LNP/BP Standards Association

RGBCON 0

Non-conference for client-validated state (RGB)

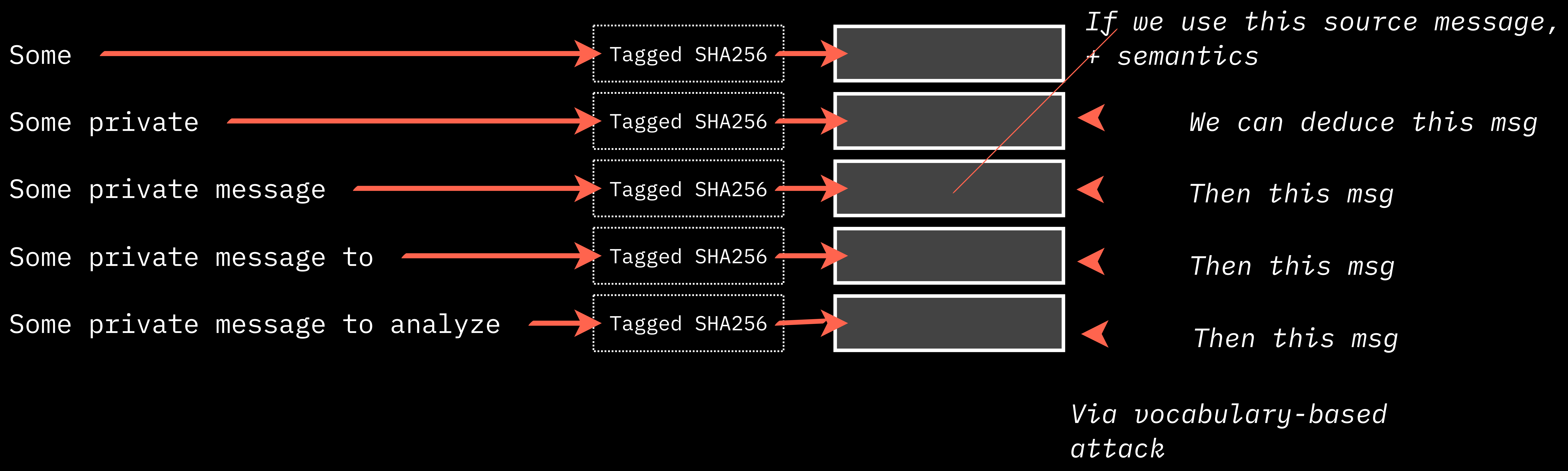
Organised by **LNP/BP Standards Association**, using donations from **Bitfinex & Tether, Fulgur Ventures, Poseidon Group**

Day I Re-cap

New developments

- Length-extension based privacy attack against public key tweaking
- Reason to allow non-squared/Schnorr public key serialisations
- More details on the external parts of the commitments
- Script commitment details

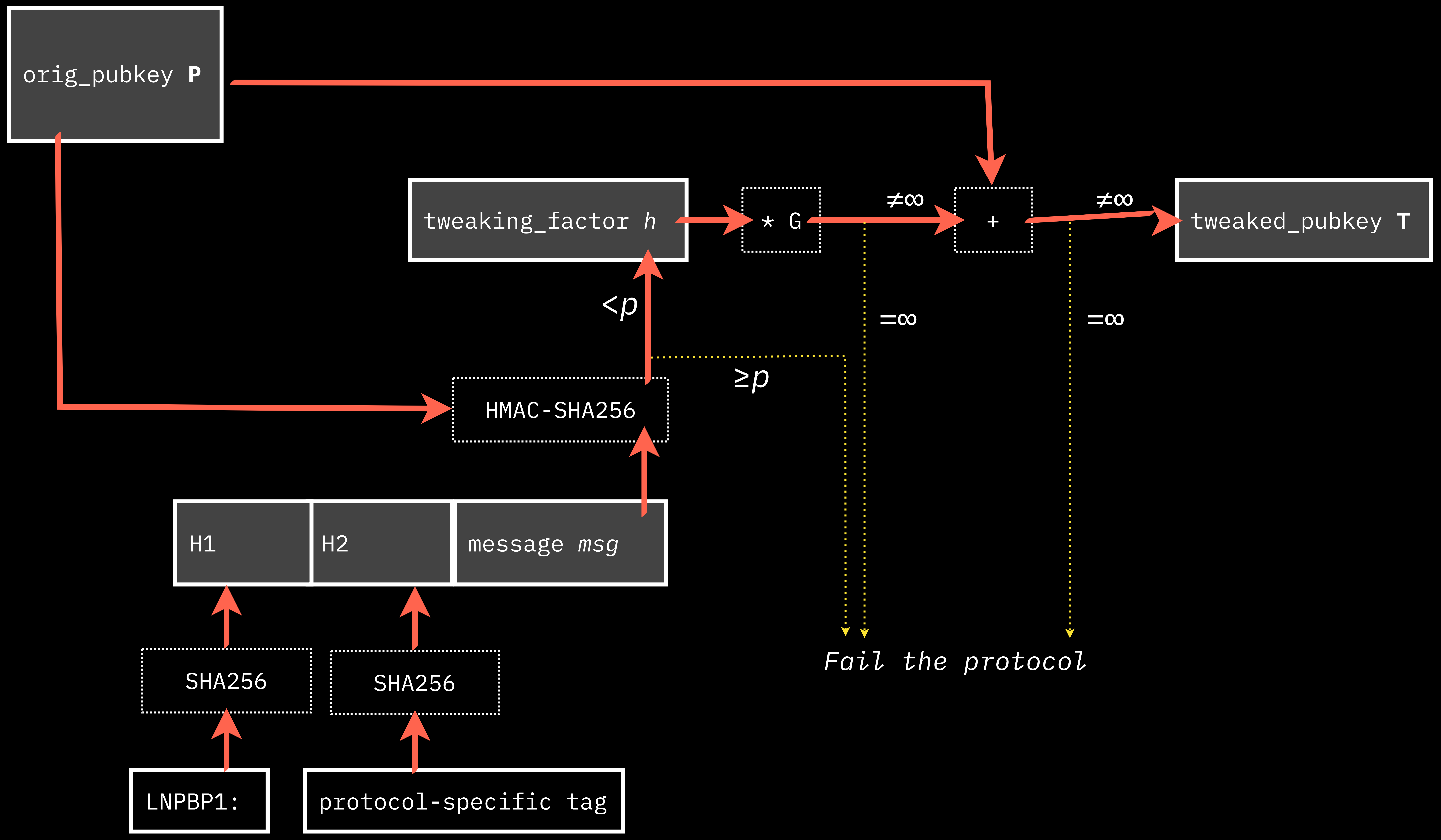
Length-extension privacy leak



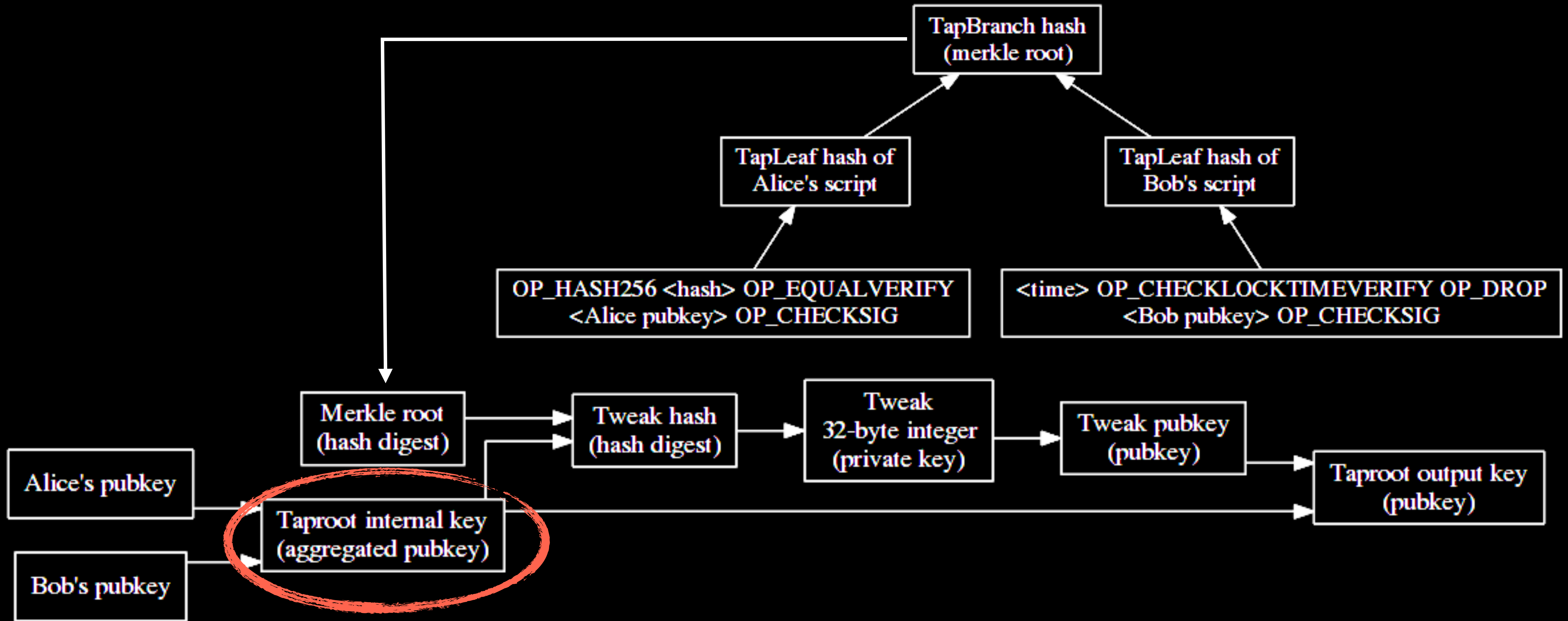
Arguments for HMAC

- Length-extension privacy leaks
- Squared public key requires more frequent public key replacements at each stage and makes tweaking procedure more complex
- We need to support older non-Schnorr public key encodings and public keys
- HMAC solves all these problems:
  - No length extension
  - Does not require any public key encoding

Results of LNPBP-1 finalisation



Taproot-specific details



Transaction commitments summary

	External
Transaction	External protocol-specific entropy + scriptPubkey-specific data
Transaction scriptPubkey type	
custom Script	Original public key(s)
OP_RETURN	Original public key
P2PK	Original public key
P2(W)PKH	Original public key
P2(W)SH	LockScript with original public keys
P2TR	Original intermediate public key, [Script Merkle root]
Public key	Original public key

