Optech Schnorr/Taproot Workshop

September 2019

Welcome!

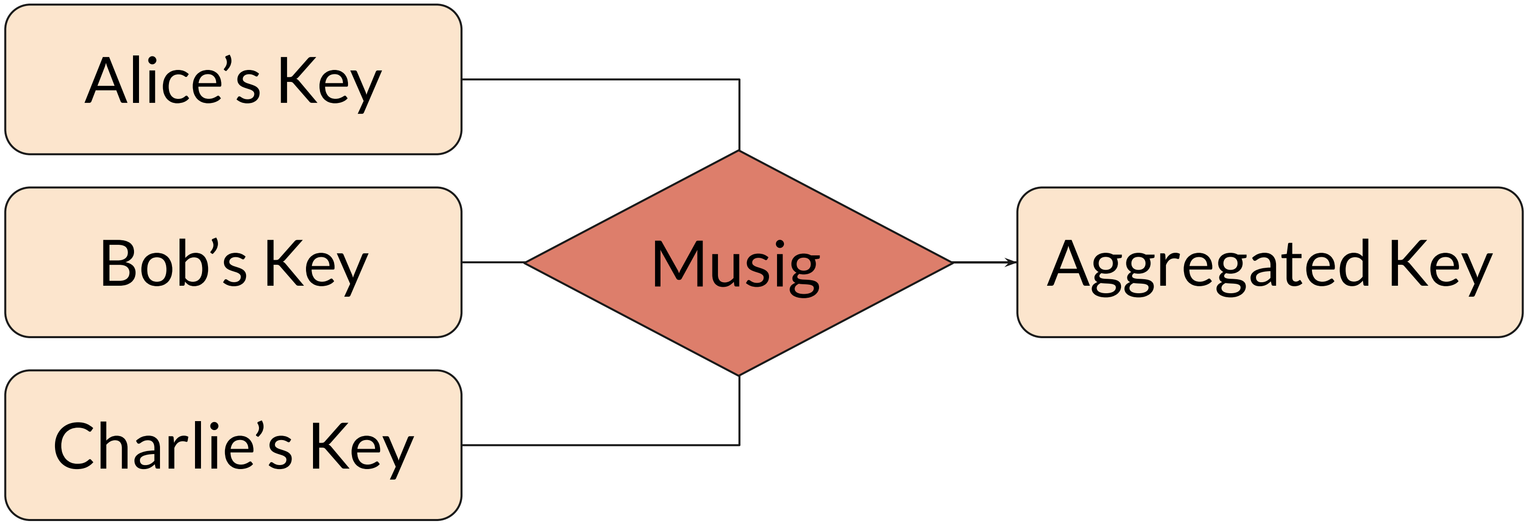
Why Schnorr/Taproot?

1	Scalability	• 30-75% savings on multisig • 2.5x faster block validation
2	Privacy and Fungibility	• All outputs and most spends indistinguishable
3	Functionality	• Very large k of n multisig • Larger scripts • Script innovation

