnemonic_type: ElectrumV2MnemonicTypes
```

```
Encode(entropy_bytes: bytes) → Mnemonic
```

Encode bytes to mnemonic phrase.

Parameters

entropy_bytes (*bytes*) – Entropy bytes

Returns

Encoded mnemonic

Return type

Mnemonic object

Raises

ValueError – If bytes length is not valid or a mnemonic cannot be generated

10.1.10.4.5 electrum_v2_mnemonic_generator

Module for Electrum v2 mnemonic generation.

class ElectrumV2MnemonicGeneratorConst

Bases: object

Class container for Electrum v2 mnemonic generator constants.

```
WORDS_NUM_TO_ENTROPY_LEN: Dict[ElectrumV2WordsNum, ElectrumV2EntropyBitLen] =
{ElectrumV2WordsNum.WORDS_NUM_12: ElectrumV2EntropyBitLen.BIT_LEN_132,
 ElectrumV2WordsNum.WORDS_NUM_24: ElectrumV2EntropyBitLen.BIT_LEN_264}
```

```
MAX_ATTEMPTS: int = 1000000
```

```
class ElectrumV2MnemonicGenerator(mnemonic_type: ElectrumV2MnemonicTypes, lang:
    ElectrumV2Languages = ElectrumV2Languages.ENGLISH)
```

Bases: object

Electrum v2 mnemonic generator class. It generates 12 or 24-words mnemonic in according to Electrum wallets.

```
m_mnemonic_encoder: ElectrumV2MnemonicEncoder
```

```
FromWordsNumber(words_num: Union[int, ElectrumV2WordsNum]) → Mnemonic
```

Generate mnemonic with the specified words number and type from random entropy.

Parameters

words_num (*int* or *ElectrumV2WordsNum*) – Number of words (12)

Returns

Generated mnemonic

Return type

Mnemonic object

Raises

ValueError – If words number is not valid

```
FromEntropy(entropy_bytes: bytes) → Mnemonic
```

Generate mnemonic from the specified entropy bytes. Because of the mnemonic encoding algorithm used by Electrum, the specified entropy will only be a starting point to find a suitable one. Therefore, it's very likely that the actual entropy bytes will be different. To get the actual entropy bytes, just decode the generated mnemonic. Please note that, to successfully generate a mnemonic, the bits of the big endian integer encoded entropy shall be at least 121 (for 12 words) or 253 (for 24 words). Otherwise, a mnemonic generation is not possible and a ValueError exception will be raised.

Parameters

entropy_bytes (*bytes*) – Entropy bytes

Returns

Generated mnemonic

Return type

Mnemonic object

Raises

ValueError – If entropy byte length is not valid or a mnemonic cannot be generated

10.1.10.4.6 electrum_v2_mnemonic_utils

Module for Electrum v2 mnemonic generation.

class ElectrumV2MnemonicUtilsConst

Bases: object

Class container for Electrum v2 mnemonic utility constants.

HMAC_KEY: bytes = b'Seed version'

class ElectrumV2MnemonicUtils

Bases: object

Class container for Electrum v2 mnemonic utility functions.

static IsValidMnemonic(mnemonic: Mnemonic, mnemonic_type: Optional[ElectrumV2MnemonicTypes] = None) → bool

Get if the specified mnemonic is valid.

Parameters

- **mnemonic** (Mnemonic) – Mnemonic
- **mnemonic_type** (ElectrumV2MnemonicTypes) – Mnemonic type

Returns

True if valid, false otherwise

Return type

bool

10.1.10.4.7 electrum_v2_mnemonic_validator

Module for Electrum v2 mnemonic validation.

class ElectrumV2MnemonicValidator(mnemonic_type: Optional[ElectrumV2MnemonicTypes] = None, lang: Optional[ElectrumV2Languages] = None)

Bases: MnemonicValidator

Electrum v2 mnemonic validator class. It validates a mnemonic phrase.

m_mnemonic_decoder: ElectrumV2MnemonicDecoder

10.1.10.4.8 electrum_v2_seed_generator

Module for Electrum v2 mnemonic seed generation.

class ElectrumV2SeedGeneratorConst

Bases: object

Class container for Electrum seed generator constants (v2).

SEED_SALT_MOD: str = 'electrum'

SEED_PBKDF2_ROUNDS: int = 2048

class ElectrumV2SeedGenerator(mnemonic: Union[str, Mnemonic], lang: Optional[ElectrumV2Languages] = None)

Bases: object

Electrum seed generator class (v2). It generates the seed from a mnemonic.

m_entropy_bytes: bytes

Generate(passphrase: str = "") → bytes

Generate the seed using the specified passphrase.

Parameters

passphrase (str, optional) – Passphrase, empty if not specified

Returns

Generated seed

Return type

bytes

10.1.11 monero

10.1.11.1 conf

10.1.11.1.1 monero_coin_conf

Module with helper class for Monero coins configuration handling.

class MoneroCoinConf(coin_names: CoinNames, addr_net_ver: bytes, int_addr_net_ver: bytes, subaddr_net_ver: bytes)

Bases: object

Monero coin configuration class.

m_addr_params: Dict[str, bytes]

classmethod FromCoinConf(coin_conf: CoinConf) → MoneroCoinConf

Construct class.

Parameters

coin_conf (CoinConf object) – Generic coin configuration object

Returns

MoneroCoinConf object

Return type

MoneroCoinConf object

m_coin_names: CoinNames

m_addr_net_ver: bytes

m_int_addr_net_ver: bytes

m_subaddr_net_ver: bytes

CoinNames() → *CoinNames*

Get coin names.

Returns

CoinNames object

Return type

CoinNames object

AddrNetVersion() → bytes

Get address net version.

Returns

Address net version

Return type

bytes

IntegratedAddrNetVersion() → bytes

Get integrated address net version.

Returns

Address net version

Return type

bytes

SubaddrNetVersion() → bytes

Get subaddress net version.

Returns

Subaddress net version

Return type

bytes

10.1.11.1.2 monero_coins

Module for Monero coins enum.

class MoneroCoins(*value*)

Bases: Enum

Enumerative for supported Monero coins.

MONERO_MAINNET = 1

MONERO_STAGENET = 2

MONERO_TESTNET = 3

10.1.11.1.3 monero_conf

Module for Monero coins configuration.

class MoneroConf

Bases: object

Class container for Monero configuration.

MainNet: *MoneroCoinConf* = <bip_utils.monero.conf.monero_coin_conf.MoneroCoinConf object>

StageNet: *MoneroCoinConf* = <bip_utils.monero.conf.monero_coin_conf.MoneroCoinConf object>

TestNet: *MoneroCoinConf* = <bip_utils.monero.conf.monero_coin_conf.MoneroCoinConf object>

10.1.11.1.4 monero_conf_getter

Module for getting Monero coins configuration.

class MoneroConfGetterConst

Bases: object

Class container for Monero configuration getter constants.

COIN_TO_CONF: Dict[*MoneroCoins*, *MoneroCoinConf*] = {<MoneroCoins.MONERO_MAINNET: 1>: <bip_utils.monero.conf.monero_coin_conf.MoneroCoinConf object>, <MoneroCoins.MONERO_STAGENET: 2>: <bip_utils.monero.conf.monero_coin_conf.MoneroCoinConf object>, <MoneroCoins.MONERO_TESTNET: 3>: <bip_utils.monero.conf.monero_coin_conf.MoneroCoinConf object>}

class MoneroConfGetter

Bases: object

Monero configuration getter class. It allows to get the Monero configuration of a specific coin.

static GetConfig(*coin_type*: *MoneroCoins*) → *MoneroCoinConf*

Get coin configuration.

Parameters

coin_type (*MoneroCoins*) – Coin type

Returns

Coin configuration

Return type

MoneroCoinConf

Raises

TypeError – If coin type is not of a MoneroCoins enumerative

10.1.11.2 mnemonic

10.1.11.2.1 monero_entropy_generator

Module for Monero entropy generation.

class MoneroEntropyBitLen(*value*)

Bases: `IntEnum`

Enumerative for Monero entropy bit lengths.

BIT_LEN_128 = 128

BIT_LEN_256 = 256

class MoneroEntropyGeneratorConst

Bases: `object`

Class container for Monero entropy generator constants.

ENTROPY_BIT_LEN: `List[MoneroEntropyBitLen]` = [`<MoneroEntropyBitLen.BIT_LEN_128: 128>`, `<MoneroEntropyBitLen.BIT_LEN_256: 256>`]

class MoneroEntropyGenerator(*bit_len: Union[int, MoneroEntropyBitLen]*)

Bases: `EntropyGenerator`

Monero entropy generator class. It generates random entropy bytes with the specified length.

static IsValidEntropyBitLen(*bit_len: Union[int, MoneroEntropyBitLen]*) → `bool`

Get if the specified entropy bit length is valid.

Parameters

bit_len (*int* or `MoneroEntropyBitLen`) – Entropy length in bits

Returns

True if valid, false otherwise

Return type

`bool`

static IsValidEntropyByteLen(*byte_len: int*) → `bool`

Get if the specified entropy byte length is valid.

Parameters

byte_len (*int*) – Entropy length in bytes

Returns

True if valid, false otherwise

Return type

`bool`

m_bit_len: `int`

10.1.11.2.2 monero_mnemonic

Module for Monero mnemonic.

class MoneroWordsNum(*value*)

Bases: `IntEnum`

Enumerative for Monero words number.

WORDS_NUM_12 = 12

WORDS_NUM_13 = 13

WORDS_NUM_24 = 24

WORDS_NUM_25 = 25

class MoneroLanguages(*value*)

Bases: `MnemonicLanguages`

Enumerative for Monero languages.

CHINESE_SIMPLIFIED = 1

DUTCH = 2

ENGLISH = 3

FRENCH = 4

GERMAN = 5

ITALIAN = 6

JAPANESE = 7

PORTUGUESE = 8

SPANISH = 9

RUSSIAN = 10

class MoneroMnemonicConst

Bases: `object`

Class container for Monero mnemonic constants.

MNEMONIC_WORD_NUM: List[`MoneroWordsNum`] = [<MoneroWordsNum.WORDS_NUM_12: 12>, <MoneroWordsNum.WORDS_NUM_13: 13>, <MoneroWordsNum.WORDS_NUM_24: 24>, <MoneroWordsNum.WORDS_NUM_25: 25>]

MNEMONIC_WORD_NUM_CHKSUM: List[`MoneroWordsNum`] = [<MoneroWordsNum.WORDS_NUM_13: 13>, <MoneroWordsNum.WORDS_NUM_25: 25>]

LANGUAGE_UNIQUE_PREFIX_LEN: Dict[`MnemonicLanguages`, int] = {<MoneroLanguages.CHINESE_SIMPLIFIED: 1>: 1, <MoneroLanguages.DUTCH: 2>: 4, <MoneroLanguages.ENGLISH: 3>: 3, <MoneroLanguages.FRENCH: 4>: 4, <MoneroLanguages.GERMAN: 5>: 4, <MoneroLanguages.ITALIAN: 6>: 4, <MoneroLanguages.JAPANESE: 7>: 4, <MoneroLanguages.PORTUGUESE: 8>: 4, <MoneroLanguages.SPANISH: 9>: 4, <MoneroLanguages.RUSSIAN: 10>: 4}

```
LANGUAGE_FILES: Dict[MnemonicLanguages, str] = {<MoneroLanguages.CHINESE_SIMPLIFIED:
1>: 'wordlist/chinese_simplified.txt', <MoneroLanguages.DUTCH: 2>:
'wordlist/dutch.txt', <MoneroLanguages.ENGLISH: 3>: 'wordlist/english.txt',
<MoneroLanguages.FRENCH: 4>: 'wordlist/french.txt', <MoneroLanguages.GERMAN: 5>:
'wordlist/german.txt', <MoneroLanguages.ITALIAN: 6>: 'wordlist/italian.txt',
<MoneroLanguages.JAPANESE: 7>: 'wordlist/japanese.txt',
<MoneroLanguages.PORTUGUESE: 8>: 'wordlist/portuguese.txt',
<MoneroLanguages.SPANISH: 9>: 'wordlist/spanish.txt', <MoneroLanguages.RUSSIAN:
10>: 'wordlist/russian.txt'}
```

```
WORDS_LIST_NUM: int = 1626
```

```
class MoneroMnemonic(mnemonic_list: List[str])
```

Bases: [Mnemonic](#)

Monero mnemonic class (alias for Mnemonic).

```
m_mnemonic_list: List[str]
```

10.1.11.2.3 monero_mnemonic_decoder

Module for Monero mnemonic decoding.

```
class MoneroMnemonicDecoder(lang: Optional[MoneroLanguages] = None)
```

Bases: [MnemonicDecoderBase](#)

Monero mnemonic decoder class. It decodes a mnemonic phrase to bytes.

Decode(mnemonic: Union[str, Mnemonic]) → bytes

Decode a mnemonic phrase to bytes (no checksum).

Parameters

mnemonic (str or Mnemonic object) – Mnemonic

Returns

Decoded bytes

Return type

bytes

Raises

- [MnemonicChecksumError](#) – If checksum is not valid
- [ValueError](#) – If mnemonic is not valid

```
m_lang: Optional[MnemonicLanguages]
```

```
m_words_list: Optional[MnemonicWordsList]
```

```
m_words_list_finder_cls: Type[MnemonicWordsListFinderBase]
```

10.1.11.2.4 monero_mnemonic_encoder

Module for Monero mnemonic encoding.

class MoneroMnemonicEncoderBase(*lang*: [MoneroLanguages](#) = *MoneroLanguages.ENGLISH*)

Bases: [MnemonicEncoderBase](#), ABC

Monero mnemonic encoder base class. It encodes bytes to the mnemonic phrase.

m_lang: [MoneroLanguages](#)

class MoneroMnemonicNoChecksumEncoder(*lang*: [MoneroLanguages](#) = *MoneroLanguages.ENGLISH*)

Bases: [MoneroMnemonicEncoderBase](#)

Monero mnemonic encoder class (no checksum). It encodes bytes to the mnemonic phrase without checksum.

Encode(*entropy_bytes*: bytes) → [Mnemonic](#)

Encode bytes to mnemonic phrase (no checksum).

Parameters

entropy_bytes (bytes) – Entropy bytes (accepted lengths in bits: 128, 256)

Returns

Encoded mnemonic (no checksum)

Return type

Mnemonic object

Raises

ValueError – If entropy is not valid

m_lang: [MoneroLanguages](#)

m_words_list: [MnemonicWordsList](#)

class MoneroMnemonicWithChecksumEncoder(*lang*: [MoneroLanguages](#) = *MoneroLanguages.ENGLISH*)

Bases: [MoneroMnemonicEncoderBase](#)

Monero mnemonic encoder class (with checksum). It encodes bytes to the mnemonic phrase with checksum.

Encode(*entropy_bytes*: bytes) → [Mnemonic](#)

Encode bytes to mnemonic phrase (with checksum).

Parameters

entropy_bytes (bytes) – Entropy bytes (accepted lengths in bits: 128, 256)

Returns

Encoded mnemonic (with checksum)

Return type

Mnemonic object

Raises

ValueError – If entropy is not valid

m_lang: [MoneroLanguages](#)

m_words_list: [MnemonicWordsList](#)

```
class MoneroMnemonicEncoder(lang: MoneroLanguages = MoneroLanguages.ENGLISH)
```

Bases: object

Monero mnemonic encoder class. Helper class to encode bytes to the mnemonic phrase with or without checksum.

m_no_chk_enc: [*MoneroMnemonicNoChecksumEncoder*](#)

m_with_chk_enc: [*MoneroMnemonicWithChecksumEncoder*](#)

EncodeNoChecksum(entropy_bytes: bytes) → *Mnemonic*

Encode bytes to mnemonic phrase (no checksum).

Parameters

entropy_bytes (bytes) – Entropy bytes (accepted lengths in bits: 128, 256)

Returns

Encoded mnemonic (no checksum)

Return type

Mnemonic object

Raises

ValueError – If bytes length is not valid

EncodeWithChecksum(entropy_bytes: bytes) → *Mnemonic*

Encode bytes to mnemonic phrase (with checksum).

Parameters

entropy_bytes (bytes) – Entropy bytes (accepted lengths in bits: 128, 256)

Returns

Encoded mnemonic (with checksum)

Return type

Mnemonic object

Raises

ValueError – If bytes length is not valid

10.1.11.2.5 monero_mnemonic_generator

Module for Monero mnemonic generation.

```
class MoneroMnemonicGeneratorConst
```

Bases: object

Class container for Monero mnemonic generator constants.

```
WORDS_NUM_TO_ENTROPY_LEN: Dict[MoneroWordsNum, MoneroEntropyBitLen] =
{MoneroWordsNum.WORDS_NUM_12: MoneroEntropyBitLen.BIT_LEN_128,
MoneroWordsNum.WORDS_NUM_13: MoneroEntropyBitLen.BIT_LEN_128,
MoneroWordsNum.WORDS_NUM_24: MoneroEntropyBitLen.BIT_LEN_256,
MoneroWordsNum.WORDS_NUM_25: MoneroEntropyBitLen.BIT_LEN_256}
```

```
class MoneroMnemonicGenerator(lang: MoneroLanguages = MoneroLanguages.ENGLISH)
```

Bases: object

Monero mnemonic generator class. Mnemonic can be generated randomly from words number or from a specified entropy.

m_mnemonic_encoder: *MoneroMnemonicEncoder*

FromWordsNumber(*words_num: Union[int, MoneroWordsNum]*) → *Mnemonic*

Generate mnemonic with the specified words number from random entropy.

Parameters

words_num (*int* or *MoneroWordsNum*) – Number of words (12, 13, 24, 25)

Returns

Generated mnemonic

Return type

Mnemonic object

Raises

ValueError – If words number is not valid

FromEntropyNoChecksum(*entropy_bytes: bytes*) → *Mnemonic*

Generate mnemonic from the specified entropy bytes (no checksum).

Parameters

entropy_bytes (*bytes*) – Entropy bytes (accepted lengths in bits: 128, 256)

Returns

Generated mnemonic (no checksum)

Return type

Mnemonic object

Raises

ValueError – If entropy byte length is not valid

FromEntropyWithChecksum(*entropy_bytes: bytes*) → *Mnemonic*

Generate mnemonic from the specified entropy bytes (with checksum).

Parameters

entropy_bytes (*bytes*) – Entropy bytes (accepted lengths in bits: 128, 256)

Returns

Generated mnemonic (with checksum)

Return type

Mnemonic object

Raises

ValueError – If entropy byte length is not valid

10.1.11.2.6 monero_mnemonic_utils

Module for Monero mnemonic utility classes.

class MoneroWordsListGetter

Bases: *MnemonicWordsListGetterBase*

Monero words list getter class. It allows to get words list by language so that they are loaded from file only once per language.

GetByLanguage(*lang: MnemonicLanguages*) → *MnemonicWordsList*

Get words list by language. Words list of a specific language are loaded from file only the first time they are requested.

Parameters

lang ([MnemonicLanguages](#)) – Language

Returns

MnemonicWordsList object

Return type

MnemonicWordsList object

Raises

- **TypeError** – If the language is not a MoneroLanguages enum
- **ValueError** – If loaded words list is not valid

m_words_lists: Dict[[MnemonicLanguages](#), [MnemonicWordsList](#)]

class MoneroWordsListFinder

Bases: [MnemonicWordsListFinderBase](#)

Monero words list finder class. It automatically finds the correct words list from a mnemonic.

classmethod FindLanguage(*mnemonic*: [Mnemonic](#)) → Tuple[[MnemonicWordsList](#), [MnemonicLanguages](#)]

Automatically find the language of the specified mnemonic and get the correct MnemonicWordsList class for it.

Parameters

mnemonic (*Mnemonic object*) – Mnemonic object

Returns

MnemonicWordsList object (index 0), mnemonic language (index 1)

Return type

tuple[[MnemonicWordsList](#), [MnemonicLanguages](#)]

Raises

ValueError – If the mnemonic language cannot be found

class MoneroMnemonicUtils

Bases: object

Utility functions for Monero mnemonic.

static ComputeChecksum(*mnemonic*: List[str], *lang*: [MnemonicLanguages](#)) → str

Compute checksum.

Parameters

- **mnemonic** (*list[str]*) – Mnemonic list of words
- **lang** ([MnemonicLanguages](#)) – Language

Returns

Checksum word

Return type

str

10.1.11.2.7 monero_mnemonic_validator

Module for Monero mnemonic validation.

class MoneroMnemonicValidator(*lang: Optional[MoneroLanguages] = None*)

Bases: [MnemonicValidator](#)

Monero mnemonic validator class. It validates a mnemonic phrase.

m_mnemonic_decoder: [MnemonicDecoderBase](#)

10.1.11.2.8 monero_seed_generator

Module for Monero seed generation.

class MoneroSeedGenerator(*mnemonic: Union[str, Mnemonic], lang: Optional[MoneroLanguages] = None*)

Bases: object

Monero seed generator class. It generates the seed from a mnemonic.

m_entropy_bytes: bytes

Generate() → bytes

Generate seed. The seed is simply the entropy bytes in Monero case. There is no really need of this method, since the seed is always the same, but it's kept in this way to have the same usage of Bip39/Substrate seed generator (i.e. MoneroSeedGenerator(mnemonic).Generate()).

Returns

Generated seed

Return type

bytes

10.1.11.3 monero

Module for Monero keys computation and derivation.

class Monero(*priv_key: Union[bytes, IPrivateKey], pub_key: Optional[Union[bytes, IPublicKey]] = None, coin_type: MoneroCoins = MoneroCoins.MONERO_MAINNET*)

Bases: object

Monero class. It allows to compute Monero keys and addresses/subaddresses.

classmethod FromSeed(*seed_bytes: bytes, coin_type: MoneroCoins = MoneroCoins.MONERO_MAINNET*) → *Monero*

Create from seed bytes.

Parameters

- **seed_bytes** (bytes) – Seed bytes
- **coin_type** ([MoneroCoins](#), optional) – Coin type (default: main net)

Returns

Monero object

Return type

Monero object

classmethod FromBip44PrivateKey(*priv_key: Union[bytes, IPrivateKey], coin_type: MoneroCoins = MoneroCoins.MONERO_MAINNET*) → *Monero*

Create from Bip44 private key bytes.

Parameters

- **priv_key** (*bytes* or *IPrivateKey*) – Private key
- **coin_type** (*MoneroCoins*, *optional*) – Coin type (default: main net)

Returns

Monero object

Return type

Monero object

classmethod FromPrivateSpendKey(*priv_skey: Union[bytes, IPrivateKey], coin_type: MoneroCoins = MoneroCoins.MONERO_MAINNET*) → *Monero*

Create from private spend key.

Parameters

- **priv_skey** (*bytes* or *IPrivateKey*) – Private spend key
- **coin_type** (*MoneroCoins*, *optional*) – Coin type (default: main net)

Returns

Monero object

Return type

Monero object

Raises

MoneroKeyError – If the key constructed from the bytes is not valid

classmethod FromWatchOnly(*priv_vkey: Union[bytes, IPrivateKey], pub_skey: Union[bytes, IPublicKey], coin_type: MoneroCoins = MoneroCoins.MONERO_MAINNET*) → *Monero*

Create from private view key and public spend key (i.e. watch-only wallet).

Parameters

- **priv_vkey** (*bytes* or *IPrivateKey*) – Private view key
- **pub_skey** (*bytes* or *IPublicKey*) – Public spend key
- **coin_type** (*MoneroCoins*, *optional*) – Coin type (default: main net)

Returns

Monero object

Return type

Monero object

Raises

MoneroKeyError – If the key constructed from the bytes is not valid

m_priv_skey: *Optional[MoneroPrivateKey]*

m_priv_vkey: *MoneroPrivateKey*

m_pub_skey: *MoneroPublicKey*

m_pub_vkey: *MoneroPublicKey*

m_coin_conf: *MoneroCoinConf*

m_subaddr: *MoneroSubaddress*

IsWatchOnly() → bool

Return if it's a watch-only object.

Returns

True if watch-only, false otherwise

Return type

bool

CoinConf() → *MoneroCoinConf*

Return coin configuration.

Returns

MoneroCoinConf object

Return type

MoneroCoinConf object

PrivateSpendKey() → *MoneroPrivateKey*

Return the private spend key.

Returns

MoneroPrivateKey object

Return type

MoneroPrivateKey object

Raises

MoneroKeyError – If the class is watch-only

PrivateViewKey() → *MoneroPrivateKey*

Return the private view key.

Returns

MoneroPrivateKey object

Return type

MoneroPrivateKey object

PublicSpendKey() → *MoneroPublicKey*

Return the public spend key.

Returns

MoneroPublicKey object

Return type

MoneroPublicKey object

PublicViewKey() → *MoneroPublicKey*

Return the public view key.

Returns

MoneroPublicKey object

Return type

MoneroPublicKey object

IntegratedAddress(*payment_id: bytes*) → str

Return the integrated address with the specified payment ID.

Parameters

payment_id (*bytes*) – Payment ID

Returns

Integrated address string

Return type

str

PrimaryAddress() → str

Return the primary address.

Returns

Primary address string

Return type

str

Subaddress(*minor_idx: int, major_idx: int = 0*) → str

Return the specified subaddress.

Parameters

- **minor_idx** (*int*) – Minor index (i.e. subaddress index)
- **major_idx** (*int, optional*) – Major index (i.e. account index, default: 0)

Returns

Subaddress string

Return type

str

Raises

ValueError – If one of the indexes is not valid

10.1.11.4 monero_ex

Module for Monero exceptions.

exception MoneroKeyError

Bases: Exception

Exception in case of Monero key error.

10.1.11.5 monero_keys

Module for Monero keys handling.

class MoneroPublicKey(*pub_key: IPublicKey*)

Bases: object

Monero public key class.

classmethod FromBytesOrKeyObject(*pub_key: Union[bytes, IPublicKey]*) → *MoneroPublicKey*

Get the public key from key bytes or object.

Parameters

pub_key (*bytes or IPublicKey*) – Public key

Returns

MoneroPublicKey object

Return type

MoneroPublicKey object

Raises

MoneroKeyError – If the key constructed from the bytes is not valid

classmethod FromBytes(*key_bytes: bytes*) → *MoneroPublicKey*

Create from bytes.

Parameters

key_bytes (*bytes*) – Key bytes

Returns

MoneroPublicKey object

Return type

MoneroPublicKey object

Raises

MoneroKeyError – If the key constructed from the bytes is not valid

classmethod FromPoint(*key_point: IPoint*) → *MoneroPublicKey*

Create from point.

Parameters

key_point (*IPoint object*) – Key point

Returns

MoneroPublicKey object

Return type

MoneroPublicKey object

Raises

Bip32KeyError – If the key constructed from the bytes is not valid

m_pub_key: *IPublicKey*

KeyObject() → *IPublicKey*

Return the key object.

Returns

Key object

Return type

IPublicKey object

RawCompressed() → *DataBytes*

Return raw compressed public key.

Returns

DataBytes object

Return type

DataBytes object

RawUncompressed() → *DataBytes*

Return raw uncompressed public key.

Returns

DataBytes object

Return type

DataBytes object

class MoneroPrivateKey(priv_key: *IPrivateKey*)

Bases: object

Monero private key class.

classmethod FromBytesOrKeyObject(priv_key: *Union[bytes, IPrivateKey]*) → *MoneroPrivateKey*

Get the private key from key bytes or object.

Parameters**priv_key** (*bytes* or *IPrivateKey*) – Private key**Returns**

MoneroPrivateKey object

Return type

MoneroPrivateKey object

Raises*MoneroKeyError* – If the key constructed from the bytes is not valid**classmethod FromBytes**(key_bytes: *bytes*) → *MoneroPrivateKey*

Create from bytes.

Parameters**key_bytes** (*bytes*) – Key bytes**Raises***MoneroKeyError* – If the key constructed from the bytes is not valid**m_priv_key:** *IPrivateKey***KeyObject()** → *IPrivateKey*

Return the key object.

Returns

Key object

Return type

IPrivateKey object

Raw() → *DataBytes*

Return raw private key.

Returns

DataBytes object

Return type

DataBytes object

PublicKey() → *MoneroPublicKey*

Get the public key correspondent to the private one.

Returns

MoneroPublicKey object

Return type

MoneroPublicKey object

10.1.11.6 monero_subaddr

Module for Monero subaddress computation.

class MoneroSubaddressConst

Bases: object

Class container for Monero subaddress constants.

SUBADDR_PREFIX: bytes = b'SubAddr\x00'

SUBADDR_MAX_IDX: int = 4294967295

SUBADDR_IDX_BYTE_LEN: int = 4

class MoneroSubaddress(priv_vkey: *MoneroPrivateKey*, pub_skey: *MoneroPublicKey*, pub_vkey: *Optional[MoneroPublicKey]* = None)

Bases: object

Monero subaddress class. It allows to compute Monero subaddresses.

m_priv_vkey: *MoneroPrivateKey*

m_pub_skey: *MoneroPublicKey*

m_pub_vkey: *MoneroPublicKey*

ComputeKeys(minor_idx: int, major_idx: int) → Tuple[*MoneroPublicKey*, *MoneroPublicKey*]

Compute the public keys of the specified subaddress.

Parameters

- **minor_idx** (int) – Minor index (i.e. subaddress index)
- **major_idx** (int) – Major index (i.e. account index)

Returns

Computed public spend key (index 0) and public view key (index 1)

Return type

tuple[*MoneroPublicKey*, *MoneroPublicKey*]

Raises

ValueError – If one of the indexes is not valid

ComputeAndEncodeKeys(minor_idx: int, major_idx: int, net_ver: bytes) → str

Compute the public keys of the specified subaddress and encode them.

Parameters

- **minor_idx** (int) – Minor index (i.e. subaddress index)
- **major_idx** (int) – Major index (i.e. account index)

- **net_ver** (*bytes*) – Net version

Returns

Encoded subaddress string

Return type

str

Raises

ValueError – If one of the indexes is not valid

10.1.12 slip

10.1.12.1 slip173

10.1.12.1.1 slip173

Module for SLIP-0173 human-readable parts. Not all the human-readable parts are defined, but only the used ones. Reference: <https://github.com/satoshilabs/slips/blob/master/slip-0173.md>

class Slip173

Bases: object

SLIP-0173 class. It defines the human-readable parts in according to SLIP-0173.

AKASH_NETWORK: str = 'akash'

AXELAR: str = 'axelar'

BAND_PROTOCOL: str = 'band'

BINANCE_CHAIN: str = 'bnb'

BITCOIN_MAINNET: str = 'bc'

BITCOIN_REGTEST: str = 'bcrt'

BITCOIN_TESTNET: str = 'tb'

CERTIK: str = 'certik'

CHIHUAHUA: str = 'chihuahua'

COSMOS: str = 'cosmos'

ELROND: str = 'erd'

FETCH_AI: str = 'fetch'

HARMONY_ONE: str = 'one'

INJECTIVE: str = 'inj'

IRIS_NETWORK: str = 'iaa'

KAVA: str = 'kava'

LITECOIN_MAINNET: str = 'ltc'

```

LITECOIN_TESTNET: str = 'tltc'

OKEX_CHAIN: str = 'ex'

OSMOSIS: str = 'osmo'

SECRET_NETWORK: str = 'secret'

STAFI: str = 'stafi'

TERRA: str = 'terra'

ZILLIQA: str = 'zil'

```

10.1.12.2 slip32

10.1.12.2.1 slip32

Module for SLIP32 extended key serialization/deserialization. Reference: <https://github.com/satoshilabs/slips/blob/master/slip-0032.md>

class Slip32KeySerConst

Bases: object

Class container for SLIP32 key serialize constants.

```

STD_KEY_NET_VERSIONS: Slip32KeyNetVersions =
<bip_utils.slip.slip32.slip32_key_net_ver.Slip32KeyNetVersions object>

```

class Slip32PrivateKeySerializer

Bases: object

SLIP32 private key serializer class. It serializes private keys.

```

static Serialize(priv_key: ~bip_utils.ecc.common.ikeys.IPrivateKey, path: ~typing.Union[str,
~bip_utils.bip.bip32.bip32_path.Bip32Path], chain_code: ~typing.Union[bytes,
~bip_utils.bip.bip32.bip32_key_data.Bip32ChainCode], key_net_ver:
~bip_utils.slip.slip32.slip32_key_net_ver.Slip32KeyNetVersions =
<bip_utils.slip.slip32.slip32_key_net_ver.Slip32KeyNetVersions object>) → str

```

Serialize a private key.

Parameters

- **priv_key** (*IPrivateKey object*) – IPrivateKey object
- **path** (*str or Bip32Path object*) – BIP32 path
- **chain_code** (*bytes or Bip32ChainCode object*) – Chain code
- **key_net_ver** (*Slip32KeyNetVersions object, optional*) – Key net versions (SLIP32 net version by default)

Returns

Serialized private key

Return type

str

class Slip32PublicKeySerializer

Bases: object

SLIP32 public key serializer class. It serializes public keys.

```
static Serialize(pub_key: ~bip_utils.ecc.common.ikeys.IPublicKey, path: ~typing.Union[str,  
~bip_utils.bip.bip32.bip32_path.Bip32Path], chain_code: ~typing.Union[bytes,  
~bip_utils.bip.bip32.bip32_key_data.Bip32ChainCode], key_net_ver:  
~bip_utils.slip.slip32.slip32_key_net_ver.Slip32KeyNetVersions =  
<bip_utils.slip.slip32.slip32_key_net_ver.Slip32KeyNetVersions object>) → str
```

Serialize a public key.

Parameters

- **pub_key** (*IPublicKey object*) – IPublicKey object
- **path** (*str or Bip32Path object*) – BIP32 path
- **chain_code** (*bytes or Bip32ChainCode object*) – Chain code
- **key_net_ver** (*Slip32KeyNetVersions object, optional*) – Key net versions (SLIP32 net version by default)

Returns

Serialized public key

Return type

str

```
class Slip32DeserializedKey(key_bytes: bytes, path: Bip32Path, chain_code: Bip32ChainCode, is_public:  
bool)
```

Bases: object

SLIP32 deserialized key class. It represents a key deserialized with the Slip32KeyDeserializer.

m_key_bytes: bytes

m_path: Bip32Path

m_chain_code: Bip32ChainCode

m_is_public: bool

KeyBytes() → bytes

Get key bytes.

Returns

Key bytes

Return type

bytes

Path() → Bip32Path

Get path.

Returns

Bip32Path object

Return type

Bip32Path object

ChainCode() → *Bip32ChainCode*

Get chain code.

Returns

Bip32ChainCode object

Return type

Bip32ChainCode object

IsPublic() → bool

Get if public.

Returns

True if the key is public, false otherwise

Return type

bool

class Slip32KeyDeserializer

Bases: object

SLIP32 key deserializer class. It deserializes an extended key.

classmethod DeserializeKey(*ser_key_str: str, key_net_ver: ~bip_utils.slip.slip32.slip32_key_net_ver.Slip32KeyNetVersions = <bip_utils.slip.slip32.slip32_key_net_ver.Slip32KeyNetVersions object>*) → *Slip32DeserializedKey*

Deserialize a key.

Parameters

- **ser_key_str** (*str*) – Serialized key string
- **key_net_ver** (*Slip32KeyNetVersions object, optional*) – Key net versions (SLIP32 net version by default)

Returns

Slip32DeserializedKey object

Return type

Slip32DeserializedKey object

Raises

ValueError – If the key net version is not valid

10.1.12.2.2 slip32_key_net_ver

Module for SLIP32 net version class.

class Slip32KeyNetVersions(*pub_net_ver: str, priv_net_ver: str*)

Bases: object

SLIP32 key net versions class. It represents a SLIP32 key net versions.

m_pub_net_ver: str

m_priv_net_ver: str

Public() → str

Get public net version.

Returns

Public net version

Return type

str

Private() → str

Get private net version.

Returns

Private net version

Return type

str

10.1.12.3 slip44

10.1.12.3.1 slip44

Module for SLIP-0044 coin types. Not all the coin types are defined, but only the used ones. Reference: <https://github.com/satoshilabs/slips/blob/master/slip-0044.md>

class Slip44

Bases: object

SLIP-0044 class. It defines the coin types in according to SLIP-0044.

BITCOIN: int = 0

TESTNET: int = 1

LITECOIN: int = 2

DOGECOIN: int = 3

DASH: int = 5

ETHEREUM: int = 60

ETHEREUM_CLASSIC: int = 61

ICON: int = 74

VERGE: int = 77

ATOM: int = 118

MONERO: int = 128

ZCASH: int = 133

RIPPLE: int = 144

BITCOIN_CASH: int = 145

STELLAR: int = 148

```
NANO: int = 165
EOS: int = 194
TRON: int = 195
BITCOIN_SV: int = 236
ALGORAND: int = 283
ZILLIQA: int = 313
TERRA: int = 330
POLKADOT: int = 354
NEAR_PROTOCOL: int = 397
ERGO: int = 429
KUSAMA: int = 434
KAVA: int = 459
FILECOIN: int = 461
BAND_PROTOCOL: int = 494
THETA: int = 500
SOLANA: int = 501
ELROND: int = 508
SECRET_NETWORK: int = 529
NINE_CHRONICLES: int = 567
APTOS: int = 637
BINANCE_CHAIN: int = 714
SUI: int = 784
VECHAIN: int = 818
NEO: int = 888
OKEX_CHAIN: int = 996
HARMONY_ONE: int = 1023
ONTOLOGY: int = 1024
TEZOS: int = 1729
CARDANO: int = 1815
AVALANCHE: int = 9000
CELO: int = 52752
PI_NETWORK: int = 314159
```

10.1.13 solana

10.1.13.1 spl_token

Module for getting account addresses of SPL tokens.

class SplTokenConst

Bases: object

Class container for SPL token constants.

DEF_PROGRAM_ID: str = 'ATokenGPvbdGVxr1b2hvZbsiqW5xWH25efTNsLJA8knL'

DEF_TOKEN_PROGRAM_ID: str = 'TokenkegQfeZyiNwAJbNbGKPFXCWuBvf9Ss623VQ5DA'

PDA_MARKER: bytes = b'ProgramDerivedAddress'

SEED_BUMP_MAX_VAL: int = 255

SEEDS_MAX_NUM: int = 16

class SplToken

Bases: object

SPL token class. It provides methods for getting the account address associated to a SPL token.

classmethod GetAssociatedTokenAddress(*wallet_addr: str, token_mint_addr: str*) → str

Get the account address associated to the specified SPL token.

Parameters

- **wallet_addr** (*str*) – Wallet address
- **token_mint_addr** (*str*) – Token mint address

Returns

Associated account address

Return type

str

Raises

ValueError – If the account address cannot be found or the specified addresses are not valid

classmethod GetAssociatedTokenAddressWithProgramId(*wallet_addr: str, token_mint_addr: str, token_program_id: str*) → str

Get the account address associated to the specified SPL token and token program ID.

Parameters

- **wallet_addr** (*str*) – Wallet address
- **token_mint_addr** (*str*) – Token mint address
- **token_program_id** (*str*) – Token program ID

Returns

Associated account address

Return type

str

Raises

ValueError – If the account address cannot be found or the specified addresses or ID are not valid

classmethod **FindPda**(*seeds: List[bytes], program_id: str*) → str

Find a valid PDA (Program Derived Address) and its corresponding bump seed.

Parameters

- **seeds** (*list[bytes]*) – List of seeds bytes
- **program_id** (*str*) – Program ID

Returns

Found PDA

Return type

str

Raises

ValueError – If the PDA cannot be found or the specified seeds or program ID are not valid

10.1.14 ss58

10.1.14.1 ss58

Module for SS58 decoding/encoding. Reference: [https://github.com/paritytech/substrate/wiki/External-Address-Format-\(SS58\)](https://github.com/paritytech/substrate/wiki/External-Address-Format-(SS58))

class **SS58Const**

Bases: object

Class container for SS58 constants.

SIMPLE_ACCOUNT_FORMAT_MAX_VAL: int = 63

FORMAT_MAX_VAL: int = 16383

RESERVED_FORMATS: Tuple[int, int] = (46, 47)

DATA_BYTE_LEN: int = 32

CHECKSUM_BYTE_LEN: int = 2

CHECKSUM_PREFIX: bytes = b'SS58PRE'

class **SS58Encoder**

Bases: object

SS58 encoder class. It provides methods for encoding to SS58 format.

static **Encode**(*data_bytes: bytes, ss58_format: int*) → str

Encode bytes into a SS58 string.

Parameters

- **data_bytes** (*bytes*) – Data bytes (32-byte length)
- **ss58_format** (*int*) – SS58 format

Returns

SS58 encoded string

Return type

str

Raises**ValueError** – If parameters are not valid**class SS58Decoder**

Bases: object

SS58 decoder class. It provides methods for decoding SS58 format.

static Decode(*data_str: str*) → Tuple[int, bytes]

Decode bytes from a SS58 string.

Parameters**data_str** (*string*) – Data string**Returns**

SS58 format and data bytes

Return type

tuple[int, bytes]

Raises

- **SS58ChecksumError** – If checksum is not valid
- **ValueError** – If the string is not a valid SS58 format

10.1.14.2 ss58_ex

Module for SS58 exceptions.

exception SS58ChecksumError

Bases: Exception

Exception in case of checksum error.

10.1.15 substrate**10.1.15.1 conf****10.1.15.1.1 substrate_coin_conf**

Module with helper class for Substrate coins configuration handling.

class SubstrateCoinConf(*coin_names: CoinNames, ss58_format: int*)

Bases: object

Substrate coin configuration class.

classmethod FromCoinConf(*coin_conf: CoinConf*) → *SubstrateCoinConf*

Construct class.

Parameters**coin_conf** (*CoinConf object*) – Generic coin configuration object**Returns**

SubstrateCoinConf object

Return type

SubstrateCoinConf object

m_coin_names: *CoinNames***m_ss58_format:** *int***m_addr_params:** *Dict[str, int]***CoinNames()** → *CoinNames*

Get coin names.

Returns

CoinNames object

Return type

CoinNames object

SS58Format() → *int*

Get SS58 format.

Returns

SS58 format

Return type*int***AddrParams()** → *Dict[str, int]*

Get the address parameters.

Returns

Address parameters

Return type*dict***10.1.15.1.2 substrate_coins**

Module for Substrate coins enum.

class SubstrateCoins(*value*)

Bases: Enum

Enumerative for supported Substrate coins.

ACALA = 1**BIFROST** = 2**CHAINX** = 3**EDGEWARE** = 4**GENERIC** = 5**KARURA** = 6**KUSAMA** = 7**MOONBEAM** = 8

```
MOONRIVER = 9
PHALA = 10
PLASM = 11
POLKADOT = 12
SORA = 13
STAFI = 14
```

10.1.15.1.3 substrate_conf

Module for Substrate coins configuration. Reference: <https://wiki.polkadot.network/docs/build-ss58-registry>

class SubstrateConf

Bases: object

Class container for Substrate configuration.

Acala: *SubstrateCoinConf* =
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>

Bifrost: *SubstrateCoinConf* =
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>

ChainX: *SubstrateCoinConf* =
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>

Edgeware: *SubstrateCoinConf* =
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>

Generic: *SubstrateCoinConf* =
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>

Karura: *SubstrateCoinConf* =
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>

Kusama: *SubstrateCoinConf* =
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>

Moonbeam: *SubstrateCoinConf* =
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>

Moonriver: *SubstrateCoinConf* =
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>

Phala: *SubstrateCoinConf* =
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>

Plasm: *SubstrateCoinConf* =
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>

Polkadot: *SubstrateCoinConf* =
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>

```

Sora: SubstrateCoinConf =
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>

Stafi: SubstrateCoinConf =
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>

```

10.1.15.1.4 substrate_conf_getter

Module for getting Substrate coins configuration.

class SubstrateConfGetterConst

Bases: object

Class container for Substrate configuration getter constants.

