pyspool Documentation

Release 0.1.0

ascribe GmbH

June 20, 2016

Contents

`pyspool` is a reference implementation of the Secure Public Online Ownership Ledger SPOOL and part of the development stack of ascribe.io

CHAPTER 1

Installation

```
pip install pyspool
```

CHAPTER 2

Contents

2.1 Overview

All hashes are in base 58 so that they can be inserted in the bitcoin blockchain.

2.1.1 Refill Wallet

The refill wallet is only used to refill the Federation Wallet with 10000 and 600 satoshi outputs. This way we maintain the Federation Wallet clean which makes it easier to manage.

2.1.2 Federation Wallet

The Federation wallet is the wallet from where all the registers in the blockchain occur. We can check the validity of transactions by looking at the origin of this transactions and see if the originating address belongs to a federation. For instance all pieces registered through ascribe.io will originate from the address `1JttRRdtAi6cDNM23Uq4BEU61R8kJeANJs` for the first version of the protocol or `1AScRhqdXMrJyxNmjEapMZi1PLFsqmLquG` for protocol version `01`.

The Federation wallet should only contain fuel outputs with the amount of 10000 satoshi and token outputs with the amount of 600 satoshi. The fuel is used to pay the fee for the transactions and tokens are used to register address on the blockchain.

Ensuring that the Federation wallet cointains outputs with these amounts makes the transactions simpler since there is no need for change. It is also easier to keep track of the unspents in order to prevent invalid transactions due to double spends (specially important when there is a high throughput of transactions).

2.2 Examples

All the addresses and transaction ids output by this examples can be checked in the bitcoin blockchain.

For this example we will be implementing the following actions:

1. Refill Federation wallet with fuel and tokens
2. `user1` registers master edition
3. `user1` registers number of editions
4. `user1` registers edition number `1`

5. `user1` transfers edition number `1` to `user2`
6. `user2` consigns edition number `1` to `user3`
7. `user3` unconsigns edition number `1` to `user2`
8. `user2` loans edition number `1` to `user3`

This is the information that we will need:

refill_pass refill wallet password

federation_pass federation wallet password

user1_pass `user1` password

user2_pass `user2` password

user3_pass `user3` password

refill_root

```
('', u'mhyCaF2HFk7CVwKmyQ8TahgVdjnHSr1pTv')
```

federation_root

```
('', u'mqXz83H4LCxjf2ie8hYNsTRByvtfV43Pa7')
```

user1_root root address for `user1` HD wallet

```
('', u'n36EiKuYYXNS9h84CnkLK3sEevZsfksaGN')
```

user1_leaf leaf address for `user1` HD wallet

```
('2015/5/28/9/28/44/982190',
 u'mjq4rZZEyJGFfzg59RLFmuNJTo3jakEDrS')
```

user2_leaf leaf address for `user2` HD wallet

```
('2015/5/28/9/28/45/184623',
 u'mhRJqLG3BwXXG7YADQD3Ng3c6coZVwoFo6')
```

user3_leaf leaf address for `user3` HD wallet

```
('2015/5/28/9/28/45/387493',
 u'mhMXXntQCorduhcygY7gLdmuunVEM59C8q')
```

file_hash file_hash, file_hash + metadata

```
(u'mhXeWEMLnEwVNnyqKrqUDPAYSuwhfyNXA7',
 u'mzQxP43Y4A6PfYeArV2mGBEucJfDtsyCk5')
```

2.2.1 Federation Wallet Refill

Before we can start with spool transactions we need to have a Federation with all the necessary fuel and tokens.

```
>>> from spool import Spool

>>> spool = Spool(testnet=True)

>>> # lets refill the federation wallet with necessary fuel*7 and tokens*11 for this example
>>> txid = spool.refill_main_wallet(refill_root, federation_root[1], 7, 11,
                                    refill_pass, min_confirmations=1, sync=True)
```

```
>>> print txid
67d22e66ee46a96e94f08bed0c857f23de39aee8b25db5fa0369c495e072e44c
```