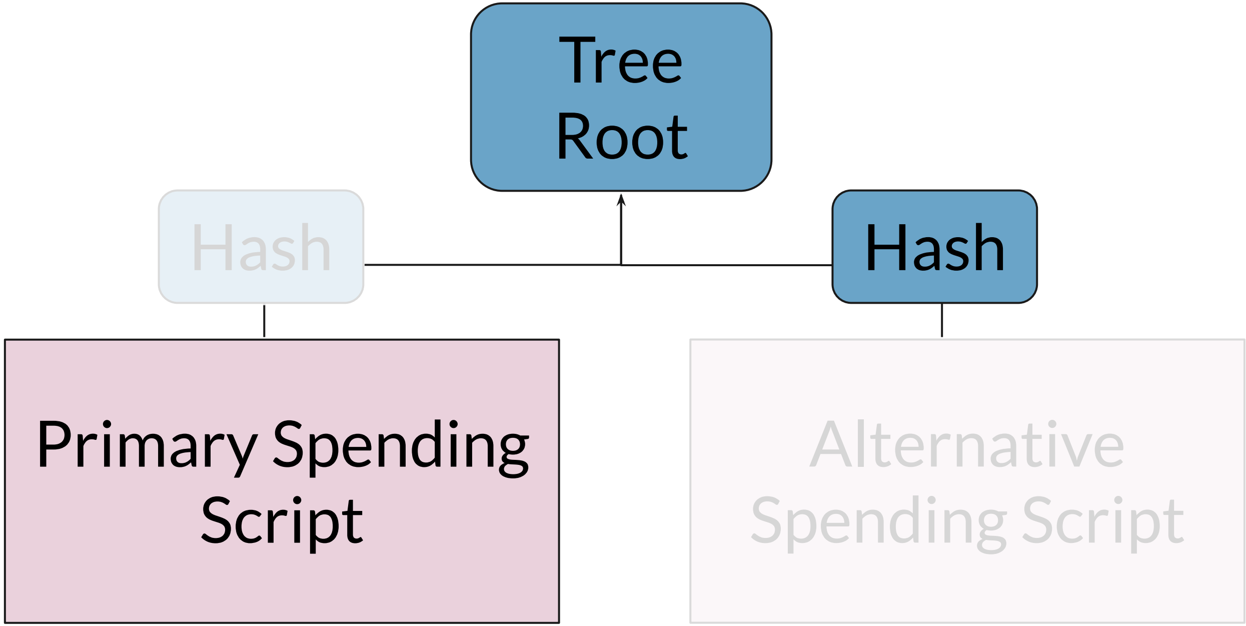
Schnorr signatures

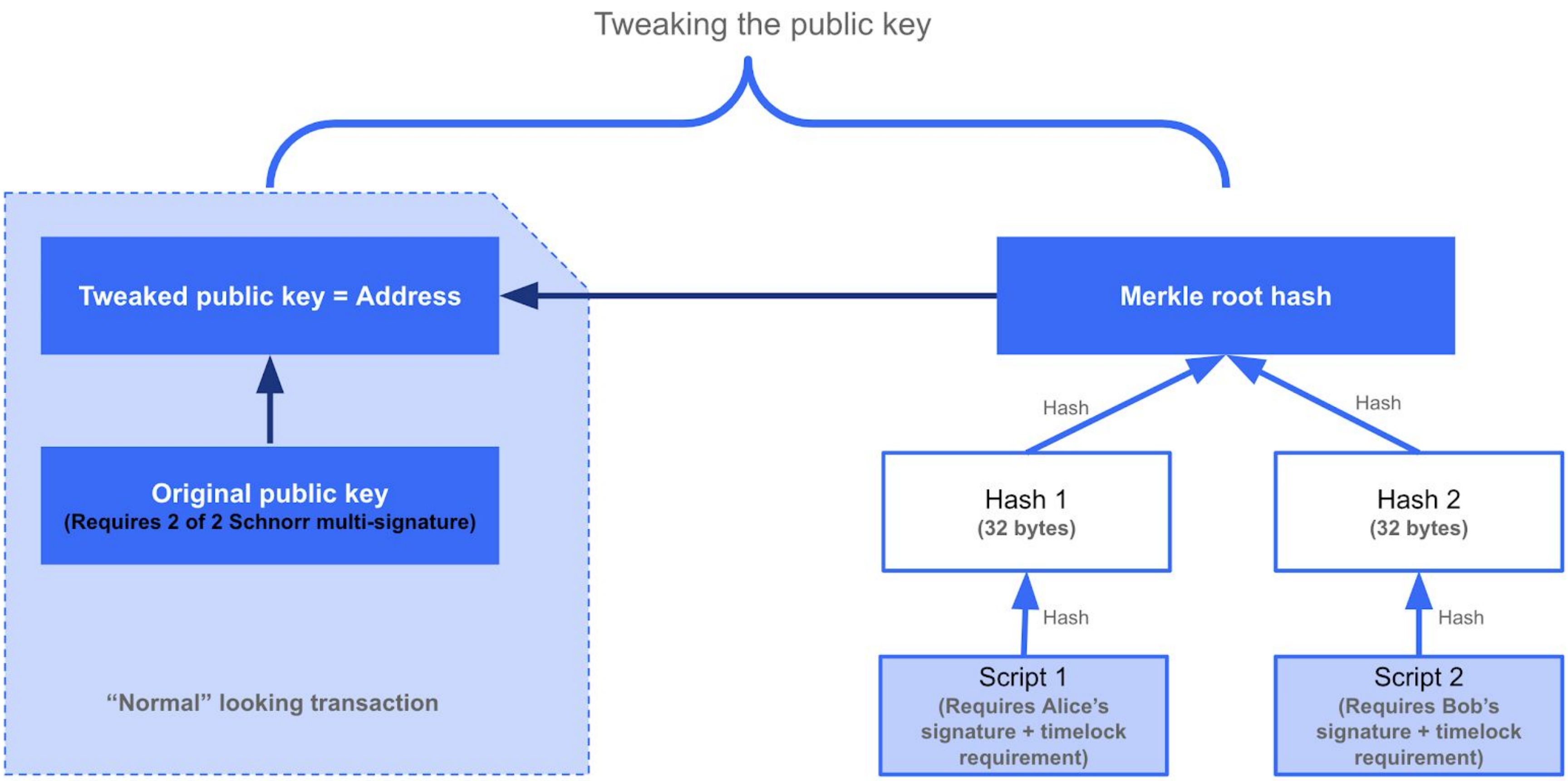
1. Better in every way than ECDSA
2. 11% smaller than existing signatures
3. Compatible with existing private keys
4. Same security assumption...with a theoretical proof
5. Verification algorithm is linear

