Bitcoin:
Programming the Future of Money

Topics in Computer Science - ITCS 4010/5010, Spring 2025

Dr. Christian Kümmerle

Lecture 20

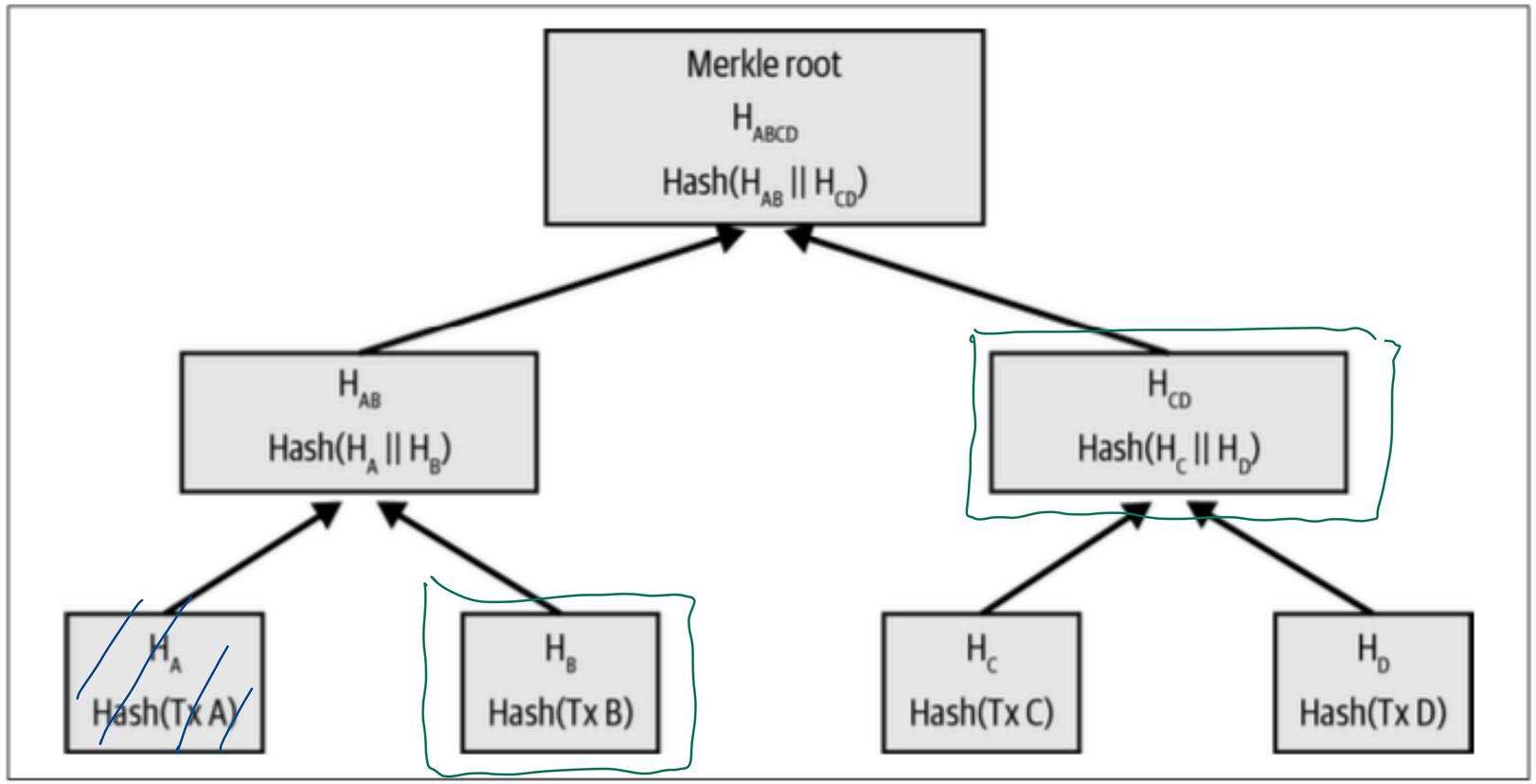
Taproot

Merkle Trees

Applications of Merkle Trees in the Bitcoin protocol

- Efficient Transaction Verification
- Data integrity
- Block header compactness
- Enables Merklized Alternative Script Trees (in Taproot)