67d22e66ee46a96e94f08bed0c857f23de39aee8b25db5fa0369c495e072e44c

2.2.2 Registrations

`user1` registers the master edition

Now that we have enough funds in the Federation wallet `user1` can ascribe his master edition. A master edition is a register with edition number `0` that ascribes the piece to user1 making him the original owner/creator of the piece. Master editions are ascribed to the user's root address of the HD wallet.

```
>>> # user1 registers the master edition of piece with file_hash
>>> txid = spool.register(federation_root, user1_root[1], file_hash,
                          federation_pass, 0, min_confirmations=1, sync=True)
>>> print txid
f67aa26b5f47e83124040970246c969d04ec9adecc5a97d60754a0f54355ee81
```

f67aa26b5f47e83124040970246c969d04ec9adecc5a97d60754a0f54355ee81

`user1` registers the number of editions

Now that `user1` has registered the master edition he can now specify how many editions of the piece will exist. `user1` can only do this once and this cannot be changed in the future. This creates digital scarcity for this particular piece

```
>>> # user1 specifies that there will be 10 editions of the piece with hash file_hash
>>> txid = spool.editions(federation_root, user1_root, file_hash,
                          federation_pass, 10, min_confirmations=1, sync=True)
>>> print txid
f1f2cdf6ef2ee2d8af13f9d45a1fd7700f9f281078c71939f78326a4b6b957dc
```

f1f2cdf6ef2ee2d8af13f9d45a1fd7700f9f281078c71939f78326a4b6b957dc

`user1` registers edition number `1`

Once the number of editions is registered `user1` can now start registering editions so that he can transfer ownership to other users. Each edition is registered to a different leaf address of `user1` HD wallet

```
>>> # user1 registers edition number 1 of piece with file_hash
>>> txid = spool.register(federation_root, user1_leaf[1], file_hash,
                          federation_pass, 1, min_confirmations=1, sync=True)
>>> print txid
2376a200a326ee7cf87b7fee7ea0f9a80c8b23cc3a0d72732b9a75517e664f23
```

2376a200a326ee7cf87b7fee7ea0f9a80c8b23cc3a0d72732b9a75517e664f23

2.2.3 Transfers

`user1` transfers edition number `1` to `user2`

Now that an edition is registered the user can transfer ownership to another user. Transfering ownserhip implies a transaction originating from `user1` wallet address holding the edition. This means that we need to fuel `user1` wallet

with the necessary funds before performing a spool transaction

```
>>> # refill user1 wallet before transfer
>>> txid = spool.refill(federation_root, user1_leaf[1], 1, 1,
                federation_pass, min_confirmations=1, sync=True)
>>> print txid
45bc2a3eecac9b5538a3b5bc325e94fcffee47c0025e78ece426aeebfac59c24

>>> # now we can transfer ownserhip of edition 1 from user1 to user2
>>> txid = spool.transfer(user1_leaf, user2_leaf[1], file_hash,
                          user1_pass, 1, min_confirmations=1, sync=True)
>>> print txid
38509a49b00f3c3c3fadedd2c5ce35ffcc05a9737a36dd1b7ff00ed1ffe5fd80
```

45bc2a3eecac9b5538a3b5bc325e94fcffee47c0025e78ece426aeebfac59c24 38509a49b00f3c3c3fadedd2c5ce35ffcc05a9737a36dd1b7ff

2.2.4 Consignments

user2 consigns edition number 1 to user3

`user2` now owns edition `1` of piece with hash `file_hash` and he can transfer ownership of the piece. Lets consign the piece to `user3`

```
>>> # refill user2 wallet before consign
>>> txid = spool.refill(federation_root, user2_leaf[1], 1, 1,
                        federation_pass, min_confirmations=1, sync=True)
>>> print txid
e07732c8af3557277f68871babc874766c511fdf898449cc9be9e505f8325f10

>>> # now we can consign edition 1 from user2 to user3
>>> txid = spool.consign(user2_leaf, user3_leaf[1], file_hash,
                         user2_pass, 1, min_confirmations=1, sync=True)
>>> print txid
3b30cea26d49eb023ccd62fb78ddd9308c9505fe0796abc0fe60989980fc5eb8
```

e07732c8af3557277f68871babc874766c511fdf898449cc9be9e505f8325f10 3b30cea26d49eb023ccd62fb78ddd9308c9505fe0796abc0f

2.2.5 Unconsignments

user3 unconsigns edition number 1 to user2

Now lets unconsign the piece with hash `file_hash` back to `user2`

```
>>> # refill user3 wallet before unconsign
>>> txid = spool.refill(federation_root, user3_leaf[1], 1, 1,
                        federation_pass, min_confirmations=1, sync=True)
>>> print txid
f0c9cf0832e7ca14012e7379da35dd2d50bd66df45c2eb089a23b10db4047dcc

>>> # user3 unconsigns edition number 1 back to user2
>>> txid = spool.unconsign(user3_leaf, user2_leaf[1], file_hash,
                           user3_pass, 1, min_confirmations=1, sync=True)
>>> print txid
11dcb46061526790e0e7cf0a83e9163d35b75461cd203858c1fd7bdb2149db0c
```

f0c9cf0832e7ca14012e7379da35dd2d50bd66df45c2eb089a23b10db4047dcc 11dcb46061526790e0e7cf0a83e9163d35b75461cd20385