Schnorr enables key and signature aggregation



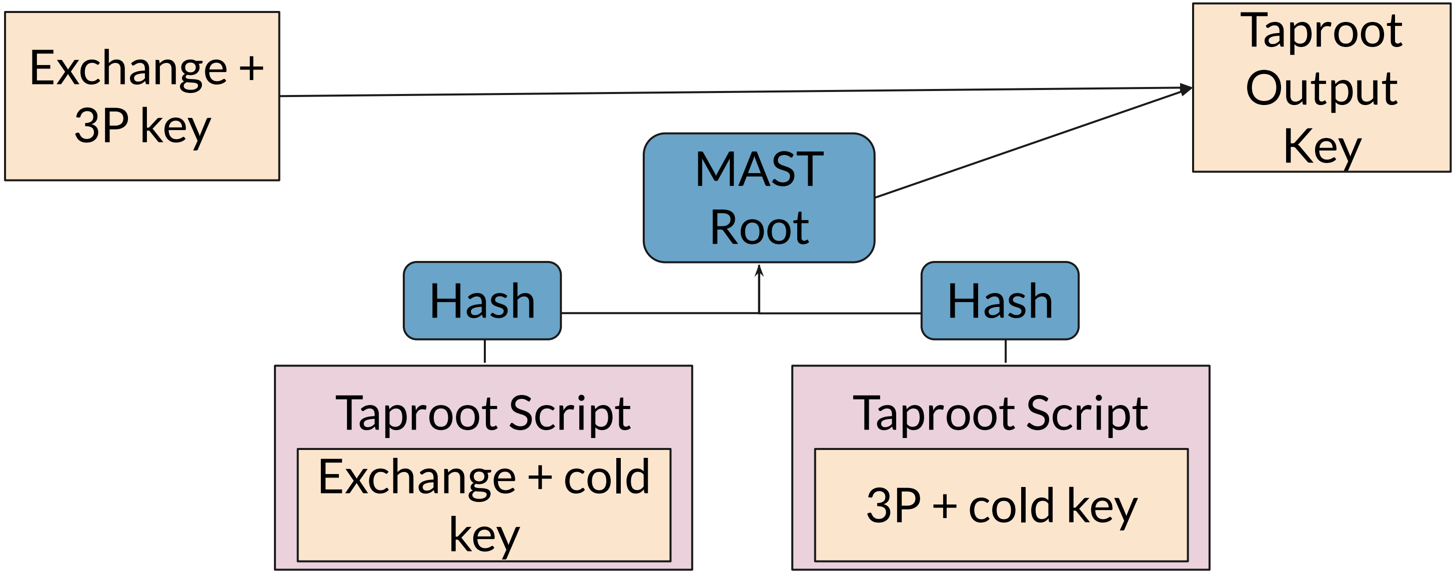
Script trees




Tweaking the public key
Tweaked public key = Address
Original public key
(Requires 2 of 2 Schnorr multi-signature)
"Normal" looking transaction
Merkle root hash
Hash
Hash
Hash 1
(32 bytes)
Hash 2
(32 bytes)
Hash
Hash
Script 1
(Requires Alice's signature + timelock requirement)
Script 2
(Requires Bob's signature + timelock requirement)

h/t BitMEX

Exchange 2-of-3 using Musig keytrees



Why Optech?

Bitcoin Optech helps Bitcoin users and businesses integrate scaling technologies.

We provide workshops, documentation, weekly newsletters, original research, case studies and announcements, analysis of Bitcoin software and services, and help facilitate improved relations between businesses and the open source community.

Why this workshop?

- Help share current thinking on schnorr/taproot
- Give engineers a chance to play with the technology
- Involve engineers in the feedback process

WARNING!

The schnorr/taproot proposal is a proposal

- **Details will change**
- **There is no roadmap**
- **The workshop code is for educational purposes only!**

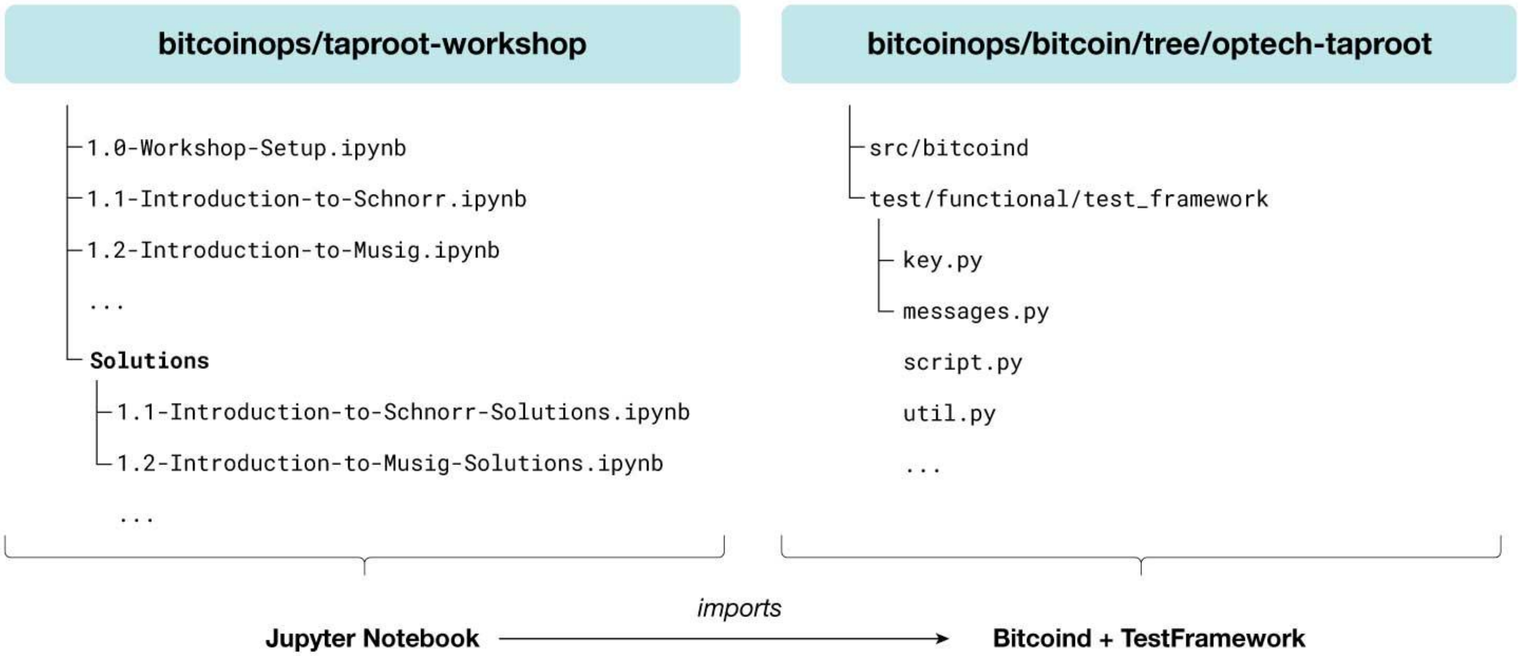
Chapter 0.1

Toolchain Setup

Did you do your homework?

- Optech Bitcoin Repository: https://github.com/bitcoinops/bitcoin/releases/tag/Taproot_V0.1.4
- Workshop Repository: https://github.com/bitcoinops/taproot-workshop
- Pull latest taproot-workshop
- $ jupyter-notebook
  - 0.1-test-notebook

Optech Schnorr & Taproot Workshop Repositories



Chapter 0.2

Elliptic Curve Math

Scalars (numbers)

- Regular arithmetic but modulo the group order (SECP256K1_ORDER)
- ***a·b mod n***
- Division done using modular inverse (i.e. Fermat's little theorem: $a^p = a$)
- Numbers can go from 0 to (group order - 1). eg:
  - (15 + 9) mod 21 = 24 mod 21 = 3
  - (-3) mod 21 = (21-3) = 18