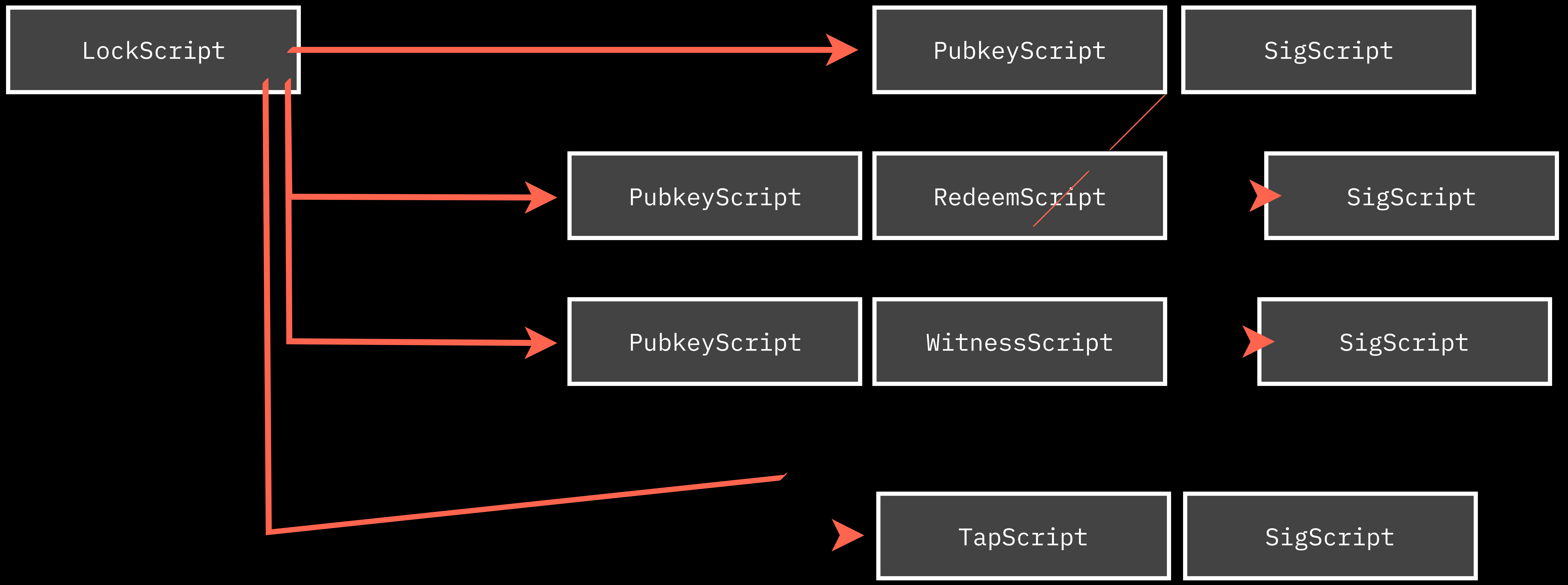
External protocol-specific entropy

- Required to be protocol specific, not specific for each transaction, otherwise double commitments are possible

Public key fingerprints in LockScript

OP_DUP	OP_HASH160	<pubkeyhash>	OP_CHECKSIG[VERIFY]		*Single pubkey hash*
OP_PUSH32/33/65	<pubkey>	OP_EQUAL	OP_CHECKSIG[VERIFY]		*Single public key in three serialisation formats*
OP_PUSH *n*	OP_PUSH_33/65	<pubkey> {m times}	OP_PUSH *m*	OP_CHECKMULTISIG[VERIFY]	*Multisig with public keys in two serialisation formats*

What is LockScript



Day II: RGB re-cap. Single-use seals. State-related protocols. Zero knowledge.

Dr Maxim Orlovsky, CEO **Pandora Core**, Secretary of LNP/BP Standards Association

Part I: RGB Re-cap.

RGB

- A set of standards and their implementation
- for client-validated state management (CVS)
- able to operate on top of Bitcoin and Lightning Network
- RGB to CVS is the same as Lightning Network to state channels: a specific set of protocols suited to work jointly.

RGB: what can be done with the owned state?

- Financial assets: IOU “stable coins”, shares, bonds, futures…
- Digital collectibles
- Voting rights
- Digital identity & reputation systems
- Better non-financial accounting systems: electricity etc
- Access/ownership tokens: datasets…
- Future contracts (Storm etc)

RGB use-cases

- Free & decentralised secondary markets
- Interoperability with industry-wide standard
- Regained privacy at a lot of levels:
  - Chain analysis tools privacy
  - Privacy with counter-parties
  - Privacy from issuers
  - Lightning Network rebalance privacy/fuzzyfication

Previous work

- Coloured coins
- Counterparty
- Mastercoin/OMNI Layer
- Confidential Assets

What we are trying to fix

- Privacy
- Potential miner’s censorship
- Blockchain pollution
- Enable Lightning network
- “Utility” Tokens -> Financial assets, collectibles

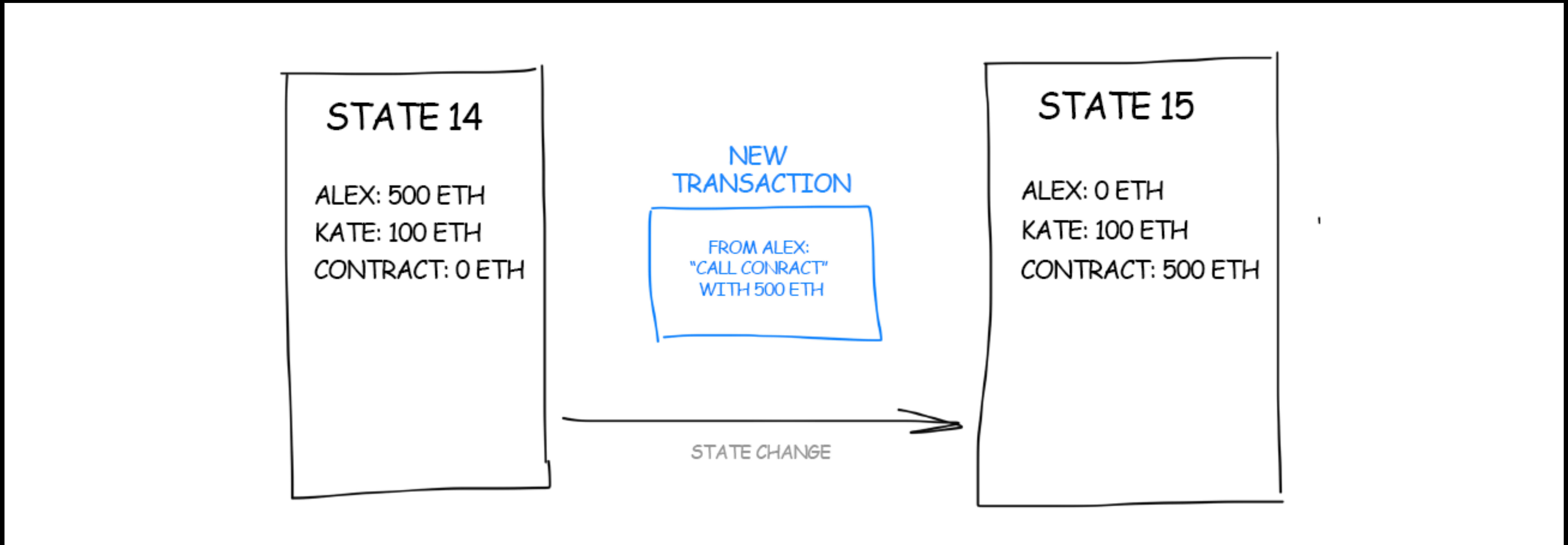
RGB main properties:

- **L1 succinctness**: Zero blockchain space consumption
- **Privacy**: transactions with seals can’t be distinguished from other transactions
- **Proper economics**: No miner incentives distortion