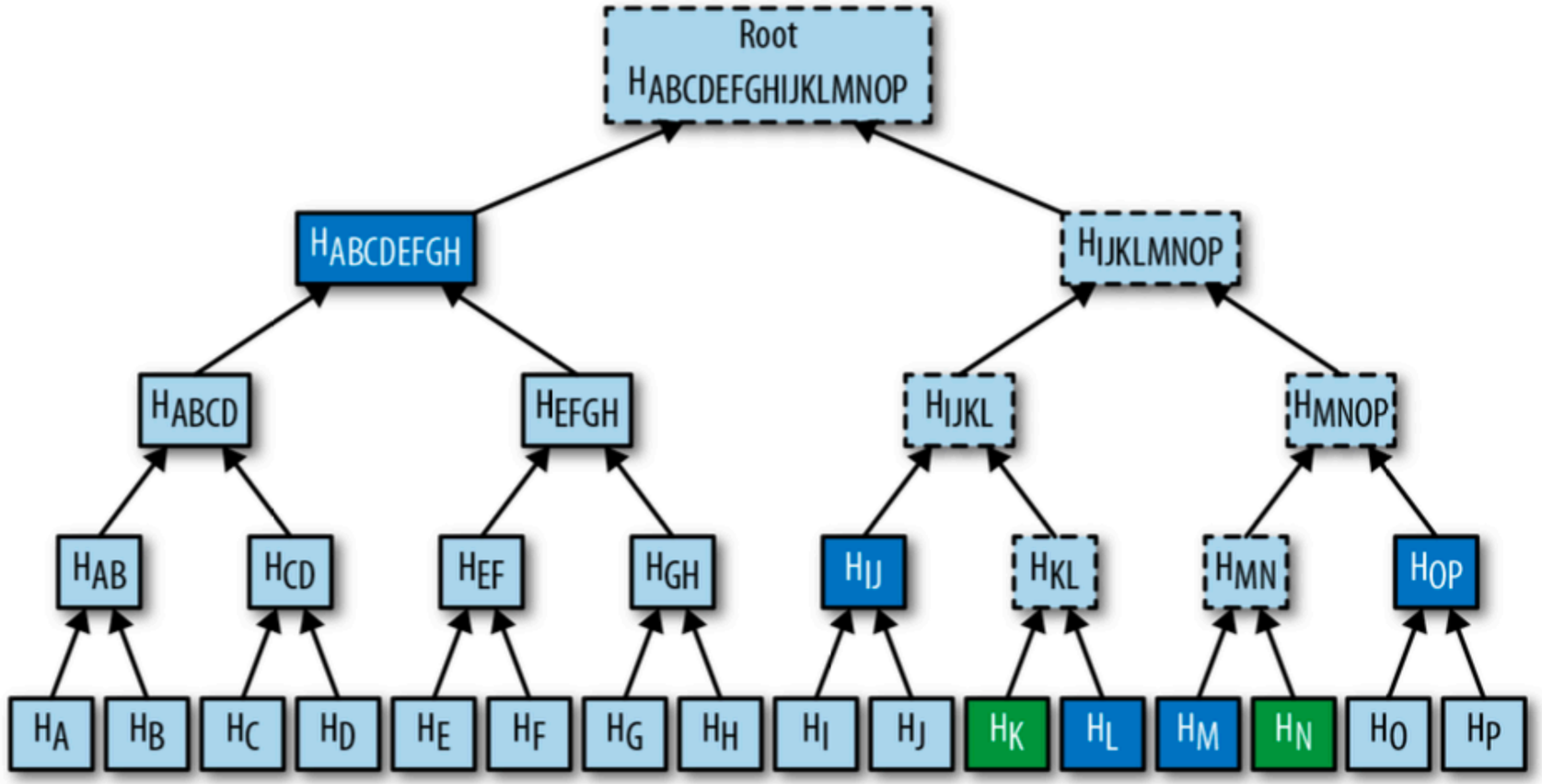
RECAP: MERKLE TREES FOR CONSTRUCTING MERKLE ROOT IN BITCOIN BLOCKS



- Verify membership of transaction in block in O(log n) time by providing “Merkle proof” of size O(log n), where n number of transactions in block.
- **(Not relevant for Bitcoin protocol):** Sorted variant can prove non-membership of data in tree in O(log n)