2.2.6 Loans

user2 loans edition number 1 to user3

Now that user2 owns the edition again lets loan it to user3 from 15-05-22 to 15-05-23

```
>>> # refill user2 wallet before loan
>>> txid = spool.refill(federation_root, user2_leaf[1], 1, 1,
                        federation_pass, min_confirmations=1, sync=True)
>>> print txid
087d85fd421db42c3efac5e6aa499edfe7386101b85314b44c86681a98c27832

>>> # user2 loans edition number 1 to user3
>>> txid = spool.loan(user2_leaf, user3_leaf[1], file_hash,
                      user2_pass, 1, '150522', '150523', min_confirmations=1, sync=True)
>>> print txid
6cc0066ee737a7104859328729cd10f8c5a5b64be3f4f8bfcab04f8a6aca4c56
```

087d85fd421db42c3efac5e6aa499edfe7386101b85314b44c86681a98c27832 6cc0066ee737a7104859328729cd10f8c5a5b64be3f4f8bfc

2.3 Library Reference

2.3.1 Spool

2.3.2 File

2.3.3 Wallet

2.3.4 Spoolverb

2.3.5 BlockchainSpider

2.3.6 Ownership

2.3.7 Exceptions

2.4 Testing

At the moment most tests rely on a bitcoin regtest node running. Depending on how the bitcoin node is run some environment variables may need to be set.

The simplest is to use the provided docker-compose.yml under the pyspool github repository.

First run the bitcoin regtest daemon in background mode:

```
$ docker-compose up -d bitcoin
```

Then run the tests:

```
$ docker-compose run --rm spool py.test -v
```

To run the tests against python 2:

```
$ docker-compose run --rm spool-py2 py.test -v
```

Note: You may need to build the image for the services `spool` and `spool-py2`. E.g.:

```
$ docker-compose build spool
```

2.4.1 Without Docker

The tests rely on four environment variables specific to bitcoin:

BITCOIN_HOST
The host of the bitcoin regtest node. Defaults to `'localhost'`.

BITCOIN_PORT
The port of the bitcoin regtest node. Defaults to `18332`.

BITCOIN_RPCUSER
The rpc user used to connect to bitcoin regtest node. Defaults to `'merlin'`.

BITCOIN_RPCPASSWORD
The password of the user, used to connect to the bitcoin regtest node. Defaults to `'secret'`.

Assuming the above default environment variables, a bitcoin regtest node can be run as follows:

```
$ bitcoind -daemon -regtest -rpcuser=merlin -rpcpassword=secret -txindex=1
```

Important: Please note the `-txindex=1` option. This ensures that all transactions are indexed and retrievable by the RPC `getrawtransaction`. Without this option some tests will fail.

Using a `bitcoin.conf`

A bitcoin.conf file can also be used. E.g.:

```
# $HOME/.bitcoin/bitcoin.conf (under linux)
rpcuser=merlin
rpcpassword=secret
txindex=1
```

Tip: The .travis.yml, docker-compose.yml, and bitcoin_regtest.conf files, under the `pyspool` github repository may be helpful to look at.

2.5 Contributing

Pull requests, feedback, and suggestions are welcome. Github repository is at https://github.com/ascribe/pyspool

You can also send inquiries directly to devel@ascribe.io

2.6 About this Documentation

This section contains instructions to build and view the documentation locally, using the `docker-compose.yml` file of the `pyspool` repository: https://github.com/ascribe/pyspool.

If you do not have a clone of the repo, you need to get one.

2.6.1 Building the documentation

To build the docs, simply run

```
$ docker-compose up bdocs
```

Or if you prefer, start a `bash` session,

```
$ docker-compose run --rm bdocs bash
```

and build the docs:

```
root@a651959a1f2d:/usr/src/app/docs# make html
```

2.6.2 Viewing the documentation

You can start a little web server to view the docs at http://localhost:40084/

```
$ docker-compose up -d vdocs
```

Note: If you are using `docker-machine` you need to replace `localhost` with the `ip` of the machine (e.g.: `docker-machine ip tm` if your machine is named `tm`).

2.6.3 Making changes

The necessary source code is mounted, which allows you to make modifications, and view the changes by simply re-building the docs, and refreshing the browser.

2.7 Background

pyspool was developed by ascribe GmbH as part of the overall ascribe technology stack. http://www.ascribe.io

2.8 License

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

E