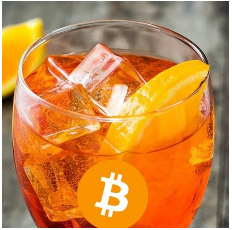
Wen Taproot?!

What do we need to get full advantage of the Taproot-enabled features?

Leonardo Comandini – leonardocomandini@gmail.com
Satoshi Spritz – Conio

February, 2023



About me

- PoliMi, math engineering, quantitative finance
- Eternity Wall, OpenTimestamps
- Blockstream, Green Wallet, Liquid Network

Presentation Structure

- What is Taproot
- Schnorr
- MAST
- Elliptic Curve Commitments (Taproot)
- Why Taproot
- Taproot Timeline
- State of the Art
- Wen (spend from) Taproot?!
- Taproot Cosigner
- Demo
- Conclusions

What is Taproot

A Bitcoin Soft-Fork that enabled:

- Schnorr
- MAST
- Elliptic Curve Commitments (Taproot)

(and more...)

Schnorr Signature Algorithm

$$sig = (R, s) \tag{1}$$

where

$$s = r + h(R, P, m)x \tag{2}$$

Verify if

$$sG == R + h(P, R, m)P \tag{3}$$

- Linearity
- Key/signature aggregation (MuSig, MuSig2, FROST, ROAST, ...)
- Security proof
- Adaptor signatures (DLC)
- and more...

MAST (Merklized Abstract Syntax Trees)

- Unbalanced Merkle tree
- Commit to an arbitrary set of scripts
- To prove the commitment, only a single script can be revealed

Benefits:

- Efficiency
- Privacy

Elliptic Curve Commitments (Taproot)

An elliptic curve point (a public key) can commit to some arbitrary data while still be used for its original purpose (e.g. signing).

$$Q = P + h(P||c)G \tag{4}$$

$$y = x + h(P||c) \tag{5}$$

- Key tweaking
- Pay-to-contract, tweak an output public key
- Sign-to-contract, tweak the nonce in the signature

Taproot

Taproot = schnorr + MAST + elliptic curve commitment

- $\{P_i\}_{i=1..m}$ set of keys
- $\{s_i\}_{i=1..n}$ set of scripts (spending conditions)
- $P = AggKey(\{P_i\}_{i=1..m})$ internal key
- $c = MAST(\{s_i\}_{i=1..n})$ Merkle root committing to the set of scripts
- $Q = P + h(P||c)G$ tweaked key

Ways of spending:

- **Key Path Spend**: produce a Schnorr signature for Q
- **Script Path Spend**: choose a script committed to c, prove its commitment and satisfy the script conditions

Why Taproot

Efficiency and **Privacy**

- Can commit to complex spending conditions with no extra cost (bandwidth and fee)
- Do not need to reveal those spending conditions if spending using another path
- If spending with key path (cheaper), single sig, multi sig and wallets with complex spending conditions all look the same, **larger anonimity set**

Taproot Timeline

- Schnorr signature paper, 1989
- Schnorr signature patent expired, Feb 2008
- MAST discussed, 2013 (BIP 114, 116, 117, 341)
- Taproot Mailing list announce, Jan 2018
- MuSig, 2018 - fixed 2019
- MuSig2, 2020
- Taproot activation, Nov 2021 (BIP 340, 341, 342, 343)

State of the Art

- Taproot support in Bitcoin Core, rust-bitcoin, BDK etc
- MuSig (n-of-n, 3 rounds)
- MuSig-DN (n-of-n, 2 rounds, ZK proofs)
- Musig2 (n-of-n, 2 rounds)
- FROST (t-of-n)
- ROAST (t-of-n, robust and asynchronous)
- No support for aggregated signatures in production libraries yet

Wen (spend from) Taproot?!

- Multisig wallets have incentives to use taproot (less fees)
- Once multisig wallets use taproot, single sig wallets can use taproot and join the anonimity set of multisig users
- Aggregated signatures are easy to verify but complex to produce
- Parties need to run a protocol to produce such signatures in which they mutually distrust

So let's start with a simple yet useful case.

Taproot Cosigner

- 2of2 between a Server and a Client
- Server is always online and ∼always cosigns
- Client can choose the script path spending conditions

E.g.

- $P = AggKey(P_c, P_s)$
- $s = and(P_c, after(144 * 60 blocks))$
- $c = MAST(s)$
- $Q = P + h(P|c)G$

Demo

- Start the Cosigning Server using `taproot-cosigner-fun` (rocket + secp256kfun + BDK)
- Get the Server xpub
- Create a Taproot descriptor with an aggregated key between the Server and the Client
- Generate an address and send some funds to it
- Create a transaction spending those funds
- Ask the Server to cosign the transaction
- Partially signs the transaction with the Client key
- Client aggregates the signatures
- Finalize and broadcast the transaction

Conclusions

- Taproot makes privacy more convenient (**!**)
- Multisig wallets should lead in Taproot adoption
- Signature aggregation protocols are complex to put into production

Resources

- taproot-cosigner-fun,
  https://github.com/LeoComandini/taproot-cosigner-fun
- secp256kfun, which includes a Rust implementation of MuSig2 and Frost,
  https://github.com/LLFourn/secp256kfun
- MuSig2, https://eprint.iacr.org/2020/1261.pdf