MERKLE TREES AND MERKLE PROOFS

Setup:



- H_K and H_N are hashes of two data pieces K and N (e.g., representing two transactions)
- By providing the values in dark blue, the fact that K and N are indeed part of Merkle tree summarized by Merkle root $H_{ABCDEFGHHIJKLMNOP}$ can be proven (**"Merkle proof"**)

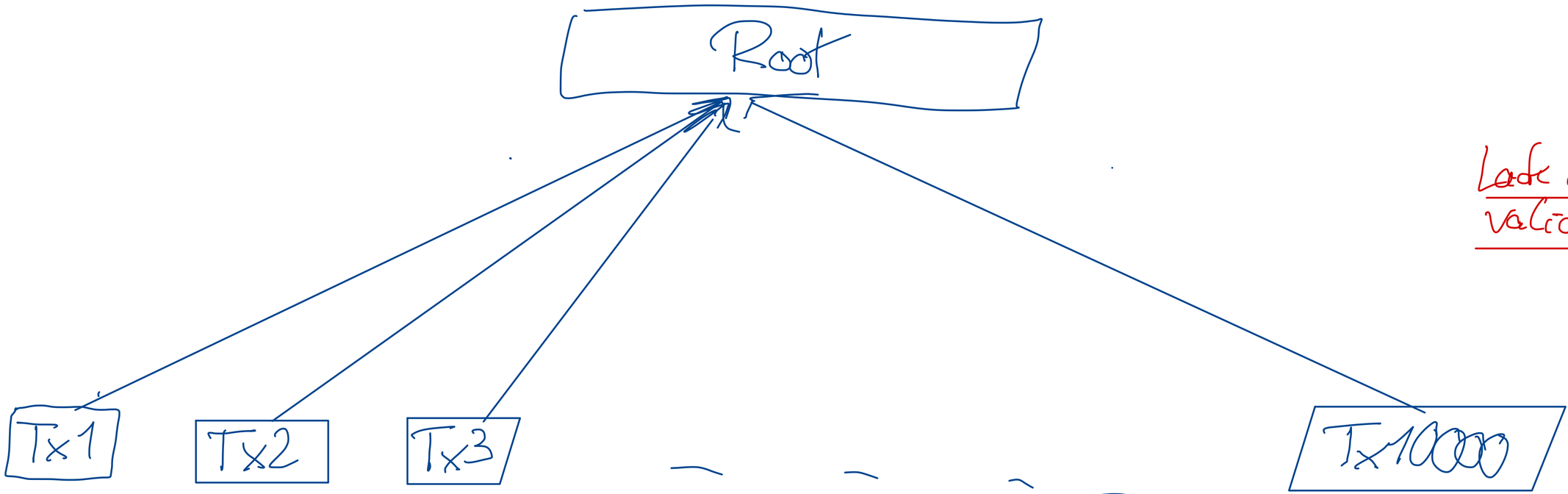
MERKLE TREES: HOW TO OBTAIN MERKLE ROOT OF LIST OF DATA

Given Ordered of list of data $l = \{A, B, C, \ldots, H\}$,
hash function $h(\cdot)$ ← sha256(sha256(·))

1. Hash all items in l using $h(\cdot)$
2. If only one hash is left → return hash
3. Consider consecutive pairs of data X, Y in list
   - compute $H_{XY} = h(X||Y)$ in list $\forall (X,Y)$ pairs
4. Create ordered list of these outputs
5. Go to step 2.

Why do we not summarize transactions as follows?

For example, as follows: Root = sha256(sha256(Tx1 || Tx2 || ... || Tx10000))



Lack of efficient validation

Answer: In such data structure there is no way to verify if Tx k, where k is some index $k \in \{1, \ldots, 10000\}$ is part of block with Root in header **unless** verifier knows all Tx's (Tx1 until Tx10000)

Taproot

TAPROOT SOFTFORK

“Taproot”: Name of most recent Bitcoin protocol upgrade

- Implemented by network in November 2021

Changes introduced:

- **Schnorr Signatures** (BIP 340)
- **Taproot/ Merklized Alternative Script Trees** (BIP 341)
  Improve privacy, efficiency, and flexibility of Bitcoin's scripting
- **Validation Rules of Taproot Scripts** (BIP 342)
  OP_CHECKSIG, OP_CHECKSIGVERIFY refers now to Schnorr signatures (instead of ECDSA), no OP_CHECKMULTIGSIG

Schnorr Signatures

In identity protocols above, we just provided a way for person with knowledge of private key e to show they know e without revealing e.