Part II: CVS

Off-chain state transitions

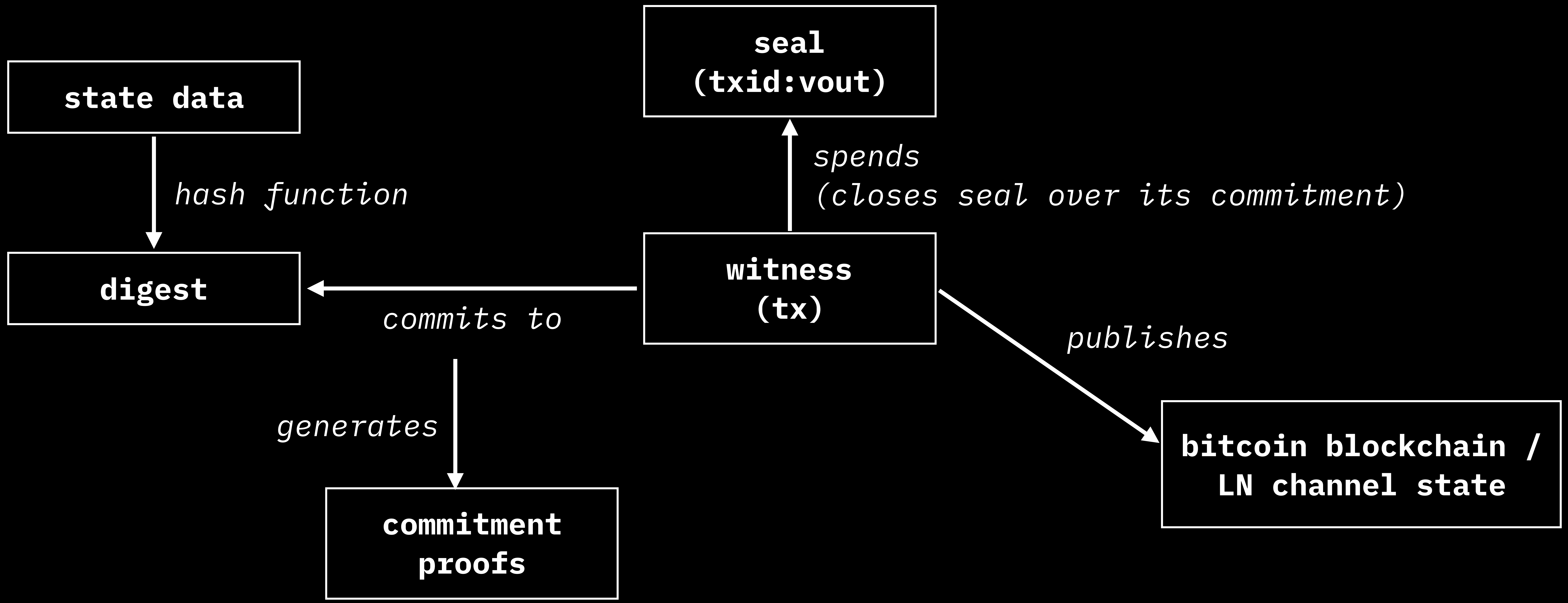
- Transaction -> state change



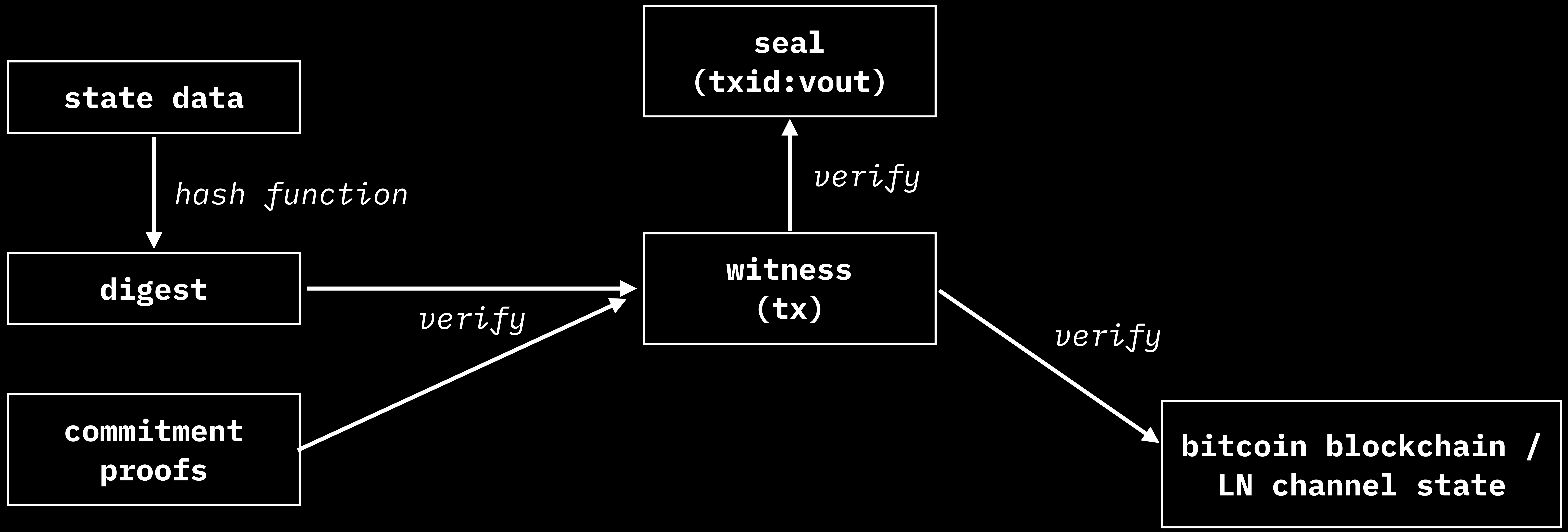
CVS Terms and definitions

- **Genesis state** (*comp.* Genesis block)
- **Owned state:** a state controlled by some well-defined party under Bitcoin script rules of some particular transaction output
- **State transition** (*comp.* Bitcoin transaction): off-chain data on the state, seals, metadata etc.
- **State transition commitment proof**: set of client-validated data, accompanied by the proper transactions from Bitcoin blockchain or a local Lightning channel, proving the security of the state transition:
    - The fact of transition
    - Absence of double commitment
- **CVS schema**: types of state data, metadata and other parameters and rules regarding possible state transition; used for validating state transitions
- **CVS graph**: graph of all state transitions from the genesis state to some owned state under the given state transition

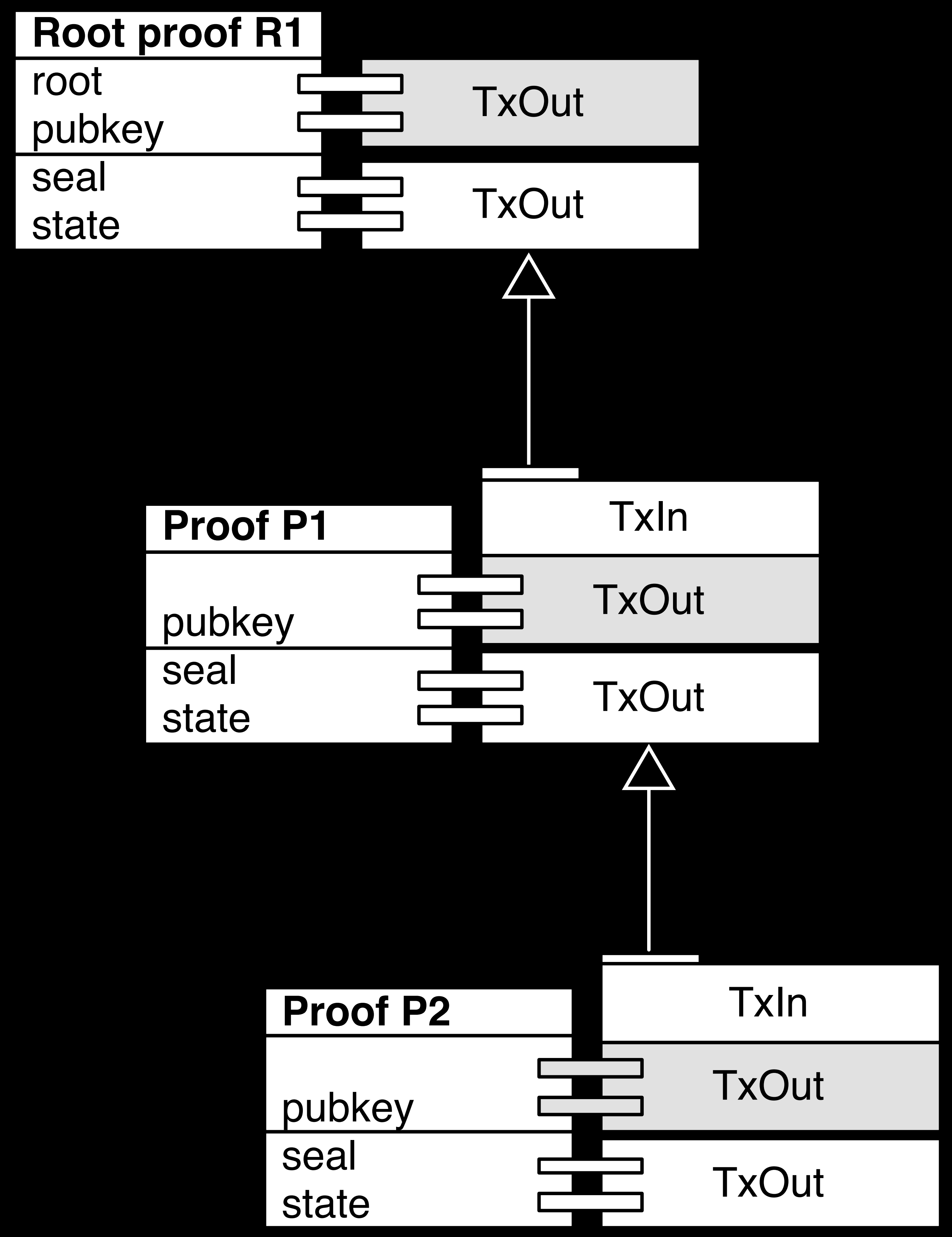
Single-use seal mechanics: “spending” state



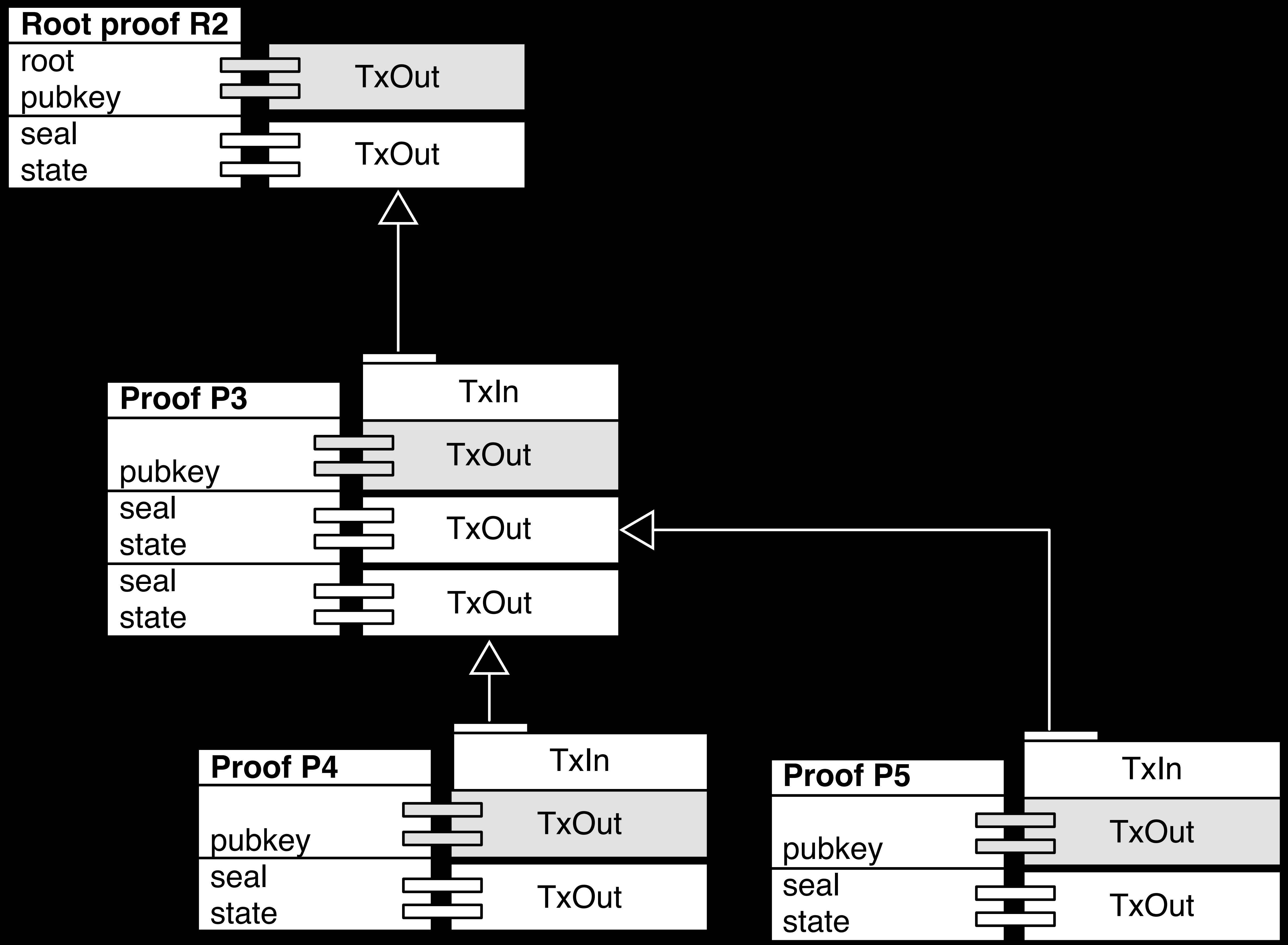
Single-use seal mechanics: “verifying” state



