



Taproot & Policy

James Chiang

Overview

Taproot (& Schnorr)

Policy (& Miniscript)

Policy & Taproot

Schnorr Properties

- **Linear Properties**
- **Validator**
 - Batch verification (Reduced Cost)
 - Security Proof (Security)
 - Non-malleable encoding (Security)
- **User**
 - “Tweakable” Musig
 - Commitments/Adaptor Signatures/DiscreteLogContracts
 - Schemes above indistinguishable from single pk(key).

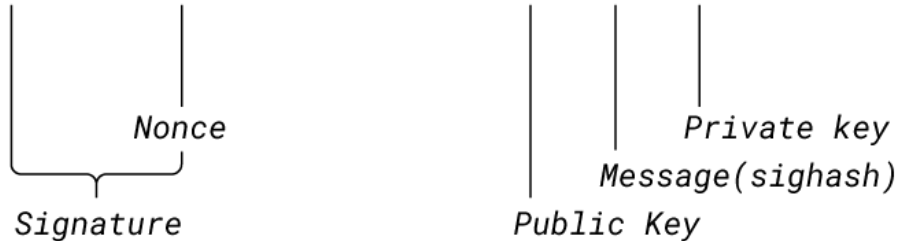
Schnorr Properties

- **Linear Properties**
- **Validator**
 - Batch verification (Reduced Cost)
 - Security Proof (Security)
 - Non-malleable encoding (Security)
- **User**
 - "Tweakable" Musig
 - Commitments/Adaptor Signatures/DiscreteLogContracts
 - Schemes above indistinguishable from single pk(key).

Tweaking Schnorr for Taproot

Schnorr Sig:

$$\mathbf{S = R + H(R|P|m) \times G}$$



Tweaking Schnorr for Taproot

Schnorr Sig:

$$S = R + H(R|Q|m)qG$$

Signature

Nonce

Public Key

Message(sighash)

Private key

Key Tweak:

$$q = x + t$$

$$Q = P + T$$

Taproot Output Key

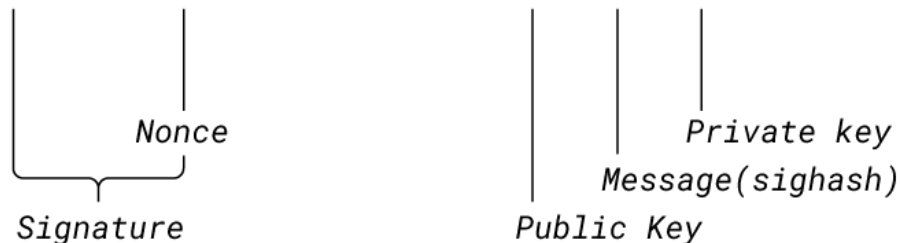
Internal Key

Tweak

Tweaking Schnorr for Taproot

Schnorr Sig:

$$\mathbf{S} = \mathbf{R} + \mathbf{H}(\mathbf{R}|\mathbf{Q}|m)\mathbf{q}\mathbf{G}$$



Key Tweak:

$$\mathbf{q} = \mathbf{x} + \mathbf{t}$$

$$\mathbf{Q} = \mathbf{P} + \mathbf{T}$$



Co-ownership with single pk:

$$\mathbf{P} = \text{musig}(\text{keys})$$

Taproot - Keypath Spend

Taproot Output Script

[01] **[33B pubkey Q]**



Spending Witness

[sig]

Taproot - Scriptpath Spend

Taproot Output Script

[01] [33B pubkey Q]



Spending Witness

[initial stack]

[script]

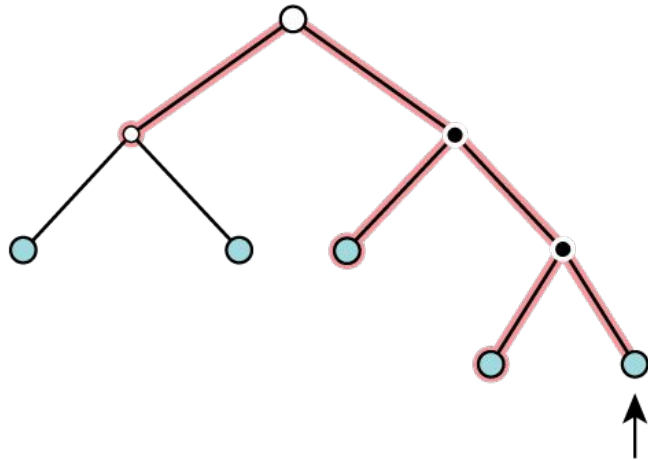
[controlblock]