```

COIN_TO_CONF: Dict[SubstrateCoins, SubstrateCoinConf] = {<SubstrateCoins.ACALA: 1>:
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>,
<SubstrateCoins.BIFROST: 2>:
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>,
<SubstrateCoins.CHAINX: 3>:
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>,
<SubstrateCoins.EDGEWARE: 4>:
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>,
<SubstrateCoins.GENERIC: 5>:
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>,
<SubstrateCoins.KARURA: 6>:
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>,
<SubstrateCoins.KUSAMA: 7>:
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>,
<SubstrateCoins.MOONBEAM: 8>:
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>,
<SubstrateCoins.MOONRIVER: 9>:
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>,
<SubstrateCoins.PHALA: 10>:
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>,
<SubstrateCoins.PLASM: 11>:
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>,
<SubstrateCoins.POLKADOT: 12>:
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>,
<SubstrateCoins.SORA: 13>:
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>,
<SubstrateCoins.STAFI: 14>:
<bip_utils.substrate.conf.substrate_coin_conf.SubstrateCoinConf object>}

```

class SubstrateConfGetter

Bases: object

Substrate configuration getter class. It allows to get the Substrate configuration of a specific coin.

static GetConfig(*coin_type*: *SubstrateCoins*) → *SubstrateCoinConf*

Get coin configuration.

Parameters

coin_type (*SubstrateCoins*) – Coin type

Returns

Coin configuration

Return type*SubstrateCoinConf***Raises****TypeError** – If coin type is not of a SubstrateCoins enumerative

10.1.15.2 mnemonic

10.1.15.2.1 substrate_bip39_seed_generator

Module for Substrate mnemonic seed generation.

```
class SubstrateBip39SeedGenerator(mnemonic: Union[str, Mnemonic], lang: Optional[Bip39Languages] = None)
```

Bases: *IBip39SeedGenerator*

Substrate BIP39 seed generator class. It implements a variant for generating seed introduced by Polkadot. Reference: <https://github.com/paritytech/substrate-bip39>

m_entropy_bytes: bytes

Generate(*passphrase: str = ""*) → bytes

Generate the seed using the specified passphrase.

Parameters

passphrase (*str, optional*) – Passphrase, empty if not specified

Returns

Generated seed

Return type

bytes

10.1.15.3 scale

10.1.15.3.1 substrate_scale_enc_base

Module for Substrate SCALE encoding base class.

```
class SubstrateScaleEncoderBase
```

Bases: ABC

Substrate SCALE encoding base class.

```
abstract classmethod Encode(value: Any) → bytes
```

Encode the specified value to bytes.

Parameters

value (*any*) – Value to be encoded

Returns

Encoded value

Return type

bytes

10.1.15.3.2 substrate_scale_enc_bytes

Module for Substrate SCALE encoding for bytes.

class SubstrateScaleBytesEncoder

Bases: *SubstrateScaleEncoderBase*

Substrate SCALE encoding class for bytes.

classmethod Encode(*value: Any*) → bytes

Encode the specified value to bytes.

Parameters

value (*any*) – Value to be encoded

Returns

Encoded value

Return type

bytes

10.1.15.3.3 substrate_scale_enc_cuint

Module for Substrate SCALE encoding for compact unsigned integers.

class SubstrateScaleCUintEncoderConst

Bases: object

Class container for Substrate SCALE encoding for compact unsigned integers constants.

SINGLE_BYTE_MODE_MAX_VAL: int = 63

TWO_BYTE_MODE_MAX_VAL: int = 16383

FOUR_BYTE_MODE_MAX_VAL: int = 1073741823

BIG_INTEGER_MODE_MAX_VAL: int =

224945689727159819140526925384299092943484855915095831655037778630591879033574393515952034305194542

class SubstrateScaleCUintEncoder

Bases: *SubstrateScaleEncoderBase*

Substrate SCALE encoding for compact unsigned integers.

classmethod Encode(*value: Any*) → bytes

Encode the specified value to bytes.

Parameters

value (*any*) – Value to be encoded

Returns

Encoded value

Return type

bytes

10.1.15.3.4 `substrate_scale_enc_uint`

Module for Substrate SCALE encoding for unsigned integers.

class `SubstrateScaleUIntEncoder`

Bases: *SubstrateScaleEncoderBase*, *ABC*

Substrate SCALE encoding class for unsigned integers.

class `SubstrateScaleU8Encoder`

Bases: *SubstrateScaleUIntEncoder*

Substrate SCALE encoding class for 8-bit unsigned integers.

classmethod `Encode(value: Any) → bytes`

Encode the specified value to bytes.

Parameters

value (*any*) – Value to be encoded

Returns

Encoded value

Return type

bytes

class `SubstrateScaleU16Encoder`

Bases: *SubstrateScaleUIntEncoder*

Substrate SCALE encoding class for 16-bit unsigned integers.

classmethod `Encode(value: Any) → bytes`

Encode the specified value to bytes.

Parameters

value (*any*) – Value to be encoded

Returns

Encoded value

Return type

bytes

class `SubstrateScaleU32Encoder`

Bases: *SubstrateScaleUIntEncoder*

Substrate SCALE encoding class for 32-bit unsigned integers.

classmethod `Encode(value: Any) → bytes`

Encode the specified value to bytes.

Parameters

value (*any*) – Value to be encoded

Returns

Encoded value

Return type

bytes

class SubstrateScaleU64Encoder

Bases: *SubstrateScaleUintEncoder*

Substrate SCALE encoding class for 64-bit unsigned integers.

classmethod Encode(*value: Any*) → bytes

Encode the specified value to bytes.

Parameters

value (*any*) – Value to be encoded

Returns

Encoded value

Return type

bytes

class SubstrateScaleU128Encoder

Bases: *SubstrateScaleUintEncoder*

Substrate SCALE encoding class for 128-bit unsigned integers.

classmethod Encode(*value: Any*) → bytes

Encode the specified value to bytes.

Parameters

value (*any*) – Value to be encoded

Returns

Encoded value

Return type

bytes

class SubstrateScaleU256Encoder

Bases: *SubstrateScaleUintEncoder*

Substrate SCALE encoding class for 256-bit unsigned integers.

classmethod Encode(*value: Any*) → bytes

Encode the specified value to bytes.

Parameters

value (*any*) – Value to be encoded

Returns

Encoded value

Return type

bytes

10.1.15.4 substrate

Module for Substrate keys computation and derivation.

class SubstrateConst

Bases: object

Class container for Substrate constants.

SEED_MIN_BYTE_LEN: int = 32

class Substrate(*priv_key: Optional[Union[bytes, IPrivateKey]], pub_key: Optional[Union[bytes, IPublicKey]], path: SubstratePath, coin_conf: SubstrateCoinConf*)

Bases: object

Substrate class. It allows to compute Substrate keys and addresses.

classmethod FromSeed(*seed_bytes: bytes, coin_type: SubstrateCoins*) → *Substrate*

Create a Substrate object from the specified seed.

Parameters

- **seed_bytes** (*bytes*) – Seed bytes
- **coin_type** (*SubstrateCoins*) – Coin type

Returns

Substrate object

Return type

Substrate object

Raises

- **TypeError** – If coin_type is not of SubstrateCoins enum
- **ValueError** – If the seed length is not valid

classmethod FromSeedAndPath(*seed_bytes: bytes, path: Union[str, SubstratePath], coin_type: SubstrateCoins*) → *Substrate*

Create a Substrate object from the specified seed and path.

Parameters

- **seed_bytes** (*bytes*) – Seed bytes
- **path** (*str or SubstratePath object*) – Path
- **coin_type** (*SubstrateCoins*) – Coin type

Returns

Substrate object

Return type

Substrate object

Raises

- **TypeError** – If coin_type is not of SubstrateCoins enum
- **ValueError** – If the seed length is not valid
- **SubstratePathError** – If the path is not valid

classmethod **FromPrivateKey**(*priv_key*: Union[bytes, IPrivateKey], *coin_type*: SubstrateCoins) → *Substrate*

Create a Substrate object from the specified private key.

Parameters

- **priv_key** (bytes or IPrivateKey) – Private key
- **coin_type** (SubstrateCoins) – Coin type

Returns

Substrate object

Return type

Substrate object

Raises

- **TypeError** – If coin_type is not of SubstrateCoins enum
- **SubstrateKeyError** – If the key is not valid

classmethod **FromPublicKey**(*pub_key*: Union[bytes, IPublicKey], *coin_type*: SubstrateCoins) → *Substrate*

Create a Substrate object from the specified public key.

Parameters

- **pub_key** (bytes or IPublicKey) – Public key
- **coin_type** (SubstrateCoins) – Coin type

Returns

Substrate object

Return type

Substrate object

Raises

- **TypeError** – If coin_type is not of SubstrateCoins enum
- **SubstrateKeyError** – If the key is not valid

m_priv_key: Optional[SubstratePrivateKey]

m_pub_key: SubstratePublicKey

m_path: SubstratePath

m_coin_conf: SubstrateCoinConf

ChildKey(*path_elem*: Union[str, SubstratePathElem]) → *Substrate*

Create and return a child key of the current one with the specified path element.

Parameters

path_elem (str or SubstratePathElem object) – Path element

Returns

Substrate object

Return type

Substrate object

Raises

SubstrateKeyError – If the index results in invalid keys

DerivePath(*path*: Union[str, SubstratePath]) → *Substrate*

Derive children keys from the specified path.

Parameters

path (str or SubstratePath object) – Path

Returns

Substrate object

Return type

Substrate object

Raises

SubstratePathError – If the path is not valid

ConvertToPublic() → None

Convert a private Substrate object into a public one.

IsPublicOnly() → bool

Get if it's public-only.

Returns

True if public-only, false otherwise

Return type

bool

CoinConf() → *SubstrateCoinConf*

Return coin configuration.

Returns

SubstrateCoinConf object

Return type

SubstrateCoinConf object

Path() → *SubstratePath*

Return path.

Returns

SubstratePath object

Return type

SubstratePath object

PrivateKey() → *SubstratePrivateKey*

Return private key object.

Returns

SubstratePrivateKey object

Return type

SubstratePrivateKey object

Raises

SubstrateKeyError – If internal key is public-only

PublicKey() → *SubstratePublicKey*

Return public key object.

Returns

SubstratePublicKey object

Return type

SubstratePublicKey object

10.1.15.5 substrate_ex

Module for Substrate exceptions.

exception SubstrateKeyError

Bases: Exception

Exception in case of Substrate key error.

exception SubstratePathError

Bases: Exception

Exception in case of Substrate path error.

10.1.15.6 substrate_keys

Module for Substrate keys handling.

class SubstratePublicKey(*pub_key: IPublicKey, coin_conf: SubstrateCoinConf*)

Bases: object

Substrate public key class.

classmethod FromBytesOrKeyObject(*pub_key: Union[bytes, IPublicKey], coin_conf: SubstrateCoinConf*) → *SubstratePublicKey*

Get the public key from key bytes or object.

Parameters

- **pub_key** (*bytes* or *IPublicKey*) – Public key
- **coin_conf** (*SubstrateCoinConf* object) – SubstrateCoinConf object

Returns

SubstratePublicKey object

Return type

SubstratePublicKey object

Raises

SubstrateKeyError – If the key constructed from the bytes is not valid

classmethod FromBytes(*key_bytes: bytes, coin_conf: SubstrateCoinConf*) → *SubstratePublicKey*

Create from bytes.

Parameters

- **key_bytes** (*bytes*) – Key bytes
- **coin_conf** (*SubstrateCoinConf* object) – SubstrateCoinConf object

Raises

SubstrateKeyError – If the key constructed from the bytes is not valid

m_pub_key: *IPublicKey*

m_coin_conf: *SubstrateCoinConf*

KeyObject() → *IPublicKey*

Return the key object.

Returns

Key object

Return type

IPublicKey object

RawCompressed() → *DataBytes*

Return raw compressed public key.

Returns

DataBytes object

Return type

DataBytes object

RawUncompressed() → *DataBytes*

Return raw uncompressed public key.

Returns

DataBytes object

Return type

DataBytes object

ToAddress() → str

Return the address correspondent to the public key.

Returns

Address string

Return type

str

class SubstratePrivateKey(priv_key: *IPrivateKey*, coin_conf: *SubstrateCoinConf*)

Bases: object

Substrate private key class.

classmethod FromBytesOrKeyObject(priv_key: *Union[bytes, IPrivateKey]*, coin_conf: *SubstrateCoinConf*) → *SubstratePrivateKey*

Get the private key from key bytes or object.

Parameters

- **priv_key** (*bytes* or *IPrivateKey*) – Private key
- **coin_conf** (*SubstrateCoinConf* object) – *SubstrateCoinConf* object

Returns

SubstratePrivateKey object

Return type

SubstratePrivateKey object

Raises*SubstrateKeyError* – If the key constructed from the bytes is not valid**classmethod** **FromBytes**(key_bytes: bytes, coin_conf: SubstrateCoinConf) → SubstratePrivateKey

Create from bytes.

Parameters

- **key_bytes** (bytes) – Key bytes
- **coin_conf** (SubstrateCoinConf object) – SubstrateCoinConf object

Raises*SubstrateKeyError* – If the key constructed from the bytes is not valid**m_priv_key**: IPrivateKey**m_coin_conf**: SubstrateCoinConf**KeyObject**() → IPrivateKey

Return the key object.

Returns

Key object

Return type

IPrivateKey object

Raw() → DataBytes

Return raw private key.

Returns

DataBytes object

Return type

DataBytes object

PublicKey() → SubstratePublicKey

Get the public key correspondent to the private one.

Returns

SubstratePublicKey object

Return type

SubstratePublicKey object

10.1.15.7 substrate_path

Module for Substrate paths parsing and handling.

class SubstratePathConst

Bases: object

Container for Substrate path constants.

ENCODED_ELEM_MAX_BYTE_LEN: int = 32**RE_PATH**: str = '\\\\/[^\s/]+'**SOFT_PATH_PREFIX**: str = '/'

```
HARD_PATH_PREFIX: str = '/'

SCALE_INT_ENCODERS: Dict[int, Type[SubstrateScaleEncoderBase]] = {8: <class
'bip_utils.substrate.scale.substrate_scale_enc_uint.SubstrateScaleU8Encoder'>, 16:
<class
'bip_utils.substrate.scale.substrate_scale_enc_uint.SubstrateScaleU16Encoder'>, 32:
<class
'bip_utils.substrate.scale.substrate_scale_enc_uint.SubstrateScaleU32Encoder'>, 64:
<class
'bip_utils.substrate.scale.substrate_scale_enc_uint.SubstrateScaleU64Encoder'>, 128:
<class
'bip_utils.substrate.scale.substrate_scale_enc_uint.SubstrateScaleU128Encoder'>,
256: <class
'bip_utils.substrate.scale.substrate_scale_enc_uint.SubstrateScaleU256Encoder'>}
```

```
class SubstratePathElem(elem: str)
```

Bases: object

Substrate path element. It represents a Substrate path element.

m_elem: str

m_is_hard: bool

IsHard() → bool

Get if the element is hard.

Returns

True if hard, false otherwise

Return type

bool

IsSoft() → bool

Get if the element is soft.

Returns

True if soft, false otherwise

Return type

bool

ChainCode() → bytes

Return the chain code.

Returns

Chain code

Return type

bytes

ToStr() → str

Get the path element as a string.

Returns

Path element as a string

Return type

str

__str__() → str

Get the path element as a string.

Returns

Path element as a string

Return type

str

class SubstratePath(elems: Optional[Sequence[Union[str, SubstratePathElem]]] = None)

Bases: object

Substrate path. It represents a Substrate path.

m_elems: List[SubstratePathElem]

AddElem(elem: Union[str, SubstratePathElem]) → SubstratePath

Return a new path object with the specified element added.

Parameters

elem (str or SubstratePathElem) – Path element

Returns

SubstratePath object

Return type

SubstratePath object

Raises

SubstratePathError – If the path element is not valid

Length() → int

Get the number of elements of the path.

Returns

Number of elements

Return type

int

ToList() → List[str]

Get the path as a list of strings.

Returns

Path as a list of strings

Return type

list[str]

ToStr() → str

Get the path as a string.

Returns

Path as a string

Return type

str

__str__() → str

Get the path as a string.

Returns

Path as a list of integers

Return type

str

__getitem__(*idx: int*) → *SubstratePathElem*

Get the specified element index.

Parameters**idx** (*int*) – Element index**Returns**

SubstratePathElem object

Return type

SubstratePathElem object

__iter__() → Iterator[*SubstratePathElem*]

Get the iterator to the current element.

Returns

Iterator to the current element

Return type

Iterator object

class SubstratePathParser

Bases: object

Substrate path parser. It parses a Substrate path and returns a SubstratePath object.

static Parse(*path: str*) → *SubstratePath*

Parse a path and return a SubstratePath object.

Parameters**path** (*str*) – Path**Returns**

SubstratePath object

Return type

SubstratePath object

Raises*SubstratePathError* – If the path element is not valid

10.1.16 utils

10.1.16.1 conf

10.1.16.1.1 coin_names

Module with helper class for coin names.

class CoinNames(*name: str, abbr: str*)

Bases: object

Helper class for representing coin names.

m_name: str

m_abbr: str

Name() → str

Get name.

Returns :

str: Name

Abbreviation() → str

Get abbreviation.

Returns

Abbreviation

Return type

str

10.1.16.2 crypto

10.1.16.2.1 aes_ecb

Module for AES-ECB encryption/decryption.

class **AesEcbEncrypter**(*key: Union[str, bytes]*)

Bases: object

AES-ECB encrypter class. It encrypts data using AES-ECB algorithm.

aes: Any

auto_pad: bool

AutoPad(*value: bool*) → None

Set the auto-pad flag.

Parameters

value (*bool*) – Flag value

Encrypt(*data: Union[str, bytes]*) → bytes

Encrypt data using AES-ECB algorithm.

Parameters

data (*str or bytes*) – Data to be encrypted

Returns

Encrypted data

Return type

bytes

static Pad(*data: Union[str, bytes]*) → bytes

Pad data using PKCS7 algorithm.

Parameters

data (*str or bytes*) – Data to be padded

Returns

Padded data

Return type

bytes

class **AesEcbDecrypter**(*key: Union[str, bytes]*)

Bases: object

AES-ECB decrypter class. It decrypts data using AES-ECB algorithm.

aes: **Any****AutoUnPad**(*value: bool*) → None

Set the auto-unpad flag.

Parameters**value** (*bool*) – Flag value**Decrypt**(*data: bytes*) → bytes

Decrypt data using AES-ECB algorithm.

Parameters**data** (*bytes*) – Data to be decrypted**Returns**

Decrypted data

Return type

bytes

static UnPad(*data: bytes*) → bytes

Unpad data using PKCS7 algorithm.

Parameters**data** (*bytes*) – Data to be unpadded**Returns**

Unpadded data

Return type

bytes

10.1.16.2.2 blake2

Module for BLAKE-2 algorithms.

class **Blake2b**

Bases: object

BLAKE2b class. It computes digests using BLAKE2b algorithm.

static QuickDigest(*data: Union[bytes, str], digest_size: int, key: Union[bytes, str] = b'', salt: Union[bytes, str] = b''*) → bytes

Compute the digest (quick version).

Parameters

- **data** (*str or bytes*) – Data
- **digest_size** (*int*) – Digest size
- **key** (*((str or bytes, optional)*) – Key (default: empty)
- **salt** (*((str or bytes, optional)*) – Salt (default: empty)

Returns

Computed digest

Return type

bytes

class Blake2b32Bases: `_Blake2bWithSpecificSize`

BLAKE2b-32 class. It computes digests using BLAKE2b-32 algorithm.

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

class Blake2b40Bases: `_Blake2bWithSpecificSize`

BLAKE2b-40 class. It computes digests using BLAKE2b-40 algorithm.

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

class Blake2b160Bases: `_Blake2bWithSpecificSize`

BLAKE2b-160 class. It computes digests using BLAKE2b-160 algorithm.

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

class Blake2b224Bases: `_Blake2bWithSpecificSize`

BLAKE2b-224 class. It computes digests using BLAKE2b-224 algorithm.

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

class Blake2b256

Bases: `_Blake2bWithSpecificSize`

BLAKE2b-256 class. It computes digests using BLAKE2b-256 algorithm.

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

class Blake2b512

Bases: `_Blake2bWithSpecificSize`

BLAKE2b-512 class. It computes digests using BLAKE2b-512 algorithm.

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

10.1.16.2.3 chacha20_poly1305

Module for ChaCha20-Poly1305 algorithm.

class ChaCha20Poly1305

Bases: `object`

ChaCha20-Poly1305 class. It decrypts/encrypts data using ChaCha20-Poly1305 algorithm.

static Decrypt(*key: Union[bytes, str], nonce: Union[bytes, str], assoc_data: Union[bytes, str], cipher_text: Union[bytes, str], tag: Union[bytes, str]*) → bytes

Decrypt data.

Parameters

- **key** (*str or bytes*) – Key
- **nonce** (*str or bytes*) – Nonce
- **assoc_data** (*str or bytes*) – Associated data
- **cipher_text** (*bytes*) – Cipher text
- **tag** (*bytes*) – Tag

Returns

Decrypted data

Return type

bytes

static Encrypt(*key: Union[bytes, str], nonce: Union[bytes, str], assoc_data: Union[bytes, str], plain_text: Union[bytes, str]*) → Tuple[bytes, bytes]

Encrypt data.

Parameters

- **key** (*str or bytes*) – Key
- **nonce** (*str or bytes*) – Nonce
- **assoc_data** (*str or bytes*) – Associated data
- **plain_text** (*str or bytes*) – Plain text

Returns

Cipher text bytes (index 0) and tag bytes (index 1)

Return type

tuple[bytes, bytes]

static KeySize() → int

Get the key size.

Returns

Key size

Return type

int

static TagSize() → int

Get the tag size.

Returns

Tag size

Return type

int

10.1.16.2.4 crc

Module for CRC algorithms.

class Crc32

Bases: object

CRC32 class. It computes digests using CRC32 algorithm.

static QuickDigest(*data: Union[bytes, str]*) → bytes

Compute the digest (quick version).

Parameters

data (*str or bytes*) – Data

Returns

Computed digest

Return type

bytes

static QuickIntDigest(*data: Union[bytes, str]*) → int

Compute the digest as integer (quick version).

Parameters

data (*str or bytes*) – Data

Returns

Computed digest

Return type

bytes

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

class XModemCrc

Bases: object

XMODEM-CRC class. It computes digests using XMODEM-CRC algorithm.

static QuickDigest(*data: Union[bytes, str]*) → bytes

Compute the digest (quick version).

Parameters

data (*str or bytes*) – Data

Returns

Computed digest

Return type

bytes

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

10.1.16.2.5 hash160

Module for HASH160 algorithm.

class Hash160

Bases: object

HASH160 class. It computes digests using HASH160 algorithm.

static QuickDigest(*data: Union[bytes, str]*) → bytes

Compute the digest (quick version).

Parameters

data (*str or bytes*) – Data

Returns

Computed digest

Return type

bytes

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

10.1.16.2.6 hmac

Module for SHA-2 algorithms.

class HmacSha256

Bases: object

HMAC-SHA256 class. It computes digests using HMAC-SHA256 algorithm.

static QuickDigest(*key: Union[bytes, str], data: Union[bytes, str]*) → bytes

Compute the digest (quick version).

Parameters

- **key** (*str or bytes*) – Key
- **data** (*str or bytes*) – Data

Returns

Computed digest

Return type

bytes

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

class HmacSha512

Bases: object

HMAC-SHA512 class. It computes digests using HMAC-SHA512 algorithm.

static QuickDigest(*key: Union[bytes, str], data: Union[bytes, str]*) → bytes

Compute the digest (quick version).

Parameters

- **key** (*str or bytes*) – Key
- **data** (*str or bytes*) – Data

Returns

Computed digest

Return type

bytes

static QuickDigestHalves(*key: Union[bytes, str], data: Union[bytes, str]*) → Tuple[bytes, bytes]

Compute the digest and return it split into two halves (quick version).

Parameters

- **key** (*str or bytes*) – Key
- **data** (*str or bytes*) – Data

Returns

Computed digest left part (index 0) and right part (index 1)

Return type

tuple[bytes, bytes]

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

10.1.16.2.7 pbkdf2

Module for PBKDF2 algorithm.

class Pbkdf2HmacSha512

Bases: object

PBKDF2 HMAC-SHA512 class. It derives keys using PBKDF2 HMAC-SHA512 algorithm.

static DeriveKey(*password: Union[bytes, str], salt: Union[bytes, str], itr_num: int, dklen: Optional[int] = None*) → bytes

Derive a key.

Parameters

- **password** (*str or bytes*) – Password
- **salt** (*str or bytes*) – Salt
- **itr_num** (*int*) – Iteration number
- **dklen** (*int, optional*) – Length of the derived key (default: SHA-512 output length)

Returns

Computed result

Return type

bytes

10.1.16.2.8 ripemd

Module for RIPEMD algorithm.

class Ripemd160

Bases: object

RIPEMD160 class. It computes digests using RIPEMD160 algorithm.

static QuickDigest(*data: Union[bytes, str]*) → bytes

Compute the digest (quick version).

Parameters

data (*str or bytes*) – Data

Returns

Computed digest

Return type

bytes

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

10.1.16.2.9 scrypt

Module for Scrypt algorithm.

class Scrypt

Bases: object

Scrypt class. It derives key using Scrypt algorithm.

static DeriveKey(*password: Union[bytes, str], salt: Union[bytes, str], key_len: int, n: int, r: int, p: int*) → bytes

Derive a key.

Parameters

- **password** (*str or bytes*) – Password
- **salt** (*str or bytes*) – Salt
- **key_len** (*int*) – Length of the derived key
- **n** (*int*) – CPU/Memory cost parameter
- **r** (*int*) – Block size parameter
- **p** (*int*) – Parallelization parameter

Returns

Computed result

Return type

bytes

10.1.16.2.10 sha2

Module for SHA-2 algorithms.

class Sha256

Bases: object

SHA256 class. It computes digests using SHA256 algorithm.

handle: Any

Update(*data_bytes: bytes*) → None

Update digest.

Parameters

data_bytes (*bytes*) – Data bytes

Digest() → bytes

Get the computed digest.

Returns

Computed digest

Return type

bytes

static QuickDigest(*data: Union[bytes, str]*) → bytes

Compute the digest (quick version).

Parameters

data (*str or bytes*) – Data

Returns

Computed digest

Return type

bytes

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

class DoubleSha256

Bases: object

Double SHA256 class. It computes digests using SHA256 algorithm twice.

static QuickDigest(*data: Union[bytes, str]*) → bytes

Compute the digest (quick version).

Parameters

data (*str or bytes*) – Data

Returns

Computed digest

Return type

bytes

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

class Sha512

Bases: object

SHA512 class. It computes digests using SHA512 algorithm.

static QuickDigest(*data: Union[bytes, str]*) → bytes

Compute the digest (quick version).

Parameters**data** (*str or bytes*) – Data**Returns**

Computed digest

Return type

bytes

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

class Sha512_256

Bases: object

SHA512/256 class. It computes digests using SHA512/256 algorithm.

static QuickDigest(*data: Union[bytes, str]*) → bytes

Compute the digest (quick version).

Parameters**data** (*str or bytes*) – Data**Returns**

Computed digest

Return type

bytes

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

10.1.16.2.11 sha3

Module for SHA-3 algorithms.

class Kekkak256

Bases: object

Kekkak-256 class. It computes digests using Kekkak-256 algorithm.

static QuickDigest(*data: Union[bytes, str]*) → bytes

Compute the digest (quick version).

Parameters

data (*str or bytes*) – Data

Returns

Computed digest

Return type

bytes

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

class Sha3_256

Bases: object

SHA3-256 class. It computes digests using SHA3-256 algorithm.

static QuickDigest(*data: Union[bytes, str]*) → bytes

Compute the digest (quick version).

Parameters

data (*str or bytes*) – Data

Returns

Computed digest

Return type

bytes

static DigestSize() → int

Get the digest size in bytes.

Returns

Digest size in bytes

Return type

int

10.1.16.3 misc

10.1.16.3.1 algo

Module with some algorithm utility functions.

class AlgoUtils

Bases: object

Class container for algorithm utility functions.

static **BinarySearch**(*arr: List, elem: Any*) → int

Binary search algorithm simply implemented by using the bisect library.

Parameters

- **arr** (*list*) – list of elements
- **elem** (*any*) – element to be searched

Returns

First index of the element, -1 if not found

Return type

int

static **Decode**(*data: Union[bytes, str], encoding: str = 'utf-8'*) → str

Decode from bytes.

Parameters

- **data** (*str or bytes*) – Data
- **encoding** (*str*) – Encoding type

Returns

String encoded to bytes

Return type

str

Raises

TypeError – If the data is neither string nor bytes

static **Encode**(*data: Union[bytes, str], encoding: str = 'utf-8'*) → bytes

Encode to bytes.

Parameters

- **data** (*str or bytes*) – Data
- **encoding** (*str*) – Encoding type

Returns

String encoded to bytes

Return type

bytes

Raises

TypeError – If the data is neither string nor bytes

static IsStringMixed(*data_str: str*) → bool

Get if the specified string is in mixed case.

Parameters

data_str (*str*) – string

Returns

True if mixed case, false otherwise

Return type

bool

10.1.16.3.2 base32

Module with helper class for Base32.

class Base32Const

Bases: object

Class container for Base32 constants.

ALPHABET: str = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ234567'

PADDING_CHAR: str = '='

class Base32Decoder

Bases: object

Base32 decoder class. It provides methods for decoding to Base32 format.

static Decode(*data: str, custom_alphabet: Optional[str] = None*) → bytes

Decode from Base32.

Parameters

- **data** (*str*) – Data
- **custom_alphabet** (*str, optional*) – Custom alphabet string

Returns

Decoded bytes

Return type

bytes

Raises

ValueError – If the Base32 string is not valid

class Base32Encoder

Bases: object

Base32 encoder class. It provides methods for encoding to Base32 format.

static Encode(*data: Union[bytes, str], custom_alphabet: Optional[str] = None*) → str

Encode to Base32.

Parameters

- **data** (*str or bytes*) – Data
- **custom_alphabet** (*str, optional*) – Custom alphabet string

Returns

Encoded string

Return type

str

static EncodeNoPadding(*data: Union[bytes, str], custom_alphabet: Optional[str] = None*) → str

Encode to Base32 by removing the final padding.

Parameters

- **data** (*str or bytes*) – Data
- **custom_alphabet** (*str, optional*) – Custom alphabet string

Returns

Encoded string

Return type

str

10.1.16.3.3 bit

Module with some bits utility functions.

class BitUtils

Bases: object

Class container for bit utility functions.

static IsBitSet(*value: int, bit_num: int*) → bool

Get if the specified bit is set.

Parameters

- **value** (*int*) – Value
- **bit_num** (*int*) – Bit number to check

Returns

True if bit is set, false otherwise

Return type

bool

static AreBitsSet(*value: int, bit_mask: int*) → bool

Get if the specified bits are set.

Parameters

- **value** (*int*) – Value
- **bit_mask** (*int*) – Bit mask to check

Returns

True if bit is set, false otherwise

Return type

bool

static SetBit(*value: int, bit_num: int*) → int

Set the specified bit.

Parameters

- **value** (*int*) – Value
- **bit_num** (*int*) – Bit number to set

Returns

Value with the specified bit set

Return type

int

static SetBits(*value: int, bit_mask: int*) → int

Set the specified bits.

Parameters

- **value** (*int*) – Value
- **bit_mask** (*int*) – Bit mask to set

Returns

Value with the specified bit set

Return type

int

static ResetBit(*value: int, bit_num: int*) → int

Reset the specified bit.

Parameters

- **value** (*int*) – Value
- **bit_num** (*int*) – Bit number to reset

Returns

Value with the specified bit reset

Return type

int

static ResetBits(*value: int, bit_mask: int*) → int

Reset the specified bits.

Parameters

- **value** (*int*) – Value
- **bit_mask** (*int*) – Bit mask to reset

Returns

Value with the specified bit reset

Return type

int

10.1.16.3.4 bytes

Module with some bytes utility functions.

class BytesUtils

Bases: object

Class container for bytes utility functions.

static Reverse(*data_bytes: bytes*) → bytes

Reverse the specified bytes.

Parameters

data_bytes (*bytes*) – Data bytes

Returns

Original bytes in the reverse order

Return type

bytes

static Xor(*data_bytes_1: bytes, data_bytes_2: bytes*) → bytes

XOR the specified bytes.

Parameters

- **data_bytes_1** (*bytes*) – Data bytes 1
- **data_bytes_2** (*bytes*) – Data bytes 2

Returns

XORed bytes

Return type

bytes

static AddNoCarry(*data_bytes_1: bytes, data_bytes_2: bytes*) → bytes

Add the specified bytes (byte-by-byte, no carry).

Parameters

- **data_bytes_1** (*bytes*) – Data bytes 1
- **data_bytes_2** (*bytes*) – Data bytes 2

Returns

XORed bytes

Return type

bytes

static MultiplyScalarNoCarry(*data_bytes: bytes, scalar: int*) → bytes

Multiply the specified bytes with the specified scalar (byte-by-byte, no carry).

Parameters

- **data_bytes** (*bytes*) – Data bytes
- **scalar** (*int*) – Scalar

Returns

XORed bytes

Return type

bytes

static ToBinaryStr(*data_bytes: bytes, zero_pad_bit_len: int = 0*) → str

Convert the specified bytes to a binary string.

Parameters

- **data_bytes** (*bytes*) – Data bytes
- **zero_pad_bit_len** (*int, optional*) – Zero pad length in bits, 0 if not specified

Returns

Binary string

Return type

str

static ToInteger(*data_bytes: bytes, endianness: typing_extensions.Literal[*little*, *big*] = 'big', signed: bool = False*) → int

Convert the specified bytes to integer.

Parameters

- **data_bytes** (*bytes*) – Data bytes
- **endianness** (*"big" or "little", optional*) – Endianness (default: big)
- **signed** (*bool, optional*) – True if signed, false otherwise (default: false)

Returns

Integer representation

Return type

int

static FromBinaryStr(*data: Union[bytes, str], zero_pad_byte_len: int = 0*) → bytes

Convert the specified binary string to bytes.

Parameters

- **data** (*str or bytes*) – Data
- **zero_pad_byte_len** (*int, optional*) – Zero pad length in bytes, 0 if not specified

Returns

Bytes representation

Return type

bytes

static ToHexString(*data_bytes: bytes, encoding: str = 'utf-8'*) → str

Convert bytes to hex string.

Parameters

- **data_bytes** (*bytes*) – Data bytes
- **encoding** (*str, optional*) – Encoding type, utf-8 by default

Returns

Bytes converted to hex string

Return type

str

static FromHexString(*data: Union[bytes, str]*) → bytes

Convert hex string to bytes.

Parameters

data (*str or bytes*) – Data bytes

Returns

bytes: Hex string converted to bytes

static FromList(*data_list: List[int]*) → bytes

Convert the specified list of integers to bytes.

Parameters

data_list (*list[int]*) – List of integers

Returns

Bytes representation

Return type

bytes

static ToList(*data_bytes: bytes*) → List[int]

Convert the specified bytes to a list of integers.

Parameters

data_bytes (*bytes*) – Data bytes

Returns

List of integers

Return type

list[int]

10.1.16.3.5 cbor_indefinite_len_array

Module for CBOR decoding/encoding indefinite length arrays. Indefinite length arrays are encoded without writing the array length, so elements shall be read until the termination byte is found.

NOTE: encoding of values greater than 2^{64} is not supported.

class CborIds(*value*)

Bases: IntEnum

Enumerative for CBOR identifiers.

UINT8 = 24

UINT16 = 25

UINT32 = 26

UINT64 = 27

INDEF_LEN_ARRAY_START = 159

INDEF_LEN_ARRAY_END = 255

class CborIndefiniteLenArrayConst

Bases: object

Class container for CBOR indefinite length arrays constants.

UINT_IDS_TO_BYTE_LEN: Dict[int, int] = {CborIds.UINT8: 2, CborIds.UINT16: 3, CborIds.UINT32: 5, CborIds.UINT64: 9}

class CborIndefiniteLenArrayDecoder

Bases: object

CBOR indefinite length arrays decoder. It decodes bytes back to array.

static Decode(*enc_bytes: bytes*) → List[int]

CBOR-decode the specified bytes.

Parameters

enc_bytes (*bytes*) – Encoded bytes

Returns

List of integers

Return type

list[int]

Raises

ValueError – If encoding is not valid

class CborIndefiniteLenArrayEncoder

Bases: object

CBOR indefinite length arrays encoder. It encodes indefinite length arrays to bytes.

static Encode(*int_seq: Sequence[int]*) → bytes

CBOR-encode the specified elements.

Parameters

int_seq (*sequence[int]*) – Collection of integers

Returns

CBOR-encoded bytes

Return type

bytes

10.1.16.3.6 data_bytes

Module with helper class for data bytes.

class DataBytes(*data_bytes: bytes*)

Bases: object

Data bytes class. It allows to get bytes in different formats.

m_data_bytes: bytes

Length() → int

Get length in bytes.

Returns

Length in bytes

Return type

int

Size() → int

Get length in bytes (same of Length()).

Returns

Length in bytes

Return type

int

ToBytes() → bytes

Get data bytes.

Returns

Data bytes

Return type

bytes

ToHex() → str

Get data bytes in hex format.

Returns

Data bytes in hex format

Return type

str

ToInt(*endianness: typing_extensions.Literal[*little*, *big*] = 'big'*) → int

Get data bytes as an integer.

Parameters**endianness** ("*big*" or "*little*", optional) – Endianness (default: big)**Returns**

Data bytes as an integer

Return type

int

__len__() → int

Get length in bytes.

Returns

Length in bytes

Return type

int

__bytes__() → bytes

Get data bytes.

Returns

Data bytes

Return type

bytes

__int__() → int

Get data bytes as integer.

Returns

Data bytes as integer

Return type

bytes

__repr__() → str

Get data bytes representation.

Returns

Data bytes representation

Return type

str

__getitem__(*idx: int*) → int

Get the element with the specified index.

Parameters

idx (*int*) – Index

Returns

Element

Return type

int

Raises

IndexError – If the index is not valid

__iter__() → Iterator[int]

Get the iterator to the current element.

Returns

Iterator to the current element

Return type

Iterator object

__eq__(*other: object*) → bool

Equality operator.

Parameters

other (*bytes, str, int or DataBytes object*) – Other object to compare

Returns

True if equal false otherwise

Return type

bool

Raises

TypeError – If the other object is not of the correct type

10.1.16.3.7 integer

Module with some integer utility functions.

class IntegerUtils

Bases: object

Class container for integer utility functions.

static **GetBytesNumber**(*data_int: int*) → int

Get the number of bytes of the specified integer.

Parameters

data_int (*int*) – Data integer

Returns

Number of bytes

Return type

int

static **ToBytes**(*data_int: int, bytes_num: Optional[int] = None, endianness: typing_extensions.Literal[*little*, *big*] = 'big', signed: bool = False*) → bytes

Convert integer to bytes.

Parameters

- **data_int** (*int*) – Data integer
- **bytes_num** (*int, optional*) – Number of bytes, automatic if None
- **endianness** ("*big*" or "*little*", *optional*) – Endianness (default: big)
- **signed** (*bool, optional*) – True if signed, false otherwise (default: false)

Returns

Bytes representation

Return type

bytes

static **FromBinaryStr**(*data: Union[bytes, str]*) → int

Convert the specified binary string to integer.

Parameters

data (*str or bytes*) – Data

Returns

Integer representation

Return type

int

static **ToBinaryStr**(*data_int: int, zero_pad_bit_len: int = 0*) → str

Convert the specified integer to a binary string.

Parameters

- **data_int** (*int*) – Data integer
- **zero_pad_bit_len** (*int, optional*) – Zero pad length in bits, 0 if not specified

Returns

Binary string

Return type

str

10.1.16.3.8 string

Module with some string utility functions.

class StringUtils

Bases: object

Class container for string utility functions.

static NormalizeNfc(*data_str: str*) → str

Normalize string using NFC.

Parameters

data_str (*str*) – Input string

Returns

Normalized string

Return type

str

static NormalizeNfkd(*data_str: str*) → str

Normalize string using NFKD.

Parameters

data_str (*str*) – Input string

Returns

Normalized string

Return type

str

10.1.16.4 mnemonic**10.1.16.4.1 entropy_generator**

Module for generic entropy generator.

class EntropyGenerator(*bit_len: int*)

Bases: object

Entropy generator class. It generates random entropy bytes with the specified length.

m_bit_len: int

Generate() → bytes

Generate random entropy bytes.

Returns

Generated entropy bytes

Return type

bytes

10.1.16.4.2 mnemonic

Module containing common classes for mnemonic.

class MnemonicLanguages(*value*)

Bases: Enum

Base enum for mnemonic languages.

class Mnemonic(*mnemonic_list: List[str]*)

Bases: object

Mnemonic class. It represents a generic mnemonic phrase. It acts as a simple container with some helper functions, so it doesn't validate the given mnemonic.

classmethod FromString(*mnemonic_str: str*) → *Mnemonic*

Create a class from mnemonic string.

Parameters

mnemonic_str (*str*) – Mnemonic string

Returns

Mnemonic object

Return type

Mnemonic

classmethod FromList(*mnemonic_list: List[str]*) → *Mnemonic*

Create a class from mnemonic list.

Parameters

mnemonic_list (*list[str]*) – Mnemonic list

Returns

Mnemonic object

Return type

Mnemonic

m_mnemonic_list: List[str]

WordsCount() → int

Get the words count.

Returns

Words count

Return type

int

ToList() → List[str]

Get the mnemonic as a list.

Returns

Mnemonic as a list

Return type

list[str]

ToStr() → str

Get the mnemonic as a string.

Returns

Mnemonic as a string

Return type

str

__str__() → str

Get the mnemonic as a string.

Returns

Mnemonic as a string

Return type

str

10.1.16.4.3 mnemonic_decoder_base

Module for mnemonic decoder base class.