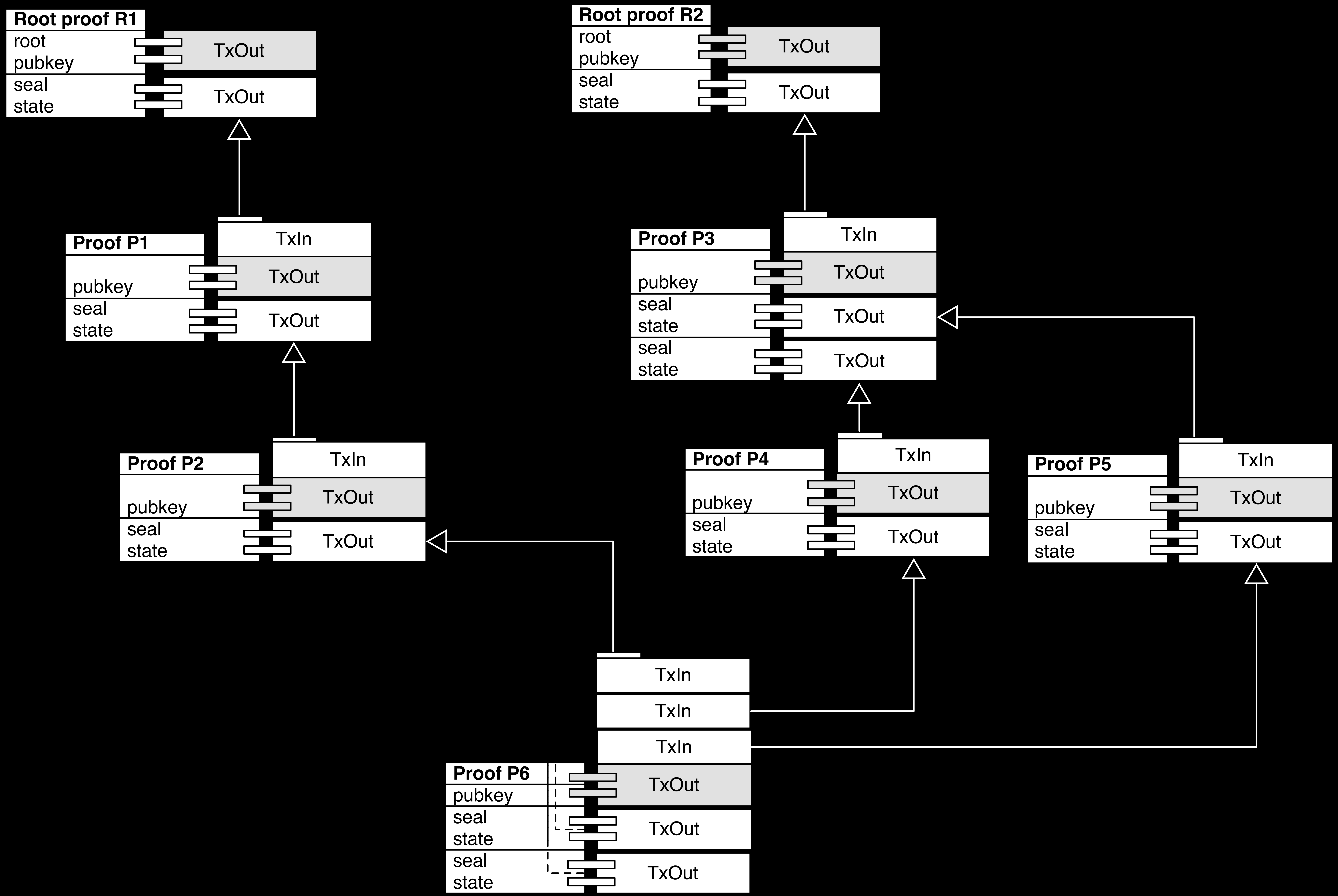
Chain of seals



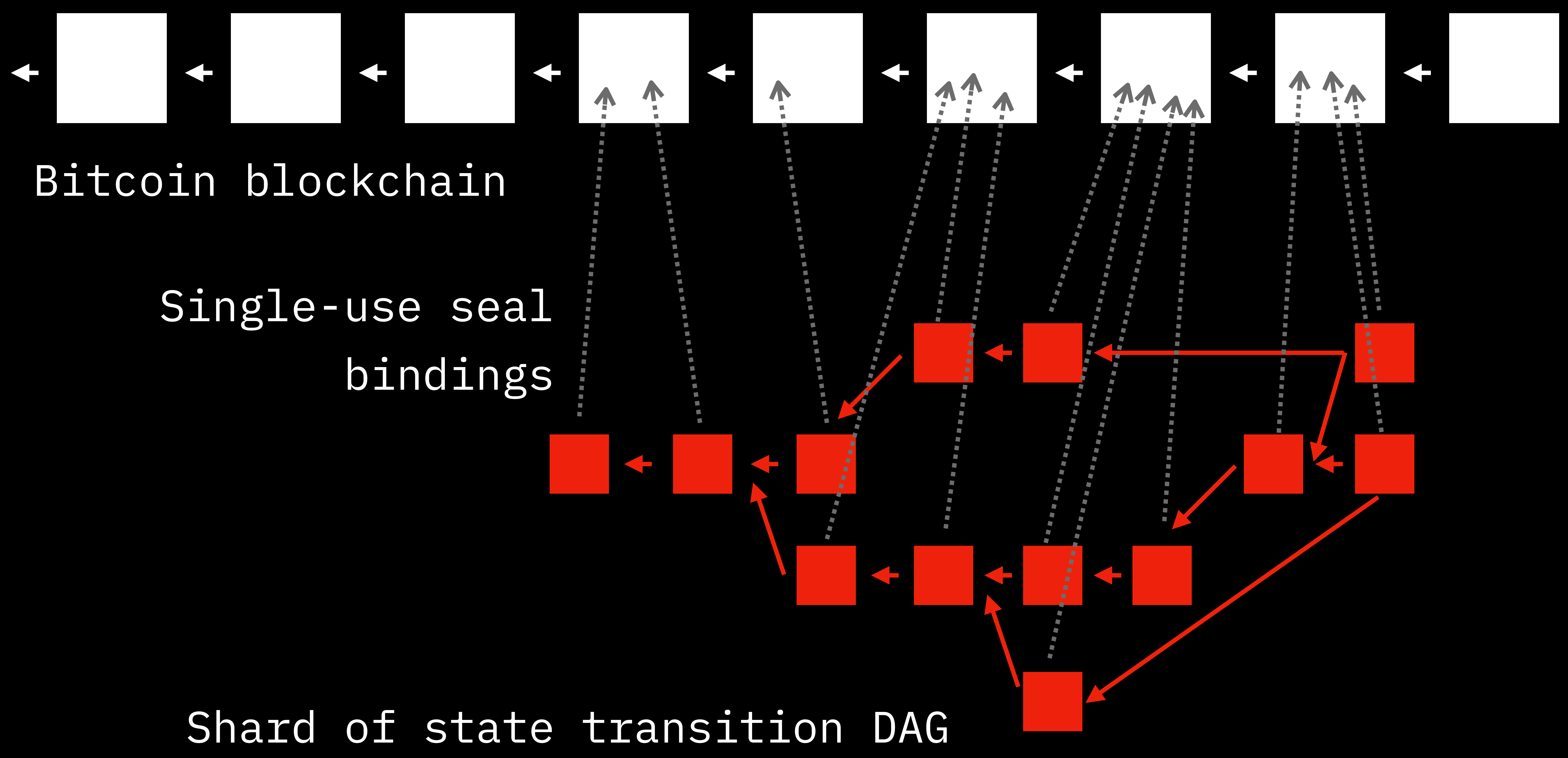
Tree of seals



DAG of seals


Root proof R1
root
pubkey
seal
state
TxOut
TxOut
Root proof R2
root
pubkey
seal
state
TxOut
TxOut
Proof P1
pubkey
seal
state
TxIn
TxOut
TxOut
Proof P3
pubkey
seal
state
seal
state
TxIn
TxOut
TxOut
TxOut
Proof P2
pubkey
seal
state
TxIn
TxOut
TxOut
Proof P4
pubkey
seal
state
TxIn
TxOut
TxOut
Proof P5
pubkey
seal
state
TxIn
TxOut
TxOut
TxIn
TxIn
TxIn
Proof P6
pubkey
seal
state
seal
state
TxOut
TxOut
TxOut

Sharded DAG on top of Bitcoin Blockchain



Questions to address:

- How do we define a state data?
- Peter Todd’s single-use seals assume seal-commitment with 1:1 correspondence; however by adding state and using Bitcoin we have:
  - Transactions that spend multiple txouts associated with different seals
  - Transactions that contain multiple defined seals
  - Transactions created under multiple (independent) genesis states

Part III: Zero knowledge

Pedersen commitments

- "Homomorphic commitments", i.e. commitments with arithmetic properties: utilise homomorphic properties of elliptic curves
- Prover can prove to verifier that $f(x, y, \ldots) = a$, where f is some set of linear equations, without providing x, y, …
- Zero-knowledge under DLP assumption
- Proofs, not arguments (under hash function collision resistance guarantees)

Pedersen commitments

- Select elliptic curve (we use Secp256k1)
- Select two generator points G, H (random public keys) such as we do not know h such as H = h * G, for instance H.x = SHA256(G.x)
- To commit to an amount a we need a random blinding factor x
- Commitment: C = x * G + a * H
- Blinding factor and amounts operate as private keys
- Now, for two commitments C1, C2:
  C3 = C1 + C2 <=> x3 = x1 + x2; a3 = a1 + a2

Pedersen commitments

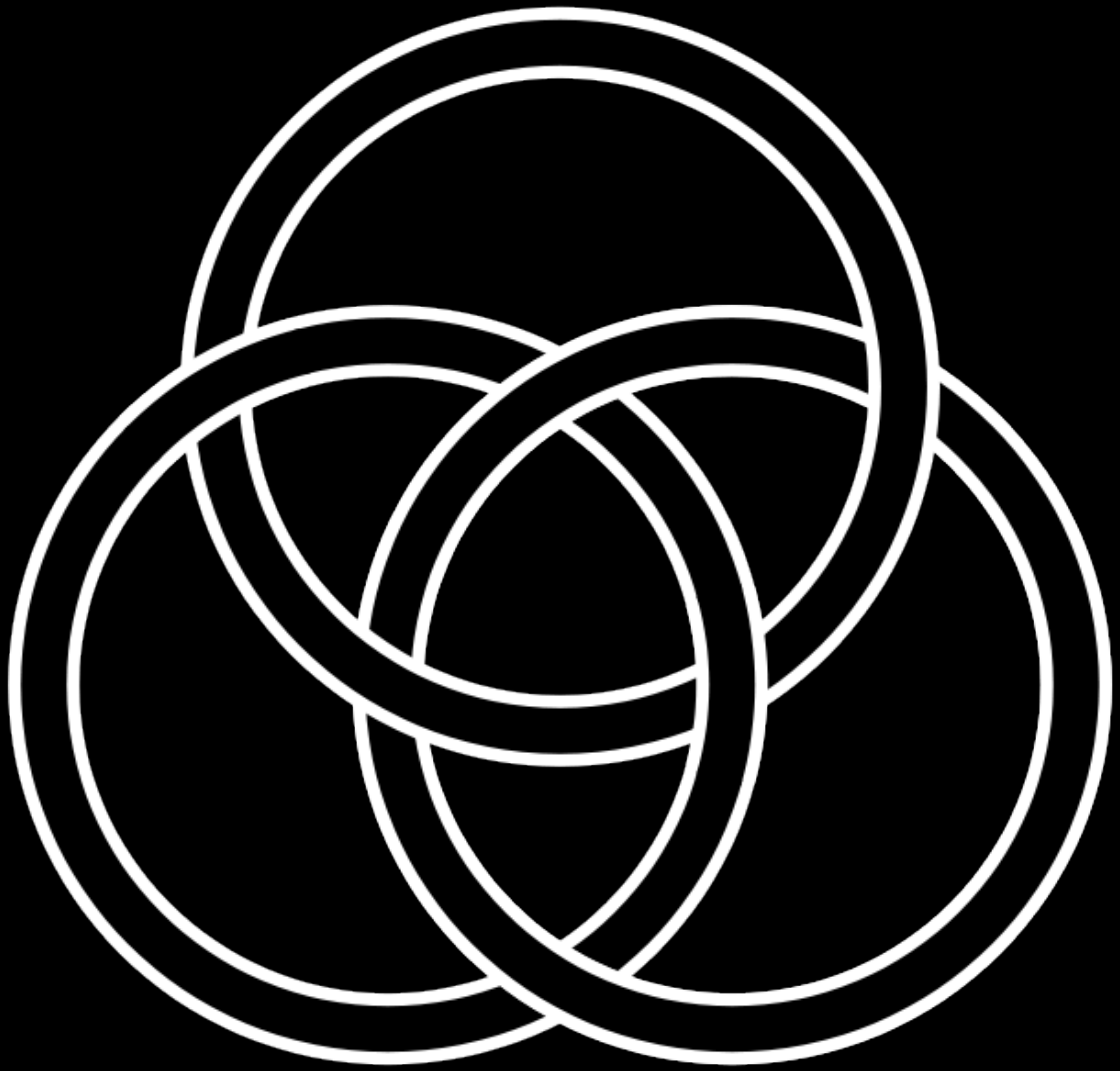
- Alice can prove to everybody that her transaction does not produce any new coins
- Problem: amounts can be negative, so she actually can produce coins in scheme like 1 + 1 = 7 + (-5)
- Solution: range proofs
- Confidential transactions v1: Borromean signatures for range proofs
- Mimblewimble: Bulletproofs (+ other enhancements)

