QRL Documentation

Release 0.60.0

The Quantum Resistant Ledger

Feb 03, 2018

Contents

This is the documentation of **QRL Node**.

Note: WORK IN PROGRESS

Note: This is the main page of your project's Sphinx documentation.

It is also possible to refer to the documentation of other Python packages with the Python domain syntax. By default you can reference the documentation of Sphinx, Python, NumPy, SciPy, matplotlib, Pandas, Scikit-Learn. You can add more by extending the `intersphinx_mapping` in your Sphinx's `conf.py`.

The pretty useful extension autodoc is activated by default and lets you include documentation from docstrings. Docstrings can be written in Google (recommended!), NumPy and classical style.

CHAPTER 1

Contents

1.1 License

```
MIT License

Copyright (c) 2017 The Quantum Resistant Ledger

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

1.2 Developers

- qrlcore developers:

#1 cyyber 309 commits 19,410 ++ 16,048 –

#2 jleni 240 commits 72,648 ++ 66,410 –

#3 surg0r 227 commits 10,857 ++ 5,095 –

#4 scottdonaldau 29 commits 241,841 ++ 240,315 –

#5 bidulemachin 13 commits 285 ++ 57 –

#6 randomshinichi 10 commits 632 ++ 481 –

#7 converghub 6 commits 204 ++ 40 –

#8 jplomas 4 commits 33 ++ 3 –

#9 Wyc0 4 commits 166 ++ 11 –

#10 leagueofdelegates 4 commits 0 ++ 0 –

#11 randomusergenerator 3 commits 173 ++ 2 –

#12 nickycakes 3 commits 24 ++ 18 –

#13 jomarip 2 commits 36 ++ 0 –

#14 r41d 2 commits 11 ++ 17 –

#15 timhovius 1 commit 2 ++ 2 –

#16 fullmatches 1 commit 0 ++ 0 –

#17 jackalyst 1 commit 1 ++ 1 –

23/10/2017

1.3 Changelog

1.3.1 Version 0.X

- DESCRIBE IMPORTANT CHANGES

1.4 qrl

1.4.1 qrl package

Subpackages

qrl.core package

Submodules

qrl.core.AddressState module

qrl.core.Block module

qrl.core.BlockHeader module

qrl.core.BlockMetadata module

qrl.core.ChainManager module

qrl.core.DifficultyTracker module

qrl.core.ESyncState module

class qrl.core.ESyncState.**ESyncState**
Bases: enum.Enum

An enumeration.

forked = 4

synced = 3

syncing = 2

unknown = 0

unsynced = 1

qrl.core.Ephemeral module

class qrl.core.Ephemeral.**Ephemeral**
Bases: object

Ephemeral Messaging Layer

qrl.core.EphemeralMessage module

qrl.core.EphemeralMetadata module

qrl.core.GenesisBlock module

qrl.core.Message module

class qrl.core.Message.**Message**(*pbdata*, *msg_type*)

Bases: object

add_peer(*msg_type*)

qrl.crypto.doctest_data.**binvec2hstr**(*data*)

qrl.crypto.hashchain module

qrl.crypto.misc module

qrl.crypto.misc.**merkle_tx_hash**(*hashes*)
merkle tree root hash of tx from pool for next POW block :param hashes: :return:

```
>>> bin2hstr(merkle_tx_hash([b'0', b'1']) ) # FIXME: This input is not realistic
'938db8c9f82c8cb58d3f3ef4fd250036a48d26a712753d2fde5abd03a85cabf4'
>>> merkle_tx_hash([
↪'938db8c9f82c8cb58d3f3ef4fd250036a48d26a712753d2fde5abd03a85cabf4'])
'938db8c9f82c8cb58d3f3ef4fd250036a48d26a712753d2fde5abd03a85cabf4'
>>> bin2hstr(merkle_tx_hash(
↪'938db8c9f82c8cb58d3f3ef4fd250036a48d26a712753d2fde5abd03a85cabf4'))
'938db8c9f82c8cb58d3f3ef4fd250036a48d26a712753d2fde5abd03a85cabf4'
>>> bin2hstr(merkle_tx_hash([b'0', b
↪'938db8c9f82c8cb58d3f3ef4fd250036a48d26a712753d2fde5abd03a85cabf4'])) # FIXME:␣
↪This input is not realistic
'40243e694d9c015d5097590bcc9df82683d8ba4006d58c6abb5e1a6bee5ec6dc'
```

qrl.crypto.misc.**sha256**(*message: bytes*) → bytes

Parameters **message** (*Union[str, unicode]*) –

Returns

Return type str

```
>>> bin2hstr(sha256(b"test"))
'9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08'
>>> bin2hstr(sha256(b"another string"))
'81e7826a5821395470e5a2fed0277b6a40c26257512319875e1d70106dcb1ca0'
```

qrl.crypto.misc.**sha256_n**(*message: bytes*, *count*) → bytes

qrl.crypto.xmss module

class qrl.crypto.xmss.**XMSS**(*tree_height*, *seed=None*, *_xmssfast=None*)
Bases: object

SIGN(*message*)

Parameters **message** –

Returns

```
>>> from qrl.crypto.doctest_data import *; bin2hstr(XMSS(4, xmss_test_seed1).
↪SIGN(str2bin("test_message"))) == xmss_sign_expected1
True
>>> from qrl.crypto.doctest_data import *; bin2hstr(XMSS(4, xmss_test_seed2).
↪SIGN(str2bin("test_message"))) == xmss_sign_expected2
True
```

static VERIFY(*message: bytes*, *signature: bytes*, *pk: bytes*)
Verify an xmss sig with shorter PK same function but verifies using shorter signature where PK: {root, hex(_public_SEED)} # main verification function.. :param pk: :type pk: :param message: :param signature: :return:

```
>>> from qrl.crypto.doctest_data import *; XMSS.VERIFY( str2bin("test_message
↪"), hstr2bin(xmss_sign_expected1), hstr2bin(xmss_pk_expected1))
True
>>> from qrl.crypto.doctest_data import *; XMSS.VERIFY( str2bin("test_messagex
↪"), hstr2bin(xmss_sign_expected1), hstr2bin(xmss_pk_expected1))
False
>>> from qrl.crypto.doctest_data import *; XMSS.VERIFY( str2bin("test_message
↪"), hstr2bin(xmss_sign_expected2), hstr2bin(xmss_pk_expected2))
True
>>> from qrl.crypto.doctest_data import *; XMSS.VERIFY( str2bin("test_messagex
↪"), hstr2bin(xmss_sign_expected2), hstr2bin(xmss_pk_expected2))
False
```

get_address() → bytes

get_hexseed()

Returns

Return type

```
>>> from qrl.crypto.doctest_data import *; XMSS(4, xmss_test_seed1).get_
↪hexseed()

↪'303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303
↪'
>>> from qrl.crypto.doctest_data import *; XMSS(4, xmss_test_seed2).get_
↪hexseed()

↪'333133313331333133313331333133313331333133313331333133313331333133313331333133313331333133313331333
↪'
```

get_index()

Returns

Return type

```
>>> from qrl.crypto.doctest_data import *; XMSS(4, xmss_test_seed1).get_
↪index()
0
>>> from qrl.crypto.doctest_data import *; XMSS(4, xmss_test_seed2).get_
↪index()
0
>>> from qrl.crypto.doctest_data import *
>>> xmss = XMSS(4, xmss_test_seed2)
>>> s = xmss.SIGN(str2bin("test"))
>>> xmss.get_index()
1
```

get_mnemonic()

Returns

Return type