Taproot - Scriptpath Spend

Taproot Output Script

[01] **[33B pubkey Q]**

$$Q = P + tG$$



Spending Witness

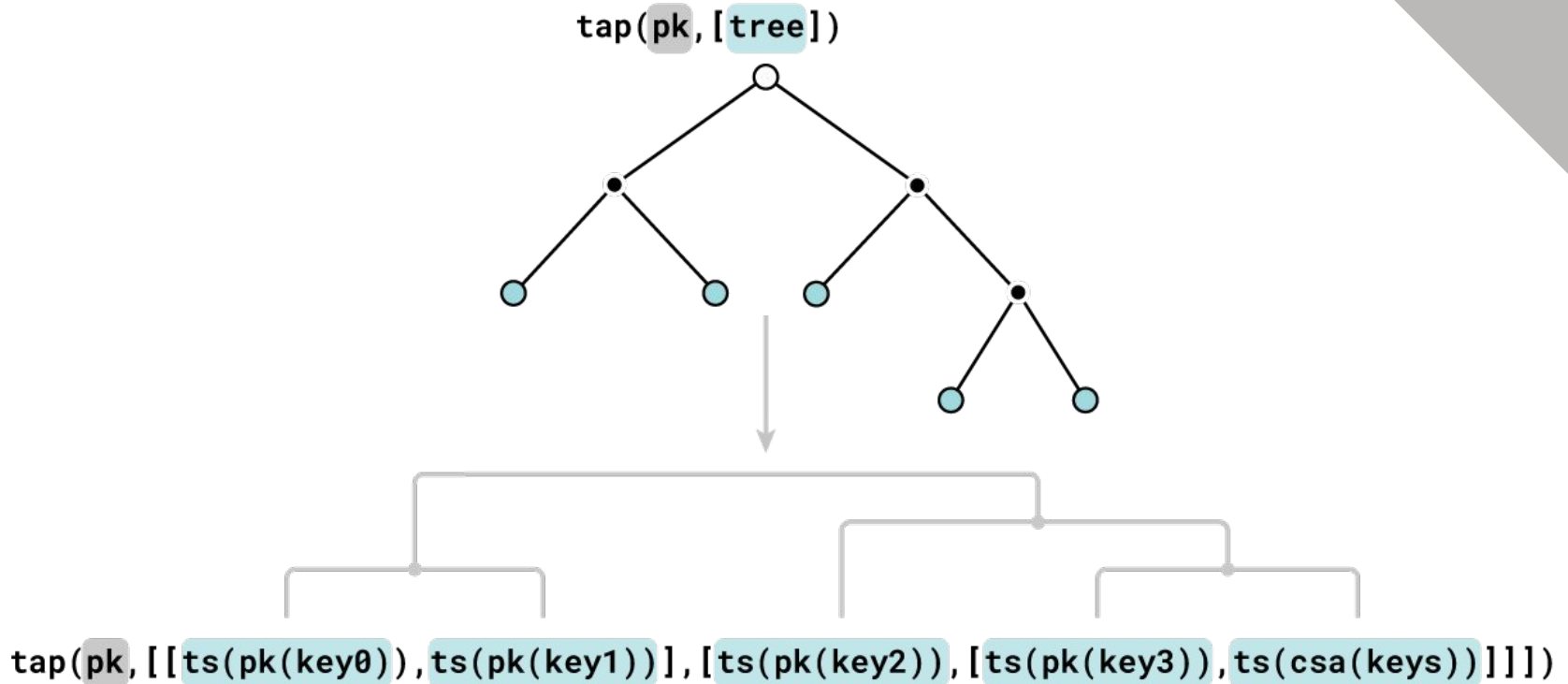
[initial stack]

[script]

[controlblock]

└ Taproot Inclusion Proof

Taproot Descriptor



Tapscript

- **Versioning**
 - Versioned Tapleaf
 - op_code upgradability (OP_SUCCESSx)
- **Checksig**
 - Verifies Schnorr signatures
 - Checkmultisig -> checksigadd (batch verifiable)
- **Malleability**
 - Schnorr signatures encoding is non-malleable
 - OP_IF , must consume 1
- **Transaction Digest**
 - Spendtype: Bit 2 set (Must be unset for keypath spend)

Overview

Taproot (& Schnorr)

Policy (& Miniscript)

Policy & Taproot

Policy Language & Miniscript

- **Proposed by Sipa (Pieter Wuille)**
 - Miniscript Presentation, Stanford 2019
 - bitcoin.sipa.be/miniscript
- **Andrew Poelstra**
 - Miniscript Compiler in Rust

Policy Language

“Non-terminal” Expressions

- **and**(EXPR, EXPR)
- **or**(EXPR, EXPR)
- **thresh**(n, EXPR, EXPR, ...)