```
class MnemonicDecoderBase(lang: Optional[MnemonicLanguages], words_list_finder_cls:
    Type[MnemonicWordsListFinderBase], words_list_getter_cls:
    Type[MnemonicWordsListGetterBase])
```

Bases: ABC

Mnemonic decoder base class. It decodes a mnemonic phrase to bytes.

m_lang: Optional[MnemonicLanguages]

m_words_list: Optional[MnemonicWordsList]

m_words_list_finder_cls: Type[MnemonicWordsListFinderBase]

abstract Decode(mnemonic: Union[str, Mnemonic]) → bytes

Decode a mnemonic phrase to bytes (no checksum).

Parameters

mnemonic (str or Mnemonic object) – Mnemonic

Returns

Decoded bytes (no checksum)

Return type

bytes

Raises

- **MnemonicChecksumError** – If checksum is not valid
- **ValueError** – If mnemonic is not valid

10.1.16.4.4 mnemonic_encoder_base

Module for mnemonic encoder base class.

```
class MnemonicEncoderBase(lang: MnemonicLanguages, words_list_getter_cls:
                          Type[MnemonicWordsListGetterBase])
```

Bases: ABC

Mnemonic encoder base class. It encodes bytes to the mnemonic phrase.

m_words_list: [MnemonicWordsList](#)

abstract Encode(*entropy_bytes*: bytes) → [Mnemonic](#)

Encode bytes to mnemonic phrase.

Parameters

entropy_bytes (bytes) – Entropy bytes

Returns

Encoded mnemonic

Return type

Mnemonic object

Raises

ValueError – If entropy is not valid

10.1.16.4.5 mnemonic_ex

Module for mnemonic exceptions.

exception MnemonicChecksumError

Bases: Exception

Exception in case of checksum error.

10.1.16.4.6 mnemonic_utils

Module containing common utility classes for mnemonic.

class MnemonicUtils

Bases: object

Class container for mnemonic utility functions.

static BytesChunkToWords(*bytes_chunk*: bytes, *words_list*: [MnemonicWordsList](#), *endianness*:
typing_extensions.Literal[*little*, *big*]) → List[str]

Get words from a bytes chunk.

Parameters

- **bytes_chunk** (bytes) – Bytes chunk
- **words_list** ([MnemonicWordsList](#) object) – Mnemonic list
- **endianness** ("big" or "little") – Bytes endianness

Returns

3 word indexes

Return type

list[str]

static WordsToBytesChunk(*word1: str, word2: str, word3: str, words_list: MnemonicWordsList, endianness: typing_extensions.Literal[*little, big*]*) → bytes

Get bytes chunk from words.

Parameters

- **word1** (*str*) – Word 1
- **word2** (*str*) – Word 2
- **word3** (*str*) – Word 3
- **words_list** (*MnemonicWordsList* object) – Mnemonic list
- **endianness** ("*big*" or "*little*") – Bytes endianness

Returns

Bytes chunk

Return type

bytes

class MnemonicWordsList(*words_list: List[str]*)

Bases: object

Mnemonic words list class.

m_idx_to_words: List[str]

m_words_to_idx: Dict[str, int]

Length() → int

Get the length of the words list.

Returns

Words list length

Return type

int

GetWordIdx(*word: str*) → int

Get the index of the specified word.

Parameters

word (*str*) – Word to be searched

Returns

Word index

Return type

int

Raises

ValueError – If the word is not found

GetWordAtIdx(*word_idx: int*) → str

Get the word at the specified index.

Parameters

word_idx (*int*) – Word index

Returns

Word at the specified index

Return type

str

class MnemonicWordsListFileReader

Bases: object

Mnemonic words list file reader class. It reads the words list from a file.

static LoadFile(*file_path: str, words_num: int*) → *MnemonicWordsList*

Load words list file correspondent to the specified language.

Parameters

- **file_path** (*str*) – File name
- **words_num** (*int*) – Number of expected words

Returns

MnemonicWordsList object

Return type

MnemonicWordsList

Raises

ValueError – If loaded words list is not valid

class MnemonicWordsListGetterBase

Bases: ABC

Mnemonic words list getter base class.

m_words_lists: Dict[*MnemonicLanguages*, *MnemonicWordsList*]

abstract GetByLanguage(*lang: MnemonicLanguages*) → *MnemonicWordsList*

Get words list by language. Words list of a specific language are loaded from file only the first time they are requested.

Parameters

lang (*MnemonicLanguages*) – Language

Returns

MnemonicWordsList object

Return type

MnemonicWordsList object

Raises

- **TypeError** – If the language is not of the correct enumerative
- **ValueError** – If loaded words list is not valid

classmethod Instance() → *MnemonicWordsListGetterBase*

Get the global class instance.

Returns

MnemonicWordsListGetterBase object

Return type

MnemonicWordsListGetterBase object

class MnemonicWordsListFinderBase

Bases: ABC

Mnemonic words list finder base class. It automatically finds the correct words list from a mnemonic.

abstract classmethod FindLanguage(*mnemonic*: [Mnemonic](#)) → Tuple[[MnemonicWordsList](#),
[MnemonicLanguages](#)]

Automatically find the language of the specified mnemonic and get the correct MnemonicWordsList class for it.

Parameters

mnemonic (*Mnemonic object*) – Mnemonic object

Returns

MnemonicWordsList object (index 0), mnemonic language (index 1)

Return type

tuple[[MnemonicWordsList](#), [MnemonicLanguages](#)]

Raises

ValueError – If the mnemonic language cannot be found

10.1.16.4.7 mnemonic_validator

Module for generic mnemonic validation.

class MnemonicValidator(*mnemonic_decoder*: [MnemonicDecoderBase](#))

Bases: object

Mnemonic validator class.

m_mnemonic_decoder: [MnemonicDecoderBase](#)

Validate(*mnemonic*: Union[str, [Mnemonic](#)]) → None

Validate the mnemonic specified at construction.

Parameters

mnemonic (*str or Mnemonic object*) – Mnemonic

Raises

- [MnemonicChecksumError](#) – If checksum is not valid
- **ValueError** – If mnemonic is not valid

IsValid(*mnemonic*: Union[str, [Mnemonic](#)]) → bool

Get if the mnemonic specified at construction is valid.

Parameters

mnemonic (*str or Mnemonic object*) – Mnemonic

Returns

True if valid, False otherwise

Return type

bool

10.1.16.5 typing

10.1.16.5.1 literal

Module with Literal type definition.

10.1.17 wif

10.1.17.1 wif

Module for WIF encoding/decoding.

class WifConst

Bases: object

Class container for WIF constants.

COMPR_PUB_KEY_SUFFIX: bytes = b'\x01'

class WifEncoder

Bases: object

WIF encoder class. It provides methods for encoding to WIF format.

static Encode(*priv_key*: Union[bytes, IPrivateKey], *net_ver*: bytes = b'\x80', *pub_key_mode*: P2PKHPubKeyModes = P2PKHPubKeyModes.COMPRESSED) → str

Encode key bytes into a WIF string.

Parameters

- **priv_key** (bytes or IPrivateKey) – Private key bytes or object
- **net_ver** (bytes, optional) – Net version (Bitcoin main net by default)
- **pub_key_mode** (WifPubKeyModes, optional) – Specify if the private key corresponds to a compressed public key

Returns

WIF encoded string

Return type

str

Raises

- **TypeError** – If *pub_key_mode* is not a WifPubKeyModes enum or the private key is not a valid Secp256k1PrivateKey
- **ValueError** – If the key is not valid

class WifDecoder

Bases: object

WIF encoder class. It provides methods for encoding to WIF format.

static Decode(*wif_str*: str, *net_ver*: bytes = b'\x80') → Tuple[bytes, P2PKHPubKeyModes]

Decode key bytes from a WIF string.

Parameters

- **wif_str** (str) – WIF string

- **net_ver** (*bytes*, *optional*) – Net version (Bitcoin main net by default)

Returns

Key bytes (index 0), public key mode (index 1)

Return type

tuple[bytes, WifPubKeyModes]

Raises

- **Base58ChecksumError** – If the base58 checksum is not valid
- **ValueError** – If the resulting key is not valid

PYTHON MODULE INDEX

b

`bip_utils.addr.ada_byron_addr`, 27
`bip_utils.addr.ada_shelley_addr`, 30
`bip_utils.addr.addr_dec_utils`, 32
`bip_utils.addr.addr_key_validator`, 34
`bip_utils.addr.algo_addr`, 36
`bip_utils.addr.aptos_addr`, 37
`bip_utils.addr.atom_addr`, 38
`bip_utils.addr.avax_addr`, 39
`bip_utils.addr.bch_addr_converter`, 40
`bip_utils.addr.egld_addr`, 41
`bip_utils.addr.eos_addr`, 42
`bip_utils.addr.ergo_addr`, 43
`bip_utils.addr.eth_addr`, 44
`bip_utils.addr.fil_addr`, 45
`bip_utils.addr.iaddr_decoder`, 46
`bip_utils.addr.iaddr_encoder`, 47
`bip_utils.addr.icx_addr`, 47
`bip_utils.addr.inj_addr`, 48
`bip_utils.addr.nano_addr`, 49
`bip_utils.addr.near_addr`, 50
`bip_utils.addr.neo_addr`, 51
`bip_utils.addr.okex_addr`, 52
`bip_utils.addr.one_addr`, 53
`bip_utils.addr.P2PKH_addr`, 21
`bip_utils.addr.P2SH_addr`, 23
`bip_utils.addr.P2TR_addr`, 25
`bip_utils.addr.P2WPKH_addr`, 26
`bip_utils.addr.sol_addr`, 54
`bip_utils.addr.substrate_addr`, 55
`bip_utils.addr.sui_addr`, 56
`bip_utils.addr.trx_addr`, 57
`bip_utils.addr.xlm_addr`, 58
`bip_utils.addr.xmr_addr`, 60
`bip_utils.addr.xrp_addr`, 62
`bip_utils.addr.xtz_addr`, 63
`bip_utils.addr.zil_addr`, 64
`bip_utils.algorand.mnemonic.algorand_entropy_generator`, 65
`bip_utils.algorand.mnemonic.algorand_mnemonic`, 66
`bip_utils.algorand.mnemonic.algorand_mnemonic_decoder`, 66
`bip_utils.algorand.mnemonic.algorand_mnemonic_encoder`, 67
`bip_utils.algorand.mnemonic.algorand_mnemonic_generator`, 67
`bip_utils.algorand.mnemonic.algorand_mnemonic_utils`, 68
`bip_utils.algorand.mnemonic.algorand_mnemonic_validator`, 69
`bip_utils.algorand.mnemonic.algorand_seed_generator`, 69
`bip_utils.base58.base58`, 70
`bip_utils.base58.base58_ex`, 72
`bip_utils.base58.base58_xmr`, 72
`bip_utils.bech32.bch_bech32`, 73
`bip_utils.bech32.bech32`, 75
`bip_utils.bech32.bech32_base`, 77
`bip_utils.bech32.bech32_ex`, 78
`bip_utils.bech32.segwit_bech32`, 79
`bip_utils.bip.bip32.base.bip32_base`, 80
`bip_utils.bip.bip32.base.ibip32_key_derivator`, 85
`bip_utils.bip.bip32.base.ibip32_mst_key_generator`, 86
`bip_utils.bip.bip32.bip32_const`, 86
`bip_utils.bip.bip32.bip32_ex`, 87
`bip_utils.bip.bip32.bip32_key_data`, 87
`bip_utils.bip.bip32.bip32_key_net_ver`, 93
`bip_utils.bip.bip32.bip32_key_ser`, 94
`bip_utils.bip.bip32.bip32_keys`, 96
`bip_utils.bip.bip32.bip32_path`, 99
`bip_utils.bip.bip32.bip32_utils`, 101
`bip_utils.bip.bip32.kholaw.bip32_kholaw_ed25519`, 102
`bip_utils.bip.bip32.kholaw.bip32_kholaw_ed25519_key_derivator`, 103
`bip_utils.bip.bip32.kholaw.bip32_kholaw_key_derivator_base`, 103
`bip_utils.bip.bip32.kholaw.bip32_kholaw_mst_key_generator`, 104
`bip_utils.bip.bip32.slip10.bip32_slip10_ed25519`, 104

104
bip_utils.bip.bip32.slip10.bip32_slip10_ed25519, 104
105
bip_utils.bip.bip32.slip10.bip32_slip10_key_derivator, 105
106
bip_utils.bip.bip32.slip10.bip32_slip10_mst_key_generator, 106
108
bip_utils.bip.bip32.slip10.bip32_slip10_nist256p1, 108
109
bip_utils.bip.bip32.slip10.bip32_slip10_secp256k1, 109
110
bip_utils.bip.bip38.bip38, 110
bip_utils.bip.bip38.bip38_addr, 112
bip_utils.bip.bip38.bip38_ec, 112
bip_utils.bip.bip38.bip38_no_ec, 114
bip_utils.bip.bip39.bip39_entropy_generator, 116
116
bip_utils.bip.bip39.bip39_mnemonic, 117
bip_utils.bip.bip39.bip39_mnemonic_decoder, 118
118
bip_utils.bip.bip39.bip39_mnemonic_encoder, 119
119
bip_utils.bip.bip39.bip39_mnemonic_generator, 119
119
bip_utils.bip.bip39.bip39_mnemonic_utils, 120
bip_utils.bip.bip39.bip39_mnemonic_validator, 121
121
bip_utils.bip.bip39.bip39_seed_generator, 121
bip_utils.bip.bip39.ibip39_seed_generator, 121
121
bip_utils.bip.bip44.bip44, 122
bip_utils.bip.bip44_base.bip44_base, 125
bip_utils.bip.bip44_base.bip44_base_ex, 130
bip_utils.bip.bip44_base.bip44_keys, 130
bip_utils.bip.bip49.bip49, 133
bip_utils.bip.bip84.bip84, 136
bip_utils.bip.bip86.bip86, 140
bip_utils.bip.conf.bip44.bip44_coins, 143
bip_utils.bip.conf.bip44.bip44_conf, 146
bip_utils.bip.conf.bip44.bip44_conf_getter, 150
150
bip_utils.bip.conf.bip49.bip49_coins, 152
bip_utils.bip.conf.bip49.bip49_conf, 153
bip_utils.bip.conf.bip49.bip49_conf_getter, 154
154
bip_utils.bip.conf.bip84.bip84_coins, 156
bip_utils.bip.conf.bip84.bip84_conf, 156
bip_utils.bip.conf.bip84.bip84_conf_getter, 156
156
bip_utils.bip.conf.bip86.bip86_coins, 157
bip_utils.bip.conf.bip86.bip86_conf, 158
bip_utils.bip.conf.bip86.bip86_conf_getter, 158
158
bip_utils.bip.conf.common.bip_bitcoin_cash_conf, 158
159
bip_utils.bip.conf.common.bip_coin_conf, 159
bip_utils.bip.conf.common.bip_coins, 162
bip_utils.bip.conf.common.bip_conf_const, 162
bip_utils.bip.conf.common.bip_litecoin_conf, 162
162
bip_utils.brainwallet.brainwallet, 163
bip_utils.brainwallet.brainwallet_algo, 164
bip_utils.brainwallet.brainwallet_algo_getter, 166
166
bip_utils.brainwallet.ibrainwallet_algo, 167
bip_utils.cardano.bip32.cardano_byron_legacy_bip32, 167
167
bip_utils.cardano.bip32.cardano_byron_legacy_key_derivator, 168
168
bip_utils.cardano.bip32.cardano_byron_legacy_mst_key_generator, 168
168
bip_utils.cardano.bip32.cardano_icarus_bip32, 169
169
bip_utils.cardano.bip32.cardano_icarus_mst_key_generator, 169
169
bip_utils.cardano.byron.cardano_byron_legacy, 170
170
bip_utils.cardano.cip1852.cip1852, 172
bip_utils.cardano.cip1852.conf.cip1852_coins, 176
176
bip_utils.cardano.cip1852.conf.cip1852_conf, 176
176
bip_utils.cardano.cip1852.conf.cip1852_conf_getter, 176
176
bip_utils.cardano.mnemonic.cardano_byron_legacy_seed_generator, 177
177
bip_utils.cardano.mnemonic.cardano_icarus_seed_generator, 178
178
bip_utils.cardano.shelley.cardano_shelley, 178
178
bip_utils.cardano.shelley.cardano_shelley_keys, 180
180
bip_utils.coin_conf.coin_conf, 182
bip_utils.coin_conf.coins_conf, 183
bip_utils.ecc.common.dummy_point, 186
bip_utils.ecc.common.ikeys, 188
bip_utils.ecc.common.ipoint, 192
bip_utils.ecc.conf, 194
bip_utils.ecc.curve.elliptic_curve, 194
bip_utils.ecc.curve.elliptic_curve_getter, 196
196
bip_utils.ecc.curve.elliptic_curve_types, 197
bip_utils.ecc.ecdsa.ecdsa_keys, 197
bip_utils.ecc.ed25519.ed25519, 197
bip_utils.ecc.ed25519.ed25519_const, 198
bip_utils.ecc.ed25519.ed25519_keys, 198
bip_utils.ecc.ed25519.ed25519_point, 201
bip_utils.ecc.ed25519.ed25519_utils, 204

Python Module Index	331
----------------------------	------------

`bip_utils.substrate.conf.substrate_coin_conf,`
280
`bip_utils.substrate.conf.substrate_coins,` 281
`bip_utils.substrate.conf.substrate_conf,` 282
`bip_utils.substrate.conf.substrate_conf_getter,`
283
`bip_utils.substrate.mnemonic.substrate_bip39_seed_generator,`
284
`bip_utils.substrate.scale.substrate_scale_enc_base,`
284
`bip_utils.substrate.scale.substrate_scale_enc_bytes,`
285
`bip_utils.substrate.scale.substrate_scale_enc_cuint,`
285
`bip_utils.substrate.scale.substrate_scale_enc_uint,`
286
`bip_utils.substrate.substrate,` 288
`bip_utils.substrate.substrate_ex,` 291
`bip_utils.substrate.substrate_keys,` 291
`bip_utils.substrate.substrate_path,` 293
`bip_utils.utils.conf.coin_names,` 296
`bip_utils.utils.crypto.aes_ecb,` 297
`bip_utils.utils.crypto.blake2,` 298
`bip_utils.utils.crypto.chacha20_poly1305,` 300
`bip_utils.utils.crypto.crc,` 301
`bip_utils.utils.crypto.hash160,` 302
`bip_utils.utils.crypto.hmac,` 303
`bip_utils.utils.crypto.pbkdf2,` 304
`bip_utils.utils.crypto.ripemd,` 305
`bip_utils.utils.crypto.scrypt,` 305
`bip_utils.utils.crypto.sha2,` 306
`bip_utils.utils.crypto.sha3,` 308
`bip_utils.utils.misc.algo,` 309
`bip_utils.utils.misc.base32,` 310
`bip_utils.utils.misc.bit,` 311
`bip_utils.utils.misc.bytes,` 313
`bip_utils.utils.misc.cbor_indefinite_len_array,`
315
`bip_utils.utils.misc.data_bytes,` 316
`bip_utils.utils.misc.integer,` 319
`bip_utils.utils.misc.string,` 320
`bip_utils.utils.mnemonic.entropy_generator,`
320
`bip_utils.utils.mnemonic.mnemonic,` 321
`bip_utils.utils.mnemonic.mnemonic_decoder_base,`
322
`bip_utils.utils.mnemonic.mnemonic_encoder_base,`
323
`bip_utils.utils.mnemonic.mnemonic_ex,` 323
`bip_utils.utils.mnemonic.mnemonic_utils,` 323
`bip_utils.utils.mnemonic.mnemonic_validator,`
326
`bip_utils.utils.typing.literal,` 327
`bip_utils.wif.wif,` 327

Symbols

__add__() (*DummyPoint* method), 187
 __add__() (*Ed25519Point* method), 203
 __add__() (*IPoint* method), 193
 __add__() (*Nist256p1Point* method), 222
 __add__() (*Secp256k1PointCoincurve* method), 231
 __add__() (*Secp256k1PointEcdsa* method), 233
 __bytes__() (*Bip32Depth* method), 89
 __bytes__() (*Bip32KeyIndex* method), 91
 __bytes__() (*DataBytes* method), 317
 __eq__() (*Bip32Depth* method), 89
 __eq__() (*Bip32KeyIndex* method), 92
 __eq__() (*DataBytes* method), 318
 __getitem__() (*Bip32Path* method), 100
 __getitem__() (*DataBytes* method), 318
 __getitem__() (*SubstratePath* method), 296
 __gt__() (*Bip32Depth* method), 89
 __int__() (*Bip32Depth* method), 88
 __int__() (*Bip32KeyIndex* method), 91
 __int__() (*DataBytes* method), 317
 __iter__() (*Bip32Path* method), 101
 __iter__() (*DataBytes* method), 318
 __iter__() (*SubstratePath* method), 296
 __len__() (*DataBytes* method), 317
 __lt__() (*Bip32Depth* method), 89
 __mul__() (*DummyPoint* method), 188
 __mul__() (*Ed25519Point* method), 203
 __mul__() (*IPoint* method), 194
 __mul__() (*Nist256p1Point* method), 222
 __mul__() (*Secp256k1PointCoincurve* method), 231
 __mul__() (*Secp256k1PointEcdsa* method), 234
 __radd__() (*DummyPoint* method), 188
 __radd__() (*Ed25519Point* method), 203
 __radd__() (*IPoint* method), 193
 __radd__() (*Nist256p1Point* method), 222
 __radd__() (*Secp256k1PointCoincurve* method), 231
 __radd__() (*Secp256k1PointEcdsa* method), 233
 __repr__() (*DataBytes* method), 318
 __rmul__() (*DummyPoint* method), 188
 __rmul__() (*Ed25519Point* method), 203
 __rmul__() (*IPoint* method), 194
 __rmul__() (*Nist256p1Point* method), 223

__rmul__() (*Secp256k1PointCoincurve* method), 231
 __rmul__() (*Secp256k1PointEcdsa* method), 234
 __str__() (*Bip32Path* method), 100
 __str__() (*Mnemonic* method), 322
 __str__() (*SubstratePath* method), 295
 __str__() (*SubstratePathElem* method), 294

A

Abbreviation() (*CoinNames* method), 297
 Acala (*CoinsConf* attribute), 183
 ACALA (*SubstrateCoins* attribute), 281
 Acala (*SubstrateConf* attribute), 282
 ACCOUNT (*Bip44Levels* attribute), 125
 Account() (*Bip44* method), 124
 Account() (*Bip44Base* method), 129
 Account() (*Bip49* method), 135
 Account() (*Bip84* method), 138
 Account() (*Bip86* method), 142
 Account() (*Cip1852* method), 174
 AdaByronAddrConst (class in *bip_utils.addr.ada_byron_addr*), 27
 AdaByronAddrDecoder (class in *bip_utils.addr.ada_byron_addr*), 28
 AdaByronAddrTypes (class in *bip_utils.addr.ada_byron_addr*), 27
 AdaByronIcarusAddr (in module *bip_utils.addr.ada_byron_addr*), 29
 AdaByronIcarusAddrEncoder (class in *bip_utils.addr.ada_byron_addr*), 28
 AdaByronLegacyAddr (in module *bip_utils.addr.ada_byron_addr*), 29
 AdaByronLegacyAddrEncoder (class in *bip_utils.addr.ada_byron_addr*), 29
 AdaShelleyAddr (in module *bip_utils.addr.ada_shelley_addr*), 32
 AdaShelleyAddrConst (class in *bip_utils.addr.ada_shelley_addr*), 30
 AdaShelleyAddrDecoder (class in *bip_utils.addr.ada_shelley_addr*), 30
 AdaShelleyAddrEncoder (class in *bip_utils.addr.ada_shelley_addr*), 31

AdaShelleyAddrHeaderTypes (class in [bip_utils.addr.ada_shelley_addr](#)), 30

AdaShelleyAddrNetworkTags (class in [bip_utils.addr.ada_shelley_addr](#)), 30

AdaShelleyRewardAddr (in module [bip_utils.addr.ada_shelley_addr](#)), 32

AdaShelleyRewardAddrDecoder (in module [bip_utils.addr.ada_shelley_addr](#)), 32

AdaShelleyRewardAddrEncoder (in module [bip_utils.addr.ada_shelley_addr](#)), 32

AdaShelleyStakingAddr (in module [bip_utils.addr.ada_shelley_addr](#)), 32

AdaShelleyStakingAddrDecoder (class in [bip_utils.addr.ada_shelley_addr](#)), 31

AdaShelleyStakingAddrEncoder (class in [bip_utils.addr.ada_shelley_addr](#)), 31

AddElem() ([Bip32Path](#) method), 99

AddElem() ([SubstratePath](#) method), 295

AddNoCarry() ([BytesUtils](#) static method), 313

ADDR_HASH_LEN ([Bip38AddrConst](#) attribute), 112

ADDR_LEN ([EthAddrConst](#) attribute), 44

AddrClass() ([BipBitcoinCashConf](#) method), 159

AddrClass() ([BipCoinConf](#) method), 161

AddrDecUtils (class in [bip_utils.addr.addr_dec_utils](#)), 32

ADDRESS_INDEX ([Bip44Levels](#) attribute), 126

AddressHash() ([Bip38Addr](#) static method), 112

AddressIndex() ([Bip44](#) method), 125

AddressIndex() ([Bip44Base](#) method), 130

AddressIndex() ([Bip49](#) method), 135

AddressIndex() ([Bip84](#) method), 139

AddressIndex() ([Bip86](#) method), 142

AddressIndex() ([CardanoShelley](#) method), 180

AddressIndex() ([Cip1852](#) method), 175

AddressKey() ([CardanoShelleyPrivateKeys](#) method), 181

AddressKey() ([CardanoShelleyPublicKeys](#) method), 180

AddrKeyValidator (class in [bip_utils.addr.addr_key_validator](#)), 34

AddrNetVersion() ([MoneroCoinConf](#) method), 256

AddrParams() ([BipBitcoinCashConf](#) method), 159

AddrParams() ([BipCoinConf](#) method), 161

AddrParams() ([BipLitecoinConf](#) method), 162

AddrParams() ([SubstrateCoinConf](#) method), 281

AddrParamsWithResolvedCalls() ([BipCoinConf](#) method), 161

aes ([AesEcbDecrypter](#) attribute), 298

aes ([AesEcbEncrypter](#) attribute), 297

AesEcbDecrypter (class in [bip_utils.utils.crypto.aes_ecb](#)), 298

AesEcbEncrypter (class in [bip_utils.utils.crypto.aes_ecb](#)), 297

AKASH_NETWORK ([Bip44Coins](#) attribute), 143

AKASH_NETWORK ([Slip173](#) attribute), 272

AkashNetwork ([Bip44Conf](#) attribute), 146

AkashNetwork ([CoinsConf](#) attribute), 183

AlgoAddr (in module [bip_utils.addr.algo_addr](#)), 36

AlgoAddrConst (class in [bip_utils.addr.algo_addr](#)), 36

AlgoAddrDecoder (class in [bip_utils.addr.algo_addr](#)), 36

AlgoAddrEncoder (class in [bip_utils.addr.algo_addr](#)), 36

ALGORAND ([Bip44Coins](#) attribute), 143

Algorand ([Bip44Conf](#) attribute), 146

Algorand ([CoinsConf](#) attribute), 183

ALGORAND ([Slip44](#) attribute), 277

AlgorandEntropyBitLen (class in [bip_utils.algorand.mnemonic.algorand_entropy_generator](#)), 65

AlgorandEntropyGenerator (class in [bip_utils.algorand.mnemonic.algorand_entropy_generator](#)), 65

AlgorandEntropyGeneratorConst (class in [bip_utils.algorand.mnemonic.algorand_entropy_generator](#)), 65

AlgorandLanguages (class in [bip_utils.algorand.mnemonic.algorand_mnemonic](#)), 66

AlgorandMnemonic (class in [bip_utils.algorand.mnemonic.algorand_mnemonic](#)), 66

AlgorandMnemonicConst (class in [bip_utils.algorand.mnemonic.algorand_mnemonic](#)), 66

AlgorandMnemonicDecoder (class in [bip_utils.algorand.mnemonic.algorand_mnemonic_decoder](#)), 66

AlgorandMnemonicEncoder (class in [bip_utils.algorand.mnemonic.algorand_mnemonic_encoder](#)), 67

AlgorandMnemonicGenerator (class in [bip_utils.algorand.mnemonic.algorand_mnemonic_generator](#)), 67

AlgorandMnemonicGeneratorConst (class in [bip_utils.algorand.mnemonic.algorand_mnemonic_generator](#)), 67

AlgorandMnemonicUtils (class in [bip_utils.algorand.mnemonic.algorand_mnemonic_utils](#)), 68

AlgorandMnemonicValidator (class in [bip_utils.algorand.mnemonic.algorand_mnemonic_validator](#)), 69

AlgorandSeedGenerator (class in [bip_utils.algorand.mnemonic.algorand_seed_generator](#)), 69

AlgorandWordsNum (class in [bip_utils.algorand.mnemonic.algorand_mnemonic](#)), 69

- 66
- AlgoUtils (class in bip_utils.utils.misc.algo), 309
- ALPHABET (Base32Const attribute), 310
- ALPHABET (Base58XmrConst attribute), 72
- ALPHABETS (Base58Const attribute), 70
- APTOS (Bip44Coins attribute), 143
- Aptos (Bip44Conf attribute), 146
- Aptos (CoinsConf attribute), 183
- APTOS (Slip44 attribute), 277
- AptosAddr (in module bip_utils.addr.aptos_addr), 37
- AptosAddrConst (class in bip_utils.addr.aptos_addr), 37
- AptosAddrDecoder (class in bip_utils.addr.aptos_addr), 37
- AptosAddrEncoder (class in bip_utils.addr.aptos_addr), 37
- ARBITRUM (Bip44Coins attribute), 143
- Arbitrum (Bip44Conf attribute), 146
- Arbitrum (CoinsConf attribute), 183
- AreBitsSet() (BitUtils static method), 311
- AreEntropyBitsEnough() (Elec-
trumV2EntropyGenerator static method), 250
- ATOM (Slip44 attribute), 276
- AtomAddr (in module bip_utils.addr.atom_addr), 38
- AtomAddrDecoder (class in bip_utils.addr.atom_addr), 38
- AtomAddrEncoder (class in bip_utils.addr.atom_addr), 38
- auto_pad (AesEcbEncrypter attribute), 297
- AutoPad() (AesEcbEncrypter method), 297
- AutoUnPad() (AesEcbDecrypter method), 298
- AVALANCHE (Slip44 attribute), 277
- AVAX_C_CHAIN (Bip44Coins attribute), 143
- AVAX_P_CHAIN (Bip44Coins attribute), 143
- AVAX_X_CHAIN (Bip44Coins attribute), 143
- AvaxCChain (Bip44Conf attribute), 146
- AvaxCChain (CoinsConf attribute), 183
- AvaxPChain (Bip44Conf attribute), 146
- AvaxPChain (CoinsConf attribute), 183
- AvaxPChainAddr (in module bip_utils.addr.avax_addr), 40
- AvaxPChainAddrDecoder (class in bip_utils.addr.avax_addr), 39
- AvaxPChainAddrEncoder (class in bip_utils.addr.avax_addr), 39
- AvaxXChain (Bip44Conf attribute), 146
- AvaxXChain (CoinsConf attribute), 183
- AvaxXChainAddr (in module bip_utils.addr.avax_addr), 40
- AvaxXChainAddrDecoder (class in bip_utils.addr.avax_addr), 39
- AvaxXChainAddrEncoder (class in bip_utils.addr.avax_addr), 40
- AXELAR (Bip44Coins attribute), 143
- Axelar (Bip44Conf attribute), 146
- Axelar (CoinsConf attribute), 183
- AXELAR (Slip173 attribute), 272
- ## B
- BAND_PROTOCOL (Bip44Coins attribute), 143
- BAND_PROTOCOL (Slip173 attribute), 272
- BAND_PROTOCOL (Slip44 attribute), 277
- BandProtocol (Bip44Conf attribute), 146
- BandProtocol (CoinsConf attribute), 183
- BASE32_ALPHABET (FilAddrConst attribute), 45
- BASE32_ALPHABET (NanoAddrConst attribute), 49
- Base32Const (class in bip_utils.utils.misc.base32), 310
- Base32Decoder (class in bip_utils.utils.misc.base32), 310
- Base32Encoder (class in bip_utils.utils.misc.base32), 310
- Base58Alphabets (class in bip_utils.base58.base58), 70
- Base58ChecksumError, 72
- Base58Const (class in bip_utils.base58.base58), 70
- Base58Decoder (class in bip_utils.base58.base58), 71
- Base58Encoder (class in bip_utils.base58.base58), 70
- Base58Utils (class in bip_utils.base58.base58), 70
- Base58XmrConst (class in bip_utils.base58.base58_xmr), 72
- Base58XmrDecoder (class in bip_utils.base58.base58_xmr), 72
- Base58XmrEncoder (class in bip_utils.base58.base58_xmr), 72
- BchAddrConverter (class in bip_utils.addr.bch_addr_converter), 40
- BchBech32Const (class in bip_utils.bech32.bch_bech32), 73
- BchBech32Decoder (class in bip_utils.bech32.bch_bech32), 74
- BchBech32Encoder (class in bip_utils.bech32.bch_bech32), 74
- BchBech32Utils (class in bip_utils.bech32.bch_bech32), 73
- BchP2PKHAddr (in module bip_utils.addr.P2PKH_addr), 23
- BchP2PKHAddrDecoder (class in bip_utils.addr.P2PKH_addr), 22
- BchP2PKHAddrEncoder (class in bip_utils.addr.P2PKH_addr), 22
- BchP2SHAddr (in module bip_utils.addr.P2SH_addr), 25
- BchP2SHAddrDecoder (class in bip_utils.addr.P2SH_addr), 24
- BchP2SHAddrEncoder (class in bip_utils.addr.P2SH_addr), 24
- BECH32 (Bech32Encodings attribute), 75
- Bech32BaseConst (class in bip_utils.bech32.bech32_base), 77

Bech32BaseUtils	(class	in	bip_utils.bip.bip32.bip32_key_data), 87
bip_utils.bech32.bech32_base), 77			
Bech32ChecksumError, 78			
Bech32Const (class in bip_utils.bech32.bech32), 75			
Bech32Decoder (class in bip_utils.bech32.bech32), 76			
Bech32DecoderBase	(class	in	bip_utils.bip.bip32.bip32_key_data), 89
bip_utils.bech32.bech32_base), 78			
Bech32Encoder (class in bip_utils.bech32.bech32), 76			
Bech32EncoderBase	(class	in	bip_utils.bip.bip32.bip32_key_net_ver), 93
bip_utils.bech32.bech32_base), 78			
Bech32Encodings (class in bip_utils.bech32.bech32), 75			
BECH32M (Bech32Encodings attribute), 75			
Bech32Utils (class in bip_utils.bech32.bech32), 75			
Bifrost (CoinsConf attribute), 183			
BIFROST (SubstrateCoins attribute), 281			
Bifrost (SubstrateConf attribute), 282			
BIG_INTEGER_MODE_MAX_VAL (SubstrateScaleCUintEncoderConst attribute), 285			
BINANCE_CHAIN (Bip44Coins attribute), 143			
BINANCE_CHAIN (Slip173 attribute), 272			
BINANCE_CHAIN (Slip44 attribute), 277			
BINANCE_SMART_CHAIN (Bip44Coins attribute), 143			
BinanceChain (Bip44Conf attribute), 147			
BinanceChain (CoinsConf attribute), 183			
BinanceSmartChain (Bip44Conf attribute), 147			
BinanceSmartChain (CoinsConf attribute), 183			
BinarySearch() (AlgoUtils static method), 309			
Bip32Base (class in bip_utils.bip.bip32.base.bip32_base), 80			
Bip32ChainCode	(class	in	bip_utils.bip.bip32.bip32_key_data), 87
Bip32Class() (BipCoinConf method), 161			
Bip32Const (class in bip_utils.bip.bip32.bip32_const), 86			
Bip32Depth (class in bip_utils.bip.bip32.bip32_key_data), 88			
Bip32DeserializedKey	(class	in	bip_utils.bip.bip32.bip32_key_ser), 95
Bip32Ed25519Blake2bSlip	(in	module	bip_utils.bip.bip32.slip10.bip32_slip10_ed25519_bip32_key_data), 105
Bip32Ed25519Kholaw	(in	module	bip_utils.bip.bip32.kholaw.bip32_kholaw_ed25519), 102
Bip32Ed25519Slip	(in	module	bip_utils.bip.bip32.slip10.bip32_slip10_ed25519), 105
Bip32FingerPrint	(class	in	bip_utils.bip.bip32.bip32_key_data), 87
Bip32Key() (Bip44PrivateKey method), 132			
Bip32Key() (Bip44PublicKey method), 130			
Bip32KeyData	(class	in	bip_utils.bip.bip32.bip32_key_data), 92
