```
BIP: 67
Layer: Applications
Title: Deterministic Pay-to-script-hash multi-signature addresses through public key sorti
Author: Thomas Kerin <me@thomaskerin.io>
        Jean-Pierre Rupp <root@haskoin.com>
        Ruben de Vries <ruben@rubensayshi.com>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0067
Status: Proposed
Type: Standards Track
Created: 2015-02-08
License: PD
```

Abstract

This BIP describes a method to deterministically generate multi-signature pay-to-script-hash transaction scripts. It focuses on defining how the public keys must be encoded and sorted so that the redeem script and corresponding P2SH address are always the same for a given set of keys and number of required signatures.

Motivation

Pay-to-script-hash (BIP-0011[1]) is a transaction type that allows funding of arbitrary scripts, where the recipient carries the cost of fee's associated with using longer, more complex scripts.

Multi-signature pay-to-script-hash transactions are defined in BIP-0016[2]. The redeem script does not require a particular ordering or encoding for public keys. This means that for a given set of keys and number of required signatures, there are as many as 2(n!) possible standard redeem scripts, each with its separate P2SH address. Adhering to an ordering and key encoding would ensure that a multi-signature "account" (set of public keys and required signature count) has a canonical P2SH address.

By adopting a sorting and encoding standard, compliant wallets will always produce the same P2SH address for the same given set of keys and required signature count, making it easier to recognize transactions involving that multi-signature account. This is particularly attractive for multisignature hierarchical-deterministic wallets, as less state is required to setup multi-signature accounts: only the number of required signatures and master public keys of participants need to be shared, and all wallets will generate the same addresses.

While most web wallets do not presently facilitate the setup of multisignature accounts with users of a different service, conventions which ensure cross-

[1] BIP-0011
[2] BIP-0016

compatibility should make it easier to achieve this.

Many wallet as a service providers use a 2of3 multi-signature schema where the user stores 1 of the keys (offline) as backup while using the other key for daily use and letting the service cosign his transactions. This standard will help in enabling a party other than the service provider to recover the wallet without any help from the service provider.

Specification

For a set of public keys, ensure that they have been received in compressed form:

```
022df8750480ad5b26950b25c7ba79d3e37d75f640f8e5d9bcd5b150a0f85014da
03e3818b65bcc73a7d64064106a859cc1a5a728c4345ff0b641209fba0d90de6e9
021f2f6e1e50cb6a953935c3601284925decd3fd21bc445712576873fb8c6ebc18
```

Sort them lexicographically according to their binary representation:

```
021f2f6e1e50cb6a953935c3601284925decd3fd21bc445712576873fb8c6ebc18
022df8750480ad5b26950b25c7ba79d3e37d75f640f8e5d9bcd5b150a0f85014da
03e3818b65bcc73a7d64064106a859cc1a5a728c4345ff0b641209fba0d90de6e9
```

..before using the resulting list of keys in a standard multisig redeem script:

```
OP_2 021f2f6e1e50cb6a953935c3601284925decd3fd21bc445712576873fb8c6ebc18 022df8750480ad5b2
```

Hash the redeem script according to BIP-0016 to get the P2SH address.

```
3Q4sF6tv9wsdqu2NtARzNCpQgwifm2rAba
```

Compatibility

- Uncompressed keys are incompatible with this specificiation. A compatible implementation should not automatically compress keys. Receiving an uncompressed key from a multisig participant should be interpreted as a sign that the user has an incompatible implementation.
- P2SH addressses do not reveal information about the script that is receiving the funds. For this reason it is not technically possible to enforce this BIP as a rule on the network. Also, it would cause a hard fork.
- Implementations that do not conform with this BIP will have compatibility issues with strictly-compliant wallets.
- Implementations which do adopt this standard will be cross-compatible when choosing multisig addressses.
- If a group of users were not entirely compliant, there is the possibility that a participant will derive an address that the others will not recognize as part of the common multisig account.

Test vectors

Two signatures are required in each of these test vectors.