“Terminal” Expressions

- **pk**(key)
- **time**(n) - (relative/absolute)
- **hash**(hex)

Policy Language

```
and(pk(key), time(100))
```


Policy Language

`and(pk(key), time(100))`

- Timelocked Pubkey Output.

Policy Language

```
or( [ ] , [ ] )  
    and( time(1000), pk(key3) )  
    thresh(2, pk(key0), pk(key1), pk(key2))
```

Policy Language

```
or( [ ] , [ ] )  
    and( time(1000), pk(key3) )  
    thresh(2, pk(key0), pk(key1), pk(key2))
```

The diagram illustrates a policy language expression: `or([] , [])`. The two square brackets are highlighted in grey. Below them, two lines of code are shown. The first line is `and(time(1000), pk(key3))`, where `time(1000)` and `pk(key3)` are highlighted in light blue. The second line is `thresh(2, pk(key0), pk(key1), pk(key2))`, where `pk(key0)`, `pk(key1)`, and `pk(key2)` are highlighted in light blue. Lines connect the dots in the brackets of the `or` function to the `and` and `thresh` functions, indicating that the `or` function takes these two sub-expressions as arguments.

- Multisignature Output
- Spendable by a back-up key after timeout

Policy Language

```
or( [ ] , [ ] )  
    and(hash(hex), pk(key0))  
    and(time(100), pk(key1))
```

The diagram illustrates the structure of a policy language expression. It shows an `or` function with two arguments in brackets. The first argument is connected to the expression `and(time(100), pk(key1))` and the second argument is connected to `and(hash(hex), pk(key0))`. The function names and their arguments are highlighted in light blue.

Policy Language

```
or( [ ] , [ ] )  
    |    |  
    |    +--- and(hash(hex), pk(key0))  
    +--- and(time(100), pk(key1))
```

- Hash Time-locked Contract (HTLC - revocation)
- Lightning Routing

Policy to Bitcoin Script

Policy

pk, hash, time

X and Y

X or Z

threshold

wrappers

...

?

Bitcoin Script

- Satisfiable non-malleable witness
- Spendable by wallets
 - Composable primitives
 - “Subset of Bitcoin Script”

Miniscript



pk, hash, time

and(X,Y)

or(X,Y)

thresh(n,X,Y,..)

Miniscript



pk, hash, time

and(X, Y)

or(X, Y)

thresh(n, X, Y, ..)

pk(key)

key

pk_h(keyhash)

DUP HASH160 keyhash EQUALVERIFY

older(n)

n CHECKSEQUENCEVERIFY

after(n)

n CHECKLOCKTIMEVERIFY

sha256(h)

SIZE 32 EQUALVERIFY SHA256 h EQUAL

ripemd160(h)

SIZE 32 EQUALVERIFY RIPEMD160 h EQUAL

...

Miniscript



pk, hash, time

and(X, Y)

or(X, Y)

thresh(n, X, Y, ..)

and_v(X, Y)

and_b(X, Y)

and_n(X, Y)

[X] [Y]

[X] [Y] BOOLAND

[X] NOTIF 0 ELSE [Y] ENDF

Miniscript



pk, hash, time

or_b(X,Z)

[X] [Z] BOOLOR

and(X,Y)

or_d(X,Z)

[X] IFDUP NOTIF [Z] ENDIF

or(X,Y)

or_c(X,Z)

[X] NOTIF [Z] ENDIF

thresh(n,X,Y,..)

or_i(X,Z)

IF [X] ELSE [Z] ENDIF

Miniscript



pk, hash, time

and(X, Y)

or(X, Y)

thresh(n, X, Y, ..)

thresh(k, ...)

[X1] ([Xi] ADD)*(n-1) k EQUAL

thresh_m(k, ...)

k key_1 ... key_n n CHECKMULTISIG

Miniscript



pk, hash, time

and(X,Y)

or(X,Y)

thresh(n,X,Y,..)