Bip32KeyDataConst	(class	in	bip_utils.bip.bip32.bip32_key_data), 92
Bip32KeyDeserializer	(class	in	bip_utils.bip.bip32.bip32_key_ser), 95
Bip32KeyError, 87			
Bip32KeyIndex	(class	in	bip_utils.bip.bip32.bip32_key_data), 89
Bip32KeyNetVersions	(class	in	bip_utils.bip.bip32.bip32_key_net_ver), 93
Bip32KeyNetVersionsConst	(class	in	bip_utils.bip.bip32.bip32_key_net_ver), 93
Bip32KeySerConst	(class	in	bip_utils.bip.bip32.bip32_key_ser), 94
Bip32KholawEd25519	(class	in	bip_utils.bip.bip32.kholaw.bip32_kholaw_ed25519), 102
Bip32KholawEd25519KeyDerivator	(class	in	bip_utils.bip.bip32.kholaw.bip32_kholaw_ed25519_key_derivator), 103
Bip32KholawEd25519KeyDerivatorBase	(class	in	bip_utils.bip.bip32.kholaw.bip32_kholaw_key_derivator_base), 103
Bip32KholawEd25519MstKeyGenerator	(class	in	bip_utils.bip.bip32.kholaw.bip32_kholaw_mst_key_generator), 104
Bip32KholawMstKeyGeneratorConst	(class	in	bip_utils.bip.bip32.kholaw.bip32_kholaw_mst_key_generator), 104
Bip32Nist256p1	(in	module	bip_utils.bip.bip32.slip10.bip32_slip10_nist256p1), 109
Bip32Object() (Bip44Base method), 126			
Bip32Object() (CardanoByronLegacy method), 170			
Bip32Object() (ElectrumV2Base method), 241			
Bip32Path (class in bip_utils.bip.bip32.bip32_path), 99			
Bip32PathConst	(class	in	bip_utils.bip.bip32.bip32_path), 99
Bip32PathError, 87			
Bip32PathParser	(class	in	bip_utils.bip.bip32.bip32_path), 101
Bip32PrivateKey	(class	in	bip_utils.bip.bip32.bip32_keys), 98
Bip32PrivateKeySerializer	(class	in	bip_utils.bip.bip32.bip32_key_ser), 94
Bip32PublicKey	(class	in	bip_utils.bip.bip32.bip32_keys), 96
Bip32PublicKeySerializer	(class	in	bip_utils.bip.bip32.bip32_key_ser), 94
Bip32Secp256k1	(in	module	bip_utils.bip.bip32.slip10.bip32_slip10_secp256k1), 110
Bip32Slip10DerivatorConst	(class	in	bip_utils.bip.bip32.slip10.bip32_slip10_key_derivator), 106
Bip32Slip10EcdsaDerivator	(class	in	bip_utils.bip.bip32.slip10.bip32_slip10_key_derivator), 106

<i>bip_utils.bip.bip32.slip10.bip32_slip10_key_deriver</i>	<i>Bip39EntropyGenerator</i> (class in <i>bip_utils.bip.bip39.bip39_entropy_generator</i>),
<i>106</i>	<i>116</i>
<i>Bip32Slip10Ed25519MstKeyGenerator</i> (class in <i>bip_utils.bip.bip32.slip10.bip32_slip10_mst_key_deriver</i>),	<i>Bip39EntropyGeneratorConst</i> (class in <i>bip_utils.bip.bip39.bip39_entropy_generator</i>),
<i>108</i>	<i>116</i>
<i>Bip32Slip10Ed25519</i> (class in <i>bip_utils.bip.bip32.slip10.bip32_slip10_ed25519</i>),	<i>Bip39Languages</i> (class in <i>bip_utils.bip.bip39.bip39_mnemonic</i>),
<i>104</i>	<i>117</i>
<i>Bip32Slip10Ed25519Blake2b</i> (class in <i>bip_utils.bip.bip32.slip10.bip32_slip10_ed25519_blake2b</i>),	<i>Bip39Mnemonic</i> (class in <i>bip_utils.bip.bip39.bip39_mnemonic</i>),
<i>105</i>	<i>118</i>
<i>Bip32Slip10Ed25519Derivator</i> (class in <i>bip_utils.bip.bip32.slip10.bip32_slip10_key_deriver</i>),	<i>Bip39MnemonicConst</i> (class in <i>bip_utils.bip.bip39.bip39_mnemonic</i>),
<i>107</i>	<i>117</i>
<i>Bip32Slip10MstKeyGeneratorConst</i> (class in <i>bip_utils.bip.bip32.slip10.bip32_slip10_mst_key_deriver</i>),	<i>Bip39MnemonicDecoder</i> (class in <i>bip_utils.bip.bip39.bip39_mnemonic_decoder</i>),
<i>108</i>	<i>118</i>
<i>Bip32Slip10Nist256p1</i> (class in <i>bip_utils.bip.bip32.slip10.bip32_slip10_nist256p1</i>),	<i>Bip39MnemonicEncoder</i> (class in <i>bip_utils.bip.bip39.bip39_mnemonic_encoder</i>),
<i>109</i>	<i>119</i>
<i>Bip32Slip10Nist256p1MstKeyGenerator</i> (class in <i>bip_utils.bip.bip32.slip10.bip32_slip10_mst_key_deriver</i>),	<i>Bip39MnemonicGenerator</i> (class in <i>bip_utils.bip.bip39.bip39_mnemonic_generator</i>),
<i>108</i>	<i>119</i>
<i>Bip32Slip10Secp256k1</i> (class in <i>bip_utils.bip.bip32.slip10.bip32_slip10_secp256k1</i>),	<i>Bip39MnemonicValidator</i> (class in <i>bip_utils.bip.bip39.bip39_mnemonic_validator</i>),
<i>110</i>	<i>121</i>
<i>Bip32Slip10Secp256k1MstKeyGenerator</i> (class in <i>bip_utils.bip.bip32.slip10.bip32_slip10_mst_key_deriver</i>),	<i>Bip39SeedGenerator</i> (class in <i>bip_utils.bip.bip39.bip39_seed_generator</i>),
<i>109</i>	<i>121</i>
<i>Bip32Utils</i> (class in <i>bip_utils.bip.bip32.bip32_utils</i>),	<i>Bip39SeedGeneratorConst</i> (class in <i>bip_utils.bip.bip39.bip39_seed_generator</i>),
<i>101</i>	<i>121</i>
<i>Bip38Addr</i> (class in <i>bip_utils.bip.bip38.bip38_addr</i>),	<i>Bip39WordsListFinder</i> (class in <i>bip_utils.bip.bip39.bip39_mnemonic_utils</i>),
<i>112</i>	<i>120</i>
<i>Bip38AddrConst</i> (class in <i>bip_utils.bip.bip38.bip38_addr</i>),	<i>Bip39WordsListGetter</i> (class in <i>bip_utils.bip.bip39.bip39_mnemonic_utils</i>),
<i>111</i>	<i>120</i>
<i>Bip38Decrypter</i> (class in <i>bip_utils.bip.bip38.bip38</i>),	<i>Bip39WordsNum</i> (class in <i>bip_utils.bip.bip39.bip39_mnemonic</i>),
<i>112</i>	<i>117</i>
<i>Bip38EcConst</i> (class in <i>bip_utils.bip.bip38.bip38_ec</i>),	<i>Bip44</i> (class in <i>bip_utils.bip.bip44.bip44</i>),
<i>112</i>	<i>122</i>
<i>Bip38EcDecrypter</i> (class in <i>bip_utils.bip.bip38.bip38_ec</i>),	<i>bip44_obj</i> (Brainwallet attribute),
<i>114</i>	<i>164</i>
<i>Bip38EcKeysGenerator</i> (class in <i>bip_utils.bip.bip38.bip38_ec</i>),	<i>Bip44Base</i> (class in <i>bip_utils.bip.bip44_base.bip44_base</i>),
<i>113</i>	<i>126</i>
<i>Bip38Encrypter</i> (class in <i>bip_utils.bip.bip38.bip38</i>),	<i>Bip44Changes</i> (class in <i>bip_utils.bip.bip44_base.bip44_base</i>),
<i>110</i>	<i>125</i>
<i>Bip38NoEcConst</i> (class in <i>bip_utils.bip.bip38.bip38_no_ec</i>),	<i>Bip44Coins</i> (class in <i>bip_utils.bip.conf.bip44.bip44_coins</i>),
<i>114</i>	<i>143</i>
<i>Bip38NoEcDecrypter</i> (class in <i>bip_utils.bip.bip38.bip38_no_ec</i>),	<i>Bip44Conf</i> (class in <i>bip_utils.bip.conf.bip44.bip44_conf</i>),
<i>115</i>	<i>146</i>
<i>Bip38NoEcEncrypter</i> (class in <i>bip_utils.bip.bip38.bip38_no_ec</i>),	<i>Bip44ConfGetter</i> (class in <i>bip_utils.bip.conf.bip44.bip44_conf_getter</i>),
<i>115</i>	<i>152</i>
<i>Bip39EntropyBitLen</i> (class in <i>bip_utils.bip.bip39.bip39_entropy_generator</i>),	<i>Bip44ConfGetterConst</i> (class in <i>bip_utils.bip.conf.bip44.bip44_conf_getter</i>),
<i>116</i>	<i>150</i>

Bip44Const (class in bip_utils.bip.bip44.bip44), 122
Bip44DepthError, 130
Bip44Levels (class in bip_utils.bip.bip44_base.bip44_base), 125
Bip44PrivateKey (class in bip_utils.bip.bip44_base.bip44_keys), 131
Bip44PublicKey (class in bip_utils.bip.bip44_base.bip44_keys), 130
Bip49 (class in bip_utils.bip.bip49.bip49), 133
Bip49Coins (class in bip_utils.bip.conf.bip49.bip49_coins), 152
Bip49Conf (class in bip_utils.bip.conf.bip49.bip49_conf), 153
Bip49ConfGetter (class in bip_utils.bip.conf.bip49.bip49_conf_getter), 155
Bip49ConfGetterConst (class in bip_utils.bip.conf.bip49.bip49_conf_getter), 154
Bip49Const (class in bip_utils.bip.bip49.bip49), 133
Bip84 (class in bip_utils.bip.bip84.bip84), 136
Bip84Coins (class in bip_utils.bip.conf.bip84.bip84_coins), 156
Bip84Conf (class in bip_utils.bip.conf.bip84.bip84_conf), 156
Bip84ConfGetter (class in bip_utils.bip.conf.bip84.bip84_conf_getter), 157
Bip84ConfGetterConst (class in bip_utils.bip.conf.bip84.bip84_conf_getter), 156
Bip84Const (class in bip_utils.bip.bip84.bip84), 136
Bip86 (class in bip_utils.bip.bip86.bip86), 140
Bip86Coins (class in bip_utils.bip.conf.bip86.bip86_coins), 157
Bip86Conf (class in bip_utils.bip.conf.bip86.bip86_conf), 158
Bip86ConfGetter (class in bip_utils.bip.conf.bip86.bip86_conf_getter), 158
Bip86ConfGetterConst (class in bip_utils.bip.conf.bip86.bip86_conf_getter), 158
Bip86Const (class in bip_utils.bip.bip86.bip86), 140
bip_utils.addr.ada_byron_addr module, 27
bip_utils.addr.ada_shelley_addr module, 30
bip_utils.addr.addr_dec_utils module, 32
bip_utils.addr.addr_key_validator module, 34
bip_utils.addr.algo_addr module, 36
bip_utils.addr.aptos_addr module, 37
bip_utils.addr.atom_addr module, 38
bip_utils.addr.avax_addr module, 39
bip_utils.addr.bch_addr_converter module, 40
bip_utils.addr.egld_addr module, 41
bip_utils.addr.eos_addr module, 42
bip_utils.addr.ergo_addr module, 43
bip_utils.addr.eth_addr module, 44
bip_utils.addr.fil_addr module, 45
bip_utils.addr.iaddr_decoder module, 46
bip_utils.addr.iaddr_encoder module, 47
bip_utils.addr.icx_addr module, 47
bip_utils.addr.inj_addr module, 48
bip_utils.addr.nano_addr module, 49
bip_utils.addr.near_addr module, 50
bip_utils.addr.neo_addr module, 51
bip_utils.addr.okex_addr module, 52
bip_utils.addr.one_addr module, 53
bip_utils.addr.P2PKH_addr module, 21
bip_utils.addr.P2SH_addr module, 23
bip_utils.addr.P2TR_addr module, 25
bip_utils.addr.P2WPKH_addr module, 26
bip_utils.addr.sol_addr module, 54
bip_utils.addr.substrate_addr module, 55
bip_utils.addr.sui_addr module, 56
bip_utils.addr.trx_addr module, 57
bip_utils.addr.xlm_addr module, 58

bip_utils.addr.xmr_addr	bip_utils.bip.bip32.bip32_key_ser
module, 60	module, 94
bip_utils.addr.xrp_addr	bip_utils.bip.bip32.bip32_keys
module, 62	module, 96
bip_utils.addr.xtz_addr	bip_utils.bip.bip32.bip32_path
module, 63	module, 99
bip_utils.addr.zil_addr	bip_utils.bip.bip32.bip32_utils
module, 64	module, 101
bip_utils.algorand.mnemonic.algorand_entropy_generator	bip_utils.bip.bip32.kholaw.bip32_kholaw_ed25519
module, 65	module, 102
bip_utils.algorand.mnemonic.algorand_mnemonic	bip_utils.bip.bip32.kholaw.bip32_kholaw_ed25519_key_deriva
module, 66	module, 103
bip_utils.algorand.mnemonic.algorand_mnemonic_decoder	bip_utils.bip.bip32.kholaw.bip32_kholaw_key_derivator_base
module, 66	module, 103
bip_utils.algorand.mnemonic.algorand_mnemonic_encoder	bip_utils.bip.bip32.kholaw.bip32_kholaw_mst_key_generator
module, 67	module, 104
bip_utils.algorand.mnemonic.algorand_mnemonic_generator	bip_utils.bip.bip32.slip10.bip32_slip10_ed25519
module, 67	module, 104
bip_utils.algorand.mnemonic.algorand_mnemonic_validator	bip_utils.bip.bip32.slip10.bip32_slip10_ed25519_blake2b
module, 68	module, 105
bip_utils.algorand.mnemonic.algorand_mnemonic_validator	bip_utils.bip.bip32.slip10.bip32_slip10_key_derivator
module, 69	module, 106
bip_utils.algorand.mnemonic.algorand_seed_generator	bip_utils.bip.bip32.slip10.bip32_slip10_mst_key_generator
module, 69	module, 108
bip_utils.base58.base58	bip_utils.bip.bip32.slip10.bip32_slip10_nist256p1
module, 70	module, 109
bip_utils.base58.base58_ex	bip_utils.bip.bip32.slip10.bip32_slip10_secp256k1
module, 72	module, 110
bip_utils.base58.base58_xmr	bip_utils.bip.bip38.bip38
module, 72	module, 110
bip_utils.bech32.bch_bech32	bip_utils.bip.bip38.bip38_addr
module, 73	module, 112
bip_utils.bech32.bech32	bip_utils.bip.bip38.bip38_ec
module, 75	module, 112
bip_utils.bech32.bech32_base	bip_utils.bip.bip38.bip38_no_ec
module, 77	module, 114
bip_utils.bech32.bech32_ex	bip_utils.bip.bip39.bip39_entropy_generator
module, 78	module, 116
bip_utils.bech32.segwit_bech32	bip_utils.bip.bip39.bip39_mnemonic
module, 79	module, 117
bip_utils.bip.bip32.base.bip32_base	bip_utils.bip.bip39.bip39_mnemonic_decoder
module, 80	module, 118
bip_utils.bip.bip32.base.ibip32_key_derivator	bip_utils.bip.bip39.bip39_mnemonic_encoder
module, 85	module, 119
bip_utils.bip.bip32.base.ibip32_mst_key_generator	bip_utils.bip.bip39.bip39_mnemonic_generator
module, 86	module, 119
bip_utils.bip.bip32.bip32_const	bip_utils.bip.bip39.bip39_mnemonic_utils
module, 86	module, 120
bip_utils.bip.bip32.bip32_ex	bip_utils.bip.bip39.bip39_mnemonic_validator
module, 87	module, 121
bip_utils.bip.bip32.bip32_key_data	bip_utils.bip.bip39.bip39_seed_generator
module, 87	module, 121
bip_utils.bip.bip32.bip32_key_net_ver	bip_utils.bip.bip39.ibip39_seed_generator
module, 93	module, 121

bip_utils.bip.bip44.bip44	bip_utils.brainwallet.ibrainwallet_algo
module, 122	module, 167
bip_utils.bip.bip44_base.bip44_base	bip_utils.cardano.bip32.cardano_byron_legacy_bip32
module, 125	module, 167
bip_utils.bip.bip44_base.bip44_base_ex	bip_utils.cardano.bip32.cardano_byron_legacy_key_derivator
module, 130	module, 168
bip_utils.bip.bip44_base.bip44_keys	bip_utils.cardano.bip32.cardano_byron_legacy_mst_key_gener
module, 130	module, 168
bip_utils.bip.bip49.bip49	bip_utils.cardano.bip32.cardano_icarus_bip32
module, 133	module, 169
bip_utils.bip.bip84.bip84	bip_utils.cardano.bip32.cardano_icarus_mst_key_generator
module, 136	module, 169
bip_utils.bip.bip86.bip86	bip_utils.cardano.byron.cardano_byron_legacy
module, 140	module, 170
bip_utils.bip.conf.bip44.bip44_coins	bip_utils.cardano.cip1852.cip1852
module, 143	module, 172
bip_utils.bip.conf.bip44.bip44_conf	bip_utils.cardano.cip1852.conf.cip1852_coins
module, 146	module, 176
bip_utils.bip.conf.bip44.bip44_conf_getter	bip_utils.cardano.cip1852.conf.cip1852_conf
module, 150	module, 176
bip_utils.bip.conf.bip49.bip49_coins	bip_utils.cardano.cip1852.conf.cip1852_conf_getter
module, 152	module, 176
bip_utils.bip.conf.bip49.bip49_conf	bip_utils.cardano.mnemonic.cardano_byron_legacy_seed_gener
module, 153	module, 177
bip_utils.bip.conf.bip49.bip49_conf_getter	bip_utils.cardano.mnemonic.cardano_icarus_seed_generator
module, 154	module, 178
bip_utils.bip.conf.bip84.bip84_coins	bip_utils.cardano.shelley.cardano_shelley
module, 156	module, 178
bip_utils.bip.conf.bip84.bip84_conf	bip_utils.cardano.shelley.cardano_shelley_keys
module, 156	module, 180
bip_utils.bip.conf.bip84.bip84_conf_getter	bip_utils.coin_conf.coin_conf
module, 156	module, 182
bip_utils.bip.conf.bip86.bip86_coins	bip_utils.coin_conf.coins_conf
module, 157	module, 183
bip_utils.bip.conf.bip86.bip86_conf	bip_utils.ecc.common.dummy_point
module, 158	module, 186
bip_utils.bip.conf.bip86.bip86_conf_getter	bip_utils.ecc.common.ikeys
module, 158	module, 188
bip_utils.bip.conf.common.bip_bitcoin_cash_conf	bip_utils.ecc.common.ipoint
module, 159	module, 192
bip_utils.bip.conf.common.bip_coin_conf	bip_utils.ecc.conf
module, 159	module, 194
bip_utils.bip.conf.common.bip_coins	bip_utils.ecc.curve.elliptic_curve
module, 162	module, 194
bip_utils.bip.conf.common.bip_conf_const	bip_utils.ecc.curve.elliptic_curve_getter
module, 162	module, 196
bip_utils.bip.conf.common.bip_litecoin_conf	bip_utils.ecc.curve.elliptic_curve_types
module, 162	module, 197
bip_utils.brainwallet.brainwallet	bip_utils.ecc.ecdsa.ecdsa_keys
module, 163	module, 197
bip_utils.brainwallet.brainwallet_algo	bip_utils.ecc.ed25519.ed25519
module, 164	module, 197
bip_utils.brainwallet.brainwallet_algo_getter	bip_utils.ecc.ed25519.ed25519_const
module, 166	module, 198

bip_utils.ecc.ed25519.ed25519_keys module, 198	bip_utils.ecc.sr25519.sr25519_const module, 234
bip_utils.ecc.ed25519.ed25519_point module, 201	bip_utils.ecc.sr25519.sr25519_keys module, 235
bip_utils.ecc.ed25519.ed25519_utils module, 204	bip_utils.ecc.sr25519.sr25519_point module, 238
bip_utils.ecc.ed25519.lib.ed25519_lib module, 204	bip_utils.electrum.electrum_v1 module, 238
bip_utils.ecc.ed25519.blake2b.ed25519_blake2b module, 208	bip_utils.electrum.electrum_v2 module, 240
bip_utils.ecc.ed25519.blake2b.ed25519_blake2b_keys module, 208	bip_utils.electrum.mnemonic_v1.electrum_v1_entropy_generator module, 244
bip_utils.ecc.ed25519.blake2b.ed25519_blake2b_keys module, 209	bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic module, 245
bip_utils.ecc.ed25519.blake2b.ed25519_blake2b_point module, 211	bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic_decode module, 246
bip_utils.ecc.ed25519.kholaw.ed25519_kholaw module, 212	bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic_encode module, 247
bip_utils.ecc.ed25519.kholaw.ed25519_kholaw_const module, 212	bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic_generator module, 247
bip_utils.ecc.ed25519.kholaw.ed25519_kholaw_keys module, 212	bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic_utils module, 248
bip_utils.ecc.ed25519.kholaw.ed25519_kholaw_point module, 214	bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic_validate module, 249
bip_utils.ecc.ed25519.monero.ed25519_monero module, 214	bip_utils.electrum.mnemonic_v1.electrum_v1_seed_generator module, 249
bip_utils.ecc.ed25519.monero.ed25519_monero_const module, 215	bip_utils.electrum.mnemonic_v2.electrum_v2_entropy_generator module, 250
bip_utils.ecc.ed25519.monero.ed25519_monero_keys module, 215	bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic module, 251
bip_utils.ecc.ed25519.monero.ed25519_monero_point module, 217	bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_decode module, 252
bip_utils.ecc.nist256p1.nist256p1 module, 217	bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_encode module, 252
bip_utils.ecc.nist256p1.nist256p1_const module, 217	bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_generator module, 253
bip_utils.ecc.nist256p1.nist256p1_keys module, 218	bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_utils module, 254
bip_utils.ecc.nist256p1.nist256p1_point module, 220	bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_validate module, 254
bip_utils.ecc.secp256k1.secp256k1 module, 223	bip_utils.electrum.mnemonic_v2.electrum_v2_seed_generator module, 254
bip_utils.ecc.secp256k1.secp256k1_const module, 223	bip_utils.monero.conf.monero_coin_conf module, 255
bip_utils.ecc.secp256k1.secp256k1_keys_coincurve module, 223	bip_utils.monero.conf.monero_coins module, 256
bip_utils.ecc.secp256k1.secp256k1_keys_ecdsa module, 226	bip_utils.monero.conf.monero_conf module, 257
bip_utils.ecc.secp256k1.secp256k1_point_coincurve module, 229	bip_utils.monero.conf.monero_conf_getter module, 257
bip_utils.ecc.secp256k1.secp256k1_point_ecdsa module, 232	bip_utils.monero.mnemonic.monero_entropy_generator module, 258
bip_utils.ecc.sr25519.sr25519 module, 234	bip_utils.monero.mnemonic.monero_mnemonic module, 259

bip_utils.monero.mnemonic.monero_mnemonic_decoder	bip_utils.substrate.substrate_ex
module, 260	module, 291
bip_utils.monero.mnemonic.monero_mnemonic_encoder	bip_utils.substrate.substrate_keys
module, 261	module, 291
bip_utils.monero.mnemonic.monero_mnemonic_generator	bip_utils.substrate.substrate_path
module, 262	module, 293
bip_utils.monero.mnemonic.monero_mnemonic_utils	bip_utils.utils.conf.coin_names
module, 263	module, 296
bip_utils.monero.mnemonic.monero_mnemonic_validator	bip_utils.utils.crypto.aes_ecb
module, 265	module, 297
bip_utils.monero.mnemonic.monero_seed_generator	bip_utils.utils.crypto.blake2
module, 265	module, 298
bip_utils.monero.monero	bip_utils.utils.crypto.chacha20_poly1305
module, 265	module, 300
bip_utils.monero.monero_ex	bip_utils.utils.crypto.crc
module, 268	module, 301
bip_utils.monero.monero_keys	bip_utils.utils.crypto.hash160
module, 268	module, 302
bip_utils.monero.monero_subaddr	bip_utils.utils.crypto.hmac
module, 271	module, 303
bip_utils.slip.slip173.slip173	bip_utils.utils.crypto.pbkdf2
module, 272	module, 304
bip_utils.slip.slip32.slip32	bip_utils.utils.crypto.ripemd
module, 273	module, 305
bip_utils.slip.slip32.slip32_key_net_ver	bip_utils.utils.crypto.scrip
module, 275	module, 305
bip_utils.slip.slip44.slip44	bip_utils.utils.crypto.sha2
module, 276	module, 306
bip_utils.solana.spl_token	bip_utils.utils.crypto.sha3
module, 278	module, 308
bip_utils.ss58.ss58	bip_utils.utils.misc.algo
module, 279	module, 309
bip_utils.ss58.ss58_ex	bip_utils.utils.misc.base32
module, 280	module, 310
bip_utils.substrate.conf.substrate_coin_conf	bip_utils.utils.misc.bit
module, 280	module, 311
bip_utils.substrate.conf.substrate_coins	bip_utils.utils.misc.bytes
module, 281	module, 313
bip_utils.substrate.conf.substrate_conf	bip_utils.utils.misc.cbor_indefinite_len_array
module, 282	module, 315
bip_utils.substrate.conf.substrate_conf_getter	bip_utils.utils.misc.data_bytes
module, 283	module, 316
bip_utils.substrate.mnemonic.substrate_bip39_seed_generator	bip_utils.utils.misc.integer
module, 284	module, 319
bip_utils.substrate.scale.substrate_scale_encoder_base	bip_utils.utils.misc.string
module, 284	module, 320
bip_utils.substrate.scale.substrate_scale_encoder_bytes	bip_utils.utils.mnemonic.entropy_generator
module, 285	module, 320
bip_utils.substrate.scale.substrate_scale_encoder_crip	bip_utils.utils.mnemonic.mnemonic
module, 285	module, 321
bip_utils.substrate.scale.substrate_scale_encoder_crip	bip_utils.utils.mnemonic.mnemonic_decoder_base
module, 286	module, 322
bip_utils.substrate.substrate	bip_utils.utils.mnemonic.mnemonic_encoder_base
module, 288	module, 323

bip_utils.utils.mnemonic.mnemonic_ex module, 323
 bip_utils.utils.mnemonic.mnemonic_utils module, 323
 bip_utils.utils.mnemonic.mnemonic_validator module, 326
 bip_utils.utils.typing.literal module, 327
 bip_utils.wif.wif module, 327
 BipBitcoinCashConf (class in bip_utils.bip.conf.common.bip_bitcoin_cash_conf), 159
 BipCoinConf (class in bip_utils.bip.conf.common.bip_coin_conf), 160
 BipCoinFctCallsConf (class in bip_utils.bip.conf.common.bip_coin_conf), 159
 BipCoins (class in bip_utils.bip.conf.common.bip_coins), 162
 BipLitecoinConf (class in bip_utils.bip.conf.common.bip_litecoin_conf), 162
 BIT_LEN_128 (Bip39EntropyBitLen attribute), 116
 BIT_LEN_128 (ElectrumV1EntropyBitLen attribute), 244
 BIT_LEN_128 (MoneroEntropyBitLen attribute), 258
 BIT_LEN_132 (ElectrumV2EntropyBitLen attribute), 250
 BIT_LEN_160 (Bip39EntropyBitLen attribute), 116
 BIT_LEN_192 (Bip39EntropyBitLen attribute), 116
 BIT_LEN_224 (Bip39EntropyBitLen attribute), 116
 BIT_LEN_256 (AlgorandEntropyBitLen attribute), 65
 BIT_LEN_256 (Bip39EntropyBitLen attribute), 116
 BIT_LEN_256 (MoneroEntropyBitLen attribute), 258
 BIT_LEN_264 (ElectrumV2EntropyBitLen attribute), 250
 BITCOIN (Base58Alphabets attribute), 70
 BITCOIN (Bip44Coins attribute), 144
 BITCOIN (Bip49Coins attribute), 152
 BITCOIN (Bip84Coins attribute), 156
 BITCOIN (Bip86Coins attribute), 157
 BITCOIN (Slip44 attribute), 276
 BITCOIN_CASH (Bip44Coins attribute), 144
 BITCOIN_CASH (Bip49Coins attribute), 152
 BITCOIN_CASH (Slip44 attribute), 276
 BITCOIN_CASH_SLP (Bip44Coins attribute), 144
 BITCOIN_CASH_SLP (Bip49Coins attribute), 152
 BITCOIN_CASH_SLP_TESTNET (Bip44Coins attribute), 146
 BITCOIN_CASH_SLP_TESTNET (Bip49Coins attribute), 152
 BITCOIN_CASH_TESTNET (Bip44Coins attribute), 146
 BITCOIN_CASH_TESTNET (Bip49Coins attribute), 152
 BITCOIN_MAINNET (Slip173 attribute), 272
 BITCOIN_REGTEST (Bip44Coins attribute), 146
 BITCOIN_REGTEST (Bip49Coins attribute), 152
 BITCOIN_REGTEST (Bip84Coins attribute), 156
 BITCOIN_REGTEST (Bip86Coins attribute), 157
 BITCOIN_REGTEST (Slip173 attribute), 272
 BITCOIN_SV (Bip44Coins attribute), 144
 BITCOIN_SV (Bip49Coins attribute), 152
 BITCOIN_SV (Slip44 attribute), 277
 BITCOIN_SV_TESTNET (Bip44Coins attribute), 146
 BITCOIN_SV_TESTNET (Bip49Coins attribute), 152
 BITCOIN_TESTNET (Bip44Coins attribute), 146
 BITCOIN_TESTNET (Bip49Coins attribute), 152
 BITCOIN_TESTNET (Bip84Coins attribute), 156
 BITCOIN_TESTNET (Bip86Coins attribute), 157
 BITCOIN_TESTNET (Slip173 attribute), 272
 BitcoinCashMainNet (Bip44Conf attribute), 147
 BitcoinCashMainNet (Bip49Conf attribute), 153
 BitcoinCashMainNet (CoinsConf attribute), 183
 BitcoinCashSlpMainNet (Bip44Conf attribute), 147
 BitcoinCashSlpMainNet (Bip49Conf attribute), 153
 BitcoinCashSlpMainNet (CoinsConf attribute), 183
 BitcoinCashSlpTestNet (Bip44Conf attribute), 147
 BitcoinCashSlpTestNet (Bip49Conf attribute), 153
 BitcoinCashSlpTestNet (CoinsConf attribute), 183
 BitcoinCashTestNet (Bip44Conf attribute), 147
 BitcoinCashTestNet (Bip49Conf attribute), 153
 BitcoinCashTestNet (CoinsConf attribute), 183
 BitcoinMainNet (Bip44Conf attribute), 147
 BitcoinMainNet (Bip49Conf attribute), 153
 BitcoinMainNet (Bip84Conf attribute), 156
 BitcoinMainNet (Bip86Conf attribute), 158
 BitcoinMainNet (CoinsConf attribute), 183
 BitcoinRegTest (Bip44Conf attribute), 147
 BitcoinRegTest (Bip49Conf attribute), 153
 BitcoinRegTest (Bip84Conf attribute), 156
 BitcoinRegTest (Bip86Conf attribute), 158
 BitcoinRegTest (CoinsConf attribute), 183
 BitcoinSvMainNet (Bip44Conf attribute), 147
 BitcoinSvMainNet (Bip49Conf attribute), 153
 BitcoinSvMainNet (CoinsConf attribute), 183
 BitcoinSvTestNet (Bip44Conf attribute), 147
 BitcoinSvTestNet (Bip49Conf attribute), 153
 BitcoinSvTestNet (CoinsConf attribute), 183
 BitcoinTestNet (Bip44Conf attribute), 147
 BitcoinTestNet (Bip49Conf attribute), 153
 BitcoinTestNet (Bip84Conf attribute), 156
 BitcoinTestNet (Bip86Conf attribute), 158
 BitcoinTestNet (CoinsConf attribute), 183
 BitUtils (class in bip_utils.utils.misc.bit), 311
 Blake2b (class in bip_utils.utils.crypto.blake2), 298
 Blake2b160 (class in bip_utils.utils.crypto.blake2), 299
 Blake2b224 (class in bip_utils.utils.crypto.blake2), 299
 Blake2b256 (class in bip_utils.utils.crypto.blake2), 299
 Blake2b32 (class in bip_utils.utils.crypto.blake2), 299
 Blake2b40 (class in bip_utils.utils.crypto.blake2), 299

- Blake2b512 (class in bip_utils.utils.crypto.blake2), 300
- BLOCK_DEC_MAX_BYTE_LEN (Base58XmrConst attribute), 72
- BLOCK_ENC_BYTE_LENS (Base58XmrConst attribute), 72
- BLOCK_ENC_MAX_BYTE_LEN (Base58XmrConst attribute), 72
- BLS (FillAddrTypes attribute), 45
- Brainwallet (class in bip_utils.brainwallet.brainwallet), 163
- BrainwalletAlgoConst (class in bip_utils.brainwallet.brainwallet_algo), 164
- BrainwalletAlgoDoubleSha256 (class in bip_utils.brainwallet.brainwallet_algo), 165
- BrainwalletAlgoGetter (class in bip_utils.brainwallet.brainwallet_algo_getter), 166
- BrainwalletAlgoGetterConst (class in bip_utils.brainwallet.brainwallet_algo_getter), 166
- BrainwalletAlgoPbkdf2HmacSha512 (class in bip_utils.brainwallet.brainwallet_algo), 165
- BrainwalletAlgos (class in bip_utils.brainwallet.brainwallet_algo), 164
- BrainwalletAlgoScrypt (class in bip_utils.brainwallet.brainwallet_algo), 165
- BrainwalletAlgoSha256 (class in bip_utils.brainwallet.brainwallet_algo), 164
- BytesChunkToWords() (MnemonicUtils static method), 323
- BytesUtils (class in bip_utils.utils.misc.bytes), 313
- ## C
- CARDANO (Slip44 attribute), 277
- CARDANO_BYRON_ICARUS (Bip44Coins attribute), 144
- CARDANO_BYRON_LEDGER (Bip44Coins attribute), 144
- CARDANO_ICARUS (Cip1852Coins attribute), 176
- CARDANO_ICARUS_TESTNET (Cip1852Coins attribute), 176
- CARDANO_LEDGER (Cip1852Coins attribute), 176
- CARDANO_LEDGER_TESTNET (Cip1852Coins attribute), 176
- CardanoByronIcarus (Bip44Conf attribute), 147
- CardanoByronLedger (Bip44Conf attribute), 147
- CardanoByronLegacy (class in bip_utils.cardano.byron.cardano_byron_legacy), 170
- CardanoByronLegacyBip32 (class in bip_utils.cardano.bip32.cardano_byron_legacy_bip32), 167
- CardanoByronLegacyConst (class in bip_utils.cardano.byron.cardano_byron_legacy), 170
- CardanoByronLegacyKeyDerivator (class in bip_utils.cardano.bip32.cardano_byron_legacy_key_derivator), 168
- CardanoByronLegacyMstKeyGenerator (class in bip_utils.cardano.bip32.cardano_byron_legacy_mst_key_generator), 168
- CardanoByronLegacyMstKeyGeneratorConst (class in bip_utils.cardano.bip32.cardano_byron_legacy_mst_key_generator), 168
- CardanoByronLegacySeedGenerator (class in bip_utils.cardano.mnemonic.cardano_byron_legacy_seed_generator), 177
- CardanoIcarusBip32 (class in bip_utils.cardano.bip32.cardano_icarus_bip32), 169
- CardanoIcarusMainNet (Cip1852Conf attribute), 176
- CardanoIcarusMasterKeyGeneratorConst (class in bip_utils.cardano.bip32.cardano_icarus_mst_key_generator), 169
- CardanoIcarusMstKeyGenerator (class in bip_utils.cardano.bip32.cardano_icarus_mst_key_generator), 169
- CardanoIcarusSeedGenerator (class in bip_utils.cardano.mnemonic.cardano_icarus_seed_generator), 178
- CardanoIcarusTestNet (Cip1852Conf attribute), 176
- CardanoLedgerMainNet (Cip1852Conf attribute), 176
- CardanoLedgerTestNet (Cip1852Conf attribute), 176
- CardanoMainNet (CoinsConf attribute), 183
- CardanoShelley (class in bip_utils.cardano.shelley.cardano_shelley), 178
- CardanoShelleyPrivateKeys (class in bip_utils.cardano.shelley.cardano_shelley_keys), 181
- CardanoShelleyPublicKeys (class in bip_utils.cardano.shelley.cardano_shelley_keys), 180
- CardanoTestNet (CoinsConf attribute), 183
- CborIds (class in bip_utils.utils.misc.cbor_indefinite_len_array), 315
- CborIndefiniteLenArrayConst (class in bip_utils.utils.misc.cbor_indefinite_len_array), 315
- CborIndefiniteLenArrayDecoder (class in bip_utils.utils.misc.cbor_indefinite_len_array), 316
- CborIndefiniteLenArrayEncoder (class in bip_utils.utils.misc.cbor_indefinite_len_array), 316
- CELO (Bip44Coins attribute), 144
- Celo (Bip44Conf attribute), 147
- Celo (CoinsConf attribute), 183
- CELO (Slip44 attribute), 277
- CERTIK (Bip44Coins attribute), 144
- Certik (Bip44Conf attribute), 147

- Certik (*CoinsConf* attribute), 184
 CERTIK (*Slip173* attribute), 272
 CHACHA20_POLY1305_ASSOC_DATA (*AdaByronAddrConst* attribute), 27
 CHACHA20_POLY1305_NONCE (*AdaByronAddrConst* attribute), 28
 ChaCha20Poly1305 (class in *bip_utils.utils.crypto.chacha20_poly1305*), 300
