Bitcoin Utilities Documentation

Release 0.0.5

Konstantinos Karasavvas

Nov 12, 2018

CONTENTS

Contents:

CHAPTER

ONE

KEYS AND ADDRESSES MODULE

class keys.**Address**(*address=None*, *hash160=None*, *script=None*)

Represents a Bitcoin address

hash160

the hash160 string representation of the address; hash160 represents two consequtive hashes of the public key or the redeam script, first a SHA-256 and then an RIPEMD-160

Type str

from_address(*address*)

instantiates an object from address string encoding

from_hash160(*hash160_str*)

instantiates an object from a hash160 hex string

from_script(*redeem_script*)

instantiates an object from a redeem_script

to_address()

returns the address's string encoding

to_hash160()

returns the address's hash160 hex string representation

Raises

- TypeError – No parameters passed
- ValueError – If an invalid address or hash160 is provided.

classmethod from_address(*address*)

Creates and address object from an address string

classmethod from_hash160(*hash160*)

Creates and address object from a hash160 string

classmethod from_script(*script*)

Creates and address object from a Script object

to_address()

Returns as address string

network_prefix = (1 byte version number) data = network_prefix + hash160_bytes data_hash = SHA-256(SHA-256(hash160_bytes)) checksum = (first 4 bytes of data_hash) address_bytes = Base58CheckEncode(data + checksum)

to_hash160()

Returns as hash160 hex string

class keys.**P2pkhAddress**(*address=None*, *hash160=None*)
Encapsulates a P2PKH address.

Check Address class for details

class keys.**P2shAddress**(*address=None*, *hash160=None*, *script=None*)
Encapsulates a P2PKH address.

Check Address class for details

class keys.**PrivateKey**(*wif=None*, *secret_exponent=None*)
Represents an ECDSA private key.

key
the raw key of 32 bytes

Type bytes

from_wif(*wif*)
creates an object from a WIF of WIFC format (string)

to_wif(*compressed=True*)
returns as WIFC (compressed) or WIF format (string)

to_bytes()
returns the key's raw bytes

sign_message(*message*, *compressed=True*)
signs the message's digest and returns the signature

sign_transaction(*tx*, *compressed=True*)
signs the transaction's digest and returns the signature

get_public_key()
returns the corresponding PublicKey object

classmethod from_wif(*wif*)
Creates key from WIFC or WIF format key

get_public_key()
Returns the corresponding PublicKey

sign_input(*tx*, *txin_index*, *script*, *sighash=1*)
Signs a transaction input with the private key

Bitcoin uses the normal DER format for transactions. Each input is signed separately (thus txin_index is required). The script of the input we wish to spend is required and replaces the transaction's script sig in order to calculate the correct transaction hash (which is what is actually signed!)

Returns a signature for that input

sign_message(*message*, *compressed=True*)
Signs the message with the private key

Bitcoin uses a compact format for message signatures (for tx sigs it uses normal DER format). The format has the normal r and s parameters that ECDSA signatures have but also includes a prefix which encodes extra information. Using the prefix the public key can be reconstructed when verifying the signature.

Prefix values: 27 - 0x1B = first key with even y 28 - 0x1C = first key with odd y 29 - 0x1D = second key with even y 30 - 0x1E = second key with odd y

If key is compressed add 4 (31 - 0x1F, 32 - 0x20, 33 - 0x21, 34 - 0x22 respectively)

Returns a Bitcoin compact signature in Base64

to_bytes()
Returns key's bytes

to_wif(*compressed=True*)
Returns key in WIFC or WIF string

key_bytes = (32 bytes number) [+ 0x01 if compressed] network_prefix = (1 byte version number) data_hash = SHA-256(SHA-256(key_bytes)) checksum = (first 4 bytes of data_hash) wif = Base58CheckEncode(key_bytes + checksum)

class keys.**PublicKey**(*hex_str*)
Represents an ECDSA public key.

key
the raw public key of 64 bytes (x, y coordinates of the ECDSA curve)

Type bytes

from_hex(*hex_str*)
creates an object from a hex string in SEC format

from_message_signature(*signature*)
NO-OP!

