```
BIP: 45
Layer: Applications
Title: Structure for Deterministic P2SH Multisignature Wallets
Author: Manuel Araoz <manu@bitpay.com>
        Ryan X. Charles <ryan@bitpay.com>
        Matias Alejo Garcia <matias@bitpay.com>
Comments-Summary: No comments yet.
Comments-URI: https://github.com/bitcoin/bips/wiki/Comments:BIP-0045
Status: Proposed
Type: Standards Track
Created: 2014-04-25
```

Abstract

This BIP defines a structure for hierarchical deterministic P2SH multi-party multi-signature wallets (HDPM wallets from now on) based on the algorithm described in BIP-0032 (BIP32 from now on) and purpose scheme described in BIP-0043 (BIP43 from now on). This BIP is a particular application of BIP43.

Motivation

The structure proposed in this document allows for standard ways to create, use, import, and store HDPM wallets. It allows to handle multiple parties sharing an m-of-n wallet, on the following assumptions:

- n parties share an m-of-n wallet.
- Each party generates their master private keys independently.
- Multisig P2SH is used for all addresses.
- BIP32 is used to derive public keys, then create a multisig script, and the corresponding P2SH address for that script.
- Address generation should not require communication between parties. (Thus, all parties must be able to generate all public keys)
- Transaction creation and signing requires communication between parties.

This BIP will allow interoperability between various HDPM wallet implementations.

Specification

We define the following levels in BIP32 path:

`m / purpose' / cosigner_index / change / address_index`

Apostrophe in the path indicates that BIP32 hardened derivation is used.

Each level has special meaning described in the chapters below.

Purpose

Purpose is a constant set to 45, following the BIP43 recommendation. It indicates that the subtree of this node is used according to this specification.

`m / 45' / *`

Hardened derivation is used at this level.

Cosigner Index

The index of the party creating a P2SH multisig address. The indices can be determined independently by lexicographically sorting the purpose public keys of each cosigner. Each cosigner creates addresses on its own branch, even though they have independent extended master public key, as explained in the "Address generation" section.

Note that the master public key is not shared amongst the cosigners. Only the hardened purpose extended public key is shared, and this is what is used to derive child extended public keys.

Software should only use indices corresponding to each of the N cosigners sequentially. For example, for a 2-of-3 HDPM wallet, having the following purpose public keys:

`03a473275a750a20b7b71ebeadfec83130c014da4b53f1c4743fcf342af6589a38`
`039863fb5f07b667d9b1ca68773c6e6cdbcac0088ffba9af46f6f6acd153d44463`
`03f76588e06c0d688617ef365d1e58a7f1aa84daa3801380b1e7f12acc9a69cd13`

it should use `m / 45 ' / 0 / *` for `039863fb5f07b667d9b1ca68773c6e6cdbcac0088ffba9af46f6f6acd153d4`
`m / 45 ' / 1 / *` for `03a473275a750a20b7b71ebeadfec83130c014da4b53f1c4743fcf342af6589a38`,
and `m / 45 ' / 2 / *` for `03f76588e06c0d688617ef365d1e58a7f1aa84daa3801380b1e7f12acc9a69cd13`,
as dictated by their lexicographical order.

Software needs to discover all used indexes when importing the seed from an external source. Such algorithm is described in "Address discovery" chapter.

Non-hardened derivation is used at this level.

Change

Constant 0 is used for external chain and constant 1 for internal chain (also known as change addresses). External chain is used for addresses that are meant to be visible outside of the wallet (e.g. for receiving payments). Internal chain is used for addresses which are not meant to be visible outside of the wallet and is used for return transaction change.

For example, if cosigner 2 wants to generate a change address, he would use `m / 45 ' / 2 / 1 / *`, and `m / 45 ' / 2 / 0 / *` for a receive address.

Non-hardened derivation is used at this level.

Address Index

Addresses are numbered from index 0 in sequentially increasing manner. This number is used as child index in BIP32 derivation.

Non-hardened derivation is used at this level.