Points on the elliptic curve

- Point = (x, y)
- G is the generator point for our group. (i.e. ***P = dG***)
- The curve points form an abelian group:
    - **Closure**: if ***A*** is a point and ***B*** is a point than ***A + B*** is a point.
    - **Associativity**: ***(A + B) + C = A + (B + C)***
    - **Identity element**: ***A + ∞ = ∞ + A = A***
    - **Inverse**: For every point ***A*** there exist another point ***B*** such that ***A + B = 0***
    - **Commutativity**: ***A + B = B + A***
- Scalar operations:
    - scalar * point: ***sG*** = {***G + G + G + G …*** s times}
    - point by point division isn't feasible and requires solving **discrete log**

Chapter 1.1

Schnorr

Schnorr

Signing:

$$e = H(R||P||m) \qquad Sig(s, kG)$$

$$\boldsymbol{s = k + ed} \qquad Sig(s, R)$$

Verifying:

$$\boldsymbol{sG = kG + edG}$$

Glossary

m - message

e = H(R||P||m)

G - generator point

d - private key

point - scalar * G = (x,y)

P - public key (***P = dG***)

k - random nonce

R - nonce point (***R = kG***)

X only R Points/Public Keys

- Secp256k1: $y^2 = x^3 + 7$
- Solve for y: $y = \pm\sqrt{x^3 + 7}$
- ***(-a) mod n = n - a***
- Even/odd only (odd-even=odd; odd-odd=even)
- Lower/higher half
- Quadratic residue

Chapter 1.2

MuSig

Naive key aggregation

$$P_1 = d_1 G, \quad P_2 = d_2 G$$

$$s_1 = k_1 + ed_1, \quad s_2 = k_2 + ed_2$$

$$s_1 + s_2 = (k_1 + k_2) + e(d_1 + d_2)$$

$$s' = k' + ed'$$

$$P' = (d_1 + d_2)G$$

Glossary

m - message

e = H(R||P||m)

G - generator point

d - private key

point - scalar * G = (x,y)

P - public key (***P = dG***)

k - random nonce

R - nonce point (***R = kG***)

Key cancellation (rogue key) attack

$$P_1 = d_1 G, \quad P_2 = d_2 G$$

$$P_2' = P_2 - P_1$$

$$P' = P_1 + (P_2 - P_1)$$

Glossary

m - message

e = H(R||P||m)

G - generator point

d - private key

point - scalar * G = (x,y)

P - public key (***P = dG***)

k - random nonce

R - nonce point (***R = kG***)

Musig coefficients

$$P_1 = d_1 G, \quad P_2 = d_2 G$$

$$c_i = H(P_1 || P_2 || P_i)$$

$$d'_1 = c_1 d_1, \quad d'_2 = c_2 d_2$$

$$P' = c_1 P_1 + c_2 P_2$$

Glossary

m - message

e = H(R||P||m)

G - generator point

d - private key

point - scalar * G = (x,y)

P - public key (***P = dG***)

k - random nonce

R - nonce point (***R = kG***)

Nonce commitments

$$R_1 = k_1 G, \qquad R_2 = k_2 G$$
$$Com_1 = H(R_1), \quad Com_2 = H(R_2)$$
$$R' = R_1 + R_2$$
$$e = H(R' || P' || m)$$

Glossary

m - message

e = H(R||P||m)

G - generator point

d - private key

point - scalar * G = (x,y)

P - public key (***P = dG***)

k - random nonce

R - nonce point (***R = kG***)

Chapter 2.1 - 2.4

Taproot

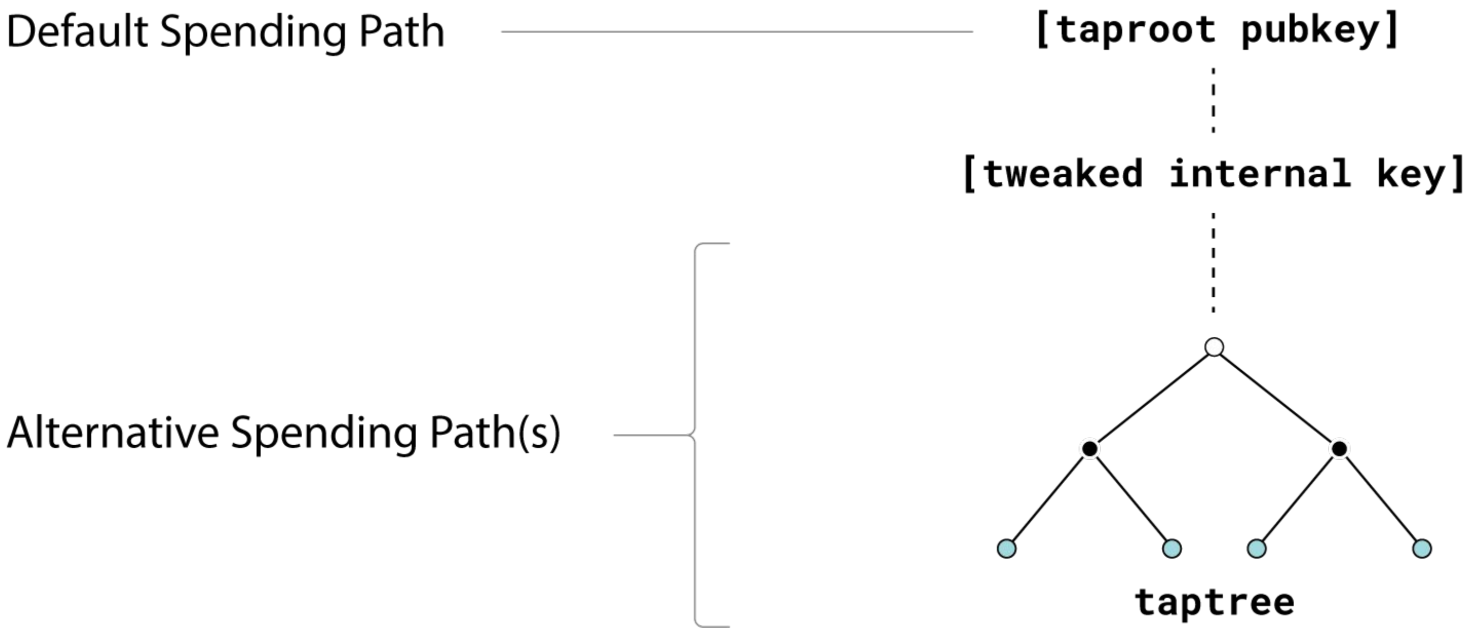
Default & Alternative Spending Paths

- **Default Spending Path**
  - Single or multi-party public keys (indistinguishable)

- **Alternative Spending Path(s)**
  - Single or multiple “hidden” alternative scripts.
  - Only the script of the spent path is revealed when spent.