Borromean signatures

A list of proofs that given number a is within the range $0..2^{64}$

64 signatures, for each of the number bits

Multiple optimizations, but still not space-efficient



Bulletproofs

- Designed by Stanford University, University Colleague London and Blockstream (Andrew Poelstra, Pieter Wuille, Greg Maxwell)
- Non-interactive zero-knowledge proofs that require no trusted setup
- Verifying a bulletproof is more time consuming than verifying a SNARK proof
- Used for range proofs in Monero and Mimblewimble, planned for the second version of Confidential Transactions
- Bulletproofs shrink the size of the cryptographic proof for confidential transactions from over 10kB to less than 1kB
- Have multiple applications outside of range proofs

RGB+CA: four dimensions of the privacy

- **Amounts:**

  Pedersen commitments + Borromean signatures range-proofs in CT/CA today

  Pedersen commitments + bulletproofs range-proofs in CT2/RGB tomorrow

- **The fact of transaction (non-public tx):**

  Client-size validation with elliptic curve commitments in RGB

- **Addresses:**

  Merklization of asset owning addresses inside the proofs

- **Assets:**

  Pedersen commitments and encapsulation for multi-asset transfers

Problem #1: Multi-asset graph transfers

Bitcoin transaction closing multiple seals over multiple messages with different genesis state

- Links unrelated proof histories from different graphs into a single graph forever: huge storage impact, DoS, leaking private information on asset ownership
- Links multiple asset transfers into a single proof, leaking private information on amounts and owners

Under LNPBP 1-3 we may have just a single commitment message, i.e. asset transfer/state change proof!

LNPBP-4: Multi-message commitment scheme with provable zero-knowledge properties

- Splits the separate assets (unrelated state histories) into multiple messages
- Commits to each of the messages and combines these commitments
- Hides information about asset types (genesis states) behind the messages
- Uses Pedersen commitments and specially-designed prove scheme to verify that each asset has just a single transfer entry (preventing double spend)

LNPBP-4: Prove distinct genesis states

- Let's identify each genesis state by some 256-bit number (like hash of the genesis state data): d_1 for genesis state #1, d_2 for state #2…
- Create Pedersen commitment $C_i = d_i{*}H + b_i{*}G$ for each d_i with some random blinding factors b_i
- For each pair of the commitments C_i, C_j compute value
  $C_{i,j} = C_i - C_j = (d_i - d_j){*}H + (b_i - b_j){*}G = d_{i,j}{*}H + b_{i,j}{*}G$
- Disclose all $C_{i,j}$ and $b_{i,j}$ to verifier so it may check that
  $C_{i,j} \neq b_{i,j} * G$, implying
  $d_i - d_j = d_{i,j} \neq 0$ and proving d_i and d_j to be distinct genesis states

LNPBP-4: Prove blinding factors

- Still, we have to prove that the provided $b_{i,j}$ are real differences on blinding factors

- Done by prover disclosing $D_{i,j} = (d_{i,j} * H)$ to verifier, so verifier may check that
  $C_{i,j} = b_{i,j} * G + D_{i,j}$

- Rainbow-table attack is possible for the known list of N genesis states, with $N^2/2$ pre-computed $D_{i,j}$ values

Workshop #5

Further LNPBP-4 strengthening

LNPBP-4: Further strengthening

- Use second round of Pedersen commitments over $B_{i,j}$, $C_{i,j}$, $D_{i,j}$
- Use double Pedersen commitment after Christophe Diederichs proposal

Even without LNPBP-4

- For most cases the asset information and history do not leak because of different protocol-specific entropy
- RGB has less space requirements than blockchain
- No problem of intermixing on Lightning channels
- RGB is more private than Blockchain or CA
- But as cypherpunks we still prefer to find a way for efficient zero-knowledge asset hiding

Part IV: State transition details

How to store the state transition?

- List seal(s)
- Define an associated state, bound to a seal
- Potentially add script, associated with seals
- Add metadata, covering all seals

Defining seals

- The state should be bound to seals
- Each new state owner will have independent seal and part of the state
- We need to obscure the exact seals (transaction outputs) to enhance the privacy of the owners
- This can be done with Merklization procedure

State types

- Amount (the sum of inputs must be equal the sum of outputs)
- Bytestring with some script-defined validation rules (more generic case)
- Maybe use just a bytestring type with validation rules defined in Simplicity?

Defining state

- The state can’t be merklerized, since for some state types (amount type) we may need to prove certain properties (like the absence of double-spend)

- We can utilise mechanics of Confidential transactions with Bulletproof range proofs as the most efficient mechanism, which will be used in Confidential Assets v2 as well

Scripting

- We need to maintain ability to add scripting conditions to sealed state in the future

- Simplicity provides the best formally-verifiable system

- The scripts can only provide additional constraints on the state transition validity; i.e. if the state transition is invalid due to wrong commitment, the script can’t make it valid

Simplicity

- Proposed and developed by Russel O’Conner, Blockstream
- Planned to be included into Elements and Liquid
- Possible “Softfork to end all forks” (Adam Back)
- Formal semantics
- Formally-verified language with proofs on execution
- Succinct (complete Schnorr signatures are just few kB)

Why not WASM

- No formal verification
- No execution constraints
- Non-succinct
- Maintained outside of Bitcoin/cypherpunk community
- Different goals (web language, instead of finance)

Plutus?

- Haskell subset
- Most of the Simplicity qualities

- Non-succinct
- Younger than Simplicity
- IOHK commercial project (while still opensource)
- No plans for adoption in other LNP/BP projects

EVM and Solidity?

Ethereum magical creatures

Governance: let’s make government eternal

Layer 3 (apps)

- dApps that nobody uses
- DeFi speculations
- Hackable DAOs
- “Unstoppable” world single-threaded computer

Layer 2 (scalability promises)

- Mystical non-existing Plasma
- Raiden: copy & paste Lightning Network
- ERC20 Bitcoin

Layer 1 (blockchain)

- Accounts
- Turing complete
- Sharding
- PoS
- Whisper
- Swarm

RGB & SPECTRUM:
BITCOIN MAXIMALISTS COMING OUT…
…TO BE JEALOUS FOR ETHEREUM MAGIC

EVM and Solidity?

NO

‘cause we are toxic maximalists!

Metadata

- Is important for special state transitions, like primary or secondary asset issuance, providing the information about total supply etc.

- Pre-defined set of types:
  - int (8, 16, 32, 64, 128, 160, 256 bit)
  - floats/doubles (16, 32, 64 bit)
  - arrays of these types
  - structures of these types

- Semantics (string encoding, integer meanings like EC point, hash) are defined at the schema level

Final state transition structure

- Number of seals, defined by this state transition
- Merkle root for the seals, defined by this state transition
- List of the state data for each of the seals
- Metadata fields (may be 0)
- Validation script bytecode in Simplicity (optional)
- Closed seals are not a part of the state transition;
  they are detected via transaction inputs!

Part V: Serialisation

Serialisation: 3 types

- **Commitment serialisation**: used to digest the original message and create or verify the commitments on each protocol level
- **Network serialisation**: used to transfer the data structure across network so other participants may parse them
- **Storage serialisation**: used to store the necessary client-validated data off chain in an efficient way

These three may be very different and require independent standards

Serialisation

	Commitment	Network	Storage
Type of standards	Common (LNPBP)	Common (LNPBP)	Vendor-specific
Important for consensus	+++	++	—
Results of hard fork	Loss of state/assets	Broken communications	None
Must optimise storage	No	Yes	Yes
Must optimise for speed	Yes	No	No
Potential attack surface	++	+	0
May cover multiple layers of protocols	No	Yes	Vendor-specific

Commitment serialisation layers for RGB

- Merklization of seals: LNPBP-?
- State transition data: LNPBP-?
- Multi-message commitments: LNPBP-4

Network serialisation for RGB

- **Ownership proof**: state transition history for a given owned state *when Alice needs to provide Bob with the proofs of her ownership of some state (like assets)*

- Per-request specific data for P2P network protocols as used in Lightning Network and Bitcoin

Rules for commitment serialisation

- Do not compress the data
- Use deterministically-defined value length
  (Bitcoin’s VarInt is a bad practice)
- No pointers/offsets/shifts, no linked lists
- Merkle trees must also commit to the depth of each branch
- Define bounds for each type validity
- Must be composed of the nested digests (prevention of length-extension attacks)
- Must be prefixed with protocol-specific tag before commitment
- First 8 bytes must deterministically define the length of the committed data

Why proper API is important?

```
pub trait Verifiable<CMT>
   where CMT: Commitment
{
   fn verify(&self, commitment: &CMT) -> bool;
}

pub trait Committable<CMT>: Verifiable<CMT>
    where CMT: StandaloneCommitment<Self>
{
   fn commit(&self) -> CMT;
}

pub trait EmbedCommittable<CMT>: Verifiable<CMT>
    where CMT: EmbeddedCommitment<Self>
{
   fn commit_embed(&self, container: &CMT::Container) -> Result<CMT, CMT::Error>;
}
```