To create Bitcoin transactions, we also need to make sure that person with knowledge of e commits to transaction data m, and this commitment needs to be publically verifiable.

SchnorrSign(e, k, m)
- $R = kG$
- Compute $z = \text{hash}(R||eG||m)$
- $s = k + z \cdot e$
- return (s, R)

private key $P = eG$ is also concatenated here to avoid that a valid signature s' for "derived public child key" $P + c$ can be created from valid signature s for public key P.

SchnorrVerify(s, P, R, m)
- Compute sG
- Compute $z = \text{hash}(R||P||m)$
- Compute $\text{festival} = zP + R$
- If $sG == \text{festival}$
  - return True
- else
  - return False

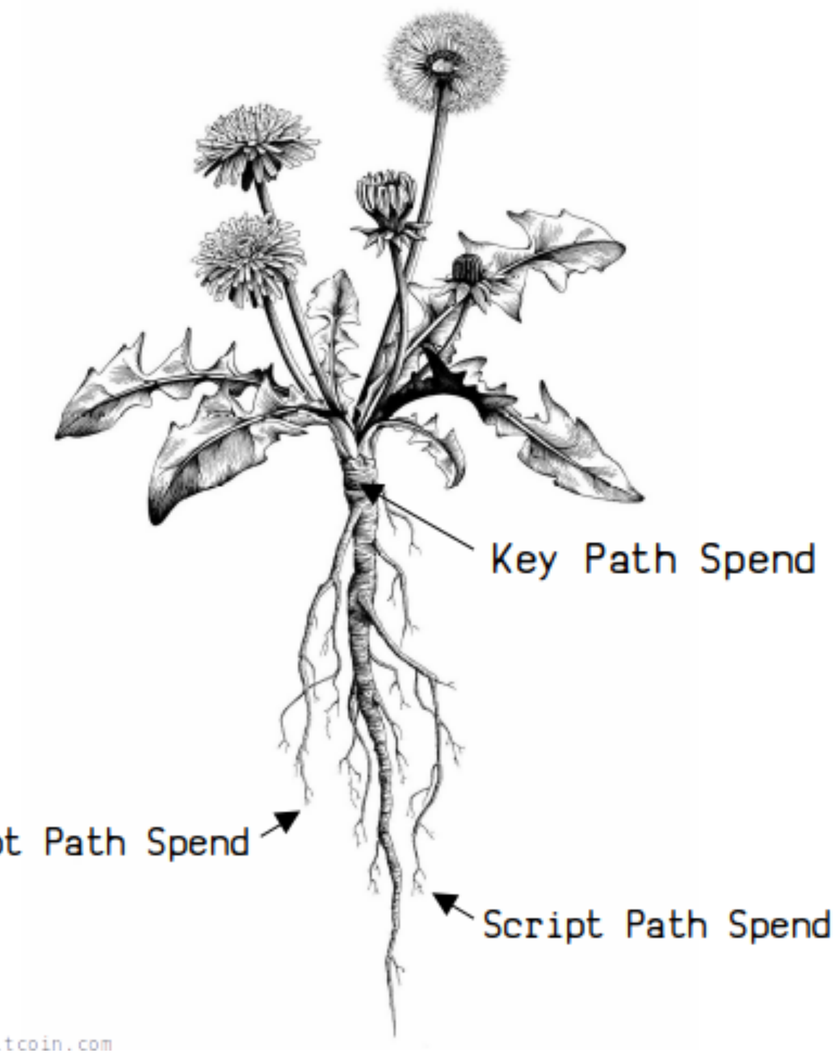
Schnorr Signature scheme as defined in BIP340 (Bitcoin Improvement Protocol 340), used with secp256k1