Taproot: Multi-party contract

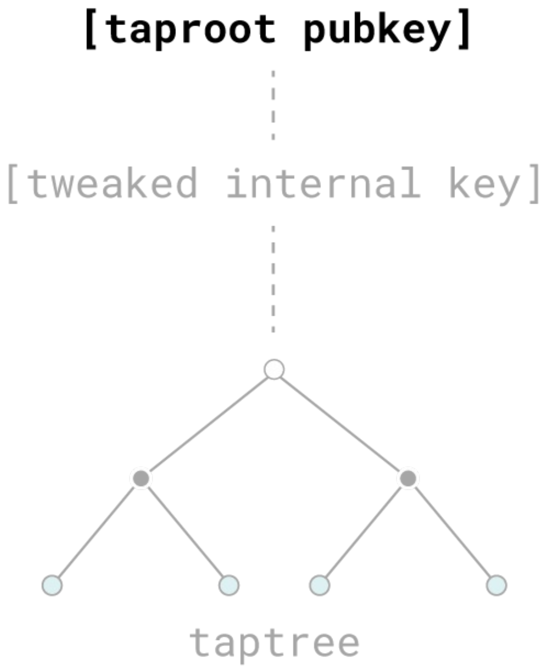
- **Default Spending Path**
  - Aggregated pubkey/signature.
  - Default spending path hides multi-party contract.
- **Alternative Spending Path(s)**
  - In aggregate, enforce the multi-party contract.
  - script_0 **OR** script_1 **OR** script_2 ...


Default Spending Path
[taproot pubkey]
[tweaked internal key]
Alternative Spending Path(s)
taptree

Chapter 2.1

Segwit Version 1

Segwit version 1



Segwit version 1

- Output script:
  - **Script:** **[01] [33B public key]**
  - *Has recently been reduced to 32B public key in bip-schnorr.*
  - *This workshop has been built with the previous 33B public key format.*

- Satisfying Witness:
  - **Key path:** **[64B BIP-schnorr signature]**
  - **Script path:** **[initial stack] [tapscript] [controlblock]**

P2PK vs P2PKH

- P2PK vs P2PKH:
  - V1 Script: [01] [33B public key]
  - **V1 Witness: [64/65B signature]**
  - V0 Script: [00] [20B pubkey hash]
  - **V0 Witness: [DER signature(ecdsa)] [public key]**
- V1 program witness: single key, MuSig, ...
- Disadvantages of P2PKH:
  - **Cost: pubkey + pubkey hash**

Taproot Sighash Flags

- **Taproot retains legacy sighash flag semantics**
  - ALL, NONE, SINGLE, ANY
  - New implied ALL sighash flag (0x00)

Taproot: Schnorr signature encoding

- **x(R), s**
  - x(R): 32B
  - s: 32B
  - *Sighashflag*: - (SIGHASH_ALL is implied) (0x00)

- **x(R), s, sighashflag**
  - x(R): 32B
  - s: 32B
  - *SIGHASH flag*: 1B (All, None, Single, Any) 0x01, 0x02, 0x03, 0x8...

v1: schnorr signature hash

- **Control**
  - Always — **epoch(0) | sighash**
- **Transaction**
  - Always — **version | locktime**
  - If !any — **prevout(s) | input amount(s) | sequence(s)**
  - If !none or !single — **outputs**
- **Input**
  - Always — **spend_type | scriptPubKey**
  - If any — **outpoint | input amount | sequence**
  - If !any — **input index**
- **Output(s)**
  - If single — **sha256(CTxOut)**

v1: schnorr signature hash

Control	
Always	**epoch(0) | sighash**
Transaction	
Always	**version | locktime**
If !any	**prevout(s) | input amount(s) | sequence(s)**
If !none or !single	**outputs**
Input	
Always	**spend_type | scriptPubKey**
If any	**outpoint | input amount | sequence**
If !any	**input index**
Output(s)	
If single	**sha256(CTxOut)**

Reusable Midstate

Chapter 2.2

Taptweak

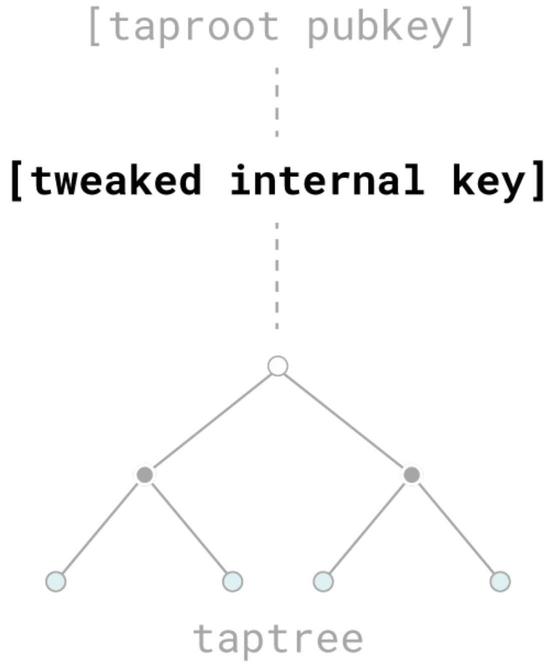
Taptweak

- **Any data can be committed to a public key tweak.**
    - Public key remains spendable.
        - Owner of private key can spend with knowledge of tweak.
        - Signing with the tweaked public key does not reveal tweak.
        - The owner of the private key can later reveal the commitment without revealing the private key.

