LNP/BP Standards Association

Modernizing RGB with Taproot

Dr Maxim Orlovsky,
Pandora Core AG

FBDE A843 3DDC 1E69 FA90 C35E FFC0 2509 47E5 C6F7

Yesterday rust **miniscript** made first release candidate (RC1) providing full Taproot support

Since February release of rust bitcoin, it was the last major obstacle on a way of **RGB completion & release**

LNP/BP Association in the following weeks will release

v0.6.0 of all out main libs & products

fully supporting Taproot:

Client-side-validation, Descriptor wallet,

BP Core, RGB Core, LNP Core,

RGB Node, LNP Node

Today we will refactor use of deterministic bitcoin commitments in RGB

- Design and exclusively use Taproot OP_RETURN
- Put all RGB commitments into a single output
- Change output selection mechanism
- Avoid interactive complexity & asset privacy leaks in LN and Coin/PayJoin protocols

Deterministic bitcoin commitments
allow provable uniqueness of the commitment

Deterministic bitcoin commitments are based on generic commitment schemes (w/o provable uniqueness):

OP_RETURN,
pay-to-contract, sign-to-contract

Commitment schemes

- **Output-based**
  - OP_RETURN commitments
  - Pay-to-contract commitments (**P2C**)
  - “Tap-to-contract” commitments – introduced in LNPBP-3
  - Tap-OP_RETURN (“**tapret**”) commitments
- **Input-based**
  - Sign-to-contract commitments (**S2C**)

Protocols producing commitments

- Confidential transactions with assets (Elements project): S2C
- Open time stamps: OP_RETURN, in the future: P2C, S2C
- RGB: P2C, S2C
- Taproot: P2C-type of tweaks
- OMNI etc: OP_RETURN
- Bitcoin-based identities: S2C
- Other future single-use-seal-based protocols: ?

Deterministic bitcoin commitment

MUST has a **uniquely** and **deterministically** defined place where commitment can be present

- inside **pubkey** or **signature** (for P2C/S2C-based commitments)
- inside **scripts** (WitnessScript/RedeemScript/TapTree)
- inside **transaction**

DBC scheme (protocol) must define:

- Which part of transaction is used for commitments (inputs, outputs)
- Deterministic selection of specific input or output containing commitment
- Deterministic & unique encoding of the commitment into the input/output data structures

Which part of transaction is best for single-use-seal DBC

Input

- Clearly defines selection of input (the same input that closes single-use-seal)
- No information aggregation from multiple parties (important for LN, PayJoin, CoinJoin)
- The only scheme possible is S2C:
  - Not compatible with Taproot (yet), many issues (MuSig, post-TapScript scripts)
  - Not compatible with zero-signature spendings (used in LN)
  - Not compatible with HSM/HW

Output

- Needs interactive commitment construction in multi-party transactions (LN, PayJoin, CoinJoin): complexity, information leackage
- Complexity in defining output with commitment
- Commitment always changes address: hard to track funds
- Big selection of different commitment schemes

Current DBC in single-use-seal & RGB

- **Allows selection between S2C & P2C,**
  but S2C scheme still not implemented and depends on upstream work in secp256k1 libraries (C code etc) – it will take many months

- In P2C:
  - **Output is selected basing on fee and asset id,**
    but if multiple RGB assets are spent, many/all outputs may be tweaked, paying to address may not be possible
  - **Fee adjustments allows changing which outputs are affected,**
    but creates recursive complexity in transaction construction and complexities LN compatibility
  - **All types of outputs are supported (bare, pre-segwit, segwit, taproot)**
    leading to huge amount of code, complexity of HW integration

RGB prototype was released in May 2021
with RGB Core & Node v0.4.0

We built consumer-facing products with it
(MyCitadel Wallet, Bitcoin Pro, RGBex.io)

The complexity of wallet implementation
due to RGB commitment schemes was
tremendous

Which part of transaction is best for single-use-seal DBC

Input

- Clearly defines select… …put (the same input that closes s… …)
- No informati… …rom mul… parties (… …N, PayJ… CoinJ…
- … …e possi…
- …mpatibl… …yet), many …ues (M… …ipt scripts)
  - Not … …ero-signature sp… …n LN)
  - Not … …e with HSM/HW

not applicable for RGB & single-use-seals

still may be used for other types of DBCs

Output

- Needs interactive commitment construction in multi-party tr… …ctions (LN, PayJoin, CoinJoin): c… …y, information leackage
- Complexity in de… …utput with commitment
- Commit… …ays changes address: hard to … …unds
- Big selection of different commitment schemes

the only option for RGB

Let’s try to solve the problems
of output-based commitments one by one

— wallet & HSM integration complexity & compatibility
— problems with multiparty transactions (LN, Coin/PayJoin)
— implementation complexity

