

---

# **Bitcoin Utilities Documentation**

***Release 0.0.4***

**Konstantinos Karasavvas**

**Nov 08, 2018**



# CONTENTS

<b>1</b>	<b>Keys and Addresses module</b>	<b>3</b>
<b>2</b>	<b>Transactions module</b>	<b>7</b>
<b>3</b>	<b>Indices and tables</b>	<b>11</b>
	<b>Python Module Index</b>	<b>13</b>
	<b>Index</b>	<b>15</b>



Contents:



## KEYS AND ADDRESSES MODULE

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

Represents a Bitcoin address

**hash160**

the hash160 string representation of the address; hash160 represents two consecutive hashes of the public key or the redeem script, first a SHA-256 and then a 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

**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

**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.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 Address object

**classmethod from\_hex** (*hex\_str*)

Creates a public key from a hex string (SEC format)

**get\_address** (*compressed=True*)

Returns the corresponding 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



## TRANSACTIONS MODULE

```
class transactions.Transaction (inputs=[], outputs=[], locktime=b'x00x00x00x00', ver-  
sion=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'\xff\xff\xff\xff')

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



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`





## PYTHON MODULE INDEX

### k

keys, [3](#)

### t

transactions, [7](#)



## A

Address (class in keys), 3  
amount (transactions.TxOutput attribute), 8

## C

copy() (transactions.Transaction class method), 7  
copy() (transactions.Transaction method), 7  
copy() (transactions.TxInput class method), 8  
copy() (transactions.TxInput method), 8  
copy() (transactions.TxOutput class method), 9  
copy() (transactions.TxOutput method), 9

## F

from\_address() (keys.Address class method), 3  
from\_address() (keys.Address method), 3  
from\_hash160() (keys.Address class method), 3  
from\_hash160() (keys.Address method), 3  
from\_hex() (keys.PublicKey class method), 5  
from\_hex() (keys.PublicKey method), 5  
from\_message\_signature() (keys.PublicKey method), 5  
from\_wif() (keys.PrivateKey class method), 4  
from\_wif() (keys.PrivateKey method), 4

## G

get\_address() (keys.PublicKey method), 5  
get\_public\_key() (keys.PrivateKey method), 4  
get\_transaction\_digest() (transactions.Transaction method), 7  
get\_txid() (transactions.Transaction method), 7, 8

## H

hash160 (keys.Address attribute), 3

## I

inputs (transactions.Transaction attribute), 7

## K

key (keys.PrivateKey attribute), 3  
key (keys.PublicKey attribute), 4  
keys (module), 3

## L

locktime (transactions.Transaction attribute), 7

## O

outputs (transactions.Transaction attribute), 7

## P

PrivateKey (class in keys), 3  
PublicKey (class in keys), 4

## S

script (transactions.Transaction attribute), 8  
script\_pubkey (transactions.TxOutput attribute), 9  
script\_sig (transactions.TxInput attribute), 8  
sequence (transactions.TxInput attribute), 8  
serialize() (transactions.Transaction method), 7, 8  
sighash (transactions.Transaction attribute), 8  
sign\_input() (keys.PrivateKey method), 4  
sign\_message() (keys.PrivateKey method), 4  
sign\_transaction() (keys.PrivateKey method), 4  
stream() (transactions.Transaction method), 7, 8  
stream() (transactions.TxInput method), 8  
stream() (transactions.TxOutput method), 9

## T

to\_address() (keys.Address method), 3  
to\_bytes() (keys.PrivateKey method), 4  
to\_bytes() (keys.PublicKey method), 5  
to\_hash160() (keys.Address method), 3  
to\_hex() (keys.PublicKey method), 5  
to\_wif() (keys.PrivateKey method), 4  
Transaction (class in transactions), 7  
transactions (module), 7  
txid (transactions.TxInput attribute), 8  
txin\_index (transactions.Transaction attribute), 8  
TxInput (class in transactions), 8  
txout\_index (transactions.TxInput attribute), 8  
TxOutput (class in transactions), 8

## V

verify() (keys.PublicKey method), 5

`verify_message()` (`keys.PublicKey` class method), [5](#)  
`verify_message()` (`keys.PublicKey` method), [5](#)  
`version` (`transactions.Transaction` attribute), [7](#)