```
>>> from qrl.crypto.doctest_data import *; XMSS(4, hstr2bin(xmss_mnemonic_
↪seed1)).get_mnemonic() == xmss_mnemonic_test1
True
>>> from qrl.crypto.doctest_data import *; XMSS(4, hstr2bin(xmss_mnemonic_
↪seed2)).get_mnemonic() == xmss_mnemonic_test2
```

```
True
>>> from qrl.crypto.doctest_data import *; XMSS(4, mnemonic2bin(xmss_mnemonic_
↪test1)).get_mnemonic() == xmss_mnemonic_test1
True
>>> from qrl.crypto.doctest_data import *; XMSS(4, mnemonic2bin(xmss_mnemonic_
↪test2)).get_mnemonic() == xmss_mnemonic_test2
True
```

get_number_signatures()

Returns

Return type

```
>>> from qrl.crypto.doctest_data import *; XMSS(4, xmss_test_seed1).get_
↪number_signatures()
16
>>> from qrl.crypto.doctest_data import *; XMSS(4, xmss_test_seed2).get_
↪number_signatures()
16
```

get_remaining_signatures()

Returns

Return type

```
>>> from qrl.crypto.doctest_data import *; XMSS(4, xmss_test_seed1).get_
↪remaining_signatures()
16
>>> from qrl.crypto.doctest_data import *; XMSS(4, xmss_test_seed2).get_
↪remaining_signatures()
16
```

get_seed()

Returns

Return type

```
>>> from qrl.crypto.doctest_data import *; bin2hstr( XMSS(4, xmss_test_seed1).
↪get_seed() )

↪'303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303030303
↪'
>>> from qrl.crypto.doctest_data import *; bin2hstr( XMSS(4, xmss_test_seed2).
↪get_seed() )

↪'333133313331333133313331333133313331333133313331333133313331333133313331333133313331333133313331333133313331333133313331333133313331333
↪'
```

get_seed_private()

Returns

Return type

```
>>> from qrl.crypto.doctest_data import *; bin2hstr( XMSS(4, xmss_test_seed1).
↪get_seed_public() )
'51ec21420dd061739e4637fd74517a46f86f89e0fb83f2526fafafe356e564ff'
```

```
>>> from qrl.crypto.doctest_data import *; bin2hstr( XMSS(4, xmss_test_seed2).
↪get_seed_public() )
'df2355c48096f2351e4d04db57b326c355345552d31b75a65ac18b1f6d7c7875'
```

get_seed_public()

Returns

Return type

```
>>> from qrl.crypto.doctest_data import *; bin2hstr( XMSS(4, xmss_test_seed1).
↪get_seed_private() )
'5f2eb95ccf6a0e3e7f472c32d234340c20b3fd379dc28b710affcc0cb2afa57b'
>>> from qrl.crypto.doctest_data import *; bin2hstr( XMSS(4, xmss_test_seed2).
↪get_seed_private() )
'ad70ef34f316aaadcbf16a64b1b381db731eb53d833745c0d3eaa1e24cf728a2'
```

get_type()

height

list_addresses()

List the addresses derived in the main tree :return:

pk()

```
>>> from qrl.crypto.doctest_data import *; bin2hstr(XMSS(4, xmss_test_seed1).
↪pk()) == xmss_pk_expected1
True
>>> from qrl.crypto.doctest_data import *; bin2hstr(XMSS(4, xmss_test_seed2).
↪pk()) == xmss_pk_expected2
True
```

set_index(*new_index*)

Returns

Return type

```
>>> from qrl.crypto.doctest_data import *
>>> xmss = XMSS(4, xmss_test_seed1)
>>> xmss.set_index(1)
>>> xmss.get_index()
1
>>> from qrl.crypto.doctest_data import *
>>> xmss = XMSS(4, xmss_test_seed1)
>>> xmss.set_index(10)
>>> xmss.get_index()
10
```

Module contents

Submodules

qrl.cli module

qrl.main module

Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

q

Index