TapTweak

- **A TapTweak is a tweak to an internal public key**
  - Default spending path: Tweak is not revealed.
  - Alternative spending paths:
    - Tweak & script branch are revealed.
    - Script branch is executed during validation.

Taptweak



Committing data to a pubkey tweak

- v1 witness program : 33B pubkey **Q**

  Q = P + H(P|c)G where **P** is the *internal key* and **c** is the *commitment*.

- Spending witness: 64B signature **(x(R), s)**

  The private key is tweaked with **H(P|c)** before signing

Public Key Tweak

$$Q = P + cG$$

✗

$$Q = x'G + c'G$$

Solve for x' *Modify c'*

Commitment Scheme

$$Q = P + H(P|c)G$$

✓

$$Q \mathrel{!=} x'G + H(x'G|c')G$$

Cannot solve for x *Modify c'*

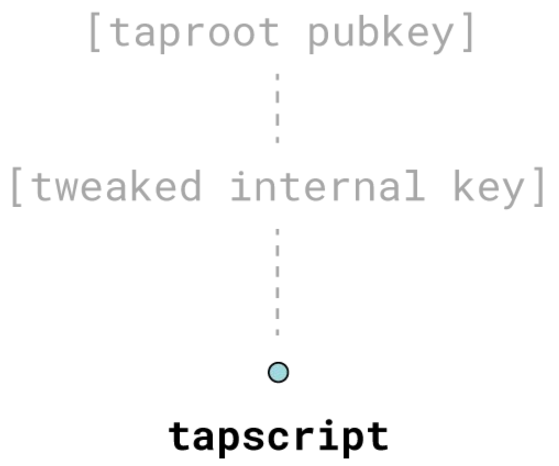
Chapter 2.3

Tapscript

Tapscript

- **TapScript is upgraded Bitcoin script.**
  - Optimized for Schnorr.
  - Allows for future TapScript versions.
  - TapScripts are committed to TapTweaks.

Tapscript



Tapscript vs. Bitcoin script

- Signature opcodes: **Perform verification of bip-schnorr signatures**
- Multisig opcodes: **Removed**
- Checksigadd opcodes: **Replace multisig opcodes. Enable signature batch verification.**
- Versioning:
  - TapLeaf version: **0xc0**
  - Upgradable opcodes: **80, 98, 126-129, 131-134, 137-138, 141-142, 149-153, 187-254**
  - Difference to NOP: **Immediate success and termination of script execution.**

Multisig with Checksigadd

- Output Script
    - **pk0**
    - **checksig**
    - pk1
    - checksigadd
    - pk2
    - checksigadd
    - 3
    - equal
- Initial Stack
    - **sig0**
    - sig1
    - sig2

Multisig with Checksigadd

- **Output Script**
  - pk1
  - checksigadd
  - pk2
  - checksigadd
  - 3
  - equal
- **Initial Stack**
  - **1**
  - sig1
  - sig2

Multisig with Checksigadd

- **Output Script**
  - 3
  - equal
- **Initial Stack**
  - 3

Tapscript Descriptors (I/II)

Pay-to-pubkey:	Satisfying Witness:
ts(pk(key))	[signature]
ts(pkhash(key, digest))	[preimage] [signature]
ts(pkolder(key, delay))	[signature] (nSequence > delay)
ts(pkhasholder(key, digest ,delay))	[preimage] [signature] (nSequence > delay)

Tapscript Descriptors (II/II)

Pay-to-pubkey:	Satisfying Witness:
ts(csa(k, keys..))	**[k signatures]**
ts(csahash(k, keys, digest))	**[hash] [k signatures]**
ts(csaolder(k, keys, delay))	**[k signatures]** (nSequence > delay)
ts(csahasholder(k, keys, digest, delay))	**[hash] [k signatures]** (nSequence > delay)

Committing a single Tapscript to a Taptweak

- **Taptweak t**
  - **Q = P + tG**
  - t = TaggedHash(“TapTweak”, P, tapleaf)
  - TapLeaf = TaggedHash(“TapLeaf”, ver, size, script)

- **TaggedHash**
  - **TaggedHash(data) =** sha256(**sha256("Tag") + sha256("Tag") + data**)
  - Collision resistance
  - 64B re-usable midstate

Taproot: Spending a single Tapscript

Spending Witness:

- [Satisfying witness elements for Tapscript]
- [Tapscript]
- [Internal Key]

Unspendable script path (WIP)

- **Problem: Hidden script path t'**
  - $Q = P1 + P2 = P1 + \mathbf{P2' + H(P1+P2'|t')}$
- **Solution: Default unspendable script path t**
  - $Q = P1 + P2 + H(P1+P2|\mathbf{t})G$
  - Not possible:
    - Hidden t': $P2 = P2' + H(Pagg|\mathbf{t'})$
    - Default t: $Q = P1 + P2 + H(P1+P2|\mathbf{t})$

https://github.com/sipa/bips/pull/49

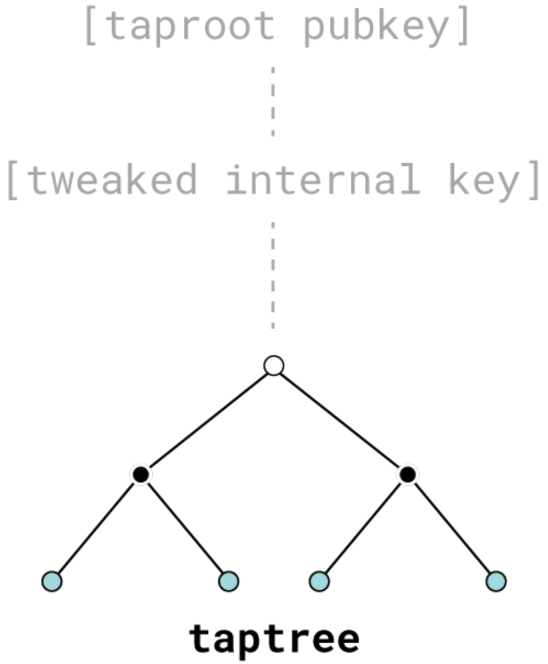
Chapter 2.4

Taptree

Taptree

- **A Taptree commits multiple Tapscripts to a Taptweak**
  - Binary merkle tree commitment structure.
  - A TapTree does not have to be balanced.
    - Allows for Tapscript specific spending cost optimizations.