 CHAIN_EXT (*Bip44Changes* attribute), 125
 CHAIN_INT (*Bip44Changes* attribute), 125
 ChainCode() (*Bip32Base* method), 84
 ChainCode() (*Bip32KeyData* method), 92
 ChainCode() (*Bip44PrivateKey* method), 132
 ChainCode() (*Bip44PublicKey* method), 131
 ChainCode() (*Slip32DeserializedKey* method), 274
 ChainCode() (*SubstratePathElem* method), 294
 CHAINCODE_BYTE_LEN (*Bip32KeyDataConst* attribute), 87
 ChainX (*CoinsConf* attribute), 184
 CHAINX (*SubstrateCoins* attribute), 281
 ChainX (*SubstrateConf* attribute), 282
 CHANGE (*Bip44Levels* attribute), 125
 Change() (*Bip44* method), 124
 Change() (*Bip44Base* method), 129
 Change() (*Bip49* method), 135
 Change() (*Bip84* method), 139
 Change() (*Bip86* method), 142
 Change() (*CardanoShelley* method), 179
 Change() (*Cip1852* method), 175
 CHARSET (*Bech32BaseConst* attribute), 77
 CheckDecode() (*Base58Decoder* static method), 71
 CheckEncode() (*Base58Encoder* static method), 71
 CHECKSUM_BYTE_LEN (*AlgoAddrConst* attribute), 36
 CHECKSUM_BYTE_LEN (*AlgorandMnemonicConst* attribute), 66
 CHECKSUM_BYTE_LEN (*Base58Const* attribute), 70
 CHECKSUM_BYTE_LEN (*EosAddrConst* attribute), 42
 CHECKSUM_BYTE_LEN (*ErgoAddrConst* attribute), 43
 CHECKSUM_BYTE_LEN (*SS58Const* attribute), 279
 CHECKSUM_BYTE_LEN (*XlmAddrConst* attribute), 58
 CHECKSUM_BYTE_LEN (*XmrAddrConst* attribute), 60
 CHECKSUM_PREFIX (*SS58Const* attribute), 279
 CHECKSUM_STR_LEN (*BchBech32Const* attribute), 73
 CHECKSUM_STR_LEN (*Bech32Const* attribute), 75
 CHECKSUM_STR_LEN (*SegwitBech32Const* attribute), 79
 CHIHUAHUA (*Bip44Coins* attribute), 144
 Chihuahua (*Bip44Conf* attribute), 147
 Chihuahua (*CoinsConf* attribute), 184
 CHIHUAHUA (*Slip173* attribute), 272
 ChildKey() (*Bip32Base* method), 82
 ChildKey() (*Substrate* method), 289
 CHINESE_SIMPLIFIED (*Bip39Languages* attribute), 117
 CHINESE_SIMPLIFIED (*ElectrumV2Languages* attribute), 251
 CHINESE_SIMPLIFIED (*MoneroLanguages* attribute), 259
 CHINESE_TRADITIONAL (*Bip39Languages* attribute), 117
 Cip1852 (class in *bip_utils.cardano.cip1852.cip1852*), 172
 Cip1852Coins (class in *bip_utils.cardano.cip1852.conf.cip1852_coins*), 176
 Cip1852Conf (class in *bip_utils.cardano.cip1852.conf.cip1852_conf*), 176
 Cip1852ConfGetter (class in *bip_utils.cardano.cip1852.conf.cip1852_conf_getter*), 177
 Cip1852ConfGetterConst (class in *bip_utils.cardano.cip1852.conf.cip1852_conf_getter*), 176
 Cip1852Const (class in *bip_utils.cardano.cip1852.cip1852*), 172
 CkdPriv() (*Bip32KholawEd25519KeyDerivatorBase* class method), 103
 CkdPriv() (*Bip32Slip10EcdsaDerivator* class method), 106
 CkdPriv() (*Bip32Slip10Ed25519Derivator* class method), 107
 CkdPriv() (*IBip32KeyDerivator* class method), 85
 CkdPub() (*Bip32KholawEd25519KeyDerivatorBase* class method), 103
 CkdPub() (*Bip32Slip10EcdsaDerivator* class method), 106
 CkdPub() (*Bip32Slip10Ed25519Derivator* class method), 107
 CkdPub() (*IBip32KeyDerivator* class method), 85
 COIN (*Bip44Levels* attribute), 125
 Coin() (*Bip44* method), 124
 Coin() (*Bip44Base* method), 129
 Coin() (*Bip49* method), 135
 Coin() (*Bip84* method), 138
 Coin() (*Bip86* method), 142
 Coin() (*Cip1852* method), 174
 COIN_TO_CONF (*Bip44ConfGetterConst* attribute), 150
 COIN_TO_CONF (*Bip49ConfGetterConst* attribute), 154
 COIN_TO_CONF (*Bip84ConfGetterConst* attribute), 156
 COIN_TO_CONF (*Bip86ConfGetterConst* attribute), 158
 COIN_TO_CONF (*Cip1852ConfGetterConst* attribute), 176
 COIN_TO_CONF (*MoneroConfGetterConst* attribute), 257
 COIN_TO_CONF (*SubstrateConfGetterConst* attribute), 283
 CoinConf (class in *bip_utils.coin_conf.coin_conf*), 182
 CoinConf() (*Bip44Base* method), 126
 CoinConf() (*Monero* method), 267

CoinConf() (*Substrate method*), 290
CoinIndex() (*BipCoinConf method*), 160
CoinNames (class in *bip_utils.utils.conf.coin_names*), 296
CoinNames() (*BipCoinConf method*), 160
CoinNames() (*CoinConf method*), 182
CoinNames() (*MoneroCoinConf method*), 255
CoinNames() (*SubstrateCoinConf method*), 281
CoinsConf (class in *bip_utils.coin_conf.coins_conf*), 183
COMPR_PUB_KEY_SUFFIX (*WifConst attribute*), 327
COMPRESSED (*P2PKHPublicKeyModes attribute*), 21
CompressedLength() (*Ed25519Blake2bPublicKey static method*), 209
CompressedLength() (*Ed25519MoneroPublicKey static method*), 215
CompressedLength() (*Ed25519PublicKey static method*), 199
CompressedLength() (*IPublicKey static method*), 189
CompressedLength() (*Nist256p1PublicKey static method*), 218
CompressedLength() (*Secp256k1PublicKeyCoincurve static method*), 224
CompressedLength() (*Secp256k1PublicKeyEcdsa static method*), 227
CompressedLength() (*Sr25519PublicKey static method*), 235
ComputeAndEncodeKeys() (*MoneroSubaddress method*), 271
ComputeChecksum() (*AlgorandMnemonicUtils static method*), 68
ComputeChecksum() (*Base58Utils static method*), 70
ComputeChecksum() (*BchBech32Utils static method*), 73
ComputeChecksum() (*Bech32Utils static method*), 76
ComputeChecksum() (*MoneroMnemonicUtils static method*), 264
ComputeChecksumWordIndex() (*AlgorandMnemonicUtils static method*), 68
ComputeKeys() (*MoneroSubaddress method*), 271
ComputePrivateKey() (*BrainwalletAlgoDoubleSha256 static method*), 165
ComputePrivateKey() (*BrainwalletAlgoPbkdf2HmacSha512 static method*), 165
ComputePrivateKey() (*BrainwalletAlgoScrypt static method*), 166
ComputePrivateKey() (*BrainwalletAlgoSha256 static method*), 165
ComputePrivateKey() (*IBrainwalletAlgo static method*), 167
Convert() (*BchAddrConverter static method*), 40
ConvertBits() (*AlgorandMnemonicUtils static method*), 68
ConvertBits() (*Bech32BaseUtils static method*), 78
ConvertFromBase32() (*Bech32BaseUtils static method*), 77
ConvertToBase32() (*Bech32BaseUtils static method*), 77
ConvertToPublic() (*Bip32Base method*), 83
ConvertToPublic() (*Substrate method*), 290
CoordinateLength() (*DummyPoint static method*), 186
CoordinateLength() (*Ed25519Point static method*), 202
CoordinateLength() (*IPoint static method*), 192
CoordinateLength() (*Nist256p1Point static method*), 221
CoordinateLength() (*Secp256k1PointCoincurve static method*), 230
CoordinateLength() (*Secp256k1PointEcdsa static method*), 232
COSMOS (*Bip44Coins attribute*), 144
Cosmos (*Bip44Conf attribute*), 147
Cosmos (*CoinsConf attribute*), 184
COSMOS (*Slip173 attribute*), 272
Crc32 (class in *bip_utils.utils.crypto.crc*), 301
Curve() (*Bip32Base class method*), 84
CURVE_ORDER (*Ed25519Blake2bConst attribute*), 208
CURVE_ORDER (*Ed25519Const attribute*), 198
CURVE_ORDER (*Ed25519KholawConst attribute*), 212
CURVE_ORDER (*Ed25519MoneroConst attribute*), 215
CURVE_ORDER (*Nist256p1Const attribute*), 217
CURVE_ORDER (*Secp256k1Const attribute*), 223
CURVE_ORDER (*Sr25519Const attribute*), 234
CurveType() (*Bip32Base static method*), 84
CurveType() (*Bip32KholawEd25519 static method*), 102
CurveType() (*Bip32Slip10Ed25519 static method*), 105
CurveType() (*Bip32Slip10Ed25519Blake2b static method*), 105
CurveType() (*Bip32Slip10Nist256p1 static method*), 109
CurveType() (*Bip32Slip10Secp256k1 static method*), 110
CurveType() (*CardanoByronLegacyBip32 static method*), 167
CurveType() (*Ed25519Blake2bPoint static method*), 211
CurveType() (*Ed25519Blake2bPrivateKey static method*), 211
CurveType() (*Ed25519Blake2bPublicKey static method*), 209
CurveType() (*Ed25519KholawPoint static method*), 214
CurveType() (*Ed25519KholawPrivateKey static method*), 213
CurveType() (*Ed25519KholawPublicKey static method*), 212
CurveType() (*Ed25519MoneroPoint static method*), 217
CurveType() (*Ed25519MoneroPrivateKey static method*), 217

method), 216
 CurveType() (Ed25519MoneroPublicKey static method), 215
 CurveType() (Ed25519Point static method), 202
 CurveType() (Ed25519PrivateKey static method), 200
 CurveType() (Ed25519PublicKey static method), 199
 CurveType() (IPoint static method), 192
 CurveType() (IPrivateKey static method), 191
 CurveType() (IPublicKey static method), 189
 CurveType() (Nist256p1Point static method), 221
 CurveType() (Nist256p1PrivateKey static method), 220
 CurveType() (Nist256p1PublicKey static method), 218
 CurveType() (Secp256k1PointCoincurve static method), 229
 CurveType() (Secp256k1PointEcdsa static method), 232
 CurveType() (Secp256k1PrivateKeyCoincurve static method), 225
 CurveType() (Secp256k1PrivateKeyEcdsa static method), 228
 CurveType() (Secp256k1PublicKeyCoincurve static method), 224
 CurveType() (Secp256k1PublicKeyEcdsa static method), 227
 CurveType() (Sr25519Point static method), 238
 CurveType() (Sr25519PrivateKey static method), 237
 CurveType() (Sr25519PublicKey static method), 235
 CZECH (Bip39Languages attribute), 117

D

DASH (Bip44Coins attribute), 144
 DASH (Bip49Coins attribute), 152
 DASH (Slip44 attribute), 276
 DASH_TESTNET (Bip44Coins attribute), 146
 DASH_TESTNET (Bip49Coins attribute), 153
 DashMainNet (Bip44Conf attribute), 147
 DashMainNet (Bip49Conf attribute), 153
 DashMainNet (CoinsConf attribute), 184
 DashTestNet (Bip44Conf attribute), 147
 DashTestNet (Bip49Conf attribute), 153
 DashTestNet (CoinsConf attribute), 184
 DATA_BYTE_LEN (SS58Const attribute), 279
 DataBytes (class in bip_utils.utils.misc.data_bytes), 316
 Decode() (AlgorandMnemonicDecoder method), 66
 Decode() (AlgoUtils static method), 309
 Decode() (Base32Decoder static method), 310
 Decode() (Base58Decoder static method), 71
 Decode() (Base58XmrDecoder static method), 72
 Decode() (BchBech32Decoder class method), 74
 Decode() (Bech32Decoder class method), 77
 Decode() (Bip39MnemonicDecoder method), 118
 Decode() (CborIndefiniteLenArrayDecoder static method), 316
 Decode() (ElectrumV1MnemonicDecoder method), 246
 Decode() (ElectrumV2MnemonicDecoder method), 252
 Decode() (MnemonicDecoderBase method), 322
 Decode() (MoneroMnemonicDecoder method), 260
 Decode() (SegwitBech32Decoder class method), 79
 Decode() (SS58Decoder static method), 280
 Decode() (WifDecoder static method), 327
 DecodeAddr() (AdaByronAddrDecoder static method), 28
 DecodeAddr() (AdaShelleyAddrDecoder static method), 30
 DecodeAddr() (AdaShelleyStakingAddrDecoder static method), 31
 DecodeAddr() (AlgoAddrDecoder static method), 36
 DecodeAddr() (AptosAddrDecoder static method), 37
 DecodeAddr() (AtomAddrDecoder static method), 38
 DecodeAddr() (AvaxPChainAddrDecoder static method), 39
 DecodeAddr() (AvaxXChainAddrDecoder static method), 39
 DecodeAddr() (BchP2PKHAddrDecoder static method), 22
 DecodeAddr() (BchP2SHAddrDecoder static method), 24
 DecodeAddr() (EgldAddrDecoder static method), 41
 DecodeAddr() (EosAddrDecoder static method), 42
 DecodeAddr() (ErgoP2PKHAddrDecoder static method), 43
 DecodeAddr() (EthAddrDecoder static method), 44
 DecodeAddr() (FilSecp256k1AddrDecoder static method), 45
 DecodeAddr() (IAddrDecoder static method), 46
 DecodeAddr() (IcxAddrDecoder static method), 47
 DecodeAddr() (InjAddrDecoder static method), 48
 DecodeAddr() (NanoAddrDecoder static method), 49
 DecodeAddr() (NearAddrDecoder static method), 50
 DecodeAddr() (NeoAddrDecoder static method), 51
 DecodeAddr() (OkexAddrDecoder static method), 52
 DecodeAddr() (OneAddrDecoder static method), 53
 DecodeAddr() (P2PKHAddrDecoder static method), 21
 DecodeAddr() (P2SHAddrDecoder static method), 23
 DecodeAddr() (P2TRAddrDecoder static method), 25
 DecodeAddr() (P2WPKHAddrDecoder static method), 26
 DecodeAddr() (SolAddrDecoder static method), 54
 DecodeAddr() (SubstrateEd25519AddrDecoder static method), 55
 DecodeAddr() (SubstrateSr25519AddrDecoder static method), 55
 DecodeAddr() (SuiAddrDecoder static method), 57
 DecodeAddr() (TrxAddrDecoder static method), 57
 DecodeAddr() (XlmAddrDecoder static method), 59
 DecodeAddr() (XmrAddrDecoder static method), 60
 DecodeAddr() (XmrIntegratedAddrDecoder static method), 61
 DecodeAddr() (XrpAddrDecoder static method), 62

- `DecodeAddr()` (*XtzAddrDecoder* static method), 63
`DecodeAddr()` (*ZilAddrDecoder* static method), 64
`DecodeWithChecksum()` (*Bip39MnemonicDecoder* method), 118
`Decrypt()` (*AesEcbDecrypter* method), 298
`Decrypt()` (*Bip38EcDecrypter* static method), 114
`Decrypt()` (*Bip38NoEcDecrypter* static method), 115
`Decrypt()` (*ChaCha20Poly1305* static method), 300
`DecryptEc()` (*Bip38Decrypter* static method), 111
`DecryptHdPath()` (*AdaByronAddrDecoder* static method), 28
`DecryptNoEc()` (*Bip38Decrypter* static method), 111
`DEF_PROGRAM_ID` (*SplTokenConst* attribute), 278
`DEF_TOKEN_PROGRAM_ID` (*SplTokenConst* attribute), 278
`DefaultPath()` (*BipCoinConf* method), 160
`Depth()` (*Bip32Base* method), 83
`Depth()` (*Bip32KeyData* method), 92
`DEPTH_BYTE_LEN` (*Bip32KeyDataConst* attribute), 87
`DeriveDefaultPath()` (*Bip44Base* method), 127
`DeriveKey()` (*Pbkdf2HmacSha512* static method), 304
`DeriveKey()` (*Scrypt* static method), 305
`DerivePath()` (*Bip32Base* method), 82
`DerivePath()` (*Substrate* method), 290
`DeserializeKey()` (*Bip32KeyDeserializer* class method), 95
`DeserializeKey()` (*Slip32KeyDeserializer* class method), 275
`Digest()` (*Sha256* method), 306
`DigestSize()` (*Blake2b160* static method), 299
`DigestSize()` (*Blake2b224* static method), 299
`DigestSize()` (*Blake2b256* static method), 300
`DigestSize()` (*Blake2b32* static method), 299
`DigestSize()` (*Blake2b40* static method), 299
`DigestSize()` (*Blake2b512* static method), 300
`DigestSize()` (*Crc32* static method), 302
`DigestSize()` (*DoubleSha256* static method), 307
`DigestSize()` (*Hash160* static method), 303
`DigestSize()` (*HmacSha256* static method), 303
`DigestSize()` (*HmacSha512* static method), 304
`DigestSize()` (*Kekkek256* static method), 308
`DigestSize()` (*Ripemd160* static method), 305
`DigestSize()` (*Sha256* static method), 306
`DigestSize()` (*Sha3_256* static method), 308
`DigestSize()` (*Sha512* static method), 307
`DigestSize()` (*Sha512_256* static method), 307
`DigestSize()` (*XModemCrc* static method), 302
`DOGE_COIN` (*Bip44Coins* attribute), 144
`DOGE_COIN` (*Bip49Coins* attribute), 152
`DOGE_COIN` (*Slip44* attribute), 276
`DOGE_COIN_TESTNET` (*Bip44Coins* attribute), 146
`DOGE_COIN_TESTNET` (*Bip49Coins* attribute), 153
`DogecoinMainNet` (*Bip44Conf* attribute), 147
`DogecoinMainNet` (*Bip49Conf* attribute), 153
`DogecoinMainNet` (*CoinsConf* attribute), 184
`DogecoinTestNet` (*Bip44Conf* attribute), 148
`DogecoinTestNet` (*Bip49Conf* attribute), 153
`DogecoinTestNet` (*CoinsConf* attribute), 184
`DOUBLE_SHA256` (*BrainwalletAlgos* attribute), 164
`DoubleSha256` (class in *bip_utils.utils.crypto.sha2*), 306
`DummyPoint` (class in *bip_utils.ecc.common.dummy_point*), 186
`DummyPointConst` (class in *bip_utils.ecc.common.dummy_point*), 186
`DUTCH` (*MoneroLanguages* attribute), 259
- ## E
- `ECASH` (*Bip44Coins* attribute), 144
`ECASH` (*Bip49Coins* attribute), 152
`ECASH_TESTNET` (*Bip44Coins* attribute), 146
`ECASH_TESTNET` (*Bip49Coins* attribute), 153
`EcashMainNet` (*Bip44Conf* attribute), 148
`EcashMainNet` (*Bip49Conf* attribute), 154
`EcashMainNet` (*CoinsConf* attribute), 184
`EcashTestNet` (*Bip44Conf* attribute), 148
`EcashTestNet` (*Bip49Conf* attribute), 154
`EcashTestNet` (*CoinsConf* attribute), 184
`EccConf` (class in *bip_utils.ecc.conf*), 194
`EcdsaKeysConst` (class in *bip_utils.ecc.ecdsa.ecdsa_keys*), 197
`ED25519` (*EllipticCurveTypes* attribute), 197
`ED25519_BLAKE2B` (*EllipticCurveTypes* attribute), 197
`ED25519_KHOLAW` (*EllipticCurveTypes* attribute), 197
`ED25519_MONERO` (*EllipticCurveTypes* attribute), 197
`Ed25519Blake2bConst` (class in *bip_utils.ecc.ed25519_blake2b.ed25519_blake2b_const*), 208
`Ed25519Blake2bPoint` (class in *bip_utils.ecc.ed25519_blake2b.ed25519_blake2b_point*), 211
`Ed25519Blake2bPrivateKey` (class in *bip_utils.ecc.ed25519_blake2b.ed25519_blake2b_keys*), 210
`Ed25519Blake2bPublicKey` (class in *bip_utils.ecc.ed25519_blake2b.ed25519_blake2b_keys*), 209
`Ed25519Const` (class in *bip_utils.ecc.ed25519.ed25519_const*), 198
`Ed25519KeysConst` (class in *bip_utils.ecc.ed25519.ed25519_keys*), 198
`Ed25519KholawConst` (class in *bip_utils.ecc.ed25519_kholaw.ed25519_kholaw_const*), 212
`Ed25519KholawKeysConst` (class in *bip_utils.ecc.ed25519_kholaw.ed25519_kholaw_keys*), 212
`Ed25519KholawPoint` (class in *bip_utils.ecc.ed25519_kholaw.ed25519_kholaw_point*), 214

Ed25519KholawPrivateKey	(class in bip_utils.ecc.ed25519_kholaw.ed25519_kholaw_keys), 213	bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic), 245
Ed25519KholawPublicKey	(class in bip_utils.ecc.ed25519_kholaw.ed25519_kholaw_keys), 212	ElectrumV1MnemonicDecoder (class in bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic_decode), 246
Ed25519MoneroConst	(class in bip_utils.ecc.ed25519_monero.ed25519_monero_const), 215	ElectrumV1MnemonicEncoder (class in bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic_encode), 247
Ed25519MoneroPoint	(class in bip_utils.ecc.ed25519_monero.ed25519_monero_point), 217	ElectrumV1MnemonicGenerator (class in bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic_generate), 247
Ed25519MoneroPrivateKey	(class in bip_utils.ecc.ed25519_monero.ed25519_monero_keys), 216	ElectrumV1MnemonicGeneratorConst (class in bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic_generate), 247
Ed25519MoneroPublicKey	(class in bip_utils.ecc.ed25519_monero.ed25519_monero_keys), 215	ElectrumV1MnemonicValidator (class in bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic_validate), 249
Ed25519Point	(class in bip_utils.ecc.ed25519.ed25519_point), 201	ElectrumV1SeedGenerator (class in bip_utils.electrum.mnemonic_v1.electrum_v1_seed_generator), 249
Ed25519PointConst	(class in bip_utils.ecc.ed25519.ed25519_point), 201	ElectrumV1SeedGeneratorConst (class in bip_utils.electrum.mnemonic_v1.electrum_v1_seed_generator), 249
Ed25519PrivateKey	(class in bip_utils.ecc.ed25519.ed25519_keys), 200	ElectrumV1WordsListFinder (class in bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic_utils), 248
Ed25519PublicKey	(class in bip_utils.ecc.ed25519.ed25519_keys), 198	ElectrumV1WordsListGetter (class in bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic_utils), 248
Ed25519Utils	(class in bip_utils.ecc.ed25519.ed25519_utils), 204	ElectrumV1WordsNum (class in bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic), 245
Edgeware (CoinsConf attribute), 184		ElectrumV2Base (class in bip_utils.electrum.electrum_v2), 240
EDGEWARE (SubstrateCoins attribute), 281		ElectrumV2EntropyBitLen (class in bip_utils.electrum.mnemonic_v2.electrum_v2_entropy_generator), 250
Edgeware (SubstrateConf attribute), 282		ElectrumV2EntropyGenerator (class in bip_utils.electrum.mnemonic_v2.electrum_v2_entropy_generator), 250
EgldAddr (in module bip_utils.addr.egld_addr), 41		ElectrumV2EntropyGeneratorConst (class in bip_utils.electrum.mnemonic_v2.electrum_v2_entropy_generator), 250
EgldAddrDecoder (class in bip_utils.addr.egld_addr), 41		ElectrumV2Languages (class in bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic), 251
EgldAddrEncoder (class in bip_utils.addr.egld_addr), 41		ElectrumV2MnemonicConst (class in bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic), 251
ElectrumV1 (class in bip_utils.electrum.electrum_v1), 238		ElectrumV2MnemonicDecoder (class in bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_decode), 251
ElectrumV1EntropyBitLen	(class in bip_utils.electrum.mnemonic_v1.electrum_v1_entropy_generator), 244	
ElectrumV1EntropyGenerator	(class in bip_utils.electrum.mnemonic_v1.electrum_v1_entropy_generator), 245	
ElectrumV1EntropyGeneratorConst	(class in bip_utils.electrum.mnemonic_v1.electrum_v1_entropy_generator), 245	
ElectrumV1Languages	(class in bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic), 245	
ElectrumV1Mnemonic	(class in bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic), 246	
ElectrumV1MnemonicConst	(class in bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic), 246	

	252	ENC_KEY_PREFIX (<i>Bip38NoEcConst</i> attribute), 114
ElectrumV2MnemonicEncoder	(class in <i>bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_encoder</i>), 67	Encode() (<i>AlgorandMnemonicEncoder</i> method), 67
	252	Encode() (<i>AlgoUtils</i> static method), 309
ElectrumV2MnemonicGenerator	(class in <i>bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_generator</i>), 70	Encode() (<i>Base32Encoder</i> static method), 310
	253	Encode() (<i>Base58Encoder</i> static method), 70
	253	Encode() (<i>Base58XmrEncoder</i> static method), 72
ElectrumV2MnemonicGeneratorConst	(class in <i>bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_generator_const</i>), 76	Encode() (<i>BchBech32Encoder</i> class method), 74
	253	Encode() (<i>Bech32Encoder</i> class method), 76
ElectrumV2MnemonicTypes	(class in <i>bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_types</i>), 316	Encode() (<i>Bip39MnemonicEncoder</i> method), 119
	251	Encode() (<i>CborIndefiniteLenArrayEncoder</i> static method), 316
ElectrumV2MnemonicUtils	(class in <i>bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_utils</i>), 261	Encode() (<i>ElectrumV1MnemonicEncoder</i> method), 247
	254	Encode() (<i>ElectrumV2MnemonicEncoder</i> method), 252
ElectrumV2MnemonicUtilsConst	(class in <i>bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_utils_const</i>), 261	Encode() (<i>MnemonicEncoderBase</i> method), 323
	254	Encode() (<i>MoneroMnemonicNoChecksumEncoder</i> method), 261
ElectrumV2MnemonicValidator	(class in <i>bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_validator</i>), 279	Encode() (<i>MoneroMnemonicWithChecksumEncoder</i> method), 261
	254	Encode() (<i>SegwitBech32Encoder</i> class method), 79
ElectrumV2SeedGenerator	(class in <i>bip_utils.electrum.mnemonic_v2.electrum_v2_seed_generator</i>), 285	Encode() (<i>SS58Encoder</i> static method), 279
	255	Encode() (<i>SubstrateScaleBytesEncoder</i> class method), 285
ElectrumV2SeedGeneratorConst	(class in <i>bip_utils.electrum.mnemonic_v2.electrum_v2_seed_generator_const</i>), 284	Encode() (<i>SubstrateScaleCUintEncoder</i> class method), 285
	254	Encode() (<i>SubstrateScaleEncoderBase</i> class method), 284
ElectrumV2Segwit	(class in <i>bip_utils.electrum.electrum_v2</i>), 243	Encode() (<i>SubstrateScaleU128Encoder</i> class method), 287
	251	Encode() (<i>SubstrateScaleU16Encoder</i> class method), 286
ElectrumV2Standard	(class in <i>bip_utils.electrum.electrum_v2</i>), 242	Encode() (<i>SubstrateScaleU256Encoder</i> class method), 287
ElectrumV2WordsNum	(class in <i>bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic</i>), 286	Encode() (<i>SubstrateScaleU32Encoder</i> class method), 286
	251	Encode() (<i>SubstrateScaleU64Encoder</i> class method), 287
EllipticCurve	(class in <i>bip_utils.ecc.curve.elliptic_curve</i>), 194	Encode() (<i>SubstrateScaleU8Encoder</i> class method), 286
EllipticCurveGetter	(class in <i>bip_utils.ecc.curve.elliptic_curve_getter</i>), 196	Encode() (<i>WifEncoder</i> static method), 327
EllipticCurveGetterConst	(class in <i>bip_utils.ecc.curve.elliptic_curve_getter</i>), 196	ENCODED_ELEM_MAX_BYTE_LEN (<i>SubstratePathConst</i> attribute), 293
EllipticCurveTypes	(class in <i>bip_utils.ecc.curve.elliptic_curve_types</i>), 197	EncodeKey() (<i>AdaByronIcarusAddrEncoder</i> static method), 29
ELROND (<i>Bip44Coins</i> attribute), 144		EncodeKey() (<i>AdaByronLegacyAddrEncoder</i> static method), 29
Elrond (<i>Bip44Conf</i> attribute), 148		EncodeKey() (<i>AdaShelleyAddrEncoder</i> static method), 31
Elrond (<i>CoinsConf</i> attribute), 184		EncodeKey() (<i>AdaShelleyStakingAddrEncoder</i> static method), 31
ELROND (<i>Slip173</i> attribute), 272		EncodeKey() (<i>AlgoAddrEncoder</i> static method), 36
ELROND (<i>Slip44</i> attribute), 277		EncodeKey() (<i>AptosAddrEncoder</i> static method), 37
ENC_BYTE_LEN (<i>Bip38EcConst</i> attribute), 113		EncodeKey() (<i>AtomAddrEncoder</i> static method), 38
ENC_KEY_BYTE_LEN (<i>Bip38NoEcConst</i> attribute), 114		EncodeKey() (<i>AvaxPChainAddrEncoder</i> static method), 39
ENC_KEY_PREFIX (<i>Bip38EcConst</i> attribute), 113		EncodeKey() (<i>AvaxXChainAddrEncoder</i> static method), 39

- 40
 EncodeKey() (BchP2PKHAddrEncoder static method), 22
 EncodeKey() (BchP2SHAddrEncoder static method), 24
 EncodeKey() (EgldAddrEncoder static method), 41
 EncodeKey() (EosAddrEncoder static method), 42
 EncodeKey() (ErgoP2PKHAddrEncoder static method), 43
 EncodeKey() (EthAddrEncoder static method), 45
 EncodeKey() (FilSecp256k1AddrEncoder static method), 46
 EncodeKey() (IAddrEncoder static method), 47
 EncodeKey() (IcxAddrEncoder static method), 48
 EncodeKey() (InjAddrEncoder static method), 48
 EncodeKey() (NanoAddrEncoder static method), 50
 EncodeKey() (NearAddrEncoder static method), 50
 EncodeKey() (NeoAddrEncoder static method), 52
 EncodeKey() (OkexAddrEncoder static method), 52
 EncodeKey() (OneAddrEncoder static method), 53
 EncodeKey() (P2PKHAddrEncoder static method), 22
 EncodeKey() (P2SHAddrEncoder static method), 24
 EncodeKey() (P2TRAddrEncoder static method), 26
 EncodeKey() (P2WPKHAddrEncoder static method), 27
 EncodeKey() (SolAddrEncoder static method), 54
 EncodeKey() (SubstrateEd25519AddrEncoder static method), 55
 EncodeKey() (SubstrateSr25519AddrEncoder static method), 56
 EncodeKey() (SuiAddrEncoder static method), 57
 EncodeKey() (TrxAddrEncoder static method), 58
 EncodeKey() (XlmAddrEncoder static method), 59
 EncodeKey() (XmrAddrEncoder static method), 60
 EncodeKey() (XmrIntegratedAddrEncoder static method), 61
 EncodeKey() (XrpAddrEncoder static method), 62
 EncodeKey() (XtzAddrEncoder static method), 63
 EncodeKey() (ZilAddrEncoder static method), 64
 EncodeNoChecksum() (MoneroMnemonicEncoder method), 262
 EncodeNoPadding() (Base32Encoder static method), 311
 EncodeWithChecksum() (MoneroMnemonicEncoder method), 262
 ENCODING_CHECKSUM_CONST (Bech32Const attribute), 75
 Encrypt() (AesEcbEncrypter method), 297
 Encrypt() (Bip38NoEcEncrypter static method), 115
 Encrypt() (ChaCha20Poly1305 static method), 300
 EncryptNoEc() (Bip38Encrypter static method), 110
 ENGLISH (AlgorandLanguages attribute), 66
 ENGLISH (Bip39Languages attribute), 117
 ENGLISH (ElectrumV1Languages attribute), 245
 ENGLISH (ElectrumV2Languages attribute), 251
 ENGLISH (MoneroLanguages attribute), 259
 ENTROPY_BIT_LEN (AlgorandEntropyGeneratorConst attribute), 65
 ENTROPY_BIT_LEN (Bip39EntropyGeneratorConst attribute), 116
 ENTROPY_BIT_LEN (ElectrumV1EntropyGeneratorConst attribute), 245
 ENTROPY_BIT_LEN (ElectrumV2EntropyGeneratorConst attribute), 250
 ENTROPY_BIT_LEN (MoneroEntropyGeneratorConst attribute), 258
 EntropyGenerator (class in bip_utils.utils.mnemonic.entropy_generator), 320
 ENUM_TO_ALGO (BrainwalletAlgoGetterConst attribute), 166
 EOS (Bip44Coins attribute), 144
 Eos (Bip44Conf attribute), 148
 Eos (CoinsConf attribute), 184
 EOS (Slip44 attribute), 277
 EosAddr (in module bip_utils.addr.eos_addr), 42
 EosAddrConst (class in bip_utils.addr.eos_addr), 42
 EosAddrDecoder (class in bip_utils.addr.eos_addr), 42
 EosAddrEncoder (class in bip_utils.addr.eos_addr), 42
 ERGO (Bip44Coins attribute), 144
 ERGO (Slip44 attribute), 277
 ERGO_TESTNET (Bip44Coins attribute), 146
 ErgoAddrConst (class in bip_utils.addr.ergo_addr), 43
 ErgoAddressTypes (class in bip_utils.addr.ergo_addr), 43
 ErgoMainNet (Bip44Conf attribute), 148
 ErgoMainNet (CoinsConf attribute), 184
 ErgoNetworkTypes (class in bip_utils.addr.ergo_addr), 43
 ErgoP2PKHAddr (in module bip_utils.addr.ergo_addr), 44
 ErgoP2PKHAddrDecoder (class in bip_utils.addr.ergo_addr), 43
 ErgoP2PKHAddrEncoder (class in bip_utils.addr.ergo_addr), 43
 ErgoTestNet (Bip44Conf attribute), 148
 ErgoTestNet (CoinsConf attribute), 184
 EthAddr (in module bip_utils.addr.eth_addr), 45
 EthAddrConst (class in bip_utils.addr.eth_addr), 44
 EthAddrDecoder (class in bip_utils.addr.eth_addr), 44
 EthAddrEncoder (class in bip_utils.addr.eth_addr), 44
 ETHEREUM (Bip44Coins attribute), 144
 Ethereum (Bip44Conf attribute), 148
 Ethereum (CoinsConf attribute), 184
 ETHEREUM (Slip44 attribute), 276
 ETHEREUM_CLASSIC (Bip44Coins attribute), 144
 ETHEREUM_CLASSIC (Slip44 attribute), 276
 EthereumClassic (Bip44Conf attribute), 148
 EthereumClassic (CoinsConf attribute), 184

F

- FANTOM_OPERA (*Bip44Coins attribute*), 144
- FantomOpera (*Bip44Conf attribute*), 148
- FantomOpera (*CoinsConf attribute*), 184
- FETCH_AI (*Bip44Coins attribute*), 144
- FETCH_AI (*Slip173 attribute*), 272
- FETCH_AI_ETH (*Bip44Coins attribute*), 144
- FetchAi (*Bip44Conf attribute*), 148
- FetchAi (*CoinsConf attribute*), 184
- FetchAiEth (*Bip44Conf attribute*), 148
- FIELD_SIZE (*P2TRConst attribute*), 25
- FilAddrConst (*class in bip_utils.addr.fil_addr*), 45
- FILECOIN (*Bip44Coins attribute*), 144
- Filecoin (*Bip44Conf attribute*), 148
- Filecoin (*CoinsConf attribute*), 184