HDPM Wallet High-level Description

Each party generates their own extended master keypair and shares the extended purpose' public key with the others, which is stored encrypted. Each party can generate any of the other's derived public keys, but only his own private keys.

Address Generation Procedure

When generating an address, each party can independently generate the N needed public keys. They do this by deriving the public key in each of the different trees, but using the same path. They can then generate the multisig script (by lexicographically sorting the public keys) and the corresponding p2sh address. In this way, each path corresponds to an address, but the public keys for that address come from different trees.

Receive address case Each cosigner generates addresses only on his own branch. One of the n cosigners wants to receive a payment, and the others are offline. He knows the last used index in his own branch, because only he generates addresses there. Thus, he can generate the public keys for all of the others using the next index, and calculate the needed script for the address.

Example: Cosigner #2 wants to receive a payment to the shared wallet. His last used index on his own branch is 4. Then, the path for the next receive address is `m/45'/2/0/5`. He uses this same path in all of the cosigners trees to generate a public key for each one, and from that he gets the new p2sh address.

Change address case Again, each cosigner generates addresses only on his own branch. One of the n cosigners wants to create an outgoing payment, for which he'll need a change address. He generates a new address using the same procedure as above, but using a separate index to track the used change addresses.

Example: Cosigner #5 wants to send a payment from the shared wallet, for which he'll need a change address. His last used change index on his own branch is 11. Then, the path for the next change address is `m/45'/5/1/12`. He uses this same path in all of the cosigners trees to generate a public key for each one, and from that he gets the new p2sh address.

Transaction creation and signing

When creating a transaction, first one of the parties creates a Transaction Proposal. This is a transaction that spends some output stored in any of the p2sh multisig addresses (corresponding to any of the copayers' branches). This proposal is sent to the other parties, who decide if they want to sign. If they approve the proposal, they can generate their needed private key for that specific address (using the same path that generated the public key in that address, but deriving the private key instead), and sign it. Once the proposal reaches m signatures, any cosigner can broadcast it to the network, becoming final. The specifics of how this proposal is structured, and the protocol to accept or reject it, belong to another BIP, in my opinion.

Address discovery

When the master seed is imported from an external source the software should start to discover the addresses in the following manner:

1. for each cosigner:
2. derive the cosigner's node (`m / 45' / cosigner_index`)
3. for both the external and internal chains on this node (`m / 45' / cosigner_index / 0` and `m / 45' / cosigner_index / 1`):
4. scan addresses of the chain; respect the gap limit described below

Please note that the algorithm uses the transaction history, not address balances, so even if the address has 0 coins, the program should continue with discovery. Opposite to BIP44, each cosigner branch needs to be checked, even if the earlier ones don't have transactions

Address gap limit

Address gap limit is currently set to 20. If the software hits 20 unused addresses (no transactions associated with that address) in a row, it expects there are no used addresses beyond this point and stops searching the address chain.

Wallet software should warn when user is trying to exceed the gap limit on an external chain by generating a new address.

Rationale

This structure provides a general way of doing HDPM wallets between m-of-n parties. Here are some explanations about the design decisions made.

The reason for using separate branches for each cosigner is we don't want two of them generating the same address and receiving simultaneous payments to it. The ideal case is that each address receives at most one payment, requested by the corresponding cosigner.

Examples

cosigner_index	change	address_index	path
first	receive	first	m / 45' / 0 / 0 / 0
first	receive	second	m / 45' / 0 / 0 / 1
first	receive	fifth	m / 45' / 0 / 0 / 4
first	change	first	m / 45' / 0 / 1 / 0
first	change	second	m / 45' / 0 / 1 / 1
second	receive	first	m / 45' / 1 / 0 / 0
third	change	tenth	m / 45' / 2 / 1 / 9

Compatible wallets

- Copay wallet (source)

Reference

- BIP32 - Hierarchical Deterministic Wallets
- BIP43 - Purpose Field for Deterministic Wallets
- Original mailing list discussion