Reducing the size of multisig backups

Leonardo Comandini – leonardocomandini@gmail.com
Satoshi Spritz

December, 2020

modern wallet backups = output descriptors

https://github.com/bitcoin/bitcoin/blob/master/doc/descriptors.md

Example

P2WSH(2of3) – public descriptor

Useful for **watch-only wallet** or **PSBT coordinator**

```
wsh(
  multi(
    2,
    XPUB_ALICE/0/*,
    XPUB_BOBBY/0/*,
    XPUB_CARLO/0/*,
  )
)
```

Example

P2WSH(2of3) – Alice's private descriptor

Useful for **signer**

```
wsh(
  multi(
    2,
    XPRV_ALICE/0/*,
    XPUB_BOBBY/0/*,
    XPUB_CARLO/0/*,
  )
)
```

A Multisig pitfall

Multisig signers **must** know the output descriptors to recognize change outputs and display the correct balance to the user before signing a transaction.

https://shiftcrypto.ch/blog/how-nearly-all-personal-hardware-wallet-multisig-setups-are-insecure/

Solution: at wallet *setup* signer persists the descriptor and sign only its outputs.

Wallet setup

- Signer (offline) generates/derives secret
- Signer exports *xpub* to Coordinator (online)
- Coordinator collects *xpubs* from each Signer, and use them to compose the *public descriptor*
- Coordinator provides the *public descriptor* to each Signer
- Signer creates the *private descriptor* from public one and persist it

https://github.com/RCasatta/firma

Descriptor size

xpub/xprv are long (111 chars)

multisig with several keys becomes very long: quite a lot of data that the signer has to transmit and persist

Pruning BIP32 keys

- depth (1 byte) **prunable!**
- fingerprint (4 bytes) **prunable!**
- child number (4 bytes) **prunable!**
- chain code (32 bytes)
- public/private key (33 bytes)

https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

Pruning BIP32 keys

- depth (1 byte) **prunable!**
- fingerprint (4 bytes) **prunable!**
- child number (4 bytes) **prunable!**
- chain code (32 bytes) **prunable!**
- public/private key (33 bytes)

https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki

BIP32 schematic

Let i be the child index. Let $(p, P = pG)$ and $(p_i, P_i = p_i G)$ be the parent and i-th child keypairs respectively. Let c and c_i be the corresponding chain codes. Let $h1, h2, h3, h4$ be hash functions so that the formulae below match the definitions given in BIP32.

```
p_i(p, c, i) = (i < 2^31)  p + h1(c, pG, i)
               (i >= 2^31) p + h2(c, p, i)

c_i(p, c, i) = (i < 2^31)  h3(c, pG, i)
               (i >= 2^31) h4(c, p, i)

P_i(P, c, i) = (i < 2^31)  P + h1(c, P, i)G
               (i >= 2^31) not possible

c_i(P, c, i) = (i < 2^31)  h3(c, P, i)
               (i >= 2^31) not possible
```

An alternative

Let h be an adequately strong hash function which converts its output to integer.

```
p_i(p, i) = (i < 2^31)  p + h(pG, i)
            (i >= 2^31) h(p, i)

P_i(P, i) = (i < 2^31)  P + h(P, i)G
            (i >= 2^31) not possible
```

https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2020-September/018211.html

A bold claim

I claim that this has the same properties as BIP32 (see emails for details)

(almost true)

Pros and Cons

If I am right...

Pros:

- shorter backups (32/33 vs 72 bytes comparison is a bit dishonest)
- user-friendly backup for child keys

Cons:

- backward incompatible

Question (and spritz) time

Cheers!