Let’s start by looking at DBC components landscape

Output-based DBCs

Specific forms of commitment:

Pay-to-contract (P2C)

pubkey	keyset
P2PK	+ Bare
P2PKH	P2SH
P2WPKH	P2WSH
P2WPKH/P2SH	P2WSH/P2SH

Return-to-contract (R2C)

op_return	tap-return
OP_RETURN	P2TR

Input-based DBCs

ECDSA S2C

ecdsasig	ecdsaset

Pre-segwit, SegWit v0

Schnorr S2C – non-existing

schnorrsig	schnorrset	musig

Taproot and post-taproot

Output-based DBCs

Specific forms of commitment:

Pay-to-contract (P2C)

pubkey	**keyset**
P2PK	+ Bare
P2PKH	P2SH
P2WPKH	P2WSH
P2WPKH/P2SH	P2WSH/P2SH

Return-to-contract (R2C)

op_return	**tap-return**
OP_RETURN	P2TR

Input-based DBCs

ECDSA S2C

ecdsasig **ecdsaset**

Pre-segwit, SegWit v0

Schnorr S2C – non-existing

schnorrsig **schnorrset** **musig**

Taproot and post-taproot

Other hypothetical schemes:

tap-annex ~~**extra-witness-item**~~

Output-based DBCs

Specific forms of commitment:

Pay-to-contract (P2C)

pubkey	keyset
P2PK P2PKH P2WPKH P2WPKH/P2SH	+ Bare P2SH P2WSH P2WSH/P2SH

Return-to-contract (R2C)

op_return	tap-return
OP_RETURN	P2TR

Determining single transaction output containing commitment:

- fee
- nseq
- nlocktime
- 1st output
- fee + protocol id
- last output

Input-based DBCs

ECDSA S2C

ecdsasig	ecdsaset
Pre-segwit, SegWit v0	

Schnorr S2C – non-existing

schnorrsig	schnorrset	musig
Taproot and post-taproot		

Other hypothetical schemes:

- tap-annex
- ~~extra-witness-item~~

Fee+protocol output selection

Why was introduced

- Dynamically select output which will contain commitment by changing fee
- Allows separate commitments under different RGB contracts to different outputs (mitigation of interactive construction)
- At the same time avoids chainanalisys from detecting which output is used for commitments (by analyzing fee)

Disadvantages

- Requires access to blockchain (fee computation)
- Makes transaction construction recurrent and complex
- With multiple RGB contracts
  - makes impossible to pay to address, since all outputs get tweaked
  - still requires interactive protocols and RGB contract data leackage

Output-based DBCs

Input-based DBCs

Specific forms of commitment:

Pay-to-contract (P2C)

- **pubkey**: P2PK, P2PKH, P2WPKH, P2WPKH/P2SH
- **keyset**: + Bare, P2SH, P2WSH, P2WSH/P2SH

Return-to-contract (R2C)

- **op_return**: OP_RETURN
- **tap-return**: P2TR

ECDSA S2C

- **ecdsasig**
- **ecdsaset**

Pre-segwit, SegWit v0

Schnorr S2C – non-existing

- **schnorrsig**
- **schnorrset**
- **musig**

Taproot and post-taproot

Determining single transaction output containing commitment:

- **fee**
- **nseq**
- **nlocktime**
- **1st output**
- **last output**
- ~~**fee + protocol id**~~

Other hypothetical schemes:

- **tap-annex**
- ~~**extra-witness-item**~~

Output-based DBCs

Specific forms of commitment:

Pay-to-contract (P2C)

pubkey	**keyset**
P2PK P2PKH P2WPKH P2WPKH/P2SH	+ Bare P2SH P2WSH P2WSH/P2SH

Return-to-contract (R2C)

op_return	**tap-return**
OP_RETURN	P2TR

Determining single transaction output containing commitment:

fee **nseq** **nlocktime** **1st output**

~~**fee + protocol id**~~ **last output**

Input-based DBCs

ECDSA S2C

ecdsasig **ecdsaset**

Pre-segwit, SegWit v0

Schnorr S2C – <u>non-existing</u>

schnorrsig **schnorrset** **musig**

Taproot and post-taproot

Other hypothetical schemes:

tap-annex ~~**extra-witness-item**~~

Pay-to-contract

- Requires wallet to keep track of key tweaks for
  - Balance computing
  - Spending outputs
- Makes compatibility with hardware wallets a complex issue (wallets need to tweak private keys to spend output)
- Not compatible with anyone-can-spend outputs (no public keys), used in LN
- Complex multisig wallet management
- Compatible with all output types