- FILECOIN (*Slip44 attribute*), 277
- FillAddrTypes (*class in bip_utils.addr.fil_addr*), 45
- FilSecp256k1Addr (*in module bip_utils.addr.fil_addr*), 46
- FilSecp256k1AddrDecoder (*class in bip_utils.addr.fil_addr*), 45
- FilSecp256k1AddrEncoder (*class in bip_utils.addr.fil_addr*), 46
- FindLanguage() (*Bip39WordsListFinder class method*), 120
- FindLanguage() (*ElectrumV1WordsListFinder class method*), 248
- FindLanguage() (*MnemonicWordsListFinderBase class method*), 326
- FindLanguage() (*MoneroWordsListFinder class method*), 264
- FindPda() (*Sp1Token class method*), 279
- FingerPrint() (*Bip32Base method*), 84
- FingerPrint() (*Bip32PublicKey method*), 97
- FINGERPRINT_BYTE_LEN (*Bip32KeyDataConst attribute*), 87
- FINGERPRINT_MASTER_KEY (*Bip32KeyDataConst attribute*), 87
- FixedLength() (*Bip32ChainCode static method*), 87
- FixedLength() (*Bip32Depth static method*), 88
- FixedLength() (*Bip32FingerPrint static method*), 87
- FixedLength() (*Bip32KeyIndex static method*), 90
- FLAG_BIT_COMPRESSED (*Bip38EcConst attribute*), 113
- FLAG_BIT_LOT_SEQ (*Bip38EcConst attribute*), 113
- FLAGBYTE_COMPRESSED (*Bip38NoEcConst attribute*), 114
- FLAGBYTE_UNCOMPRESSED (*Bip38NoEcConst attribute*), 114
- FORMAT_MAX_VAL (*SS58Const attribute*), 279
- FOUR_BYTE_MODE_MAX_VAL (*SubstrateScaleCUintEncoderConst attribute*), 285
- FRENCH (*Bip39Languages attribute*), 117
- FRENCH (*MoneroLanguages attribute*), 259
- FromBinaryStr() (*BytesUtils static method*), 314
- FromBinaryStr() (*IntegerUtils static method*), 319
- FromBip44PrivateKey() (*Monero class method*), 265
- FromBytes() (*Bip32KeyIndex class method*), 90
- FromBytes() (*Bip32PrivateKey class method*), 98
- FromBytes() (*Bip32PublicKey class method*), 96
- FromBytes() (*DummyPoint class method*), 186
- FromBytes() (*Ed25519Blake2bPrivateKey class method*), 210
- FromBytes() (*Ed25519Blake2bPublicKey class method*), 209
- FromBytes() (*Ed25519KholawPrivateKey class method*), 213
- FromBytes() (*Ed25519MoneroPrivateKey class method*), 216
- FromBytes() (*Ed25519Point class method*), 201
- FromBytes() (*Ed25519PrivateKey class method*), 200
- FromBytes() (*Ed25519PublicKey class method*), 198
- FromBytes() (*IPoint class method*), 192
- FromBytes() (*IPrivateKey class method*), 190
- FromBytes() (*IPublicKey class method*), 188
- FromBytes() (*MoneroPrivateKey class method*), 270
- FromBytes() (*MoneroPublicKey class method*), 269
- FromBytes() (*Nist256p1Point class method*), 220
- FromBytes() (*Nist256p1PrivateKey class method*), 219
- FromBytes() (*Nist256p1PublicKey class method*), 218
- FromBytes() (*Secp256k1PointCoincurve class method*), 229
- FromBytes() (*Secp256k1PointEcdsa class method*), 232
- FromBytes() (*Secp256k1PrivateKeyCoincurve class method*), 225
- FromBytes() (*Secp256k1PrivateKeyEcdsa class method*), 228
- FromBytes() (*Secp256k1PublicKeyCoincurve class method*), 223
- FromBytes() (*Secp256k1PublicKeyEcdsa class method*), 226
- FromBytes() (*Sr25519PrivateKey class method*), 236
- FromBytes() (*Sr25519PublicKey class method*), 235
- FromBytes() (*SubstratePrivateKey class method*), 293
- FromBytes() (*SubstratePublicKey class method*), 291
- FromBytesOrKeyObject() (*Bip32PrivateKey class method*), 98
- FromBytesOrKeyObject() (*Bip32PublicKey class method*), 96
- FromBytesOrKeyObject() (*MoneroPrivateKey class method*), 270
- FromBytesOrKeyObject() (*MoneroPublicKey class method*), 268
- FromBytesOrKeyObject() (*SubstratePrivateKey class method*), 292
- FromBytesOrKeyObject() (*SubstratePublicKey class method*), 291
- FromCip1852Object() (*CardanoShelley class method*), 178

- FromCoinConf() (*MoneroCoinConf* class method), 255
 FromCoinConf() (*SubstrateCoinConf* class method), 280
 FromCoordinates() (*DummyPoint* class method), 186
 FromCoordinates() (*Ed25519Point* class method), 201
 FromCoordinates() (*IPoint* class method), 192
 FromCoordinates() (*Nist256p1Point* class method), 221
 FromCoordinates() (*Sepc256k1PointCoincurve* class method), 229
 FromCoordinates() (*Sepc256k1PointEcdsa* class method), 232
 FromEntropy() (*AlgorandMnemonicGenerator* method), 68
 FromEntropy() (*Bip39MnemonicGenerator* method), 119
 FromEntropy() (*ElectrumV1MnemonicGenerator* method), 248
 FromEntropy() (*ElectrumV2MnemonicGenerator* method), 253
 FromEntropyNoChecksum() (*MoneroMnemonicGenerator* method), 263
 FromEntropyWithChecksum() (*MoneroMnemonicGenerator* method), 263
 FromExtendedKey() (*Bip32Base* class method), 81
 FromExtendedKey() (*Bip44* class method), 122
 FromExtendedKey() (*Bip44Base* class method), 127
 FromExtendedKey() (*Bip49* class method), 133
 FromExtendedKey() (*Bip84* class method), 137
 FromExtendedKey() (*Bip86* class method), 140
 FromExtendedKey() (*Cip1852* class method), 173
 FromHexString() (*BytesUtils* static method), 314
 FromList() (*BytesUtils* static method), 315
 FromList() (*Mnemonic* class method), 321
 FromPoint() (*Bip32PublicKey* class method), 96
 FromPoint() (*Ed25519Blake2bPublicKey* class method), 209
 FromPoint() (*Ed25519PublicKey* class method), 198
 FromPoint() (*IPublicKey* class method), 189
 FromPoint() (*MoneroPublicKey* class method), 269
 FromPoint() (*Nist256p1PublicKey* class method), 218
 FromPoint() (*Sepc256k1PublicKeyCoincurve* class method), 224
 FromPoint() (*Sepc256k1PublicKeyEcdsa* class method), 226
 FromPoint() (*Sr25519PublicKey* class method), 235
 FromPrivateKey() (*Bip32Base* class method), 81
 FromPrivateKey() (*Bip44* class method), 123
 FromPrivateKey() (*Bip44Base* class method), 128
 FromPrivateKey() (*Bip49* class method), 133
 FromPrivateKey() (*Bip84* class method), 137
 FromPrivateKey() (*Bip86* class method), 140
 FromPrivateKey() (*Cip1852* class method), 173
 FromPrivateKey() (*ElectrumV1* class method), 238
 FromPrivateKey() (*Substrate* class method), 288
 FromPrivateKeySpendKey() (*Monero* class method), 266
 FromPublicKey() (*Bip32Base* class method), 82
 FromPublicKey() (*Bip44* class method), 123
 FromPublicKey() (*Bip44Base* class method), 128
 FromPublicKey() (*Bip49* class method), 134
 FromPublicKey() (*Bip84* class method), 137
 FromPublicKey() (*Bip86* class method), 141
 FromPublicKey() (*Cip1852* class method), 173
 FromPublicKey() (*ElectrumV1* class method), 238
 FromPublicKey() (*Substrate* class method), 289
 FromSeed() (*Bip32Base* class method), 80
 FromSeed() (*Bip44* class method), 122
 FromSeed() (*Bip44Base* class method), 127
 FromSeed() (*Bip49* class method), 133
 FromSeed() (*Bip84* class method), 136
 FromSeed() (*Bip86* class method), 140
 FromSeed() (*CardanoByronLegacy* class method), 170
 FromSeed() (*Cip1852* class method), 172
 FromSeed() (*ElectrumV1* class method), 238
 FromSeed() (*ElectrumV2Base* class method), 240
 FromSeed() (*Monero* class method), 265
 FromSeed() (*Substrate* class method), 288
 FromSeedAndPath() (*Bip32Base* class method), 80
 FromSeedAndPath() (*Substrate* class method), 288
 FromString() (*Mnemonic* class method), 321
 FromType() (*EllipticCurveGetter* static method), 196
 FromWatchOnly() (*Monero* class method), 266
 FromWordsNumber() (*AlgorandMnemonicGenerator* method), 67
 FromWordsNumber() (*Bip39MnemonicGenerator* method), 119
 FromWordsNumber() (*ElectrumV1MnemonicGenerator* method), 247
 FromWordsNumber() (*ElectrumV2MnemonicGenerator* method), 253
 FromWordsNumber() (*MoneroMnemonicGenerator* method), 263
- ## G
- Generate() (*AlgorandSeedGenerator* method), 69
 Generate() (*Bip39SeedGenerator* method), 121
 Generate() (*Brainwallet* class method), 163
 Generate() (*CardanoByronLegacySeedGenerator* method), 177
 Generate() (*CardanoIcarusSeedGenerator* method), 178
 Generate() (*ElectrumV1SeedGenerator* method), 249
 Generate() (*ElectrumV2SeedGenerator* method), 255
 Generate() (*EntropyGenerator* method), 320
 Generate() (*IBip39SeedGenerator* method), 121
 Generate() (*MoneroSeedGenerator* method), 265
 Generate() (*SubstrateBip39SeedGenerator* method), 284

- GenerateFromSeed() (Bip32KholawEd25519MstKeyGenerator class method), 104
- GenerateFromSeed() (Bip32Slip10Ed2519MstKeyGenerator class method), 108
- GenerateFromSeed() (Bip32Slip10Nist256p1MstKeyGenerator class method), 108
- GenerateFromSeed() (Bip32Slip10Secp256k1MstKeyGenerator class method), 109
- GenerateFromSeed() (CardanoByronLegacyMstKeyGenerator class method), 168
- GenerateFromSeed() (CardanoIcarusMstKeyGenerator class method), 169
- GenerateFromSeed() (IBip32MstKeyGenerator class method), 86
- GenerateIntermediatePassphrase() (Bip38EcKeysGenerator static method), 113
- GeneratePrivateKey() (Bip38EcKeysGenerator static method), 113
- GeneratePrivateKeyEc() (Bip38Encrypter static method), 111
- GenerateWithCustomAlgo() (Brainwallet class method), 163
- GENERATOR (Ed25519Blake2bConst attribute), 208
- GENERATOR (Ed25519Const attribute), 198
- GENERATOR (Ed25519KholawConst attribute), 212
- GENERATOR (Ed25519MoneroConst attribute), 215
- GENERATOR (Nist256p1Const attribute), 217
- GENERATOR (Secp256k1Const attribute), 223
- GENERATOR (Sr25519Const attribute), 234
- Generator() (EllipticCurve method), 195
- GENERIC (SubstrateCoins attribute), 281
- Generic (SubstrateConf attribute), 282
- GenericSubstrate (CoinsConf attribute), 184
- GERMAN (MoneroLanguages attribute), 259
- GetAddress() (CardanoByronLegacy method), 172
- GetAddress() (ElectrumV1 method), 240
- GetAddress() (ElectrumV2Base method), 242
- GetAddress() (ElectrumV2Segwit method), 244
- GetAddress() (ElectrumV2Standard method), 243
- GetAlgo() (BrainwalletAlgoGetter static method), 166
- GetAssociatedTokenAddress() (SplToken class method), 278
- GetAssociatedTokenAddressWithProgramId() (SplToken class method), 278
- GetByLanguage() (Bip39WordsListGetter method), 120
- GetByLanguage() (ElectrumV1WordsListGetter method), 248
- GetByLanguage() (MnemonicWordsListGetterBase method), 325
- GetByLanguage() (MoneroWordsListGetter method), 263
- GetBytesNumber() (IntegerUtils static method), 319
- GetConfig() (Bip44ConfGetter static method), 152
- GetConfig() (Bip49ConfGetter static method), 155
- GetConfig() (Bip84ConfGetter static method), 157
- GetConfig() (Bip86ConfGetter static method), 158
- GetConfig() (Cip1852ConfGetter static method), 177
- GetConfig() (MoneroConfGetter static method), 257
- GetConfig() (SubstrateConfGetter static method), 283
- GetPrivateKey() (CardanoByronLegacy method), 171
- GetPrivateKey() (ElectrumV1 method), 239
- GetPrivateKey() (ElectrumV2Base method), 241
- GetPrivateKey() (ElectrumV2Segwit method), 243
- GetPrivateKey() (ElectrumV2Standard method), 242
- GetPublicKey() (CardanoByronLegacy method), 171
- GetPublicKey() (ElectrumV1 method), 240
- GetPublicKey() (ElectrumV2Base method), 241
- GetPublicKey() (ElectrumV2Segwit method), 244
- GetPublicKey() (ElectrumV2Standard method), 242
- GetWordAtIdx() (MnemonicWordsList method), 324
- GetWordIdx() (MnemonicWordsList method), 324
- ## H
- handle (Sha256 attribute), 306
- HARD_PATH_PREFIX (SubstratePathConst attribute), 293
- Harden() (Bip32KeyIndex method), 90
- HARDENED_CHARS (Bip32PathConst attribute), 99
- HardenIndex() (Bip32KeyIndex static method), 89
- HardenIndex() (Bip32Utils static method), 101
- HARMONY_ONE (Slip173 attribute), 272
- HARMONY_ONE (Slip44 attribute), 277
- HARMONY_ONE_ATOM (Bip44Coins attribute), 144
- HARMONY_ONE_ETH (Bip44Coins attribute), 144
- HARMONY_ONE_METAMASK (Bip44Coins attribute), 144
- HarmonyOne (CoinsConf attribute), 184
- HarmonyOneAtom (Bip44Conf attribute), 148
- HarmonyOneEth (Bip44Conf attribute), 148
- HarmonyOneMetamask (Bip44Conf attribute), 148
- Hash160 (class in bip_utils.utils.crypto.hash160), 302
- HASH_ITR_NUM (ElectrumV1SeedGeneratorConst attribute), 249
- HD_PATH_KEY_PBKDF2_OUT_BYTE_LEN (CardanoByronLegacyConst attribute), 170
- HD_PATH_KEY_PBKDF2_ROUNDS (CardanoByronLegacyConst attribute), 170
- HD_PATH_KEY_PBKDF2_SALT (CardanoByronLegacyConst attribute), 170
- HdPathFromAddress() (CardanoByronLegacy method), 170
- HdPathKey() (CardanoByronLegacy method), 170
- HMAC_KEY (ElectrumV2MnemonicUtilsConst attribute), 254
- HMAC_KEY_ED25519_BYTES (Bip32Slip10MstKeyGeneratorConst attribute), 108
- HMAC_KEY_NIST256P1_BYTES (Bip32Slip10MstKeyGeneratorConst attribute), 108

- 108
HMAC_KEY_SECP256K1_BYTES
(Bip32Slip10MstKeyGeneratorConst attribute), 108
HMAC_MESSAGE_FORMAT (CardanoByronLegacyMstKeyGeneratorConst attribute), 168
HmacSha256 (class in bip_utils.utils.crypto.hmac), 303
HmacSha512 (class in bip_utils.utils.crypto.hmac), 303
HrpExpand() (BchBech32Utils static method), 73
HrpExpand() (Bech32Utils static method), 75
HUOBI_CHAIN (Bip44Coins attribute), 144
HuobiChain (Bip44Conf attribute), 148
HuobiChain (CoinsConf attribute), 184
- I
- IAddrDecoder (class in bip_utils.addr.iaddr_decoder), 46
IAddrEncoder (class in bip_utils.addr.iaddr_encoder), 47
IBip32KeyDerivator (class in bip_utils.bip.bip32.base.ibip32_key_derivator), 85
IBip32MstKeyGenerator (class in bip_utils.bip.bip32.base.ibip32_mst_key_generator), 86
IBip39SeedGenerator (class in bip_utils.bip.bip39.ibip39_seed_generator), 121
IBrainwalletAlgo (class in bip_utils.brainwallet.ibrainwallet_algo), 167
ICON (Bip44Coins attribute), 144
Icon (Bip44Conf attribute), 148
Icon (CoinsConf attribute), 184
ICON (Slip44 attribute), 276
IcxAddr (in module bip_utils.addr.icx_addr), 48
IcxAddrConst (class in bip_utils.addr.icx_addr), 47
IcxAddrDecoder (class in bip_utils.addr.icx_addr), 47
IcxAddrEncoder (class in bip_utils.addr.icx_addr), 48
Increase() (Bip32Depth method), 88
INDEF_LEN_ARRAY_END (CborIds attribute), 315
INDEF_LEN_ARRAY_START (CborIds attribute), 315
Index() (Bip32Base method), 84
Index() (Bip32KeyData method), 92
InjAddr (in module bip_utils.addr.inj_addr), 49
InjAddrDecoder (class in bip_utils.addr.inj_addr), 48
InjAddrEncoder (class in bip_utils.addr.inj_addr), 48
INJECTIVE (Bip44Coins attribute), 144
Injective (Bip44Conf attribute), 148
Injective (CoinsConf attribute), 184
INJECTIVE (Slip173 attribute), 272
Instance() (MnemonicWordsListGetterBase class method), 325
int_decode() (in module bip_utils.ecc.ed25519.lib.ed25519_lib), 204
int_encode() (in module bip_utils.ecc.ed25519.lib.ed25519_lib), 205
INT_PASS_ENC_BYTE_LEN (Bip38EcConst attribute), 113
INT_PASS_MAGIC_NO_LOT_SEQ (Bip38EcConst attribute), 113
INT_PASS_MAGIC_WITH_LOT_SEQ (Bip38EcConst attribute), 113
IntDecode() (Ed25519Utils static method), 204
IntegerUtils (class in bip_utils.utils.misc.integer), 319
IntegratedAddress() (Monero method), 267
IntegratedAddrNetVersion() (MoneroCoinConf method), 256
IntEncode() (Ed25519Utils static method), 204
IPoint (class in bip_utils.ecc.common.ipoint), 192
IPrivateKey (class in bip_utils.ecc.common.ikeys), 190
IPublicKey (class in bip_utils.ecc.common.ikeys), 188
IRIS_NET (Bip44Coins attribute), 144
IRIS_NETWORK (Slip173 attribute), 272
IrisNet (Bip44Conf attribute), 148
IrisNet (CoinsConf attribute), 184
IsAbsolute() (Bip32Path method), 100
IsBitSet() (BitUtils static method), 311
IsHard() (SubstratePathElem method), 294
IsHardened() (Bip32KeyIndex method), 91
IsHardenedIndex() (Bip32KeyIndex static method), 90
IsHardenedIndex() (Bip32Utils static method), 102
IsLevel() (Bip44Base method), 127
IsMasterKey() (Bip32FingerPrint method), 88
IsPublic() (Bip32DeserializedKey method), 95
IsPublic() (Slip32DeserializedKey method), 275
IsPublicDerivationSupported() (Bip32Base class method), 84
IsPublicDerivationSupported() (Bip32KholawEd25519KeyDerivatorBase static method), 103
IsPublicDerivationSupported() (Bip32Slip10EcdsaDerivator static method), 106
IsPublicDerivationSupported() (Bip32Slip10Ed25519Derivator static method), 107
IsPublicDerivationSupported() (IBip32KeyDerivator static method), 85
IsPublicOnly() (Bip32Base method), 83
IsPublicOnly() (Bip44Base method), 126
IsPublicOnly() (CardanoShelley method), 179
IsPublicOnly() (ElectrumV1 method), 239
IsPublicOnly() (ElectrumV2Base method), 241
IsPublicOnly() (Substrate method), 290
IsSoft() (SubstratePathElem method), 294
IsStringMixed() (AlgoUtils static method), 309

- IsTestNet() (*BipCoinConf* method), 160
 IsValid() (*MnemonicValidator* method), 326
 IsValidBytes() (*IPrivateKey* class method), 191
 IsValidBytes() (*IPublicKey* class method), 189
 IsValidEntropyBitLen() (*AlgorandEntropyGenerator* static method), 65
 IsValidEntropyBitLen() (*Bip39EntropyGenerator* static method), 116
 IsValidEntropyBitLen() (*ElectrumV1EntropyGenerator* static method), 245
 IsValidEntropyBitLen() (*ElectrumV2EntropyGenerator* static method), 250
 IsValidEntropyBitLen() (*MoneroEntropyGenerator* static method), 258
 IsValidEntropyByteLen() (*AlgorandEntropyGenerator* static method), 65
 IsValidEntropyByteLen() (*Bip39EntropyGenerator* static method), 116
 IsValidEntropyByteLen() (*ElectrumV1EntropyGenerator* static method), 245
 IsValidEntropyByteLen() (*ElectrumV2EntropyGenerator* static method), 250
 IsValidEntropyByteLen() (*MoneroEntropyGenerator* static method), 258
 IsValidMnemonic() (*ElectrumV2MnemonicUtils* static method), 254
 IsValidPoint() (*IPublicKey* class method), 189
 IsWatchOnly() (*Monero* method), 267
 ITALIAN (*Bip39Languages* attribute), 117
 ITALIAN (*MoneroLanguages* attribute), 259
- ## J
- JAPANESE (*MoneroLanguages* attribute), 259
- ## K
- Karura (*CoinsConf* attribute), 184
 KARURA (*SubstrateCoins* attribute), 281
 Karura (*SubstrateConf* attribute), 282
 KAVA (*Bip44Coins* attribute), 144
 Kava (*Bip44Conf* attribute), 148
 Kava (*CoinsConf* attribute), 184
 KAVA (*Slip173* attribute), 272
 KAVA (*Slip44* attribute), 277
 Kekkak256 (*class in bip_utils.crypto.sha3*), 308
 KEY_HASH_BYTE_LEN (*IdxAddrConst* attribute), 47
 KEY_INDEX_BYTE_LEN (*Bip32KeyDataConst* attribute), 87
 KEY_INDEX_HARDENED_BIT_NUM (*Bip32KeyDataConst* attribute), 87
 KEY_INDEX_MAX_VAL (*Bip32KeyDataConst* attribute), 87
 KEY_NET_VERSION_BYTE_LEN (*Bip32KeyNetVersionsConst* attribute), 93
 KEY_TYPE (*SuiAddrConst* attribute), 56
 KeyBytes() (*Bip32DeserializedKey* method), 95
 KeyBytes() (*Slip32DeserializedKey* method), 274
 KeyData() (*Bip32DeserializedKey* method), 95
 KeyIdentifier() (*Bip32PublicKey* method), 97
 KeyNetVersions() (*Bip32Base* method), 83
 KeyNetVersions() (*BipCoinConf* method), 161
 KeyNetVersions() (*BipLitecoinConf* method), 162
 KeyObject() (*Bip32PrivateKey* method), 98
 KeyObject() (*Bip32PublicKey* method), 97
 KeyObject() (*MoneroPrivateKey* method), 270
 KeyObject() (*MoneroPublicKey* method), 269
 KeyObject() (*SubstratePrivateKey* method), 293
 KeyObject() (*SubstratePublicKey* method), 292
 KeySize() (*ChaCha20Poly1305* static method), 301
 KHOLAW_KEY_NET_VERSIONS (*Bip32Const* attribute), 86
 KOREAN (*Bip39Languages* attribute), 117
 Kusama (*CoinsConf* attribute), 184
 KUSAMA (*Slip44* attribute), 277
 KUSAMA (*SubstrateCoins* attribute), 281
 Kusama (*SubstrateConf* attribute), 282
 KUSAMA_ED25519_SLIP (*Bip44Coins* attribute), 144
 KusamaEd25519Slip (*Bip44Conf* attribute), 149
- ## L
- LANGUAGE_FILES (*Bip39MnemonicConst* attribute), 117
 LANGUAGE_FILES (*ElectrumV1MnemonicConst* attribute), 246
 LANGUAGE_FILES (*MoneroMnemonicConst* attribute), 259
 LANGUAGE_UNIQUE_PREFIX_LEN (*MoneroMnemonicConst* attribute), 259
 Length() (*Bip32KeyNetVersions* static method), 93
 Length() (*Bip32Path* method), 100
 Length() (*DataBytes* method), 316
 Length() (*Ed25519Blake2bPrivateKey* static method), 211
 Length() (*Ed25519KholawPrivateKey* static method), 213
 Length() (*Ed25519PrivateKey* static method), 200
 Length() (*IPrivateKey* static method), 191
 Length() (*MnemonicWordsList* method), 324
 Length() (*Nist256p1PrivateKey* static method), 220
 Length() (*Secp256k1PrivateKeyCoincurve* static method), 225
 Length() (*Secp256k1PrivateKeyEcdsa* static method), 228
 Length() (*Sr25519PrivateKey* static method), 237
 Length() (*SubstratePath* method), 295
 Level() (*Bip44Base* method), 126

LITECOIN (*Bip44Coins attribute*), 145
 LITECOIN (*Bip49Coins attribute*), 152
 LITECOIN (*Bip84Coins attribute*), 156
 LITECOIN (*Slip44 attribute*), 276
 LITECOIN_MAINNET (*Slip173 attribute*), 272
 LITECOIN_TESTNET (*Bip44Coins attribute*), 146
 LITECOIN_TESTNET (*Bip49Coins attribute*), 153
 LITECOIN_TESTNET (*Bip84Coins attribute*), 156
 LITECOIN_TESTNET (*Slip173 attribute*), 272
 LitecoinMainNet (*Bip44Conf attribute*), 149
 LitecoinMainNet (*Bip49Conf attribute*), 154
 LitecoinMainNet (*Bip84Conf attribute*), 156
 LitecoinMainNet (*CoinsConf attribute*), 184
 LitecoinTestNet (*Bip44Conf attribute*), 149
 LitecoinTestNet (*Bip49Conf attribute*), 154
 LitecoinTestNet (*Bip84Conf attribute*), 156
 LitecoinTestNet (*CoinsConf attribute*), 184
 LoadFile() (*MnemonicWordsListFileReader static method*), 325
 LOT_NUM_MAX_VAL (*Bip38EcConst attribute*), 112
 LOT_NUM_MIN_VAL (*Bip38EcConst attribute*), 112

M

m_abbr (*CoinNames attribute*), 296
 m_addr_cls (*BipCoinConf attribute*), 160
 m_addr_cls_legacy (*BipBitcoinCashConf attribute*), 159
 m_addr_net_ver (*MoneroCoinConf attribute*), 255
 m_addr_params (*BipCoinConf attribute*), 160
 m_addr_params (*MoneroCoinConf attribute*), 255
 m_addr_params (*SubstrateCoinConf attribute*), 281
 m_alt_key_net_ver (*BipLitecoinConf attribute*), 162
 m_any_addr_params_fct_call (*BipCoinConf attribute*), 160
 m_bip32_acc (*ElectrumV2Segwit attribute*), 243
 m_bip32_cls (*BipCoinConf attribute*), 160
 m_bip32_obj (*Bip44 attribute*), 125
 m_bip32_obj (*Bip44Base attribute*), 126
 m_bip32_obj (*Bip49 attribute*), 136
 m_bip32_obj (*Bip84 attribute*), 139
 m_bip32_obj (*Bip86 attribute*), 143
 m_bip32_obj (*CardanoByronLegacy attribute*), 170
 m_bip32_obj (*Cip1852 attribute*), 175
 m_bip32_obj (*ElectrumV2Base attribute*), 240
 m_bip32_obj (*ElectrumV2Standard attribute*), 243
 m_bip_obj (*CardanoShelley attribute*), 178
 m_bip_sk_obj (*CardanoShelley attribute*), 178
 m_bit_len (*AlgorandEntropyGenerator attribute*), 65
 m_bit_len (*Bip39EntropyGenerator attribute*), 116
 m_bit_len (*ElectrumV1EntropyGenerator attribute*), 245
 m_bit_len (*ElectrumV2EntropyGenerator attribute*), 251
 m_bit_len (*EntropyGenerator attribute*), 320
 m_bit_len (*MoneroEntropyGenerator attribute*), 258
 m_chain_code (*Bip32KeyData attribute*), 92
 m_chain_code (*Slip32DeserializedKey attribute*), 274
 m_coin_conf (*Bip44 attribute*), 125
 m_coin_conf (*Bip44Base attribute*), 126
 m_coin_conf (*Bip44PrivateKey attribute*), 131
 m_coin_conf (*Bip44PublicKey attribute*), 130
 m_coin_conf (*Bip49 attribute*), 136
 m_coin_conf (*Bip84 attribute*), 139
 m_coin_conf (*Bip86 attribute*), 143
 m_coin_conf (*CardanoShelleyPrivateKeys attribute*), 181
 m_coin_conf (*CardanoShelleyPublicKeys attribute*), 180
 m_coin_conf (*Cip1852 attribute*), 175
 m_coin_conf (*Monero attribute*), 267
 m_coin_conf (*Substrate attribute*), 289
 m_coin_conf (*SubstratePrivateKey attribute*), 293
 m_coin_conf (*SubstratePublicKey attribute*), 292
 m_coin_idx (*BipCoinConf attribute*), 160
 m_coin_name (*CoinConf attribute*), 182
 m_coin_names (*BipCoinConf attribute*), 160
 m_coin_names (*MoneroCoinConf attribute*), 255
 m_coin_names (*SubstrateCoinConf attribute*), 281
 m_data_bytes (*Bip32ChainCode attribute*), 87
 m_data_bytes (*Bip32FingerPrint attribute*), 88
 m_data_bytes (*DataBytes attribute*), 316
 m_def_path (*BipCoinConf attribute*), 160
 m_depth (*Bip32Depth attribute*), 88
 m_depth (*Bip32KeyData attribute*), 92
 m_elem (*SubstratePathElem attribute*), 294
 m_elems (*Bip32Path attribute*), 99
 m_elems (*SubstratePath attribute*), 295
 m_enc_bytes (*Ed25519Blake2bPoint attribute*), 212
 m_enc_bytes (*Ed25519KholawPoint attribute*), 214
 m_enc_bytes (*Ed25519MoneroPoint attribute*), 217
 m_enc_bytes (*Ed25519Point attribute*), 201
 m_entropy_bytes (*AlgorandSeedGenerator attribute*), 69
 m_entropy_bytes (*CardanoIcarusSeedGenerator attribute*), 178
 m_entropy_bytes (*ElectrumV2SeedGenerator attribute*), 255
 m_entropy_bytes (*MoneroSeedGenerator attribute*), 265
 m_entropy_bytes (*SubstrateBip39SeedGenerator attribute*), 284
 m_ext_key (*Ed25519KholawPrivateKey attribute*), 213
 m_fct_names (*BipCoinFctCallsConf attribute*), 159
 m_generator (*EllipticCurve attribute*), 195
 m_idx (*Bip32KeyIndex attribute*), 90
 m_idx_to_words (*MnemonicWordsList attribute*), 324
 m_index (*Bip32KeyData attribute*), 92
 m_int_addr_net_ver (*MoneroCoinConf attribute*), 255

`m_is_absolute` (*Bip32Path* attribute), 99
`m_is_generator` (*Ed25519Blake2bPoint* attribute), 212
`m_is_generator` (*Ed25519KholawPoint* attribute), 214
`m_is_generator` (*Ed25519MoneroPoint* attribute), 217
`m_is_generator` (*Ed25519Point* attribute), 201
`m_is_hard` (*SubstratePathElem* attribute), 294
`m_is_public` (*Bip32DeserializedKey* attribute), 95
`m_is_public` (*Slip32DeserializedKey* attribute), 274
`m_is_testnet` (*BipCoinConf* attribute), 160
`m_key_bytes` (*Bip32DeserializedKey* attribute), 95
`m_key_bytes` (*Slip32DeserializedKey* attribute), 274
`m_key_data` (*Bip32DeserializedKey* attribute), 95
`m_key_net_ver` (*BipCoinConf* attribute), 160
`m_lang` (*AlgorandMnemonicDecoder* attribute), 66
`m_lang` (*Bip39MnemonicDecoder* attribute), 118
`m_lang` (*ElectrumV1MnemonicDecoder* attribute), 246
`m_lang` (*MnemonicDecoderBase* attribute), 322
`m_lang` (*MoneroMnemonicDecoder* attribute), 260
`m_lang` (*MoneroMnemonicEncoderBase* attribute), 261
`m_lang` (*MoneroMnemonicNoChecksumEncoder* attribute), 261
`m_lang` (*MoneroMnemonicWithChecksumEncoder* attribute), 261
`m_mnemonic` (*Bip39SeedGenerator* attribute), 121
`m_mnemonic_decoder` (*AlgorandMnemonicValidator* attribute), 69
`m_mnemonic_decoder` (*Bip39MnemonicValidator* attribute), 121
`m_mnemonic_decoder` (*ElectrumV1MnemonicValidator* attribute), 249
`m_mnemonic_decoder` (*ElectrumV2MnemonicValidator* attribute), 254
`m_mnemonic_decoder` (*MnemonicValidator* attribute), 326
`m_mnemonic_decoder` (*MoneroMnemonicValidator* attribute), 265
`m_mnemonic_encoder` (*AlgorandMnemonicGenerator* attribute), 67
`m_mnemonic_encoder` (*Bip39MnemonicGenerator* attribute), 119
`m_mnemonic_encoder` (*ElectrumV1MnemonicGenerator* attribute), 247
`m_mnemonic_encoder` (*ElectrumV2MnemonicGenerator* attribute), 253
`m_mnemonic_encoder` (*MoneroMnemonicGenerator* attribute), 262
`m_mnemonic_list` (*AlgorandMnemonic* attribute), 66
`m_mnemonic_list` (*Bip39Mnemonic* attribute), 118
`m_mnemonic_list` (*ElectrumV1Mnemonic* attribute), 246
`m_mnemonic_list` (*ElectrumV2Mnemonic* attribute), 252
`m_mnemonic_list` (*Mnemonic* attribute), 321
`m_mnemonic_list` (*MoneroMnemonic* attribute), 260
`m_mnemonic_type` (*ElectrumV2MnemonicDecoder* attribute), 252
`m_mnemonic_type` (*ElectrumV2MnemonicEncoder* attribute), 252
`m_name` (*CoinNames* attribute), 296
`m_name` (*EllipticCurve* attribute), 194
`m_no_chk_enc` (*MoneroMnemonicEncoder* attribute), 262
`m_order` (*EllipticCurve* attribute), 195
`m_params` (*CoinConf* attribute), 182
`m_parent_fprint` (*Bip32KeyData* attribute), 92
`m_path` (*Slip32DeserializedKey* attribute), 274
`m_path` (*Substrate* attribute), 289
`m_point` (*Nist256p1Point* attribute), 221
`m_point` (*Secp256k1PointEcdsa* attribute), 232
`m_point_cls` (*EllipticCurve* attribute), 195
`m_priv_addr_key` (*CardanoShelleyPrivateKeys* attribute), 181
`m_priv_key` (*Bip32Base* attribute), 82
`m_priv_key` (*Bip32KholawEd25519* attribute), 102
`m_priv_key` (*Bip32PrivateKey* attribute), 98
`m_priv_key` (*Bip32Slip10Ed25519* attribute), 105
`m_priv_key` (*Bip32Slip10Ed25519Blake2b* attribute), 105
`m_priv_key` (*Bip32Slip10Nist256p1* attribute), 109
`m_priv_key` (*Bip32Slip10Secp256k1* attribute), 110
`m_priv_key` (*Bip44PrivateKey* attribute), 131
`m_priv_key` (*CardanoByronLegacyBip32* attribute), 167
`m_priv_key` (*CardanoIcarusBip32* attribute), 169
`m_priv_key` (*ElectrumV1* attribute), 239
`m_priv_key` (*MoneroPrivateKey* attribute), 270
`m_priv_key` (*Substrate* attribute), 289
`m_priv_key` (*SubstratePrivateKey* attribute), 293
`m_priv_key_cls` (*EllipticCurve* attribute), 195
`m_priv_net_ver` (*Bip32KeyNetVersions* attribute), 93
`m_priv_net_ver` (*Slip32KeyNetVersions* attribute), 275
`m_priv_sk_key` (*CardanoShelleyPrivateKeys* attribute), 181
`m_priv_skey` (*Monero* attribute), 266
`m_priv_vkey` (*Monero* attribute), 266
`m_priv_vkey` (*MoneroSubaddress* attribute), 271
`m_pub_addr_key` (*CardanoShelleyPublicKeys* attribute), 180
`m_pub_key` (*Bip32Base* attribute), 82
`m_pub_key` (*Bip32KholawEd25519* attribute), 102
`m_pub_key` (*Bip32PublicKey* attribute), 97
`m_pub_key` (*Bip32Slip10Ed25519* attribute), 105
`m_pub_key` (*Bip32Slip10Ed25519Blake2b* attribute), 105
`m_pub_key` (*Bip32Slip10Nist256p1* attribute), 109
`m_pub_key` (*Bip32Slip10Secp256k1* attribute), 110
`m_pub_key` (*Bip44PublicKey* attribute), 130