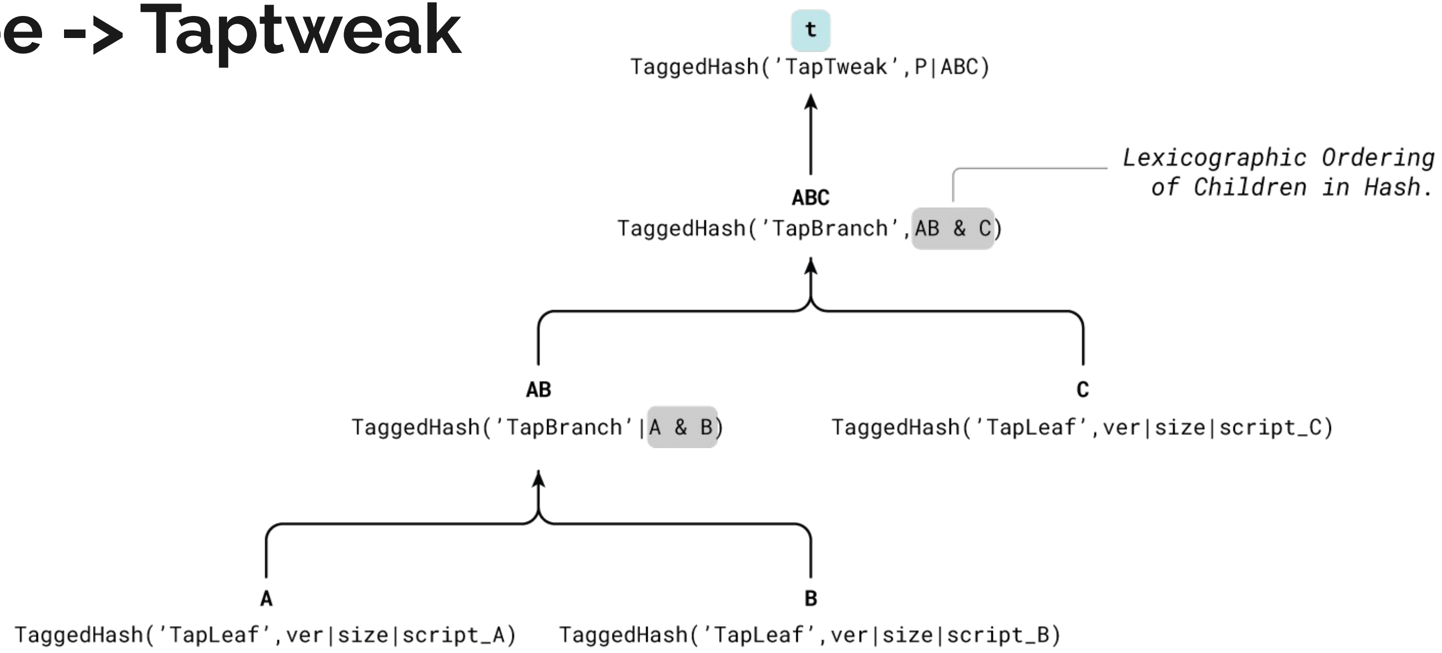
Taptree



Committing Tapscripts to a Taptweak

- TapTweak t
  - **Q = P + tG**
  - t = TaggedHash(“TapTweak”, P, Tapbranch)
  - Tapbranch is the root node of the TapTree

Taptree -> Taptweak



Committing Tapscripts to a Taptweak

- TapTweak t
  - **Q = P + tG**
  - t = TaggedHash(“TapTweak”, P, Root)
  - Root is root node of TapTree
- TapTree
  - Binary tree
  - Siblings ordered lexicographically
  - Internal nodes are tagged “TapBranch”
  - Leaf nodes are tagged “TapLeaf”
  - TapScripts are committed to leaf nodes

Protects against preimage attacks.

Taproot Descriptors

- Taproot Descriptor: **tp(P, [taptree descriptor])**
    - *P = Internal Pubkey*
    - *Tweak is implied from taptree descriptor*
- Taptree Descriptor: **[tapscript0, [tapscript1, tapscript 2]]**
    - *TapBranch represented by* [child_node0, child_node1]
    - *TapBranch are composable* [tapscript0, [tapscript1, tapscript2]]