ECDSA VS. SCHNORR SIGNATURES IN BITCOIN

- **Advantages of Schnorr Signatures over ECDSA**
  - Shorter serializations (leading to memory savings)
  - Provable security guarantees based on:
    - ✦Difficulty of discrete logrithm problem for the secp256k1 elliptic curve, and
    - ✦Appropriateness of “Random Oracle Model” for SHA-256 hash function
  - Linearity of signatures
  - Efficient batch verification
  - Ability for “Scriptless” Multi-Signature Schemes (not standard yet)

TAPROOT

Intuition: There is one “standard” spending condition, but also many non-standard ones



PREREQUISITE: TAGGED HASHES

Issue of cryptographic hash function "reuse":

Hash functions are used in different contexts, e.g.,

- For definition of challenge scalar z in Schnorr signatures*:

$$z = \text{sha256}(\text{sha256}(R \,||\, P \,||\, m))$$

- For nonce generation in Schnorr signatures:

$$k = \text{sha256}(\text{sha256}(t \,||\, P \,||\, m)), \text{ where } t = e \text{ XOR } \text{rand}$$

could accidentally Leak information that could be used for attacks.

⇒ **Tagged Hash:** Given a string tag t, a hash function h, and input m, the tagged hash of m with tag t is

$$h_t(m) := h(h(t) \,||\, h(t) \,||\, m).$$

* In BIP340, technically, "R" and "P" correspond to x-coordinate of R and P.

($R = kG$)

public nonce ↑ ↑ private nonce

↑ random input

Example tags in Taproot:

- t = "BIP0340/challenge"
- t = "TapLeaf"
- t = "BIP0340/nonce"

PREREQUISITE: KEY TWEAKS

Idea: Given private-public key pair (e, P) satisfying $P = eG$ and a (32-byte) tweak c, obtain new, so-called "tweaked public key" Q and matching "tweaked private key" e_Q with $Q = e_Q G$ such that

- Q can be obtained from P and h only.

This is similar to the derivation of public keys in hierarchical deterministic wallets (BIP32).

Tweaked Public Key:

$$Q = P + \text{hash}(P_x \| c)G$$

Tweaked Private Key:

$$e_Q = e + \text{hash}((eG)_x \| c)$$

More details: BIP341.

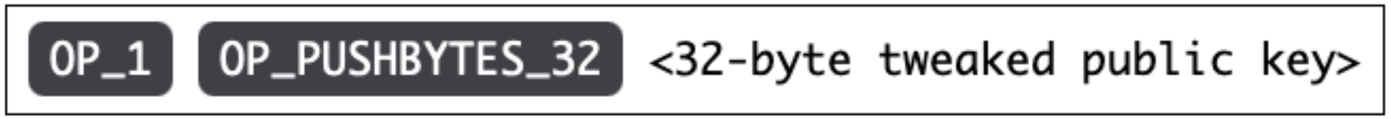
OVERVIEW OF BITCOIN ADDRESS FORMATS

Type	First Seen	BTC Supply*	Use*	Encoding	Prefix	Characters
P2PK	Jan 2009	9% (1.7M)	Obsolete			
P2PKH	Jan 2009	43% (8.3M)	Decreasing	Base58	1	26 – 34
P2MS	Jan 2012	Negligible	Obsolete			
P2SH	Apr 2012	24% (4.6M)	Decreasing	Base58	3	34
P2WPKH	Aug 2017	20% (3.8M)	Increasing	Bech32	bc1q	42
P2WSH	Aug 2017	4% (0.8M)	Increasing	Bech32	bc1q	62
P2TR	Nov 2021	0.1% (0.02M)	Increasing	Bech32m	bc1p	62

Source: https://unchained.com/blog/bitcoin-address-types-compared/

TAPROOT ADDRESSES (P2TR)

A P2TR ScriptPubKey (output script) has the following pattern:



recall why only 32 bytes as Schnorr public key!

contains information of Q

where
- $Q = P + hash_{TapTweak}(P_x \| m)G$
- m is root of a Merkleized Alternative Script Tree.

MOTIVATION: COMPLEX REDEEM SCRIPTS

Consider following setup:

- Mohammed, a company owner in Dubai, operates an import/export business; he wishes to construct a company capital account with flexible rules. The scheme he creates requires different levels of authorization depending on timelocks. The participants in the multisig scheme are Mohammed, his two partners Saeed and Zaira, and their company lawyer.