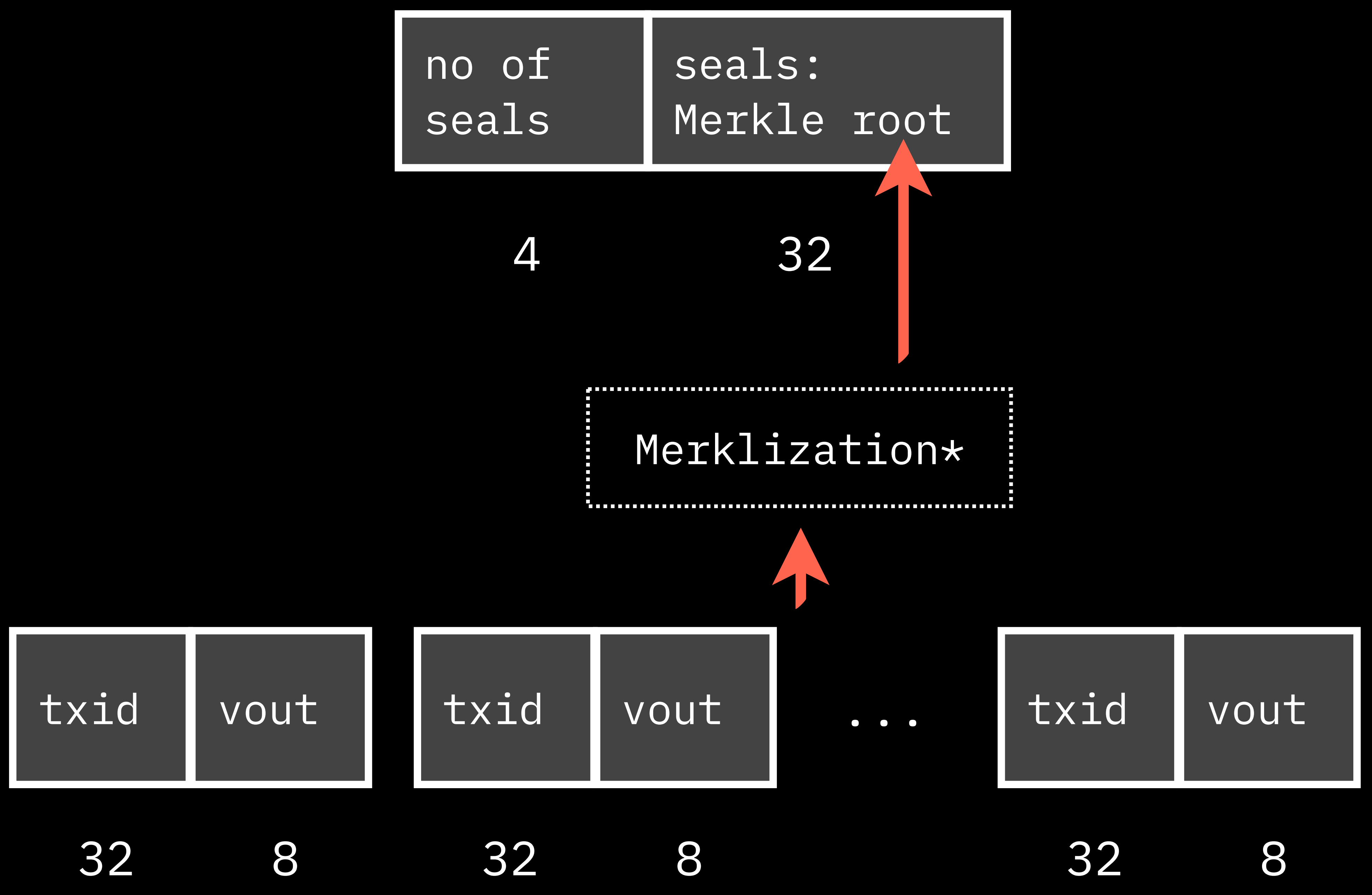
State transition commitment serialisation

no of seals
4

Every piece of data is a hash

- Fixed-size value
- A lot of space for optimization via batching and framing

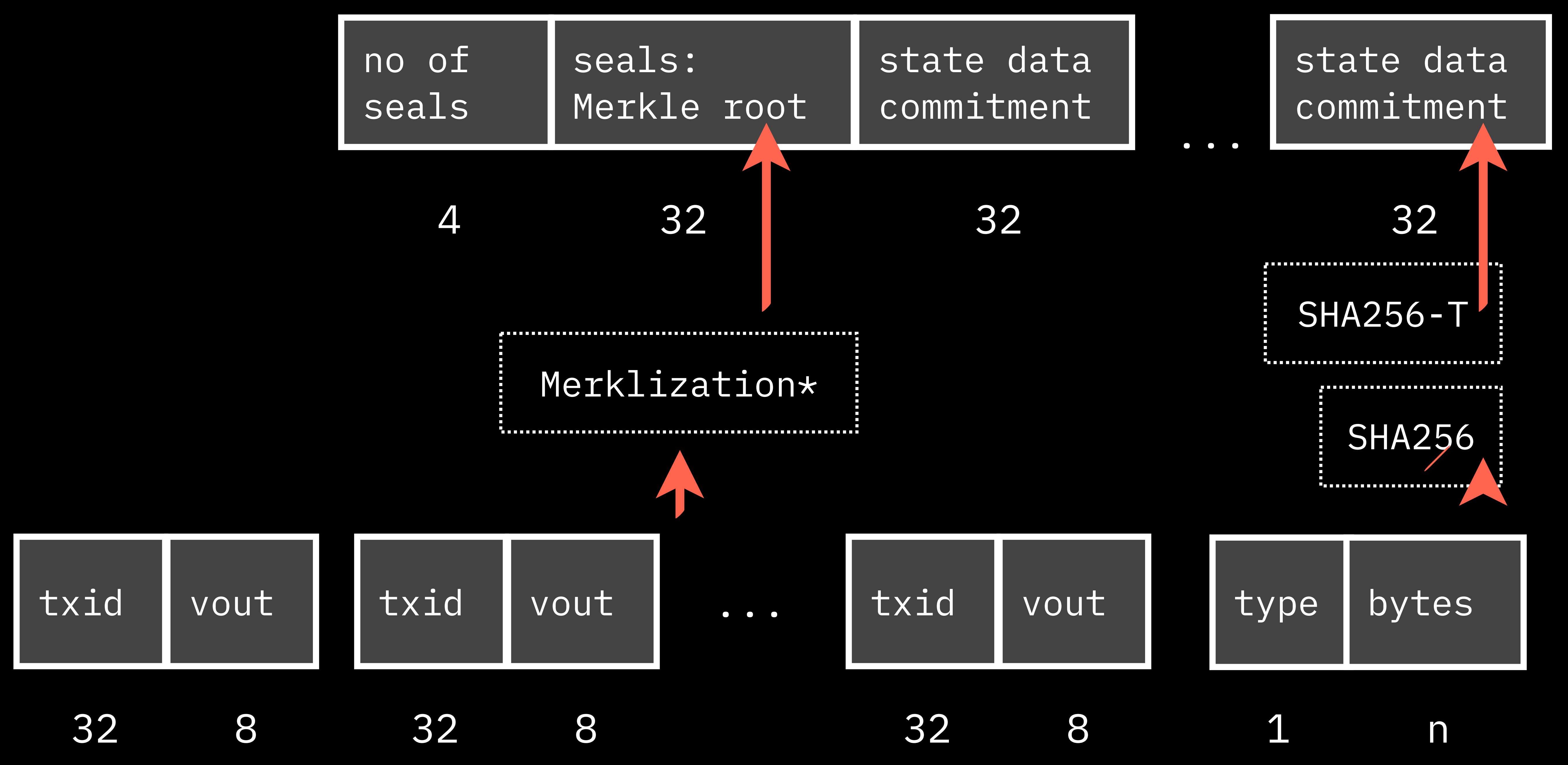
State transition commitment serialization



Merkle tree*

- Uses tagged hashes as used in BIP-Taproot
- Commits to the branch depth

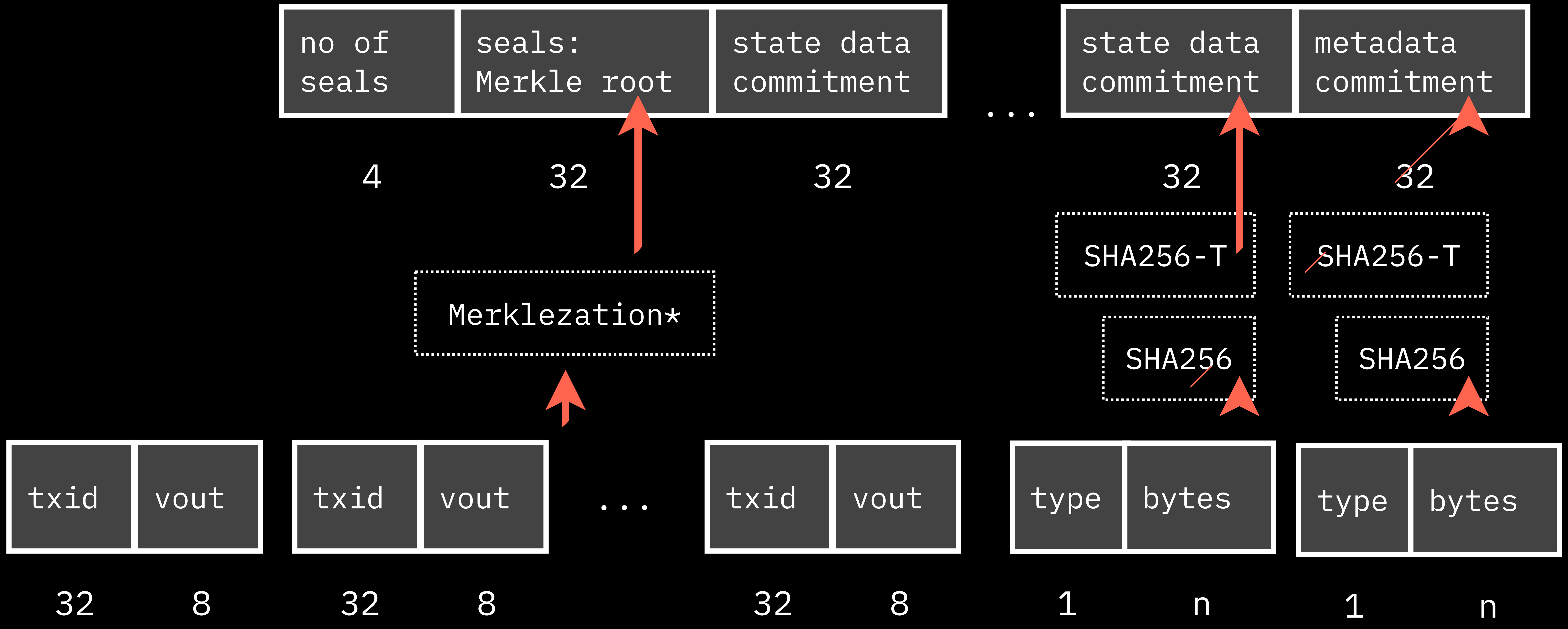
State transition commitment serialisation