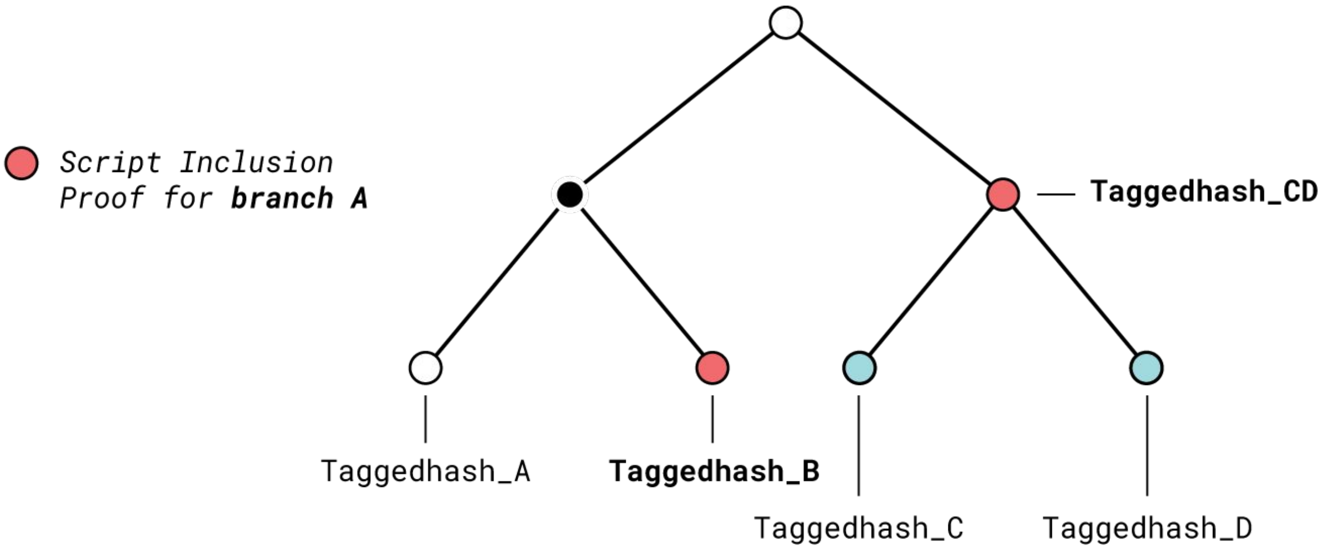
Taproot: spending a script path

- Taproot descriptor: tp(P, [[**script_A**, script_B], [script_C, script_D]])
- Satisfying witness for script 1: **[Satisfying witness elements for script_A]**
- **[script_A]**
- **[controlblock]**

  └── *Internal Key*

  └── *Inclusion proof for script A*

Tapscript inclusion proof



Chapter 3.1

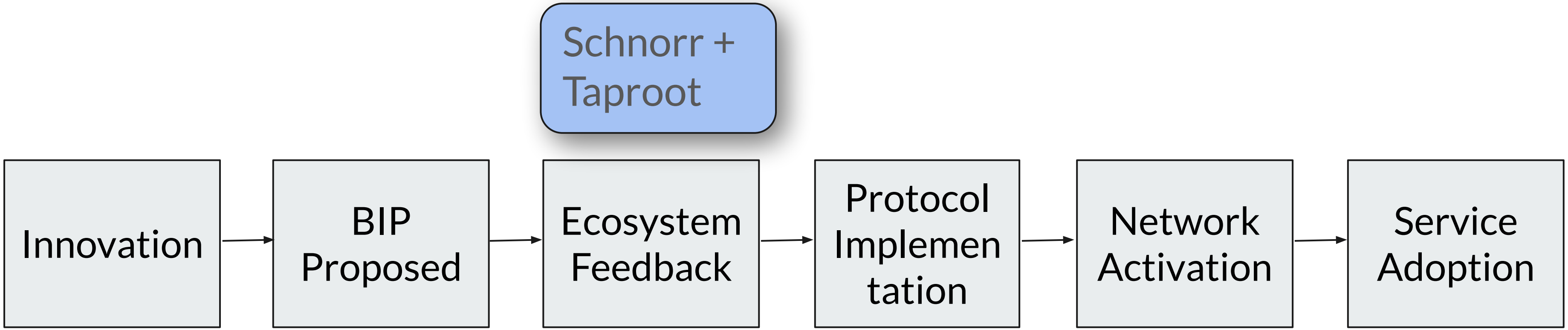
Case Study

Discussion

Where to find out more

- Draft BIPs: https://github.com/sipa/bips/tree/bip-schnorr
- Reference implementation: https://github.com/sipa/bitcoin/tree/taproot
- Mailing list: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/

Bitcoin Consensus Upgrade Lifecycle



Mailing list

[bitcoin-dev] Taproot proposal

Pieter Wuille pieter.wuille at gmail.com
Mon May 6 17:57:57 UTC 2019

- Previous message: [bitcoin-dev] Bitcoin Knots 0.18.0.knots20190502 released
- Next message: [bitcoin-dev] Taproot proposal
- **Messages sorted by:** [date] [thread] [subject] [author]

```
Hello everyone,

Here are two BIP drafts that specify a proposal for a Taproot
softfork. A number of ideas are included:

* Taproot to make all outputs and cooperative spends indistinguishable
from eachother.
* Merkle branches to hide the unexecuted branches in scripts.
* Schnorr signatures enable wallet software to use key
aggregation/thresholds within one input.
* Improvements to the signature hashing algorithm (including signing
all input amounts).
* Replacing OP_CHECKMULTISIG(VERIFY) with OP_CHECKSIGADD, to support
batch validation.
* Tagged hashing for domain separation (avoiding issues like
CVE-2012-2459 in Merkle trees).
* Extensibility through leaf versions, OP_SUCCESS opcodes, and
upgradable pubkey types.
```

- [bitcoin-dev] Taproot proposal *Pieter Wuille*
    - [bitcoin-dev] Taproot proposal *Luke Dashjr*
        - [bitcoin-dev] Taproot proposal *Sjors Provoost*
            - [bitcoin-dev] Taproot proposal *ZmnSCPxj*
            - [bitcoin-dev] Taproot proposal *ZmnSCPxj*
            - [bitcoin-dev] Taproot proposal *Pieter Wuille*
            - [bitcoin-dev] Taproot proposal *ZmnSCPxj*
        - [bitcoin-dev] Taproot proposal *ZmnSCPxj*
            - [bitcoin-dev] Taproot proposal *Johnson Lau*
            - [bitcoin-dev] Taproot proposal *ZmnSCPxj*
        - [bitcoin-dev] Taproot proposal *Anthony Towns*
        - [bitcoin-dev] Taproot proposal *Luke Dashjr*
    - [bitcoin-dev] Taproot proposal *Russell O'Connor*
        - [bitcoin-dev] Taproot proposal *Pieter Wuille*
            - [bitcoin-dev] Taproot proposal *Russell O'Connor*
    - [bitcoin-dev] Taproot proposal *John Newbery*

Questions?

Why this workshop?

- Help share current thinking on schnorr/taproot
- Give engineers a chance to play with the technology
- Involve engineers in the feedback process

Contributions welcome!

https://github.com/bitcoinops/taproot-workshop