  The three partners make decisions based on a majority rule, so two of the three must agree. However, in the case of a problem with their keys, they want their lawyer to be able to recover the funds with one of the three partner signatures.

  Finally, if all partners are unavailable or incapacitated for a while, they want the lawyer to be able to manage the account directly after he gains access to the capital account's transaction records.

2 of-3 multisig

APPLICATION OF TAPROOT SPENDING CONDITION

Redeem script (as e.g. used in P2SH):

```
01  OP_IF
02    OP_IF
03      2
04    OP_ELSE
05      <30 days> OP_CHECKSEQUENCEVERIFY OP_DROP
06      <Lawyer's Pubkey> OP_CHECKSIGVERIFY
07      1
08    OP_ENDIF
09    <Mohammed's Pubkey> <Saeed's Pubkey> <Zaira's Pubkey> 3 OP_CHECKMULTISIG
10  OP_ELSE
11    <90 days> OP_CHECKSEQUENCEVERIFY OP_DROP
12    <Lawyer's Pubkey> OP_CHECKSIG
13  OP_ENDIF
```

Script 1

pubkey

Script 2

Script 3

Q: How to encode script such that only the spending condition that is used needs to be revealed?

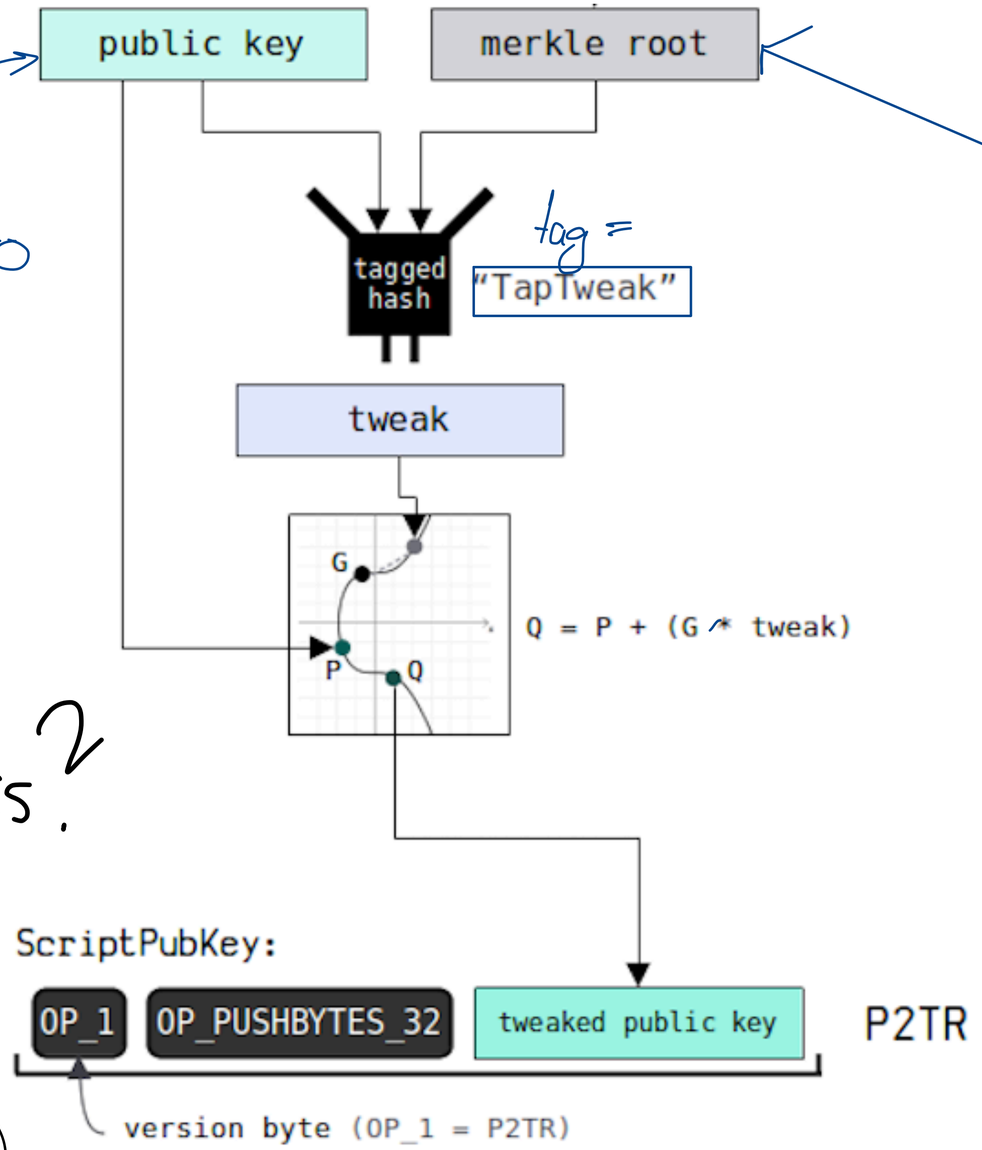
STRUCTURE OF P2TR SPENDING CONDITIONS

In example:

▷ Encode case that all three parties need to sign, i.e., a 3-of-3 multisig.

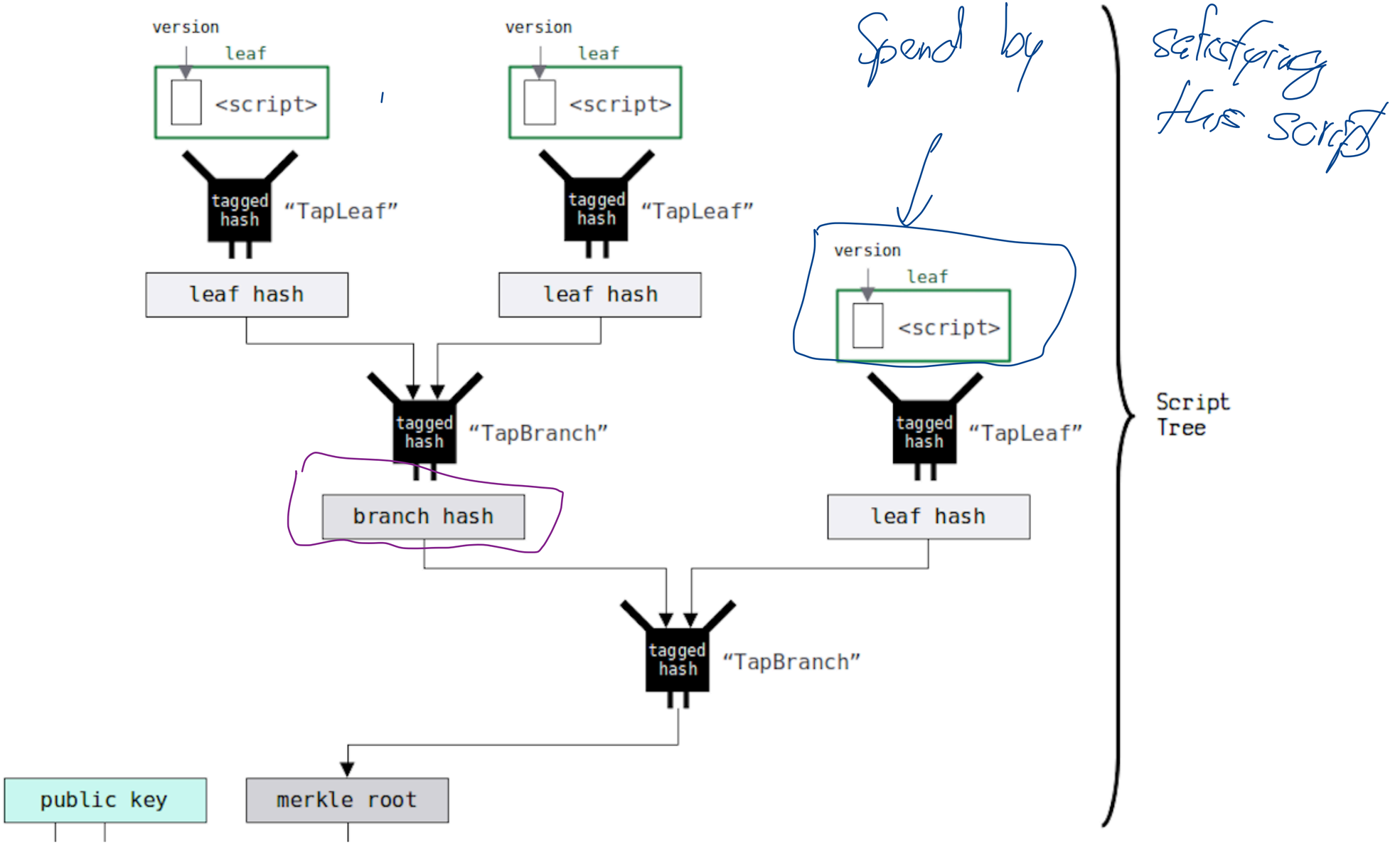
Q: How to implement this?