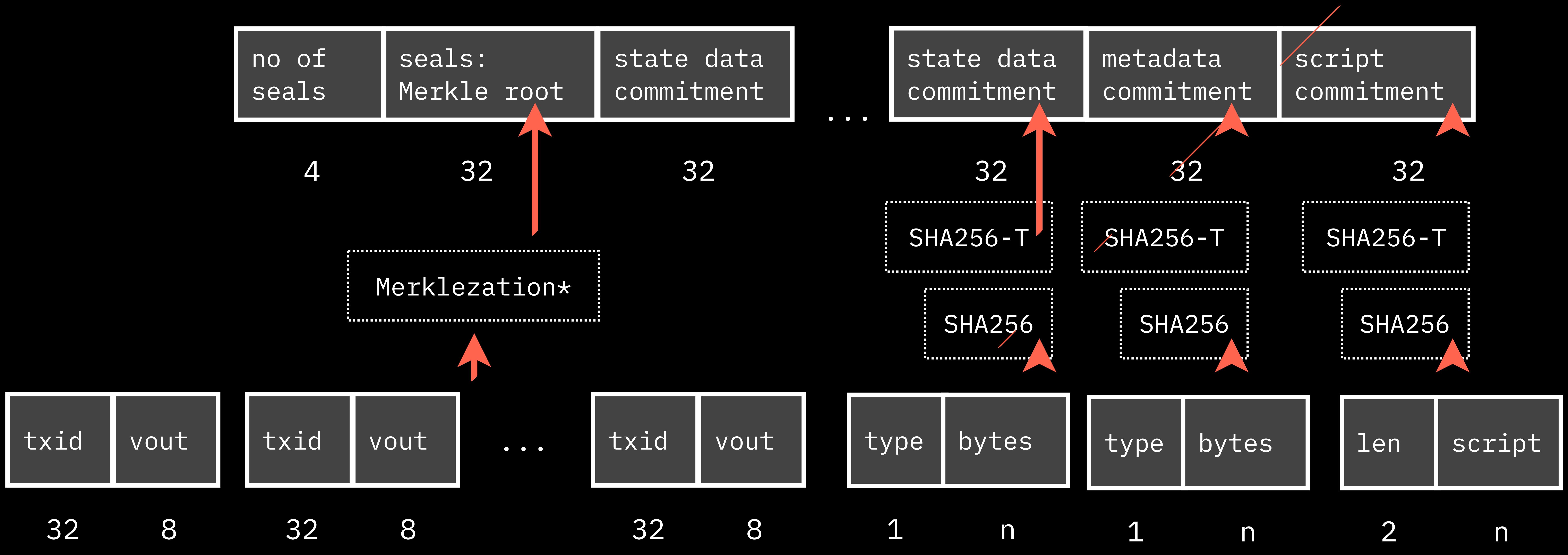
Double + tagged hashes

- Prevention of length-extension attacks

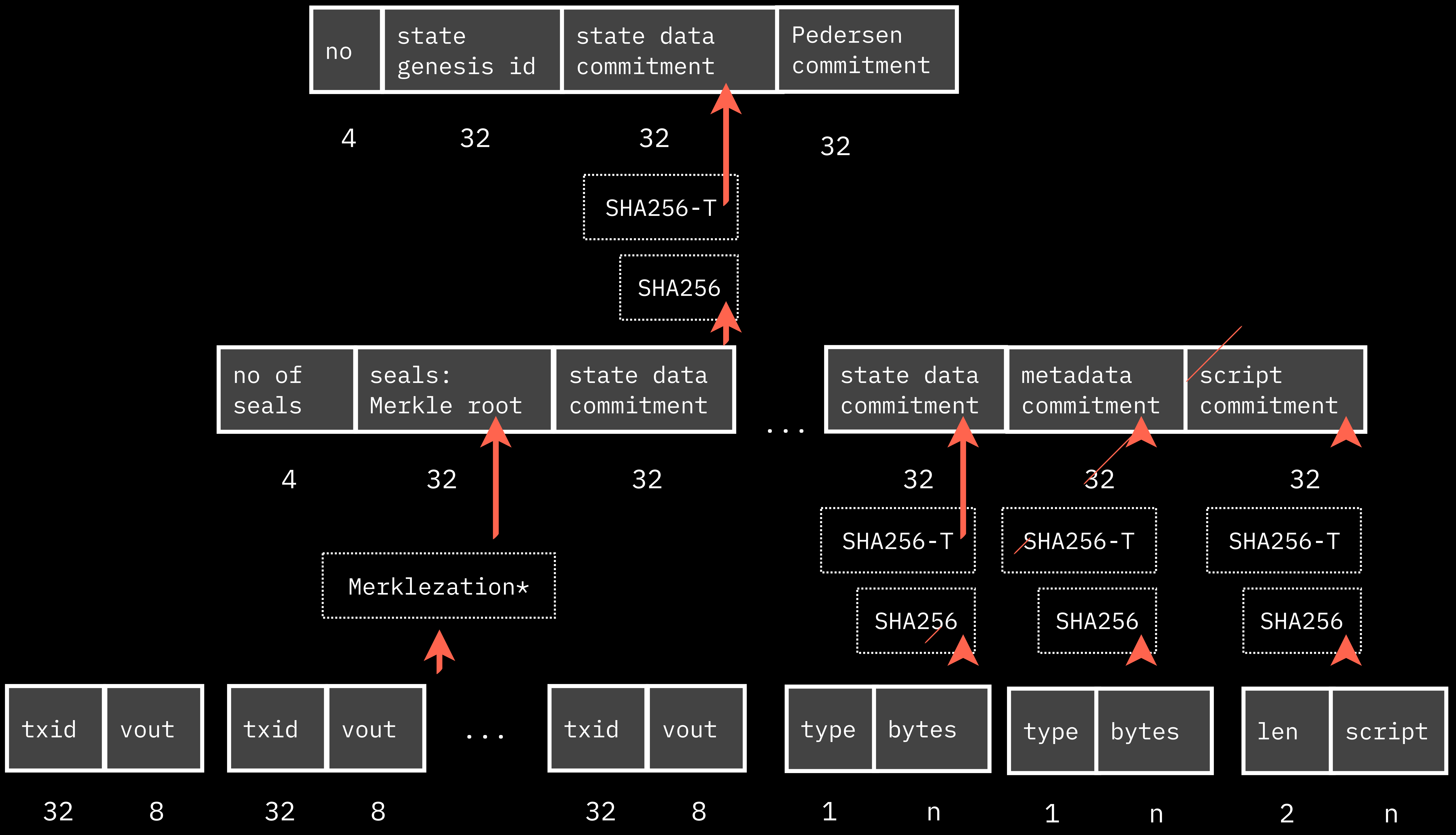
State transition commitment serialisation



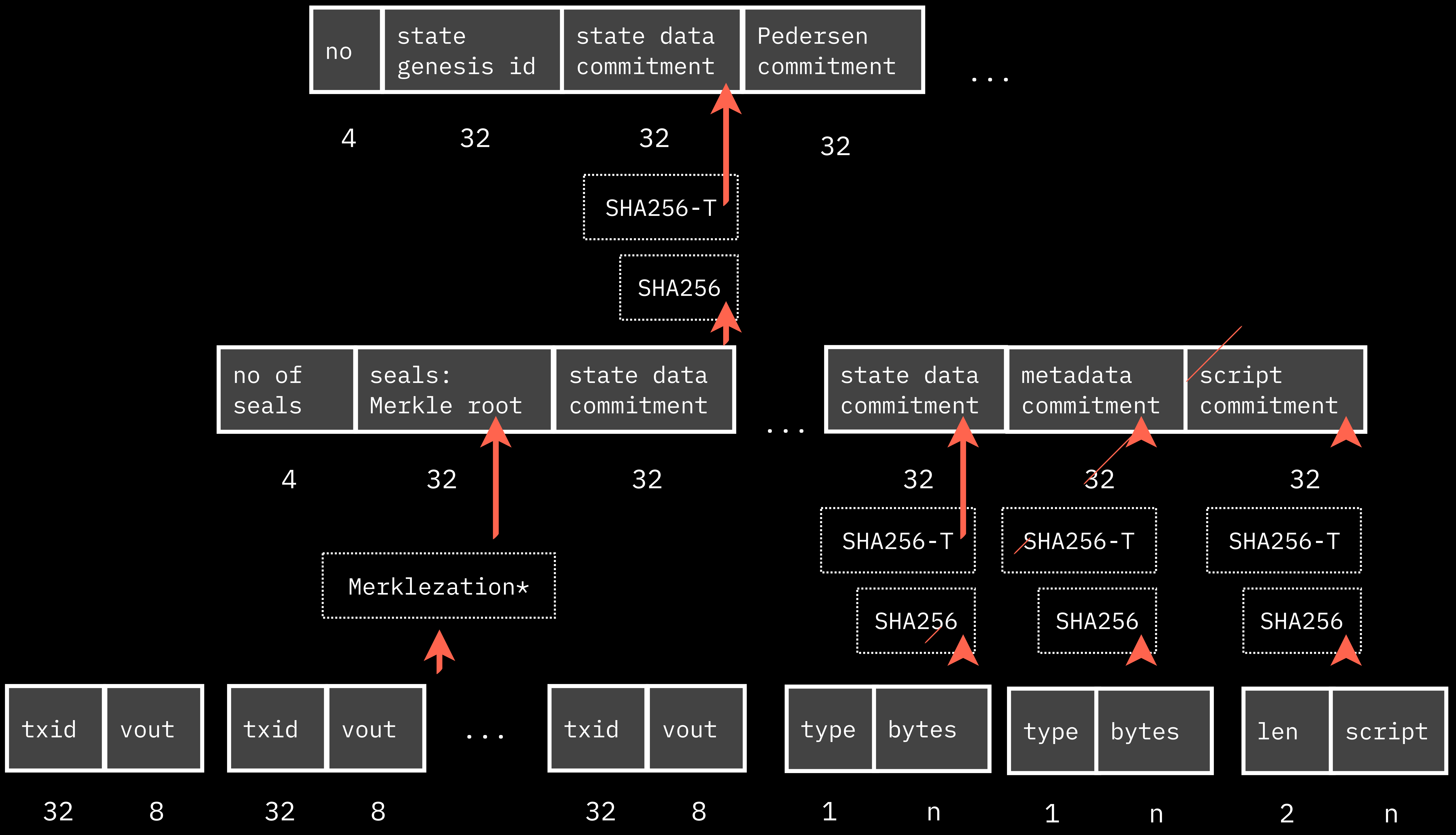
State transition commitment serialisation



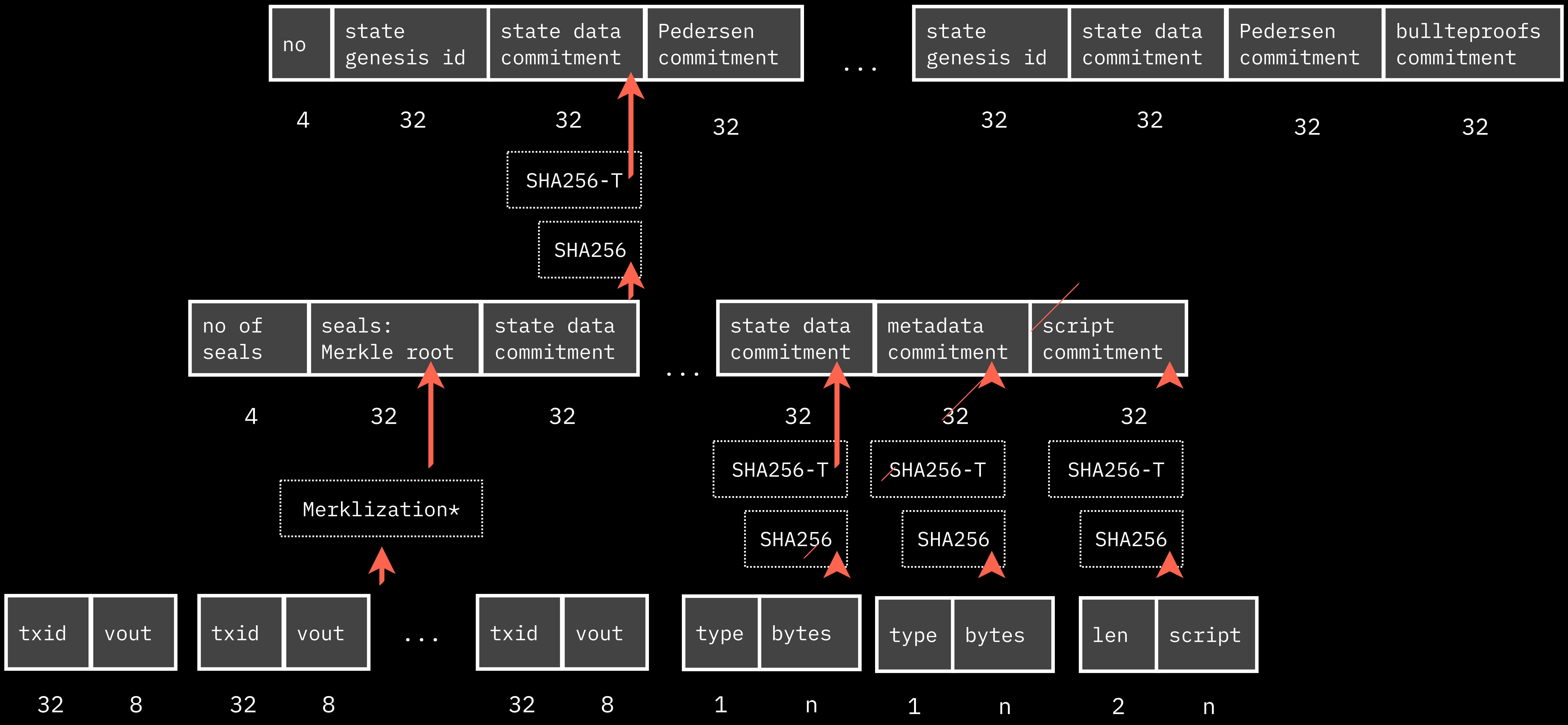
State transition commitment serialisation