Wrappers

a:X	TOALTSTACK [X] FROMALTSTACK
s:X	SWAP [X]
c:X	[X] CHECKSIG
t:X	[X] 1
...	...

Overview

Taproot (& Schnorr)

Policy (& Miniscript)

Policy & Taproot

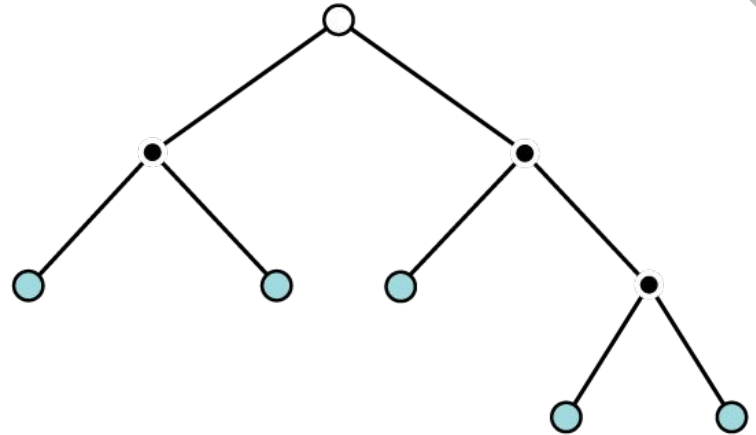
Policy to Taproot & Tapscript

Policy

pk, hash, time
and(X,Y)
or(X,Z)
threshold
...

?

Tapscript(s)

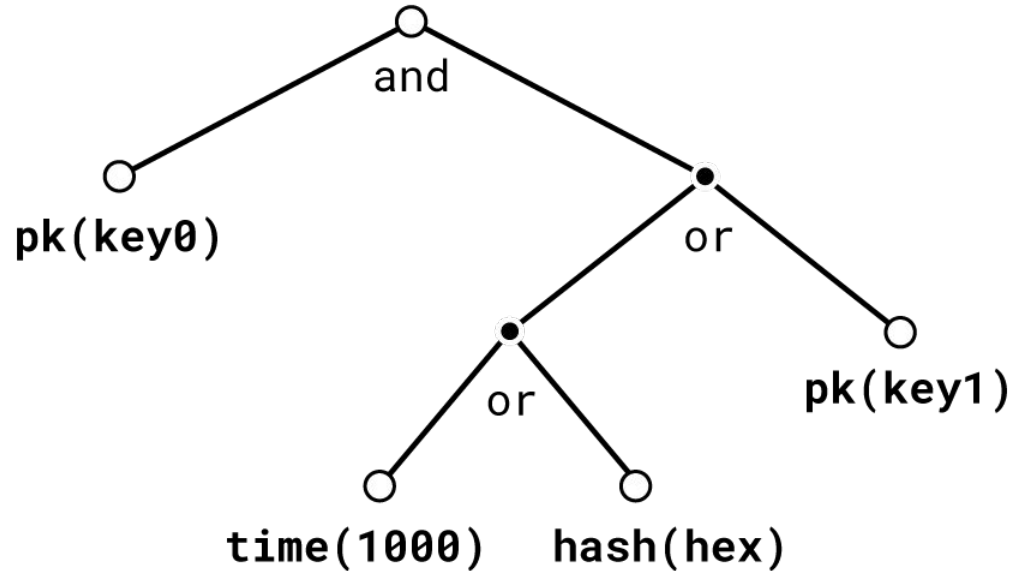


Policy to Taproot & Tapscript

```
and(pk(key0), or(or(time(1000), hash(hex)), pk(key1)))
```