↳ Key aggregation protocol (e.g., MuSig, MuSig2)



▷ Encode information of leaf redeem script into a "tree of scripts" summarized by a Merkle root m.

STRUCTURE OF P2TR SPENDING CONDITIONS



MERKLIZED ALTERNATIVE SCRIPT TREE FOR EXAMPLE

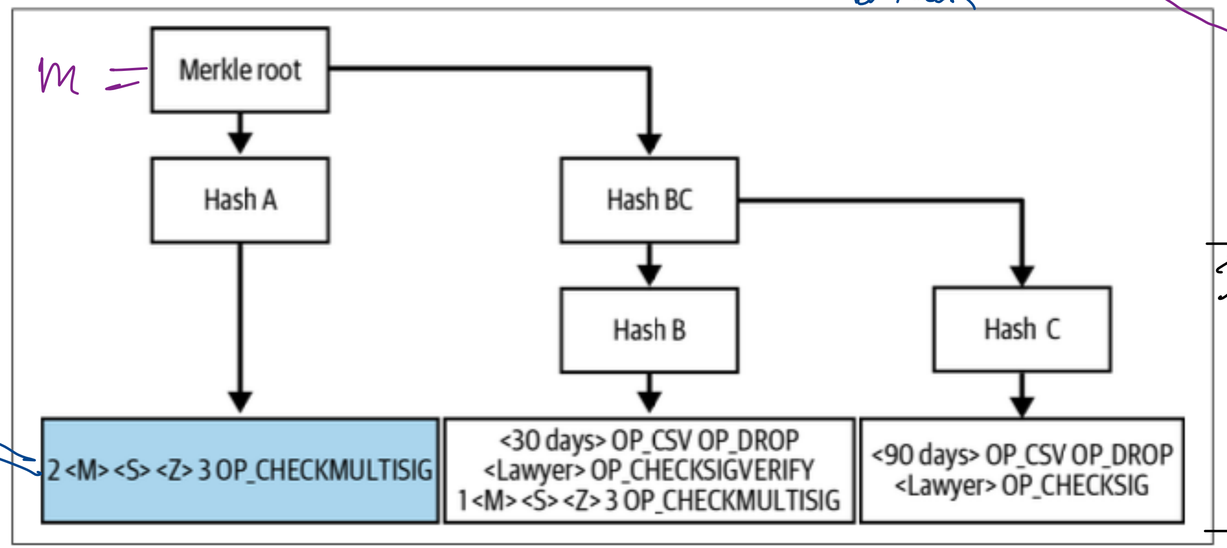
When spending the output, there are two options of what to include in the "Witness" part of transaction:

- If "Keypath spending": ▷ Signature for Q, where $Q = P + hash_{TapTweak}(P||m)G$.

(64 bytes)

⚠ Q itself does not need to appear in Witness field as already in locking script / Script Pub Key.

m = Merkle root



In example: E.g.
▷ Script inputs: OP_0 <Sig M> <Sig S>
▷ Merkle Path: <Hash BC>

OR

S =

▷ If "Scriptpath spending":

1. Script Inputs (satisfying leaf script)
2. Leaf script S
3. "Control Block" → 1 control byte (indicating parity of Q)
   → (Untweaked) public key P
   → Merkle Path ⚠ Not: Merkle root m

See also:

"Taproot" on

Learnmeabitcoin.com.