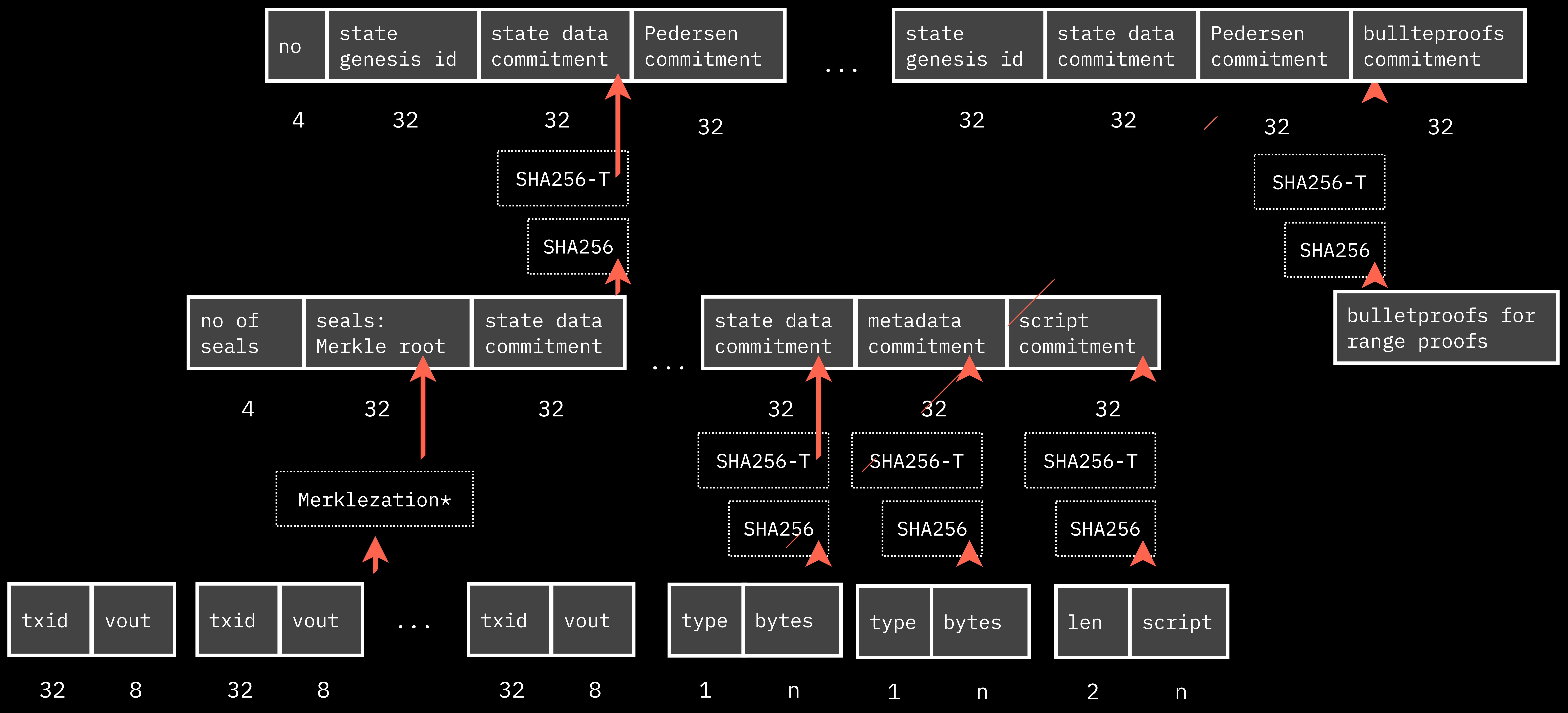
State transition commitment serialisation



State transition commitment serialisation



State transition commitment serialization



Part IV: Schemata

Schemata

- Defines types of state transitions
- Defines types of seals transitions
- Defines semantics for state and Simplicity scripts for validating the state
- Defines semantics for metadata
- Defines/references Simplicity script modules
- Defines additional constraints on each type of state transition:
    - Which seals may be defined by the state transition
    - Which state may be associated with each seals
    - Which metadata is required, optional and prohibited
    - Which additional scripts and with which constraints may be defined

Schemata describes the actual requirements for the state transition validation outside of the level of Bitcoin script commitments.

It allows simple updates without software modifications, so that wallets, explorers, LN nodes etc could accept new types of assets without any code changes

Sample: Fungible assets schema draft

```
name: Fungible assets
standard: LNPBP-6
schema_ver: 1
field_types:
  ver: u16
  schema: u256
  network: u8:0-6
  ticker: utf8:16 # Strings always have a fixed length specified after a colon;
                  # for shorter values the rest is extended with 0 bits
  title: utf8:64
  description: utf8:256
  url: utf8:1024
  fractional_bits: u8:0-256 # Integers always have a valid value bounds attached
  max_supply: amount # Amount is a special type, always equivalent to u256, plus requiring special validation rules
                     # (for any operation, sum of outputs must be equal to the sum of inputs, and must not overflow)
  dust_limit: amoun
seal_types:
  assets: amount
  issuance: none
  pruning: none
```

Sample: Fungible assets schema draft

```yaml
proof_types:
  - name: primary_issue
    fields:
      ver: single
      schema: single
      network: single
      ticker: single
      title: single
      description: optional
      url: optional
      fractional_bits: single
      max_supply: optional
      dust_limit: single
    seals:
      assets: many
      issuance: optional
      pruning: single
    script: none
```

```yaml
- name: secondary_issue
  closes:
    issuance: single
  fields:
    description: optional
    url: optional
  seals:
    assets: many
    issuance: optional
    pruning: single
  script: none
```

```yaml
- name: history_prune
  closes:
    pruning: single
  fields: [ ]
  seals:
    assets: many
    pruning: single
  script: none

- name: asset_transfer
  closes:
    assets: many
  fields:
  seals:
    assets: many
  script: none
```

Encoded schema

```
00000000  03 52 47 42 01 00 00 00  00 00 00 00 00 00 00 00  |.RGB............|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  00 00 00 00 00 00 00 09  03 76 65 72 01 06 73 63  |.........ver..sc|
00000030  68 65 6d 61 10 06 74 69  63 6b 65 72 0b 05 74 69  |hema..ticker..ti|
00000040  74 6c 65 0b 0b 64 65 73  63 72 69 70 74 69 6f 6e  |tle..description|
00000050  0b 03 75 72 6c 0b 0a 6d  61 78 5f 73 75 70 70 6c  |..url..max_suppl|
00000060  79 0a 0a 64 75 73 74 5f  6c 69 6d 69 74 09 09 73  |y..dust_limit..s|
00000070  69 67 6e 61 74 75 72 65  31 04 06 61 73 73 65 74  |ignature1..asset|
00000080  73 01 09 69 6e 66 6c 61  74 69 6f 6e 00 07 75 70  |s..inflation..up|
00000090  67 72 61 64 65 00 07 70  72 75 6e 69 6e 67 00 05  |grade..pruning..|
000000a0  0d 70 72 69 6d 61 72 79  5f 69 73 73 75 65 07 02  |.primary_issue..|
000000b0  00 01 03 00 01 04 00 01  05 00 01 06 00 01 07 01  |................|
000000c0  01 08 00 01 00 04 00 01  ff 01 00 01 02 01 01 03  |................|
000000d0  01 01 0f 73 65 63 6f 6e  64 61 72 79 5f 69 73 73  |...secondary_iss|
000000e0  75 65 02 05 00 01 08 00  01 01 01 01 01 03 00 01  |ue..............|
000000f0  ff 01 00 01 03 01 01 0e  75 70 67 72 61 64 65 5f  |........upgrade_|
00000100  73 69 67 6e 61 6c 03 00  01 01 01 00 01 08 00 01  |signal..........|
00000110  01 02 01 01 01 02 01 01  0d 68 69 73 74 6f 72 79  |.........history|
00000120  5f 70 72 75 6e 65 00 01  03 01 01 02 00 01 ff 03  |_prune..........|
00000130  01 01 0e 61 73 73 65 74  5f 74 72 61 6e 73 66 65  |...asset_transfe|
00000140  72 01 00 00 01 01 00 01  ff 01 00 00 ff           |r............|
```

Sample: Genesis state for a fungible asset issue


```yaml
type_name: primary_issue
fields:
  ver: 1
  schema: sm1p9au5tw58z34aejm6hcjn5fnlvu2pdunq2vux5ymzks33yffrazxskfnvz5
  network: bitcoin:testnet
  ticker: PLS
  title: Private Company Ltd Shares
  description: Sample asset of no value
  fractional_bits: 0
  dust_limit: 1
  max_supply: 100_000_000
seals:
  - type_name: assets
    outpoint: 5700bdccfc6209a5460dc124403eed6c3f5ba58da0123b392ab0b1fa23306f27:0
    amount: 1_000_000
  - type_name: issuance
    outpoint: 5700bdccfc6209a5460dc124403eed6c3f5ba58da0123b392ab0b1fa23306f27:1
  - type_name: pruning
    outpoint: 5700bdccfc6209a5460dc124403eed6c3f5ba58da0123b392ab0b1fa23306f27:2
```

Client-side state validation process

- Ensure the correctness of the genesis state (trusted publishers) and used Schema
- For each state transition:
    - Check its commitment validity on 4 levels: multi message commitment (LNPBP-4), transaction commitment (LNPBP-3), script commitment (LNPBP-2), public key tweak (LNPBP-1)
    - Check the closed single-use seals validity (“inputs”) and validity of the related state transitions
    - Check the defined single-use seals validity (“outputs”)
    - Check the validity of the new state according to its internal rules (including Simplicity script)
    - Check the validity of the state transition according to the Schema (correct inputs, correct outputs, types of state, metadata and script constraints)