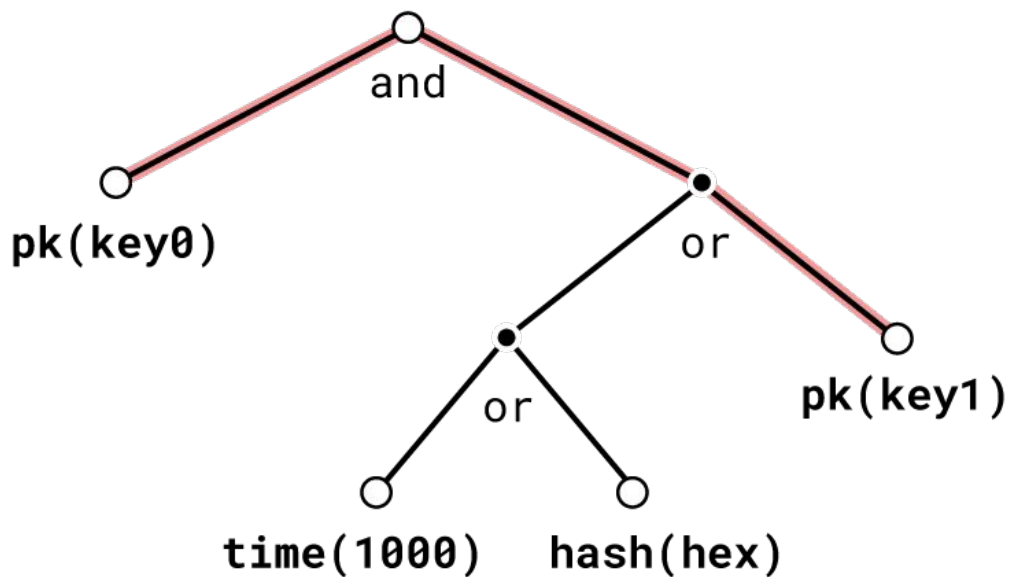
Policy Abstract Syntax Tree

`and(pk(key0), or(or(time(1000), hash(hex)), pk(key1)))`



Policy Abstract Syntax Tree

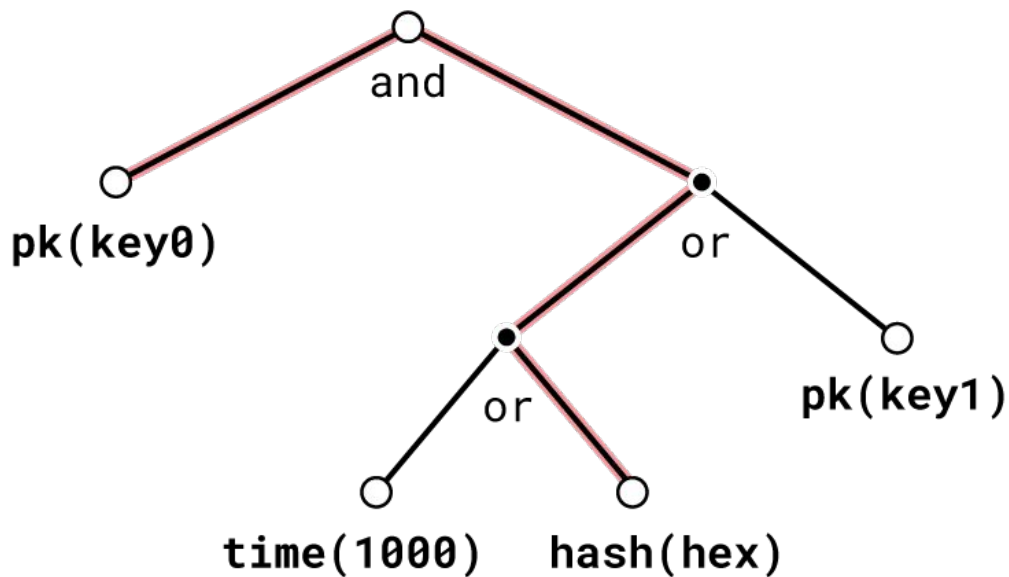
`and(pk(key0), or(or(time(1000), hash(hex)), pk(key1)))`