`m_pub_key` (*CardanoByronLegacyBip32* attribute), 167
`m_pub_key` (*CardanoIcarusBip32* attribute), 169
`m_pub_key` (*ElectrumV1* attribute), 239

- `m_pub_key` (*MoneroPublicKey* attribute), 269
- `m_pub_key` (*Secp256k1PointCoincurve* attribute), 229
- `m_pub_key` (*Substrate* attribute), 289
- `m_pub_key` (*SubstratePublicKey* attribute), 292
- `m_pub_key_cls` (*EllipticCurve* attribute), 195
- `m_pub_net_ver` (*Bip32KeyNetVersions* attribute), 93
- `m_pub_net_ver` (*Slip32KeyNetVersions* attribute), 275
- `m_pub_sk_key` (*CardanoShelleyPublicKeys* attribute), 180
- `m_pub_skey` (*Monero* attribute), 266
- `m_pub_skey` (*MoneroSubaddress* attribute), 271
- `m_pub_vkey` (*Monero* attribute), 266
- `m_pub_vkey` (*MoneroSubaddress* attribute), 271
- `m_seed` (*ElectrumV1SeedGenerator* attribute), 249
- `m_ser_seed_bytes` (*CardanoByronLegacySeedGenerator* attribute), 177
- `m_sign_key` (*Ed25519Blake2bPrivateKey* attribute), 210
- `m_sign_key` (*Ed25519KholawPrivateKey* attribute), 213
- `m_sign_key` (*Ed25519MoneroPrivateKey* attribute), 216
- `m_sign_key` (*Ed25519PrivateKey* attribute), 200
- `m_sign_key` (*Nist256p1PrivateKey* attribute), 219
- `m_sign_key` (*Secp256k1PrivateKeyCoincurve* attribute), 225
- `m_sign_key` (*Secp256k1PrivateKeyEcdsa* attribute), 228
- `m_sign_key` (*Sr25519PrivateKey* attribute), 237
- `m_ss58_format` (*SubstrateCoinConf* attribute), 281
- `m_subaddr` (*Monero* attribute), 267
- `m_subaddr_net_ver` (*MoneroCoinConf* attribute), 255
- `m_use_alt_key_net_ver` (*BipLitecoinConf* attribute), 162
- `m_use_depr_addr` (*BipLitecoinConf* attribute), 162
- `m_use_legacy_addr` (*BipBitcoinCashConf* attribute), 159
- `m_ver_key` (*Ed25519Blake2bPublicKey* attribute), 209
- `m_ver_key` (*Ed25519KholawPublicKey* attribute), 213
- `m_ver_key` (*Ed25519MoneroPublicKey* attribute), 216
- `m_ver_key` (*Ed25519PublicKey* attribute), 199
- `m_ver_key` (*Nist256p1PublicKey* attribute), 218
- `m_ver_key` (*Secp256k1PublicKeyCoincurve* attribute), 224
- `m_ver_key` (*Secp256k1PublicKeyEcdsa* attribute), 227
- `m_ver_key` (*Sr25519PublicKey* attribute), 235
- `m_wif_net_ver` (*BipCoinConf* attribute), 160
- `m_with_chk_enc` (*MoneroMnemonicEncoder* attribute), 262
- `m_words_list` (*AlgorandMnemonicDecoder* attribute), 67
- `m_words_list` (*AlgorandMnemonicEncoder* attribute), 67
- `m_words_list` (*Bip39MnemonicDecoder* attribute), 118
- `m_words_list` (*Bip39MnemonicEncoder* attribute), 119
- `m_words_list` (*ElectrumV1MnemonicDecoder* attribute), 246
- `m_words_list` (*ElectrumV1MnemonicEncoder* attribute), 247
- `m_words_list` (*MnemonicDecoderBase* attribute), 322
- `m_words_list` (*MnemonicEncoderBase* attribute), 323
- `m_words_list` (*MoneroMnemonicDecoder* attribute), 260
- `m_words_list` (*MoneroMnemonicNoChecksumEncoder* attribute), 261
- `m_words_list` (*MoneroMnemonicWithChecksumEncoder* attribute), 261
- `m_words_list_finder_cls` (*AlgorandMnemonicDecoder* attribute), 67
- `m_words_list_finder_cls` (*Bip39MnemonicDecoder* attribute), 118
- `m_words_list_finder_cls` (*ElectrumV1MnemonicDecoder* attribute), 246
- `m_words_list_finder_cls` (*MnemonicDecoderBase* attribute), 322
- `m_words_list_finder_cls` (*MoneroMnemonicDecoder* attribute), 260
- `m_words_lists` (*Bip39WordsListGetter* attribute), 120
- `m_words_lists` (*ElectrumV1WordsListGetter* attribute), 248
- `m_words_lists` (*MnemonicWordsListGetterBase* attribute), 325
- `m_words_lists` (*MoneroWordsListGetter* attribute), 264
- `m_words_to_idx` (*MnemonicWordsList* attribute), 324
- `m_x` (*DummyPoint* attribute), 186
- `m_x` (*Ed25519Blake2bPoint* attribute), 212
- `m_x` (*Ed25519KholawPoint* attribute), 214
- `m_x` (*Ed25519MoneroPoint* attribute), 217
- `m_x` (*Ed25519Point* attribute), 201
- `m_x` (*Sr25519Point* attribute), 238
- `m_y` (*DummyPoint* attribute), 186
- `m_y` (*Ed25519Blake2bPoint* attribute), 212
- `m_y` (*Ed25519KholawPoint* attribute), 214
- `m_y` (*Ed25519MoneroPoint* attribute), 217
- `m_y` (*Ed25519Point* attribute), 201
- `m_y` (*Sr25519Point* attribute), 238
- `MAIN_NET_KEY_NET_VERSIONS` (*Bip32Const* attribute), 86
- `MAINNET` (*AdaShelleyAddrNetworkTags* attribute), 30
- `MAINNET` (*ErgoNetworkTypes* attribute), 43
- `MainNet` (*MoneroConf* attribute), 257
- `MASTER` (*Bip44Levels* attribute), 125
- `MASTER_CHAR` (*Bip32PathConst* attribute), 99
- `MASTER_KEY_HMAC_KEY` (*Bip32KholawMstKeyGeneratorConst* attribute), 104
- `MasterPrivateKey()` (*CardanoByronLegacy* method), 171
- `MasterPrivateKey()` (*ElectrumV1* method), 239
- `MasterPrivateKey()` (*ElectrumV2Base* method), 241
- `MasterPublicKey()` (*CardanoByronLegacy* method),

171
MasterPublicKey() (*ElectrumV1 method*), 239
MasterPublicKey() (*ElectrumV2Base method*), 241
MAX_ATTEMPTS (*ElectrumV2MnemonicGeneratorConst attribute*), 253
METIS (*Bip44Coins attribute*), 145
Metis (*Bip44Conf attribute*), 149
Metis (*CoinsConf attribute*), 185
Mnemonic (*class in bip_utils.utils.mnemonic.mnemonic*), 321
MNEMONIC_WORD_NUM (*AlgorandMnemonicConst attribute*), 66
MNEMONIC_WORD_NUM (*Bip39MnemonicConst attribute*), 117
MNEMONIC_WORD_NUM (*ElectrumV1MnemonicConst attribute*), 246
MNEMONIC_WORD_NUM (*ElectrumV2MnemonicConst attribute*), 251
MNEMONIC_WORD_NUM (*MoneroMnemonicConst attribute*), 259
MNEMONIC_WORD_NUM_CHKSUM (*MoneroMnemonicConst attribute*), 259
MnemonicChecksumError, 323
MnemonicDecoderBase (*class in bip_utils.utils.mnemonic.mnemonic_decoder_base*), 322
MnemonicEncoderBase (*class in bip_utils.utils.mnemonic.mnemonic_encoder_base*), 323
MnemonicLanguages (*class in bip_utils.utils.mnemonic.mnemonic*), 321
MnemonicUtils (*class in bip_utils.utils.mnemonic.mnemonic_utils*), 323
MnemonicValidator (*class in bip_utils.utils.mnemonic.mnemonic_validator*), 326
MnemonicWordsList (*class in bip_utils.utils.mnemonic.mnemonic_utils*), 324
MnemonicWordsListFileReader (*class in bip_utils.utils.mnemonic.mnemonic_utils*), 325
MnemonicWordsListFinderBase (*class in bip_utils.utils.mnemonic.mnemonic_utils*), 325
MnemonicWordsListGetterBase (*class in bip_utils.utils.mnemonic.mnemonic_utils*), 325
module
 bip_utils.addr.ada_byron_addr, 27
 bip_utils.addr.ada_shelley_addr, 30
 bip_utils.addr.addr_dec_utils, 32
 bip_utils.addr.addr_key_validator, 34
 bip_utils.addr.algo_addr, 36
 bip_utils.addr.aptos_addr, 37
 bip_utils.addr.atom_addr, 38
 bip_utils.addr.avax_addr, 39
 bip_utils.addr.bch_addr_converter, 40
 bip_utils.addr.egld_addr, 41
 bip_utils.addr.eos_addr, 42
 bip_utils.addr.ergo_addr, 43
 bip_utils.addr.eth_addr, 44
 bip_utils.addr.fil_addr, 45
 bip_utils.addr.iaddr_decoder, 46
 bip_utils.addr.iaddr_encoder, 47
 bip_utils.addr.icx_addr, 47
 bip_utils.addr.inj_addr, 48
 bip_utils.addr.nano_addr, 49
 bip_utils.addr.near_addr, 50
 bip_utils.addr.neo_addr, 51
 bip_utils.addr.okex_addr, 52
 bip_utils.addr.one_addr, 53
 bip_utils.addr.P2PKH_addr, 21
 bip_utils.addr.P2SH_addr, 23
 bip_utils.addr.P2TR_addr, 25
 bip_utils.addr.P2WPKH_addr, 26
 bip_utils.addr.sol_addr, 54
 bip_utils.addr.substrate_addr, 55
 bip_utils.addr.sui_addr, 56
 bip_utils.addr.trx_addr, 57
 bip_utils.addr.xlm_addr, 58
 bip_utils.addr.xmr_addr, 60
 bip_utils.addr.xrp_addr, 62
 bip_utils.addr.xtz_addr, 63
 bip_utils.addr.zil_addr, 64
 bip_utils.algorand.mnemonic.algorand_entropy_generator, 65
 bip_utils.algorand.mnemonic.algorand_mnemonic, 66
 bip_utils.algorand.mnemonic.algorand_mnemonic_decoder, 66
 bip_utils.algorand.mnemonic.algorand_mnemonic_encoder, 67
 bip_utils.algorand.mnemonic.algorand_mnemonic_generator, 67
 bip_utils.algorand.mnemonic.algorand_mnemonic_utils, 68
 bip_utils.algorand.mnemonic.algorand_mnemonic_validator, 69
 bip_utils.algorand.mnemonic.algorand_seed_generator, 69
 bip_utils.base58.base58, 70
 bip_utils.base58.base58_ex, 72
 bip_utils.base58.base58_xmr, 72
 bip_utils.bech32.bch_bech32, 73
 bip_utils.bech32.bech32, 75
 bip_utils.bech32.bech32_base, 77

bip_utils.bech32.bech32_ex, 78	bip_utils.bip.bip39.ibip39_seed_generator,
bip_utils.bech32.segwit_bech32, 79	121
bip_utils.bip.bip32.base.bip32_base, 80	bip_utils.bip.bip44.bip44, 122
bip_utils.bip.bip32.base.ibip32_key_derivator,	bip_utils.bip.bip44_base.bip44_base, 125
85	bip_utils.bip.bip44_base.bip44_base_ex,
bip_utils.bip.bip32.base.ibip32_mst_key_generator,	130
86	bip_utils.bip.bip44_base.bip44_keys, 130
bip_utils.bip.bip32.bip32_const, 86	bip_utils.bip.bip49.bip49, 133
bip_utils.bip.bip32.bip32_ex, 87	bip_utils.bip.bip84.bip84, 136
bip_utils.bip.bip32.bip32_key_data, 87	bip_utils.bip.bip86.bip86, 140
bip_utils.bip.bip32.bip32_key_net_ver, 93	bip_utils.bip.conf.bip44.bip44_coins, 143
bip_utils.bip.bip32.bip32_key_ser, 94	bip_utils.bip.conf.bip44.bip44_conf, 146
bip_utils.bip.bip32.bip32_keys, 96	bip_utils.bip.conf.bip44.bip44_conf_getter,
bip_utils.bip.bip32.bip32_path, 99	150
bip_utils.bip.bip32.bip32_utils, 101	bip_utils.bip.conf.bip49.bip49_coins, 152
bip_utils.bip.bip32.kholaw.bip32_kholaw_ed25519,	bip_utils.bip.conf.bip49.bip49_conf, 153
102	bip_utils.bip.conf.bip49.bip49_conf_getter,
bip_utils.bip.bip32.kholaw.bip32_kholaw_ed25519_key_derivator,	154
103	bip_utils.bip.conf.bip84.bip84_coins, 156
bip_utils.bip.bip32.kholaw.bip32_kholaw_key_derivator,	bip_utils.bip.conf.bip84.bip84_conf, 156
103	bip_utils.bip.conf.bip84.bip84_conf_getter,
bip_utils.bip.bip32.kholaw.bip32_kholaw_mst_key_generator,	157
104	bip_utils.bip.conf.bip86.bip86_coins, 157
bip_utils.bip.bip32.slip10.bip32_slip10_ed25519,	bip_utils.bip.conf.bip86.bip86_conf, 158
104	bip_utils.bip.conf.bip86.bip86_conf_getter,
bip_utils.bip.bip32.slip10.bip32_slip10_ed25519_blake2b,	158
105	bip_utils.bip.conf.common.bip_bitcoin_cash_conf,
bip_utils.bip.bip32.slip10.bip32_slip10_key_derivator,	159
106	bip_utils.bip.conf.common.bip_coin_conf,
bip_utils.bip.bip32.slip10.bip32_slip10_mst_key_generator,	160
108	bip_utils.bip.conf.common.bip_coins, 162
bip_utils.bip.bip32.slip10.bip32_slip10_nist256p1,	bip_utils.bip.conf.common.bip_conf_const,
109	162
bip_utils.bip.bip32.slip10.bip32_slip10_secp256k1,	bip_utils.bip.conf.common.bip_litecoin_conf,
110	162
bip_utils.bip.bip38.bip38, 110	bip_utils.brainwallet.brainwallet, 163
bip_utils.bip.bip38.bip38_addr, 112	bip_utils.brainwallet.brainwallet_algo,
bip_utils.bip.bip38.bip38_ec, 112	164
bip_utils.bip.bip38.bip38_no_ec, 114	bip_utils.brainwallet.brainwallet_algo_getter,
bip_utils.bip.bip39.bip39_entropy_generator,	166
116	bip_utils.brainwallet.ibrainwallet_algo,
bip_utils.bip.bip39.bip39_mnemonic, 117	167
bip_utils.bip.bip39.bip39_mnemonic_decoder,	bip_utils.cardano.bip32.cardano_byron_legacy_bip32,
118	167
bip_utils.bip.bip39.bip39_mnemonic_encoder,	bip_utils.cardano.bip32.cardano_byron_legacy_key_derivator,
119	168
bip_utils.bip.bip39.bip39_mnemonic_generator,	bip_utils.cardano.bip32.cardano_byron_legacy_mst_key_generator,
119	168
bip_utils.bip.bip39.bip39_mnemonic_utils,	bip_utils.cardano.bip32.cardano_icarus_bip32,
120	169
bip_utils.bip.bip39.bip39_mnemonic_validator,	bip_utils.cardano.bip32.cardano_icarus_mst_key_generator,
121	169
bip_utils.bip.bip39.bip39_seed_generator,	bip_utils.cardano.byron.cardano_byron_legacy,
121	170

bip_utils.cardano.cip1852.cip1852, 172	bip_utils.ecc.ed25519_monero.ed25519_monero_keys, 215
bip_utils.cardano.cip1852.conf.cip1852_coins, 176	bip_utils.ecc.ed25519_monero.ed25519_monero_point, 217
bip_utils.cardano.cip1852.conf.cip1852_conf, 176	bip_utils.ecc.nist256p1.nist256p1, 217
bip_utils.cardano.cip1852.conf.cip1852_conf_getter, 176	bip_utils.ecc.nist256p1.nist256p1_const, 217
bip_utils.cardano.mnemonic.cardano_byron_legacy_seed_generator, 177	bip_utils.ecc.nist256p1.nist256p1_keys, 218
bip_utils.cardano.mnemonic.cardano_icarus_seed_generator, 178	bip_utils.ecc.nist256p1.nist256p1_point, 220
bip_utils.cardano.shelley.cardano_shelley, 178	bip_utils.ecc.secp256k1.secp256k1, 223
bip_utils.cardano.shelley.cardano_shelley_keys, 180	bip_utils.ecc.secp256k1.secp256k1_const, 223
bip_utils.coin_conf.coin_conf, 182	bip_utils.ecc.secp256k1.secp256k1_keys_coincurve, 223
bip_utils.coin_conf.coins_conf, 183	bip_utils.ecc.secp256k1.secp256k1_keys_ecdsa, 226
bip_utils.ecc.common.dummy_point, 186	bip_utils.ecc.secp256k1.secp256k1_point_coincurve, 229
bip_utils.ecc.common.ikeys, 188	bip_utils.ecc.secp256k1.secp256k1_point_ecdsa, 232
bip_utils.ecc.common.ipoint, 192	bip_utils.ecc.sr25519.sr25519, 234
bip_utils.ecc.conf, 194	bip_utils.ecc.sr25519.sr25519_const, 234
bip_utils.ecc.curve.elliptic_curve, 194	bip_utils.ecc.sr25519.sr25519_keys, 235
bip_utils.ecc.curve.elliptic_curve_getter, 196	bip_utils.ecc.sr25519.sr25519_point, 238
bip_utils.ecc.curve.elliptic_curve_types, 197	bip_utils.electrum.electrum_v1, 238
bip_utils.ecc.ecdsa.ecdsa_keys, 197	bip_utils.electrum.electrum_v2, 240
bip_utils.ecc.ed25519.ed25519, 197	bip_utils.electrum.mnemonic_v1.electrum_v1_entropy_gen, 244
bip_utils.ecc.ed25519.ed25519_const, 198	bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic, 245
bip_utils.ecc.ed25519.ed25519_keys, 198	bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic_de, 246
bip_utils.ecc.ed25519.ed25519_point, 201	bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic_en, 247
bip_utils.ecc.ed25519.ed25519_utils, 204	bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic_ge, 247
bip_utils.ecc.ed25519.lib.ed25519_lib, 204	bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic_ut, 248
bip_utils.ecc.ed25519_blake2b.ed25519_blake2b, 208	bip_utils.electrum.mnemonic_v1.electrum_v1_mnemonic_va, 249
bip_utils.ecc.ed25519_blake2b.ed25519_blake2b_ipoint, 208	bip_utils.electrum.mnemonic_v1.electrum_v1_seed_genera, 249
bip_utils.ecc.ed25519_blake2b.ed25519_blake2b_keys, 209	bip_utils.electrum.mnemonic_v2.electrum_v2_entropy_gen, 250
bip_utils.ecc.ed25519_blake2b.ed25519_blake2b_ipoint, 211	bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic, 251
bip_utils.ecc.ed25519_kholaw.ed25519_kholaw, 212	bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_de, 252
bip_utils.ecc.ed25519_kholaw.ed25519_kholaw_const, 212	bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_en, 252
bip_utils.ecc.ed25519_kholaw.ed25519_kholaw_keys, 212	bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_ge, 253
bip_utils.ecc.ed25519_kholaw.ed25519_kholaw_point, 214	
bip_utils.ecc.ed25519_monero.ed25519_monero, 214	
bip_utils.ecc.ed25519_monero.ed25519_monero_const, 215	

bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_util, 254
 bip_utils.electrum.mnemonic_v2.electrum_v2_mnemonic_util_substrate, 254
 bip_utils.electrum.mnemonic_v2.electrum_v2_seed_generator, 254
 bip_utils.monero.conf.monero_coin_conf, 255
 bip_utils.monero.conf.monero_coins, 256
 bip_utils.monero.conf.monero_conf, 257
 bip_utils.monero.conf.monero_conf_getter, 257
 bip_utils.monero.mnemonic.monero_entropy_generator, 258
 bip_utils.monero.mnemonic.monero_mnemonic, 259
 bip_utils.monero.mnemonic.monero_mnemonic_decoder, 260
 bip_utils.monero.mnemonic.monero_mnemonic_encoder, 261
 bip_utils.monero.mnemonic.monero_mnemonic_generator, 262
 bip_utils.monero.mnemonic.monero_mnemonic_util, 263
 bip_utils.monero.mnemonic.monero_mnemonic_validator, 265
 bip_utils.monero.mnemonic.monero_seed_generator, 265
 bip_utils.monero.monero, 265
 bip_utils.monero.monero_ex, 268
 bip_utils.monero.monero_keys, 268
 bip_utils.monero.monero_subaddr, 271
 bip_utils.slip.slip173.slip173, 272
 bip_utils.slip.slip32.slip32, 273
 bip_utils.slip.slip32.slip32_key_net_ver, 275
 bip_utils.slip.slip44.slip44, 276
 bip_utils.solana.spl_token, 278
 bip_utils.ss58.ss58, 279
 bip_utils.ss58.ss58_ex, 280
 bip_utils.substrate.conf.substrate_coin_conf, 280
 bip_utils.substrate.conf.substrate_coins, 281
 bip_utils.substrate.conf.substrate_conf, 282
 bip_utils.substrate.conf.substrate_conf_generator, 283
 bip_utils.substrate.mnemonic.substrate_bip39_seed_generator, 284
 bip_utils.substrate.scale.substrate_scale_monero_base, 284
 bip_utils.substrate.scale.substrate_scale_enc_byte, 285
 bip_utils.substrate.scale.substrate_scale_enc_cuint, 285
 bip_utils.substrate.scale.substrate_scale_enc_uint, 286
 bip_utils.substrate.substrate, 288
 bip_utils.substrate.substrate_ex, 291
 bip_utils.substrate.substrate_keys, 291
 bip_utils.substrate.substrate_path, 293
 bip_utils.utils.conf.coin_names, 296
 bip_utils.utils.crypto.aes_ecb, 297
 bip_utils.utils.crypto.blake2, 298
 bip_utils.utils.crypto.chacha20_poly1305, 300
 bip_utils.utils.crypto.crc, 301
 bip_utils.utils.crypto.hash160, 302
 bip_utils.utils.crypto.hmac, 303
 bip_utils.utils.crypto.pbkdf2, 304
 bip_utils.utils.crypto.ripemd, 305
 bip_utils.utils.crypto.scrip, 305
 bip_utils.utils.crypto.sha2, 306
 bip_utils.utils.crypto.sha3, 308
 bip_utils.utils.misc.algo, 309
 bip_utils.utils.misc.base32, 310
 bip_utils.utils.misc.bit, 311
 bip_utils.utils.misc.bytes, 313
 bip_utils.utils.misc.cbor_indefinite_len_array, 315
 bip_utils.utils.misc.data_bytes, 316
 bip_utils.utils.misc.integer, 319
 bip_utils.utils.misc.string, 320
 bip_utils.utils.mnemonic.entropy_generator, 320
 bip_utils.utils.mnemonic.mnemonic, 321
 bip_utils.utils.mnemonic.mnemonic_decoder_base, 322
 bip_utils.utils.mnemonic.mnemonic_encoder_base, 323
 bip_utils.utils.mnemonic.mnemonic_ex, 323
 bip_utils.utils.mnemonic.mnemonic_util, 323
 bip_utils.utils.mnemonic.mnemonic_validator, 326
 bip_utils.utils.typing.literal, 327
 bip_utils.wif.wif, 327
 Monero (class in bip_utils.monero.monero), 265
 MONERO (Slip44 attribute), 276
 MONERO_ED25519_SLIP (Bip44Coins attribute), 145
 MONERO_MAINNET (MoneroCoins attribute), 256
 MONERO_SEED_GENERATOR (Bip44Coins attribute), 145
 MONERO_STAGENET (MoneroCoins attribute), 256
 MONERO_TESTNET (MoneroCoins attribute), 256
 MoneroCoinConf (class in bip_utils.monero.conf.monero_coin_conf), 255

MoneroCoins	(class in bip_utils.monero.conf.monero_coins),	256	MoneroMnemonicWithChecksumEncoder	(class in bip_utils.monero.mnemonic.monero_mnemonic_encoder),	261
MoneroConf	(class in bip_utils.monero.conf.monero_conf),	257	MoneroPrivateKey	(class in bip_utils.monero.monero_keys),	270
MoneroConfGetter	(class in bip_utils.monero.conf.monero_conf_getter),	257	MoneroPublicKey	(class in bip_utils.monero.monero_keys),	268
MoneroConfGetterConst	(class in bip_utils.monero.conf.monero_conf_getter),	257	MoneroSecp256k1	(Bip44Conf attribute),	149
MoneroEd25519Slip	(Bip44Conf attribute),	149	MoneroSeedGenerator	(class in bip_utils.monero.mnemonic.monero_seed_generator),	265
MoneroEntropyBitLen	(class in bip_utils.monero.mnemonic.monero_entropy_generator),	258	MoneroStageNet	(CoinsConf attribute),	185
MoneroEntropyGenerator	(class in bip_utils.monero.mnemonic.monero_entropy_generator),	258	MoneroSubaddress	(class in bip_utils.monero.monero_subaddr),	271
MoneroEntropyGeneratorConst	(class in bip_utils.monero.mnemonic.monero_entropy_generator),	258	MoneroSubaddressConst	(class in bip_utils.monero.monero_subaddr),	271
MoneroKeyError,	268		MoneroTestNet	(CoinsConf attribute),	185
MoneroLanguages	(class in bip_utils.monero.mnemonic.monero_mnemonic),	259	MoneroWordsListFinder	(class in bip_utils.monero.mnemonic.monero_mnemonic_utils),	264
MoneroMainNet	(CoinsConf attribute),	185	MoneroWordsListGetter	(class in bip_utils.monero.mnemonic.monero_mnemonic_utils),	263
MoneroMnemonic	(class in bip_utils.monero.mnemonic.monero_mnemonic),	260	MoneroWordsNum	(class in bip_utils.monero.mnemonic.monero_mnemonic),	259
MoneroMnemonicConst	(class in bip_utils.monero.mnemonic.monero_mnemonic),	259	Moonbeam	(CoinsConf attribute),	185
MoneroMnemonicDecoder	(class in bip_utils.monero.mnemonic.monero_mnemonic_decoder),	260	MOONBEAM	(SubstrateCoins attribute),	281
MoneroMnemonicEncoder	(class in bip_utils.monero.mnemonic.monero_mnemonic_encoder),	261	Moonbeam	(SubstrateConf attribute),	282
MoneroMnemonicEncoderBase	(class in bip_utils.monero.mnemonic.monero_mnemonic_encoder_base),	261	Moonriver	(CoinsConf attribute),	185
MoneroMnemonicGenerator	(class in bip_utils.monero.mnemonic.monero_mnemonic_generator),	262	MOONRIVER	(SubstrateCoins attribute),	281
MoneroMnemonicGeneratorConst	(class in bip_utils.monero.mnemonic.monero_mnemonic_generator),	262	Moonriver	(SubstrateConf attribute),	282
MoneroMnemonicNoChecksumEncoder	(class in bip_utils.monero.mnemonic.monero_mnemonic_encoder),	261	MultiplyScalarNoCarry()	(BytesUtils static method),	
MoneroMnemonicUtils	(class in bip_utils.monero.mnemonic.monero_mnemonic_utils),	264	MULTIVERSX	(Bip44Coins attribute),	145
MoneroMnemonicValidator	(class in bip_utils.monero.mnemonic.monero_mnemonic_validator),		N		
			NAME	(Ed25519Blake2bConst attribute),	208
			NAME	(Ed25519Const attribute),	198
			NAME	(Ed25519KholawConst attribute),	212
			NAME	(Ed25519MoneroConst attribute),	215
			NAME	(Nist256p1Const attribute),	217
			NAME	(Secp256k1Const attribute),	223
			NAME	(Slip25519Const attribute),	234
			Name()	(CoinNames method),	297
			Name()	(EllipticCurve method),	195
			NANO	(Bip44Coins attribute),	145
			Nano	(Bip44Conf attribute),	149
			Nano	(CoinsConf attribute),	185
			NANO	(Slip44 attribute),	276
			NanoAddr	(in module bip_utils.addr.nano_addr),	50
			NanoAddrConst	(class in bip_utils.addr.nano_addr),	49

- NanoAddrDecoder (class in bip_utils.addr.nano_addr), 49
- NanoAddrEncoder (class in bip_utils.addr.nano_addr), 49
- NEAR_PROTOCOL (Bip44Coins attribute), 145
- NEAR_PROTOCOL (Slip44 attribute), 277
- NearAddr (in module bip_utils.addr.near_addr), 51
- NearAddrDecoder (class in bip_utils.addr.near_addr), 50
- NearAddrEncoder (class in bip_utils.addr.near_addr), 50
- NearProtocol (Bip44Conf attribute), 149
- NearProtocol (CoinsConf attribute), 185
- NEO (Bip44Coins attribute), 145
- Neo (Bip44Conf attribute), 149
- Neo (CoinsConf attribute), 185
- NEO (Slip44 attribute), 277
- NeoAddr (in module bip_utils.addr.neo_addr), 52
- NeoAddrConst (class in bip_utils.addr.neo_addr), 51
- NeoAddrDecoder (class in bip_utils.addr.neo_addr), 51
- NeoAddrEncoder (class in bip_utils.addr.neo_addr), 51
- NETWORK_TAG_TO_ADDR_HRP (AdaShelleyAddrConst attribute), 30
- NETWORK_TAG_TO_REWARD_ADDR_HRP (AdaShelleyAddrConst attribute), 30
- NINE_CHRONICLES (Slip44 attribute), 277
- NINE_CHRONICLES_GOLD (Bip44Coins attribute), 145
- NineChroniclesGold (Bip44Conf attribute), 149
- NineChroniclesGold (CoinsConf attribute), 185
- NIST256P1 (EllipticCurveTypes attribute), 197
- Nist256p1Const (class in bip_utils.ecc.nist256p1.nist256p1_const), 217
- Nist256p1Point (class in bip_utils.ecc.nist256p1.nist256p1_point), 220
- Nist256p1PrivateKey (class in bip_utils.ecc.nist256p1.nist256p1_keys), 219
- Nist256p1PublicKey (class in bip_utils.ecc.nist256p1.nist256p1_keys), 218
- NormalizeNfc() (StringUtils static method), 320
- NormalizeNfkd() (StringUtils static method), 320
- ## O
- OKEX_CHAIN (Slip173 attribute), 273
- OKEX_CHAIN (Slip44 attribute), 277
- OKEX_CHAIN_ATOM (Bip44Coins attribute), 145
- OKEX_CHAIN_ATOM_OLD (Bip44Coins attribute), 145
- OKEX_CHAIN_ETH (Bip44Coins attribute), 145
- OkexAddr (in module bip_utils.addr.okex_addr), 53
- OkexAddrDecoder (class in bip_utils.addr.okex_addr), 52
- OkexAddrEncoder (class in bip_utils.addr.okex_addr), 52
- OkexChain (CoinsConf attribute), 185
- OkexChainAtom (Bip44Conf attribute), 149
- OkexChainAtomOld (Bip44Conf attribute), 149
- OkexChainEth (Bip44Conf attribute), 149
- OneAddr (in module bip_utils.addr.one_addr), 54
- OneAddrDecoder (class in bip_utils.addr.one_addr), 53
- OneAddrEncoder (class in bip_utils.addr.one_addr), 53
- ONTOLOGY (Bip44Coins attribute), 145
- Ontology (Bip44Conf attribute), 149
- Ontology (CoinsConf attribute), 185
- ONTOLOGY (Slip44 attribute), 277
- OPTIMISM (Bip44Coins attribute), 145
- Optimism (Bip44Conf attribute), 149
- Optimism (CoinsConf attribute), 185
- Order() (EllipticCurve method), 195
- OSMOSIS (Bip44Coins attribute), 145
- Osmosis (Bip44Conf attribute), 149
- Osmosis (CoinsConf attribute), 185
- OSMOSIS (Slip173 attribute), 273
- OWNER_SALT_NO_LOT_SEQ_BYTE_LEN (Bip38EcConst attribute), 112
- OWNER_SALT_WITH_LOT_SEQ_BYTE_LEN (Bip38EcConst attribute), 112
- ## P
- P2PKH (ErgoAddressTypes attribute), 43
- P2PKHAddr (in module bip_utils.addr.P2PKH_addr), 23
- P2PKHAddrDecoder (class in bip_utils.addr.P2PKH_addr), 21
- P2PKHAddrEncoder (class in bip_utils.addr.P2PKH_addr), 21
- P2PKHPublicKeyModes (class in bip_utils.addr.P2PKH_addr), 21
- P2SH (ErgoAddressTypes attribute), 43
- P2SHAddr (in module bip_utils.addr.P2SH_addr), 25
- P2SHAddrConst (class in bip_utils.addr.P2SH_addr), 23
- P2SHAddrDecoder (class in bip_utils.addr.P2SH_addr), 23
- P2SHAddrEncoder (class in bip_utils.addr.P2SH_addr), 23
- P2TRAddr (in module bip_utils.addr.P2TR_addr), 26
- P2TRAddrDecoder (class in bip_utils.addr.P2TR_addr), 25
- P2TRAddrEncoder (class in bip_utils.addr.P2TR_addr), 26
- P2TRConst (class in bip_utils.addr.P2TR_addr), 25
- P2WPKHAddr (in module bip_utils.addr.P2WPKH_addr), 27
- P2WPKHAddrConst (class in bip_utils.addr.P2WPKH_addr), 26
- P2WPKHAddrDecoder (class in bip_utils.addr.P2WPKH_addr), 26

P2WPKHAddrEncoder (class *bip_utils.addr.P2WPKH_addr*), 27
Pad() (*AesEcbEncrypter* static method), 297
PADDING_CHAR (*Base32Const* attribute), 310
ParamByKey() (*CoinConf* method), 182
ParentFingerPrint() (*Bip32Base* method), 84
ParentFingerPrint() (*Bip32KeyData* method), 93
Parse() (*Bip32PathParser* static method), 101
Parse() (*SubstratePathParser* static method), 296
Path() (*Slip32DeserializedKey* method), 274
Path() (*Substrate* method), 290
PAYLOAD_PAD_DEC (*NanoAddrConst* attribute), 49
PAYLOAD_PAD_ENC (*NanoAddrConst* attribute), 49
PAYLOAD_TAG (*AdaByronAddrConst* attribute), 28
PAYMENT (*AdaShelleyAddrHeaderTypes* attribute), 30
PAYMENT_ID_BYTE_LEN (*XmrAddrConst* attribute), 60
PBKDF2_HMAC_SHA512 (*BrainwalletAlgos* attribute), 164
PBKDF2_HMAC_SHA512_DEF_ITR_NUM (*BrainwalletAlgoConst* attribute), 164
PBKDF2_HMAC_SHA512_KEY_LEN (*BrainwalletAlgoConst* attribute), 164
PBKDF2_OUT_BYTE_LEN (*CardanoIcarusMasterKeyGeneratorConst* attribute), 169
PBKDF2_PASSWORD (*CardanoIcarusMasterKeyGeneratorConst* attribute), 169
PBKDF2_ROUNDS (*CardanoIcarusMasterKeyGeneratorConst* attribute), 169
Pbkdf2HmacSha512 (class *bip_utils.utils.crypto.pbkdf2*), 304
PDA_MARKER (*SplTokenConst* attribute), 278
Phala (*CoinsConf* attribute), 185
PHALA (*SubstrateCoins* attribute), 282
Phala (*SubstrateConf* attribute), 282
PI_NETWORK (*Bip44Coins* attribute), 145
PI_NETWORK (*Slip44* attribute), 277
PiNetwork (*Bip44Conf* attribute), 149
PiNetwork (*CoinsConf* attribute), 185
Plasm (*CoinsConf* attribute), 185
PLASM (*SubstrateCoins* attribute), 282
Plasm (*SubstrateConf* attribute), 282
Point() (*Bip32PublicKey* method), 97
Point() (*Ed25519Blake2bPublicKey* method), 210
Point() (*Ed25519KholawPublicKey* method), 213
Point() (*Ed25519MoneroPublicKey* method), 216
Point() (*Ed25519PublicKey* method), 199
Point() (*IPublicKey* method), 190
Point() (*Nist256p1PublicKey* method), 219
Point() (*Secp256k1PublicKeyCoincurve* method), 225
Point() (*Secp256k1PublicKeyEcdsa* method), 228
Point() (*Sr25519PublicKey* method), 236
point_add() (in module *bip_utils.ecc.ed25519.lib.ed25519_lib*), 207
point_bytes_to_coord() (in module *bip_utils.ecc.ed25519.lib.ed25519_lib*), 205
POINT_COORD_BYTE_LEN (*DummyPointConst* attribute), 186
POINT_COORD_BYTE_LEN (*EcdsaKeysConst* attribute), 197
POINT_COORD_BYTE_LEN (*Ed25519PointConst* attribute), 201
point_coord_to_bytes() (in module *bip_utils.ecc.ed25519.lib.ed25519_lib*), 206
point_decode() (in module *bip_utils.ecc.ed25519.lib.ed25519_lib*), 206
point_decode_no_check() (in module *bip_utils.ecc.ed25519.lib.ed25519_lib*), 206
point_encode() (in module *bip_utils.ecc.ed25519.lib.ed25519_lib*), 206
point_is_decoded_bytes() (in module *bip_utils.ecc.ed25519.lib.ed25519_lib*), 205
point_is_encoded_bytes() (in module *bip_utils.ecc.ed25519.lib.ed25519_lib*), 205
point_is_generator() (in module *bip_utils.ecc.ed25519.lib.ed25519_lib*), 206
point_is_on_curve() (in module *bip_utils.ecc.ed25519.lib.ed25519_lib*), 207
point_is_valid_bytes() (in module *bip_utils.ecc.ed25519.lib.ed25519_lib*), 205
point_scalar_mul() (in module *bip_utils.ecc.ed25519.lib.ed25519_lib*), 207
point_scalar_mul_base() (in module *bip_utils.ecc.ed25519.lib.ed25519_lib*), 207
PointClass() (*EllipticCurve* method), 195
Polkadot (*CoinsConf* attribute), 185
POLKADOT (*Slip44* attribute), 277
POLKADOT (*SubstrateCoins* attribute), 282
Polkadot (*SubstrateConf* attribute), 282
POLKADOT_ED25519_SLIP (*Bip44Coins* attribute), 145
PolkadotEd25519Slip (*Bip44Conf* attribute), 149
POLYGON (*Bip44Coins* attribute), 145
Polygon (*Bip44Conf* attribute), 149
Polygon (*CoinsConf* attribute), 185
PolyMod() (*BchBech32Utils* static method), 73
PolyMod() (*Bech32Utils* static method), 75
PORTUGUESE (*Bip39Languages* attribute), 117