Vector 1

- List
  - 02ff12471208c14bd580709cb2358d98975247d8765f92bc25eab3b2763ed605f8
  - 02fe6f0a5a297eb38c391581c4413e084773ea23954d93f7753db7dc0adc188b2f
- Sorted
  - 02fe6f0a5a297eb38c391581c4413e084773ea23954d93f7753db7dc0adc188b2f
  - 02ff12471208c14bd580709cb2358d98975247d8765f92bc25eab3b2763ed605f8
- Script
  - 522102fe6f0a5a297eb38c391581c4413e084773ea23954d93f7753db7dc0adc188b2f2102ff12471208c14bd58
- Address
  - 39bgKC7RFbpoCRbtD5KEdkYKtNyhpsNa3Z

Vector 2 (Already sorted, no action required)

- List:
  - 02632b12f4ac5b1d1b72b2a3b508c19172de44f6f46bcee50ba33f3f9291e47ed0
  - 027735a29bae7780a9755fae7a1c4374c656ac6a69ea9f3697fda61bb99a4f3e77
  - 02e2cc6bd5f45edd43bebe7cb9b675f0ce9ed3efe613b177588290ad188d11b404
- Sorted:
  - 02632b12f4ac5b1d1b72b2a3b508c19172de44f6f46bcee50ba33f3f9291e47ed0
  - 027735a29bae7780a9755fae7a1c4374c656ac6a69ea9f3697fda61bb99a4f3e77
  - 02e2cc6bd5f45edd43bebe7cb9b675f0ce9ed3efe613b177588290ad188d11b404
- Script
  - 522102632b12f4ac5b1d1b72b2a3b508c19172de44f6f46bcee50ba33f3f9291e47ed021027735a29bae7780a9
- Address
  - 3CKHTjBKxCARLzwABMu9yD85kvtm7WnMfH

Vector 3:

- List:
  - 0300000000000000000000000000000000000041414141414141414141414141
  - 0200000000000000000000000000000000000041414141414141414141414141
  - 0200000000000000000000000000000000000041414141414141414141414140
  - 0300000000000000000000000000000000000041414141414141414141414140
- Sorted:
  - 0200000000000000000000000000000000000041414141414141414141414140
  - 0200000000000000000000000000000000000041414141414141414141414141
  - 0300000000000000000000000000000000000041414141414141414141414140
  - 0300000000000000000000000000000000000041414141414141414141414141
- Script
  - 5221020000000000000000000000000000000000041414141414141414141414140210200000000000000000
- Address
  - 32V85igBri9zcfBRVupVvwK18NFtS37FuD

Vector 4: (from bitcore)

- List:
  - 022df8750480ad5b26950b25c7ba79d3e37d75f640f8e5d9bcd5b150a0f85014da
  - 03e3818b65bcc73a7d64064106a859cc1a5a728c4345ff0b641209fba0d90de6e9
  - 021f2f6e1e50cb6a953935c3601284925decd3fd21bc445712576873fb8c6ebc18
- Sorted:
  - 021f2f6e1e50cb6a953935c3601284925decd3fd21bc445712576873fb8c6ebc18
  - 022df8750480ad5b26950b25c7ba79d3e37d75f640f8e5d9bcd5b150a0f85014da
  - 03e3818b65bcc73a7d64064106a859cc1a5a728c4345ff0b641209fba0d90de6e9
- Script
  - 5221021f2f6e1e50cb6a953935c3601284925decd3fd21bc445712576873fb8c6ebc1821022df8750480ad5b26
- Address
  - 3Q4sF6tv9wsdqu2NtARzNCpQgwifm2rAba

Acknowledgements

The authors wish to thank BtcDrak and Luke-Jr for their involvement & contributions in the early discussions of this BIP.

Usage & Implementations

- BIP-0045 - Structure for Deterministic P2SH Multisignature Wallets
- Bitcore
- Haskoin - Bitcoin implementation in Haskell
- Armory
- BitcoinJ

References

Copyright