`and(pk(key0), pk(key1))`

Policy Abstract Syntax Tree

`and(pk(key0), or(or(time(1000), hash(hex)), pk(key1)))`



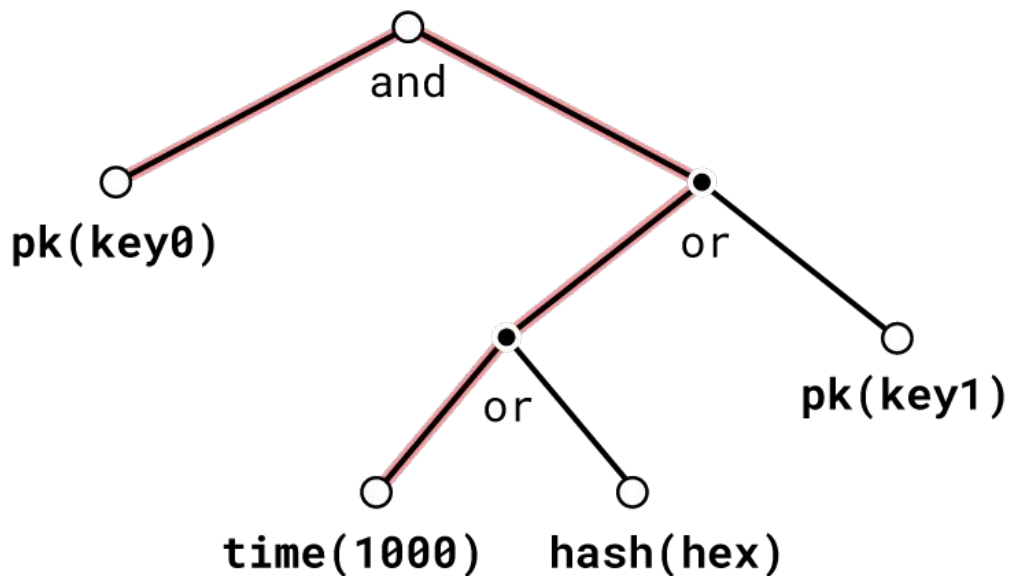
`and(pk(key0), pk(key1))`

OR

`and(pk(key0), hash(hex))`

Policy Abstract Syntax Tree

`and(pk(key0), or(or(time(1000), hash(hex)), pk(key1)))`



`and(pk(key0), pk(key1))`

OR

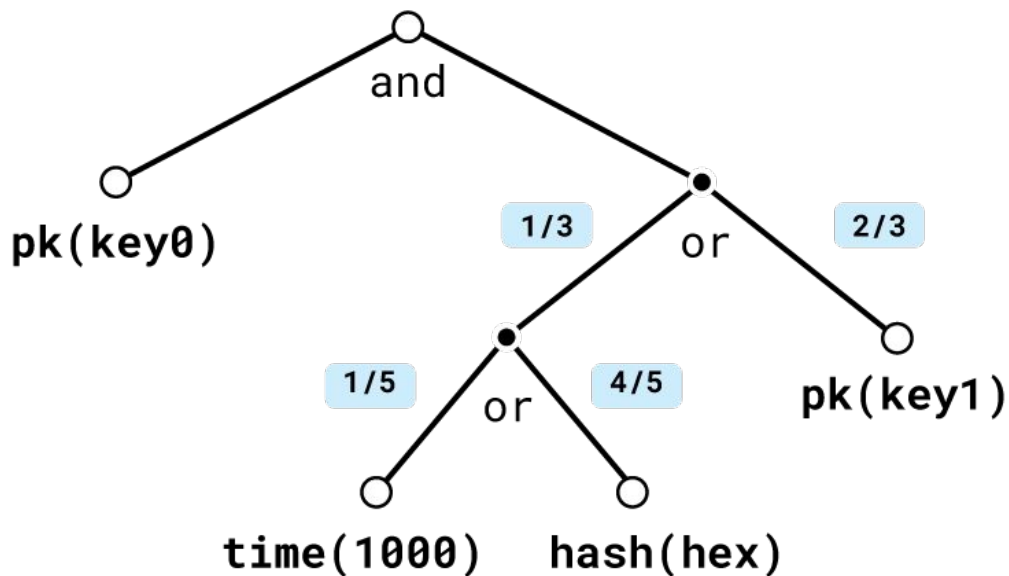
`and(pk(key0), hash(hex))`

OR

`and(pk(key0), time(1000))`

Policy Abstract Syntax Tree

