# Towards a Taproot Implementation

Tom Dudzik, Ryan Prinster
MAS.S62 Spring 2018

# Motivation: Privacy

Greg Maxwell:

Usually N-of-N (cooperative case), and other less used scripts (non-cooperative) cases. Can and should be represented as an OR between these two cases.

B/C anonymity set of people who use fancy signature scheme is small

P2PKH more private

# Motivation: Efficiency

Ideally, Pay-to-Script-Hash (P2SH) indistinguishable from Pay-to-Pubkey-Hash (P2PKH).

Make a special delegating CHECKSIG for this.

This is taproot.

# Taproot

Make Pay-to-Script-Hash (P2SH) and Pay-to-Pubkey-Hash (P2PKH) appear indistinguishable

$c = j + H(z \,||\, J)$

$C = J + H(z \,||\, J)G$

C - New Public Key
J - Old Public Key
z - Script
G - Generator Point for secp256k1 curve

# P2PKH and P2SH

Spend as a P2PKH

Sign with $c = j + H(z \,||\, J)$, where j is the old private key.

Spend as P2SH

Reveal script z, then sign/provide for the script.

Can verify that $C = J + H(z \,||\, J)G$

# Implementation

-elements sidechain
    -supports Schnorr
    -needs rebase to 0.16

-use Bitcoin Core v0.16
    -can use new bech32 address format
    -define new witness version → segwit
    - ex: 1 <pubkey hash>

# Implementation

New output type: TX_TAPROOT

-in script/standard.cpp

-add output type to `Solver()` function

scriptSig blank in segwit, use witness script

# Implementation

Wallet functionality

- -RPC: `getnewaddress` -> ‘`tweakaddress`’
- -modify `IsMine()` so wallet can spend

Invoke libsecp256k1 in key classes

# Mini Demo?

Proof of Concept