verify_message(*address*, *signature*, *message*)
Class method that constructs the public key, confirms the address and verifies the signature

to_hex(*compressed=True*)
returns the key as hex string (in SEC format - compressed by default)

to_bytes()
returns the key's raw bytes

get_address(*compressed=True)*)
returns the corresponding P2pkhAddress object

classmethod from_hex(*hex_str*)
Creates a public key from a hex string (SEC format)

get_address(*compressed=True*)
Returns the corresponding P2PKH Address (default compressed)

to_bytes()
Returns key's bytes

to_hex(*compressed=True*)
Returns public key as a hex string (SEC format - compressed by default)

verify(*signature*, *message*)
Verifies a that the message was signed with this public key's corresponding private key.

classmethod verify_message(*address*, *signature*, *message*)
Creates a public key from a message signature and verifies message

Bitcoin uses a compact format for message signatures (for tx sigs it uses normal DER format). The format has the normal r and s parameters that ECDSA signatures have but also includes a prefix which encodes extra information. Using the prefix the public key can be reconstructed from the signature.

Prefix values: 27 - 0x1B = first key with even y 28 - 0x1C = first key with odd y 29 - 0x1D = second key with even y 30 - 0x1E = second key with odd y

If key is compressed add 4 (31 - 0x1F, 32 - 0x20, 33 - 0x21, 34 - 0x22 respectively)

Raises ValueError – If signature is invalid

CHAPTER

TWO

TRANSACTIONS MODULE

class transactions.**Transaction**(*inputs=[]*, *outputs=[]*, *locktime=b'x00x00x00x00'*, *version=b'x02x00x00x00'*)

Represents a Bitcoin transaction

inputs
A list of all the transaction inputs

Type list (*TxInput*)

outputs
A list of all the transaction outputs

Type list (*TxOutput*)

locktime
The transaction's locktime parameter

Type bytes

version
The transaction version

Type bytes

stream()
Converts Transaction to bytes

serialize()
Converts Transaction to hex string

get_txid()
Calculates txid and returns it

copy()
creates a copy of the object (classmethod)

get_transaction_digest(*txin_index*, *script*, *sighash*)
returns the transaction input's digest that is to be signed according to sighash

classmethod copy(*tx*)
Deep copy of Transaction

get_transaction_digest(*txin_index*, *script*, *sighash=1*)
Returns the transaction's digest for signing.

SIGHASH types (see constants.py): SIGHASH_ALL - signs all inputs and outputs (default) SIGHASH_NONE - signs all of the inputs SIGHASH_SINGLE - signs all inputs but only txin_index output SIGHASH_ANYONECANPAY (only combined with one of the above) - with ALL - signs all

outputs but only txin_index input - with NONE - signs only the txin_index input - with SINGLE - signs txin_index input and output

txin_index
The index of the input that we wish to sign
Type int

script
The scriptPubKey of the UTXO that we want to spend
Type list (string)

sighash
The type of the signature hash to be created
Type int

get_txid()
Hashes the serialized tx to get a unique id

serialize()
Converts to hex string

stream()
Converts to bytes

class transactions.**TxInput**(*txid*, *txout_index*, *script_sig=<bitcoinutils.script.Script object>*, *sequence=b'xffxffxffxff'*)
Represents a transaction input.

A transaction input requires a transaction id of a UTXO and the index of that UTXO.

txid
the transaction id as a hex string (little-endian as displayed by tools)

Type str

txout_index
the index of the UTXO that we want to spend

Type int

script_sig
the op code and data of the script as string

Type list (strings)

sequence
the input sequence (for timelocks, RBF, etc.)

Type bytes

stream()
converts TxInput to bytes

copy()
creates a copy of the object (classmethod)

classmethod copy(*txin*)
Deep copy of TxInput

stream()
Converts to bytes

class transactions.**TxOutput**(*amount*, *script_pubkey*)
Represents a transaction output

amount
the value we want to send to this output (in BTC)

Type float

script_pubkey
the script that will lock this amount

Type list (string)

stream()
converts TxInput to bytes

copy()
creates a copy of the object (classmethod)

classmethod copy(*txout*)
Deep copy of TxOutput

stream()
Converts to bytes

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

V