PORTUGUESE (*ElectrumV2Languages* attribute), 251
PORTUGUESE (*MoneroLanguages* attribute), 259
PREFIX_BYTE (*NeoAddrConst* attribute), 51
PrimaryAddress() (*Monero* method), 268
PRIV_KEY (*XlmAddrTypes* attribute), 58
PRIV_KEY_BYTE_LEN (*EcdsaKeysConst* attribute), 197
PRIV_KEY_BYTE_LEN (*Ed25519KeysConst* attribute), 198
PRIV_KEY_BYTE_LEN (*Ed25519KholawKeysConst* attribute), 212
PRIV_KEY_BYTE_LEN (*Sr25519KeysConst* attribute), 235
PRIV_KEY_PREFIX (*Bip32Slip10DerivatorConst* attribute), 106

Private() (*Bip32KeyNetVersions method*), 93
 Private() (*Slip32KeyNetVersions method*), 276
 PrivateKey() (*Bip32Base method*), 83
 PrivateKey() (*Bip44Base method*), 126
 PrivateKey() (*Brainwallet method*), 164
 PrivateKey() (*Substrate method*), 290
 PrivateKeyClass() (*EllipticCurve method*), 195
 PrivateKeys() (*CardanoShelley method*), 179
 PrivateSpendKey() (*Monero method*), 267
 PrivateViewKey() (*Monero method*), 267
 PUB_KEY (*XlmAddrTypes attribute*), 58
 PUB_KEY_BYTE_LEN (*Ed25519KeysConst attribute*), 198
 PUB_KEY_BYTE_LEN (*Sr25519KeysConst attribute*), 235
 PUB_KEY_COMPRESSED_BYTE_LEN (*EcdsaKeysConst attribute*), 197
 PUB_KEY_PREFIX (*Ed25519KeysConst attribute*), 198
 PUB_KEY_UNCOMPRESSED_BYTE_LEN (*EcdsaKeysConst attribute*), 197
 PUB_KEY_UNCOMPRESSED_PREFIX (*EcdsaKeysConst attribute*), 197
 Public() (*Bip32KeyNetVersions method*), 93
 Public() (*Slip32KeyNetVersions method*), 275
 PUBLIC_KEY (*AdaByronAddrTypes attribute*), 27
 PublicKey() (*Bip32Base method*), 83
 PublicKey() (*Bip32PrivateKey method*), 99
 PublicKey() (*Bip44Base method*), 126
 PublicKey() (*Bip44PrivateKey method*), 132
 PublicKey() (*Brainwallet method*), 164
 PublicKey() (*Ed25519Blake2bPrivateKey method*), 211
 PublicKey() (*Ed25519KholawPrivateKey method*), 214
 PublicKey() (*Ed25519MoneroPrivateKey method*), 216
 PublicKey() (*Ed25519PrivateKey method*), 201
 PublicKey() (*IPrivateKey method*), 191
 PublicKey() (*MoneroPrivateKey method*), 270
 PublicKey() (*Nist256p1PrivateKey method*), 220
 PublicKey() (*Secp256k1PrivateKeyCoincurve method*), 226
 PublicKey() (*Secp256k1PrivateKeyEcdsa method*), 229
 PublicKey() (*Sr25519PrivateKey method*), 237
 PublicKey() (*Substrate method*), 290
 PublicKey() (*SubstratePrivateKey method*), 293
 PublicKeyClass() (*EllipticCurve method*), 195
 PublicKeys() (*CardanoShelley method*), 178
 PublicKeys() (*CardanoShelleyPrivateKeys method*), 182
 PublicSpendKey() (*Monero method*), 267
 PublicViewKey() (*Monero method*), 267
 PURPOSE (*Bip44Const attribute*), 122
 PURPOSE (*Bip44Levels attribute*), 125
 PURPOSE (*Bip49Const attribute*), 133
 PURPOSE (*Bip84Const attribute*), 136
 PURPOSE (*Bip86Const attribute*), 140
 PURPOSE (*Cip1852Const attribute*), 172

Purpose() (*Bip44 method*), 123
 Purpose() (*Bip44Base method*), 129
 Purpose() (*Bip49 method*), 134
 Purpose() (*Bip84 method*), 138
 Purpose() (*Bip86 method*), 141
 Purpose() (*Cip1852 method*), 174

Q

QuickDigest() (*Blake2b static method*), 298
 QuickDigest() (*Crc32 static method*), 301
 QuickDigest() (*DoubleSha256 static method*), 306
 QuickDigest() (*Hash160 static method*), 302
 QuickDigest() (*HmacSha256 static method*), 303
 QuickDigest() (*HmacSha512 static method*), 303
 QuickDigest() (*Kekkek256 static method*), 308
 QuickDigest() (*Ripemd160 static method*), 305
 QuickDigest() (*Sha256 static method*), 306
 QuickDigest() (*Sha3_256 static method*), 308
 QuickDigest() (*Sha512 static method*), 307
 QuickDigest() (*Sha512_256 static method*), 307
 QuickDigest() (*XModemCrc static method*), 302
 QuickDigestHalves() (*HmacSha512 static method*), 304
 QuickIntDigest() (*Crc32 static method*), 301

R

RADIX (*Base58Const attribute*), 70
 Raw() (*Bip32PrivateKey method*), 99
 Raw() (*Bip44PrivateKey method*), 132
 Raw() (*DummyPoint method*), 187
 Raw() (*Ed25519Blake2bPrivateKey method*), 211
 Raw() (*Ed25519KholawPrivateKey method*), 214
 Raw() (*Ed25519Point method*), 202
 Raw() (*Ed25519PrivateKey method*), 200
 Raw() (*IPoint method*), 193
 Raw() (*IPrivateKey method*), 191
 Raw() (*MoneroPrivateKey method*), 270
 Raw() (*Nist256p1Point method*), 222
 Raw() (*Nist256p1PrivateKey method*), 220
 Raw() (*Secp256k1PointCoincurve method*), 230
 Raw() (*Secp256k1PointEcdsa method*), 233
 Raw() (*Secp256k1PrivateKeyCoincurve method*), 226
 Raw() (*Secp256k1PrivateKeyEcdsa method*), 228
 Raw() (*Sr25519PrivateKey method*), 237
 Raw() (*SubstratePrivateKey method*), 293
 RawCompressed() (*Bip32PublicKey method*), 97
 RawCompressed() (*Bip44PublicKey method*), 131
 RawCompressed() (*Ed25519Blake2bPublicKey method*), 210
 RawCompressed() (*Ed25519MoneroPublicKey method*), 215
 RawCompressed() (*Ed25519PublicKey method*), 199
 RawCompressed() (*IPublicKey method*), 190
 RawCompressed() (*MoneroPublicKey method*), 269

RawCompressed() (*Nist256p1PublicKey method*), 219
 RawCompressed() (*Secp256k1PublicKeyCoincurve method*), 224
 RawCompressed() (*Secp256k1PublicKeyEcdsa method*), 227
 RawCompressed() (*Sr25519PublicKey method*), 236
 RawCompressed() (*SubstratePublicKey method*), 292
 RawDecoded() (*DummyPoint method*), 187
 RawDecoded() (*Ed25519Point method*), 203
 RawDecoded() (*IPoint method*), 193
 RawDecoded() (*Nist256p1Point method*), 222
 RawDecoded() (*Secp256k1PointCoincurve method*), 230
 RawDecoded() (*Secp256k1PointEcdsa method*), 233
 RawEncoded() (*DummyPoint method*), 187
 RawEncoded() (*Ed25519Point method*), 202
 RawEncoded() (*IPoint method*), 193
 RawEncoded() (*Nist256p1Point method*), 222
 RawEncoded() (*Secp256k1PointCoincurve method*), 230
 RawEncoded() (*Secp256k1PointEcdsa method*), 233
 RawUncompressed() (*Bip32PublicKey method*), 97
 RawUncompressed() (*Bip44PublicKey method*), 131
 RawUncompressed() (*Ed25519Blake2bPublicKey method*), 210
 RawUncompressed() (*Ed25519PublicKey method*), 199
 RawUncompressed() (*IPublicKey method*), 190
 RawUncompressed() (*MoneroPublicKey method*), 270
 RawUncompressed() (*Nist256p1PublicKey method*), 219
 RawUncompressed() (*Secp256k1PublicKeyCoincurve method*), 225
 RawUncompressed() (*Secp256k1PublicKeyEcdsa method*), 227
 RawUncompressed() (*Sr25519PublicKey method*), 236
 RawUncompressed() (*SubstratePublicKey method*), 292
 RE_PATH (*SubstratePathConst attribute*), 293
 REDEMPTION (*AdaByronAddrTypes attribute*), 27
 RESERVED_FORMATS (*SS58Const attribute*), 279
 ResetBit() (*BitUtils static method*), 312
 ResetBits() (*BitUtils static method*), 312
 ResolveCalls() (*BipCoinFctCallsConf method*), 159
 Reverse() (*BytesUtils static method*), 313
 REWARD (*AdaShelleyAddrHeaderTypes attribute*), 30
 RewardKey() (*CardanoShelleyPrivateKeys method*), 181
 RewardKey() (*CardanoShelleyPublicKeys method*), 180
 RewardObject() (*CardanoShelley method*), 179
 Ripemd160 (*class in bip_utils.utils.crypto.ripemd*), 305
 RIPPLE (*Base58Alphabets attribute*), 70
 RIPPLE (*Bip44Coins attribute*), 145
 Ripple (*Bip44Conf attribute*), 149
 Ripple (*CoinsConf attribute*), 185
 RIPPLE (*Slip44 attribute*), 276
 RUSSIAN (*MoneroLanguages attribute*), 259

S

scalar_is_valid() (*in module bip_utils.ecc.ed25519.lib.ed25519_lib*), 208
 scalar_reduce() (*in module bip_utils.ecc.ed25519.lib.ed25519_lib*), 208
 ScalarReduce() (*Ed25519Utils static method*), 204
 SCALE_INT_ENCODERS (*SubstratePathConst attribute*), 294
 SCRIPT_BYTES (*P2SHAddrConst attribute*), 23
 SCRIPT (*BrainwalletAlgos attribute*), 164
 Scrypt (*class in bip_utils.utils.crypto.scrypt*), 305
 SCRIPT_DEF_N (*BrainwalletAlgoConst attribute*), 164
 SCRIPT_DEF_P (*BrainwalletAlgoConst attribute*), 164
 SCRIPT_DEF_R (*BrainwalletAlgoConst attribute*), 164
 SCRIPT_HALVES_KEY_LEN (*Bip38EcConst attribute*), 113
 SCRIPT_HALVES_N (*Bip38EcConst attribute*), 113
 SCRIPT_HALVES_P (*Bip38EcConst attribute*), 113
 SCRIPT_HALVES_R (*Bip38EcConst attribute*), 113
 SCRIPT_KEY_LEN (*Bip38NoEcConst attribute*), 115
 SCRIPT_KEY_LEN (*BrainwalletAlgoConst attribute*), 164
 SCRIPT_N (*Bip38NoEcConst attribute*), 115
 SCRIPT_P (*Bip38NoEcConst attribute*), 115
 SCRIPT_PREFACTOR_KEY_LEN (*Bip38EcConst attribute*), 113
 SCRIPT_PREFACTOR_N (*Bip38EcConst attribute*), 113
 SCRIPT_PREFACTOR_P (*Bip38EcConst attribute*), 113
 SCRIPT_PREFACTOR_R (*Bip38EcConst attribute*), 113
 SCRIPT_R (*Bip38NoEcConst attribute*), 115
 SECP256K1 (*EllipticCurveTypes attribute*), 197
 SECP256K1 (*FillAddrTypes attribute*), 45
 Secp256k1Const (*class in bip_utils.ecc.secp256k1.secp256k1_const*), 223
 Secp256k1PointCoincurve (*class in bip_utils.ecc.secp256k1.secp256k1_point_coincurve*), 229
 Secp256k1PointEcdsa (*class in bip_utils.ecc.secp256k1.secp256k1_point_ecdsa*), 232
 Secp256k1PrivateKeyCoincurve (*class in bip_utils.ecc.secp256k1.secp256k1_keys_coincurve*), 225
 Secp256k1PrivateKeyEcdsa (*class in bip_utils.ecc.secp256k1.secp256k1_keys_ecdsa*), 228
 Secp256k1PublicKeyCoincurve (*class in bip_utils.ecc.secp256k1.secp256k1_keys_coincurve*), 223
 Secp256k1PublicKeyEcdsa (*class in bip_utils.ecc.secp256k1.secp256k1_keys_ecdsa*), 226
 SECRET_NETWORK (*Slip173 attribute*), 273
 SECRET_NETWORK (*Slip44 attribute*), 277

- SECRET_NETWORK_NEW (*Bip44Coins* attribute), 145
- SECRET_NETWORK_OLD (*Bip44Coins* attribute), 145
- SecretNetwork (*CoinsConf* attribute), 185
- SecretNetworkNew (*Bip44Conf* attribute), 149
- SecretNetworkOld (*Bip44Conf* attribute), 149
- SEED_B_BYTE_LEN (*Bip38EcConst* attribute), 113
- SEED_BUMP_MAX_VAL (*SplTokenConst* attribute), 278
- SEED_BYTE_LEN (*CardanoByronLegacyMstKeyGeneratorConst* attribute), 168
- SEED_MIN_BYTE_LEN (*Bip32KholawMstKeyGeneratorConst* attribute), 104
- SEED_MIN_BYTE_LEN (*Bip32Slip10MstKeyGeneratorConst* attribute), 108
- SEED_MIN_BYTE_LEN (*SubstrateConst* attribute), 288
- SEED_PBKDF2_ROUNDS (*Bip39SeedGeneratorConst* attribute), 121
- SEED_PBKDF2_ROUNDS (*ElectrumV2SeedGeneratorConst* attribute), 254
- SEED_SALT_MOD (*Bip39SeedGeneratorConst* attribute), 121
- SEED_SALT_MOD (*ElectrumV2SeedGeneratorConst* attribute), 254
- SEEDS_MAX_NUM (*SplTokenConst* attribute), 278
- SEGWIT (*ElectrumV2MnemonicTypes* attribute), 251
- SEGWIT_2FA (*ElectrumV2MnemonicTypes* attribute), 251
- SegwitBech32Const (class in *bip_utils.bech32.segwit_bech32*), 79
- SegwitBech32Decoder (class in *bip_utils.bech32.segwit_bech32*), 79
- SegwitBech32Encoder (class in *bip_utils.bech32.segwit_bech32*), 79
- SEPARATOR (*BchBech32Const* attribute), 73
- SEPARATOR (*Bech32Const* attribute), 75
- SEPARATOR (*SegwitBech32Const* attribute), 79
- SEQ_NUM_MAX_VAL (*Bip38EcConst* attribute), 112
- SEQ_NUM_MIN_VAL (*Bip38EcConst* attribute), 112
- Serialize() (*Bip32PrivateKeySerializer* static method), 94
- Serialize() (*Bip32PublicKeySerializer* static method), 94
- Serialize() (*Slip32PrivateKeySerializer* static method), 273
- Serialize() (*Slip32PublicKeySerializer* static method), 274
- SERIALIZED_PRIV_KEY_BYTE_LEN (*Bip32KeySerConst* attribute), 94
- SERIALIZED_PUB_KEY_BYTE_LEN (*Bip32KeySerConst* attribute), 94
- SetBit() (*BitUtils* static method), 311
- SetBits() (*BitUtils* static method), 312
- SHA256 (*BrainwalletAlgos* attribute), 164
- Sha256 (class in *bip_utils.utils.crypto.sha2*), 306
- SHA256_BYTE_LEN (*ZilAddrConst* attribute), 64
- Sha3_256 (class in *bip_utils.utils.crypto.sha3*), 308
- Sha512 (class in *bip_utils.utils.crypto.sha2*), 307
- Sha512_256 (class in *bip_utils.utils.crypto.sha2*), 307
- SIMPLE_ACCOUNT_FORMAT_MAX_VAL (*SS58Const* attribute), 279
- SINGLE_BYTE_MODE_MAX_VAL (*SubstrateScaleCUintEncoderConst* attribute), 285
- SINGLE_SIG_SUFFIX_BYTE (*AptosAddrConst* attribute), 37
- Size() (*DataBytes* method), 317
- Slip173 (class in *bip_utils.slip.slip173.slip173*), 272
- Slip32DeserializedKey (class in *bip_utils.slip.slip32.slip32*), 274
- Slip32KeyDeserializer (class in *bip_utils.slip.slip32.slip32*), 275
- Slip32KeyNetVersions (class in *bip_utils.slip.slip32.slip32_key_net_ver*), 275
- Slip32KeySerConst (class in *bip_utils.slip.slip32.slip32*), 273
- Slip32PrivateKeySerializer (class in *bip_utils.slip.slip32.slip32*), 273
- Slip32PublicKeySerializer (class in *bip_utils.slip.slip32.slip32*), 273
- Slip44 (class in *bip_utils.slip.slip44.slip44*), 276
- SOFT_PATH_PREFIX (*SubstratePathConst* attribute), 293
- SolAddr (in module *bip_utils.addr.sol_addr*), 55
- SolAddrDecoder (class in *bip_utils.addr.sol_addr*), 54
- SolAddrEncoder (class in *bip_utils.addr.sol_addr*), 54
- SOLANA (*Bip44Coins* attribute), 145
- Solana (*Bip44Conf* attribute), 150
- Solana (*CoinsConf* attribute), 185
- SOLANA (*Slip44* attribute), 277
- Sora (*CoinsConf* attribute), 185
- SORA (*SubstrateCoins* attribute), 282
- Sora (*SubstrateConf* attribute), 282
- SPANISH (*Bip39Languages* attribute), 117
- SPANISH (*ElectrumV2Languages* attribute), 251
- SPANISH (*MoneroLanguages* attribute), 259
- SPEC_NAME (*Bip44Const* attribute), 122
- SPEC_NAME (*Bip49Const* attribute), 133
- SPEC_NAME (*Bip84Const* attribute), 136
- SPEC_NAME (*Bip86Const* attribute), 140
- SPEC_NAME (*Cip1852Const* attribute), 172
- SpecName() (*Bip44* static method), 125
- SpecName() (*Bip44Base* static method), 130
- SpecName() (*Bip49* static method), 136
- SpecName() (*Bip84* static method), 139
- SpecName() (*Bip86* static method), 143
- SpecName() (*Cip1852* static method), 175
- SplitDecodedBytes() (*AdaByronAddrDecoder* static method), 28
- SplitPartsByChecksum() (*AddrDecUtils* static method), 33
- SplToken (class in *bip_utils.solana.spl_token*), 278

SplTokenConst (class in bip_utils.solana.spl_token), 278

Sr25519 (EllipticCurveTypes attribute), 197

Sr25519Const (class in bip_utils.ecc.sr25519.sr25519_const), 234

Sr25519KeysConst (class in bip_utils.ecc.sr25519.sr25519_keys), 235

Sr25519Point (class in bip_utils.ecc.sr25519.sr25519_point), 238

Sr25519PrivateKey (class in bip_utils.ecc.sr25519.sr25519_keys), 236

Sr25519PublicKey (class in bip_utils.ecc.sr25519.sr25519_keys), 235

SS58ChecksumError, 280

SS58Const (class in bip_utils.ss58.ss58), 279

SS58Decoder (class in bip_utils.ss58.ss58), 280

SS58Encoder (class in bip_utils.ss58.ss58), 279

SS58Format() (SubstrateCoinConf method), 281

STAFI (Bip44Coins attribute), 145

Stafi (Bip44Conf attribute), 150

Stafi (CoinsConf attribute), 185

STAFI (Slip173 attribute), 273

STAFI (SubstrateCoins attribute), 282

Stafi (SubstrateConf attribute), 283

StageNet (MoneroConf attribute), 257

StakingKey() (CardanoShelleyPrivateKeys method), 182

StakingKey() (CardanoShelleyPublicKeys method), 180

StakingObject() (CardanoShelley method), 179

STANDARD (ElectrumV2MnemonicTypes attribute), 251

STANDARD_2FA (ElectrumV2MnemonicTypes attribute), 251

START_BYTE (EthAddrConst attribute), 44

STD_KEY_NET_VERSIONS (Slip32KeySerConst attribute), 273

STELLAR (Bip44Coins attribute), 145

Stellar (Bip44Conf attribute), 150

Stellar (CoinsConf attribute), 185

STELLAR (Slip44 attribute), 276

StringUtils (class in bip_utils.utils.misc.string), 320

SUBADDR_IDX_BYTE_LEN (MoneroSubaddressConst attribute), 271

SUBADDR_MAX_IDX (MoneroSubaddressConst attribute), 271

SUBADDR_PREFIX (MoneroSubaddressConst attribute), 271

Subaddress() (Monero method), 268

SubaddrNetVersion() (MoneroCoinConf method), 256

Substrate (class in bip_utils.substrate.substrate), 288

SubstrateBip39SeedGenerator (class in bip_utils.substrate.mnemonic.substrate_bip39_seed_generator), 284

SubstrateCoinConf (class in bip_utils.substrate.conf.substrate_coin_conf), 280

SubstrateCoins (class in bip_utils.substrate.conf.substrate_coins), 281

SubstrateConf (class in bip_utils.substrate.conf.substrate_conf), 282

SubstrateConfGetter (class in bip_utils.substrate.conf.substrate_conf_getter), 283

SubstrateConfGetterConst (class in bip_utils.substrate.conf.substrate_conf_getter), 283

SubstrateConst (class in bip_utils.substrate.substrate), 288

SubstrateEd25519Addr (in module bip_utils.addr.substrate_addr), 56

SubstrateEd25519AddrDecoder (class in bip_utils.addr.substrate_addr), 55

SubstrateEd25519AddrEncoder (class in bip_utils.addr.substrate_addr), 55

SubstrateKeyError, 291

SubstratePath (class in bip_utils.substrate.substrate_path), 295

SubstratePathConst (class in bip_utils.substrate.substrate_path), 293

SubstratePathElem (class in bip_utils.substrate.substrate_path), 294

SubstratePathError, 291

SubstratePathParser (class in bip_utils.substrate.substrate_path), 296

SubstratePrivateKey (class in bip_utils.substrate.substrate_keys), 292

SubstratePublicKey (class in bip_utils.substrate.substrate_keys), 291

SubstrateScaleBytesEncoder (class in bip_utils.substrate.scale.substrate_scale_enc_bytes), 285

SubstrateScaleCUintEncoder (class in bip_utils.substrate.scale.substrate_scale_enc_cuint), 285

SubstrateScaleCUintEncoderConst (class in bip_utils.substrate.scale.substrate_scale_enc_cuint), 285

SubstrateScaleEncoderBase (class in bip_utils.substrate.scale.substrate_scale_enc_base), 284

SubstrateScaleU128Encoder (class in bip_utils.substrate.scale.substrate_scale_enc_uint), 287

SubstrateScaleU16Encoder (class in bip_utils.substrate.scale.substrate_scale_enc_uint), 286

SubstrateScaleU256Encoder (class in ToAddress() (CardanoShelleyPublicKeys method), 181
 bip_utils.substrate.scale.substrate_scale_enc_uintToAddress() (SubstratePublicKey method), 292
 287 ToBinaryStr() (BytesUtils static method), 314
 SubstrateScaleU32Encoder (class in ToBinaryStr() (IntegerUtils static method), 319
 bip_utils.substrate.scale.substrate_scale_enc_uintToBytes() (Bip32Depth method), 88
 286 ToBytes() (Bip32KeyIndex method), 91
 SubstrateScaleU64Encoder (class in ToBytes() (DataBytes method), 317
 bip_utils.substrate.scale.substrate_scale_enc_uintToBytes() (IntegerUtils static method), 319
 286 ToExtended() (Bip32PrivateKey method), 99
 SubstrateScaleU8Encoder (class in ToExtended() (Bip32PublicKey method), 98
 bip_utils.substrate.scale.substrate_scale_enc_uintToExtended() (Bip44PrivateKey method), 132
 286 ToExtended() (Bip44PublicKey method), 131
 SubstrateScaleUIntEncoder (class in ToHex() (DataBytes method), 317
 bip_utils.substrate.scale.substrate_scale_enc_uintToHexString() (BytesUtils static method), 314
 286 ToInt() (Bip32Depth method), 88
 SubstrateSr25519Addr (in module ToInt() (Bip32KeyIndex method), 91
 bip_utils.addr.substrate_addr), 56 ToInt() (DataBytes method), 317
 SubstrateSr25519AddrDecoder (class in ToInteger() (BytesUtils static method), 314
 bip_utils.addr.substrate_addr), 55 ToList() (Bip32Path method), 100
 SubstrateSr25519AddrEncoder (class in ToList() (BytesUtils static method), 315
 bip_utils.addr.substrate_addr), 56 ToList() (Mnemonic method), 321
 SUFFIX_BYTE (NeoAddrConst attribute), 51 ToList() (SubstratePath method), 295
 SUI (Bip44Coins attribute), 145 ToRewardAddress() (CardanoShelleyPublicKeys
 Sui (Bip44Conf attribute), 150 method), 181
 Sui (CoinsConf attribute), 185 ToStakingAddress() (CardanoShelleyPublicKeys
 SUI (Slip44 attribute), 277 method), 181
 SuiAddr (in module bip_utils.addr.sui_addr), 57 ToStr() (Bip32Path method), 100
 SuiAddrConst (class in bip_utils.addr.sui_addr), 56 ToStr() (Mnemonic method), 321
 SuiAddrDecoder (class in bip_utils.addr.sui_addr), 56 ToStr() (SubstratePath method), 295
 SuiAddrEncoder (class in bip_utils.addr.sui_addr), 57 ToStr() (SubstratePathElem method), 294
 ToWif() (Bip44PrivateKey method), 132
 TRON (Bip44Coins attribute), 145
 Tron (Bip44Conf attribute), 150
 Tron (CoinsConf attribute), 185
 TRON (Slip44 attribute), 277
 TrxAddr (in module bip_utils.addr.trx_addr), 58
 TrxAddrDecoder (class in bip_utils.addr.trx_addr), 57
 TrxAddrEncoder (class in bip_utils.addr.trx_addr), 58
 TWO_BYTE_MODE_MAX_VAL (SubstrateScaleCUintEn-
 coderConst attribute), 285
 TYPE_TO_INSTANCE (EllipticCurveGetterConst at-
 tribute), 196
 TYPE_TO_PREFIX (ElectrumV2MnemonicConst at-
 tribute), 251
 TZ1 (XtzAddrPrefixes attribute), 63
 TZ2 (XtzAddrPrefixes attribute), 63
 TZ3 (XtzAddrPrefixes attribute), 63
 U
 UINT16 (CborIds attribute), 315
 UINT32 (CborIds attribute), 315
 UINT64 (CborIds attribute), 315
 UINT8 (CborIds attribute), 315

T
 TagSize() (ChaCha20Poly1305 static method), 301
 TAP_TWEAK_SHA256 (P2TRConst attribute), 25
 TERRA (Bip44Coins attribute), 145
 Terra (Bip44Conf attribute), 150
 Terra (CoinsConf attribute), 185
 TERRA (Slip173 attribute), 273
 TERRA (Slip44 attribute), 277
 TEST_NET_KEY_NET_VERSIONS (Bip32Const attribute),
 86
 TESTNET (AdaShelleyAddrNetworkTags attribute), 30
 TESTNET (ErgoNetworkTypes attribute), 43
 TestNet (MoneroConf attribute), 257
 TESTNET (Slip44 attribute), 276
 TEZOS (Bip44Coins attribute), 145
 Tezos (Bip44Conf attribute), 150
 Tezos (CoinsConf attribute), 185
 TEZOS (Slip44 attribute), 277
 THETA (Bip44Coins attribute), 145
 Theta (Bip44Conf attribute), 150
 Theta (CoinsConf attribute), 185
 THETA (Slip44 attribute), 277
 ToAddress() (Bip44PublicKey method), 131

- UINT_IDS_TO_BYTE_LEN (*CborIndefiniteLenArrayConst attribute*), 316
- UNCOMPRESSED (*P2PKHPubKeyModes attribute*), 21
- UncompressedLength() (*Ed25519Blake2bPublicKey static method*), 209
- UncompressedLength() (*Ed25519MoneroPublicKey static method*), 215
- UncompressedLength() (*Ed25519PublicKey static method*), 199
- UncompressedLength() (*IPublicKey static method*), 190
- UncompressedLength() (*Nist256p1PublicKey static method*), 218
- UncompressedLength() (*Secp256k1PublicKeyCoincurve static method*), 224
- UncompressedLength() (*Secp256k1PublicKeyEcdsa static method*), 227
- UncompressedLength() (*Sr25519PublicKey static method*), 236
- UnderlyingObject() (*DummyPoint method*), 187
- UnderlyingObject() (*Ed25519Blake2bPrivateKey method*), 211
- UnderlyingObject() (*Ed25519Blake2bPublicKey method*), 210
- UnderlyingObject() (*Ed25519KholawPrivateKey method*), 213
- UnderlyingObject() (*Ed25519Point method*), 202
- UnderlyingObject() (*Ed25519PrivateKey method*), 200
- UnderlyingObject() (*Ed25519PublicKey method*), 199
- UnderlyingObject() (*IPoint method*), 192
- UnderlyingObject() (*IPrivateKey method*), 191
- UnderlyingObject() (*IPublicKey method*), 190
- UnderlyingObject() (*Nist256p1Point method*), 221
- UnderlyingObject() (*Nist256p1PrivateKey method*), 220
- UnderlyingObject() (*Nist256p1PublicKey method*), 219
- UnderlyingObject() (*Secp256k1PointCoincurve method*), 230
- UnderlyingObject() (*Secp256k1PointEcdsa method*), 232
- UnderlyingObject() (*Secp256k1PrivateKeyCoincurve method*), 226
- UnderlyingObject() (*Secp256k1PrivateKeyEcdsa method*), 228
- UnderlyingObject() (*Secp256k1PublicKeyCoincurve method*), 224
- UnderlyingObject() (*Secp256k1PublicKeyEcdsa method*), 227
- UnderlyingObject() (*Sr25519PrivateKey method*), 237
- UnderlyingObject() (*Sr25519PublicKey method*), 236
- Unharden() (*Bip32KeyIndex method*), 91
- UnhardenIndex() (*Bip32KeyIndex static method*), 90
- UnhardenIndex() (*Bip32Utils static method*), 102
- UnPad() (*AesEcbDecrypter static method*), 298
- Update() (*Sha256 method*), 306
- USE_COINCURVE (*EccConf attribute*), 194
- UseAlternateKeyNetVersions() (*BipLitecoinConf method*), 162
- UseDeprecatedAddress() (*BipLitecoinConf method*), 162
- UseLegacyAddress() (*BipBitcoinCashConf method*), 159
- ## V
- Validate() (*MnemonicValidator method*), 326
- ValidateAndGetEd25519Blake2bKey() (*AddrKeyValidator static method*), 34
- ValidateAndGetEd25519Key() (*AddrKeyValidator static method*), 34
- ValidateAndGetEd25519MoneroKey() (*AddrKeyValidator static method*), 34
- ValidateAndGetNist256p1Key() (*AddrKeyValidator static method*), 34
- ValidateAndGetSecp256k1Key() (*AddrKeyValidator static method*), 35
- ValidateAndGetSr25519Key() (*AddrKeyValidator static method*), 35
- ValidateAndRemovePrefix() (*AddrDecUtils static method*), 32
- ValidateChecksum() (*AddrDecUtils static method*), 33
- ValidateLength() (*AddrDecUtils static method*), 33
- ValidatePubKey() (*AddrDecUtils static method*), 33
- VECHAIN (*Bip44Coins attribute*), 145
- VeChain (*Bip44Conf attribute*), 150
- VeChain (*CoinsConf attribute*), 185
- VECHAIN (*Slip44 attribute*), 277
- VERGE (*Bip44Coins attribute*), 145
- Verge (*Bip44Conf attribute*), 150
- Verge (*CoinsConf attribute*), 186
- VERGE (*Slip44 attribute*), 276
- VerifyChecksum() (*BchBech32Utils static method*), 74
- VerifyChecksum() (*Bech32Utils static method*), 76
- ## W
- WifConst (*class in bip_utils.wif.wif*), 327
- WifDecoder (*class in bip_utils.wif.wif*), 327
- WifEncoder (*class in bip_utils.wif.wif*), 327
- WifNetVersion() (*BipCoinConf method*), 161
- WITNESS_PROG_MAX_BYTE_LEN (*SegwitBech32Const attribute*), 79
- WITNESS_PROG_MIN_BYTE_LEN (*SegwitBech32Const attribute*), 79
- WITNESS_VER (*P2TRConst attribute*), 25

- WITNESS_VER (*P2WPKHAddrConst* attribute), 26
- WITNESS_VER_BECH32 (*SegwitBech32Const* attribute), 79
- WITNESS_VER_MAX_VAL (*SegwitBech32Const* attribute), 79
- WITNESS_VER_ZERO_DATA_BYTE_LEN (*SegwitBech32Const* attribute), 79
- WORD_BIT_LEN (*Bip39MnemonicConst* attribute), 118
- WORD_BIT_LEN (*ElectrumV2MnemonicConst* attribute), 251
- WORDS_LIST_NUM (*Bip39MnemonicConst* attribute), 117
- WORDS_LIST_NUM (*ElectrumV1MnemonicConst* attribute), 246
- WORDS_LIST_NUM (*MoneroMnemonicConst* attribute), 260
- WORDS_NUM_12 (*Bip39WordsNum* attribute), 117
- WORDS_NUM_12 (*ElectrumV1WordsNum* attribute), 245
- WORDS_NUM_12 (*ElectrumV2WordsNum* attribute), 251
- WORDS_NUM_12 (*MoneroWordsNum* attribute), 259
- WORDS_NUM_13 (*MoneroWordsNum* attribute), 259
- WORDS_NUM_15 (*Bip39WordsNum* attribute), 117
- WORDS_NUM_18 (*Bip39WordsNum* attribute), 117
- WORDS_NUM_21 (*Bip39WordsNum* attribute), 117
- WORDS_NUM_24 (*Bip39WordsNum* attribute), 117
- WORDS_NUM_24 (*ElectrumV2WordsNum* attribute), 251
- WORDS_NUM_24 (*MoneroWordsNum* attribute), 259
- WORDS_NUM_25 (*AlgorandWordsNum* attribute), 66
- WORDS_NUM_25 (*MoneroWordsNum* attribute), 259
- WORDS_NUM_TO_ENTROPY_LEN (*AlgorandMnemonicGeneratorConst* attribute), 67
- WORDS_NUM_TO_ENTROPY_LEN (*ElectrumV1MnemonicGeneratorConst* attribute), 247
- WORDS_NUM_TO_ENTROPY_LEN (*ElectrumV2MnemonicGeneratorConst* attribute), 253
- WORDS_NUM_TO_ENTROPY_LEN (*MoneroMnemonicGeneratorConst* attribute), 262
- WordsCount() (*Mnemonic* method), 321
- WordsToBytesChunk() (*MnemonicUtils* static method), 324
- X**
- X() (*DummyPoint* method), 187
- X() (*Ed25519Point* method), 202
- X() (*IPoint* method), 193
- X() (*Nist256p1Point* method), 221
- X() (*Secp256k1PointCoincurve* method), 230
- X() (*Secp256k1PointEcdsa* method), 233
- XlmAddr (in module *bip_utils.addr.xlm_addr*), 59
- XlmAddrConst (class in *bip_utils.addr.xlm_addr*), 58
- XlmAddrDecoder (class in *bip_utils.addr.xlm_addr*), 59
- XlmAddrEncoder (class in *bip_utils.addr.xlm_addr*), 59
- XlmAddrTypes (class in *bip_utils.addr.xlm_addr*), 58
- XModemCrc (class in *bip_utils.utils.crypto.crc*), 302
- XmrAddr (in module *bip_utils.addr.xmr_addr*), 61
- XmrAddrConst (class in *bip_utils.addr.xmr_addr*), 60
- XmrAddrDecoder (class in *bip_utils.addr.xmr_addr*), 60
- XmrAddrEncoder (class in *bip_utils.addr.xmr_addr*), 60
- XmrIntegratedAddr (in module *bip_utils.addr.xmr_addr*), 61
- XmrIntegratedAddrDecoder (class in *bip_utils.addr.xmr_addr*), 60
- XmrIntegratedAddrEncoder (class in *bip_utils.addr.xmr_addr*), 61
- Xor() (*BytesUtils* static method), 313
- XrpAddr (in module *bip_utils.addr.xrp_addr*), 62
- XrpAddrDecoder (class in *bip_utils.addr.xrp_addr*), 62
- XrpAddrEncoder (class in *bip_utils.addr.xrp_addr*), 62
- XtzAddr (in module *bip_utils.addr.xtz_addr*), 64
- XtzAddrDecoder (class in *bip_utils.addr.xtz_addr*), 63
- XtzAddrEncoder (class in *bip_utils.addr.xtz_addr*), 63
- XtzAddrPrefixes (class in *bip_utils.addr.xtz_addr*), 63
- Y**
- Y() (*DummyPoint* method), 187
- Y() (*Ed25519Point* method), 202
- Y() (*IPoint* method), 193
- Y() (*Nist256p1Point* method), 221
- Y() (*Secp256k1PointCoincurve* method), 230
- Y() (*Secp256k1PointEcdsa* method), 233
- Z**
- ZCASH (*Bip44Coins* attribute), 146
- ZCASH (*Bip49Coins* attribute), 152
- ZCASH (*Slip44* attribute), 276
- ZCASH_TESTNET (*Bip44Coins* attribute), 146
- ZCASH_TESTNET (*Bip49Coins* attribute), 153
- ZcashMainNet (*Bip44Conf* attribute), 150
- ZcashMainNet (*Bip49Conf* attribute), 154
- ZcashMainNet (*CoinsConf* attribute), 186
- ZcashTestNet (*Bip44Conf* attribute), 150
- ZcashTestNet (*Bip49Conf* attribute), 154
- ZcashTestNet (*CoinsConf* attribute), 186
- ZilAddr (in module *bip_utils.addr.zil_addr*), 64
- ZilAddrConst (class in *bip_utils.addr.zil_addr*), 64
- ZilAddrDecoder (class in *bip_utils.addr.zil_addr*), 64
- ZilAddrEncoder (class in *bip_utils.addr.zil_addr*), 64
- ZILLIQA (*Bip44Coins* attribute), 146
- Zilliqa (*Bip44Conf* attribute), 150
- Zilliqa (*CoinsConf* attribute), 186
- ZILLIQA (*Slip173* attribute), 273
- ZILLIQA (*Slip44* attribute), 277