`and(pk(key0), or(or(time(1000), hash(hex)), pk(key1)))`



2/3 `and(pk(key0), pk(key1))`

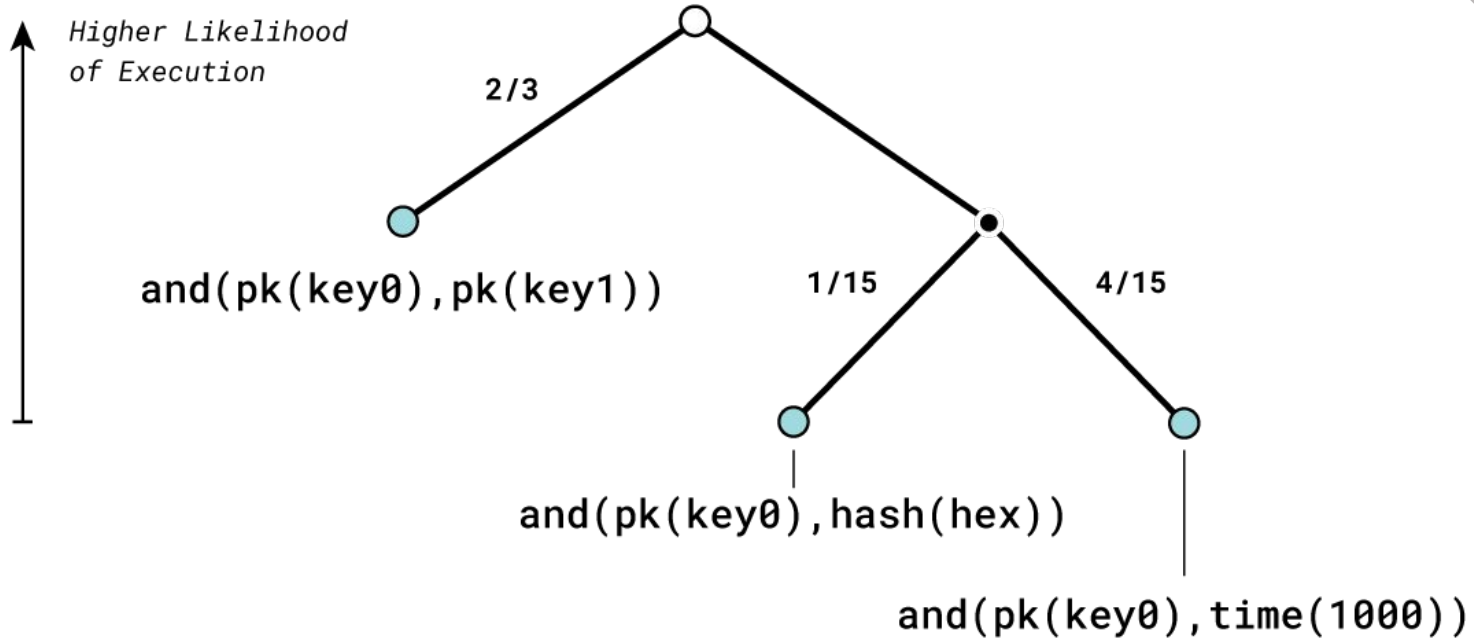
OR

1/15 `and(pk(key0), hash(hex))`

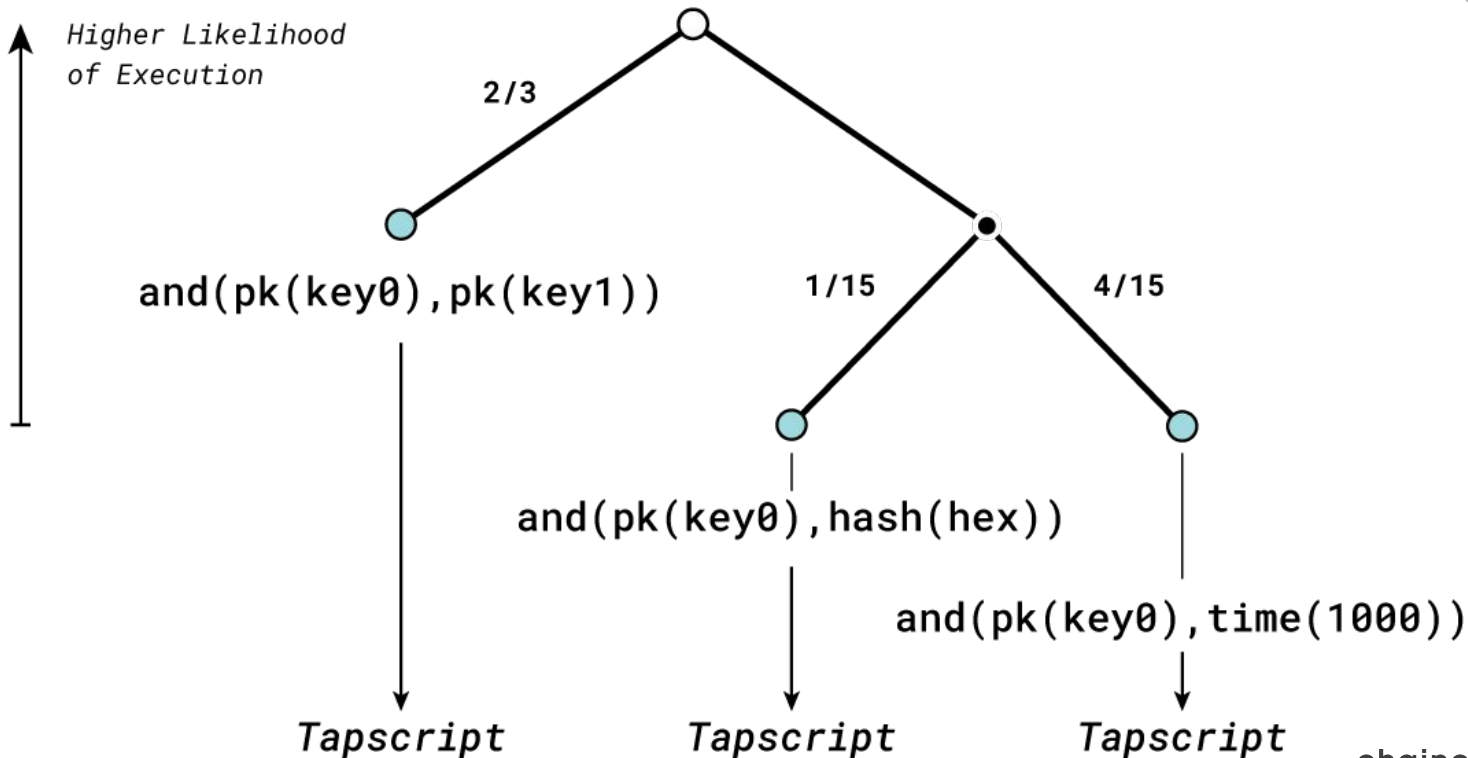
OR

4/15 `and(pk(key0), time(1000))`

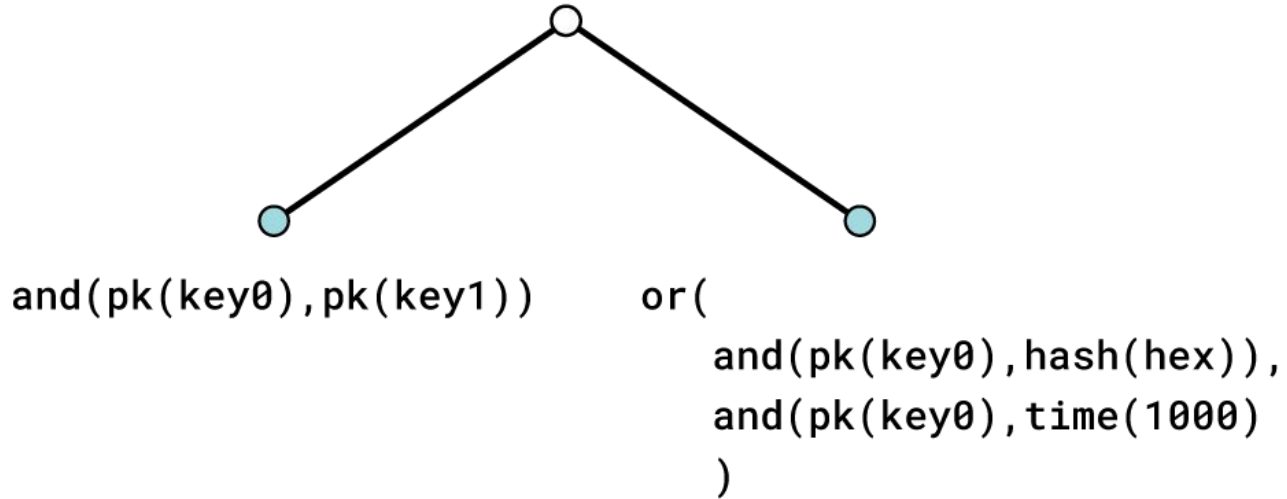
Tapscript/Taptree Compilation



Tapscript/Taptree Compilation



Tapscript/Taptree Compilation



Tapscript/Taptree Compilation



```
or(  
  and(pk(key0), pk(key1)),  
  or(  
    and(pk(key0), hash(hex)),  
    and(pk(key0), time(1000))  
  )  
)
```



Tapscript/Taptree Compilation

*Privacy
Reduction*



○
`or(
 and(pk(key0), pk(key1)),
 or(
 and(pk(key0), hash(hex)),
 and(pk(key0), time(1000))
)
)`

*Compiler determines
Taproot tree depth
for optimal cost.*



Policy Language & Taproot

- **Privacy:**
 - Exclusion of other conditions from same Tapscript
 - `pk(key) : [hash(hex), time(100)]`
- **Cost optimization**
 - Probability-weighted spending cost for script paths.

Thank you and questions?