Return-to-contract

- Requires wallet to keep track of commitment values for
  - Balance computing
- No hardware wallet compatibility problems
- Compatible with anyone-can-spend outputs
- Does not affect multisig wallet management
- Compatible with specific output types (OP_RETURN, P2TR)

Output-based DBCs

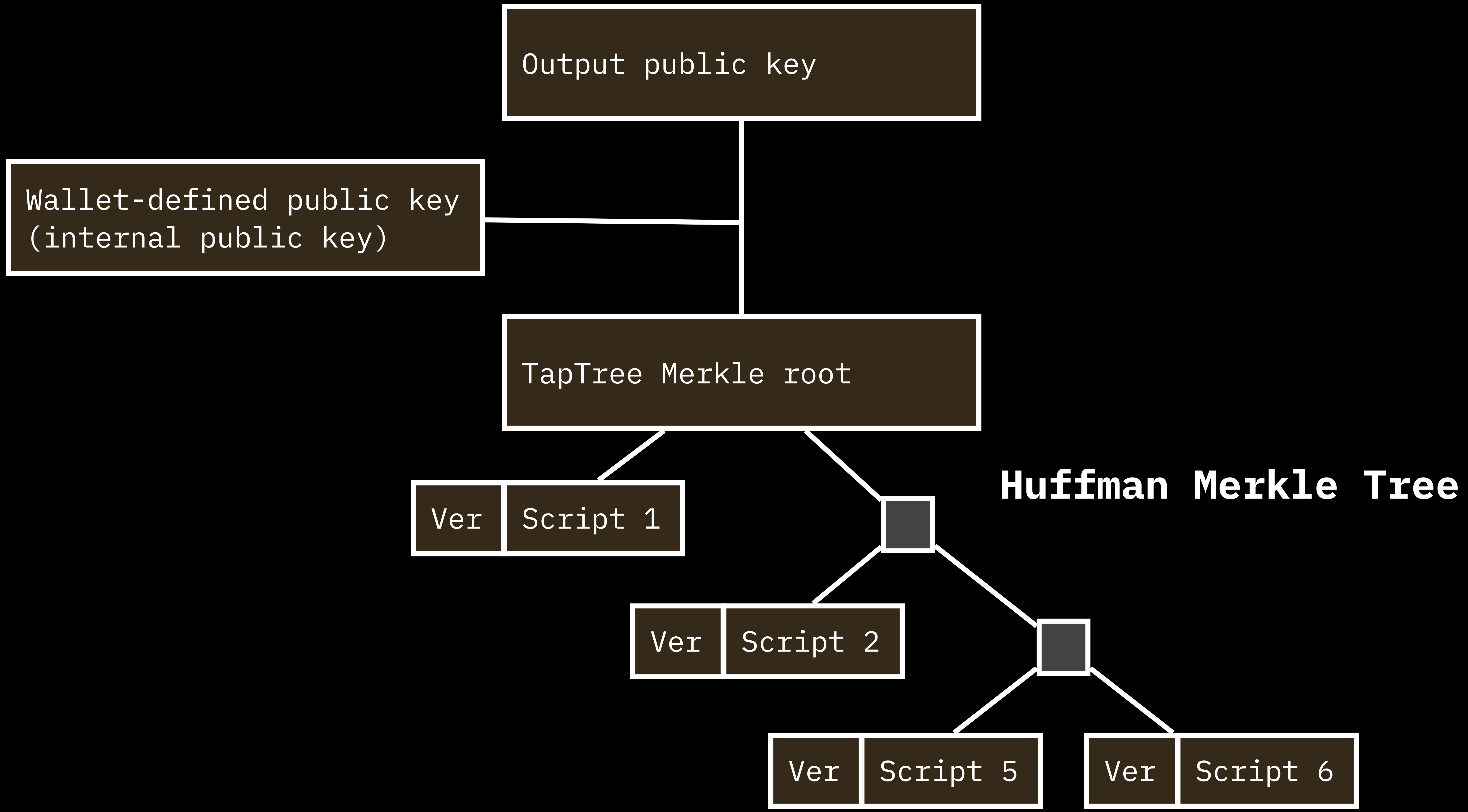
Specific forms of commitment:

Pay-to-contract (P2C)		Return-to-contract (R2C)	
pubkey	**keyset**	**op_return**	**tap-return**
P2PK P2PKH P2WPKH P2WPKH/P2SH	+ Bare P2SH P2WSH P2WSH/P2SH	OP_RETURN	P2TR

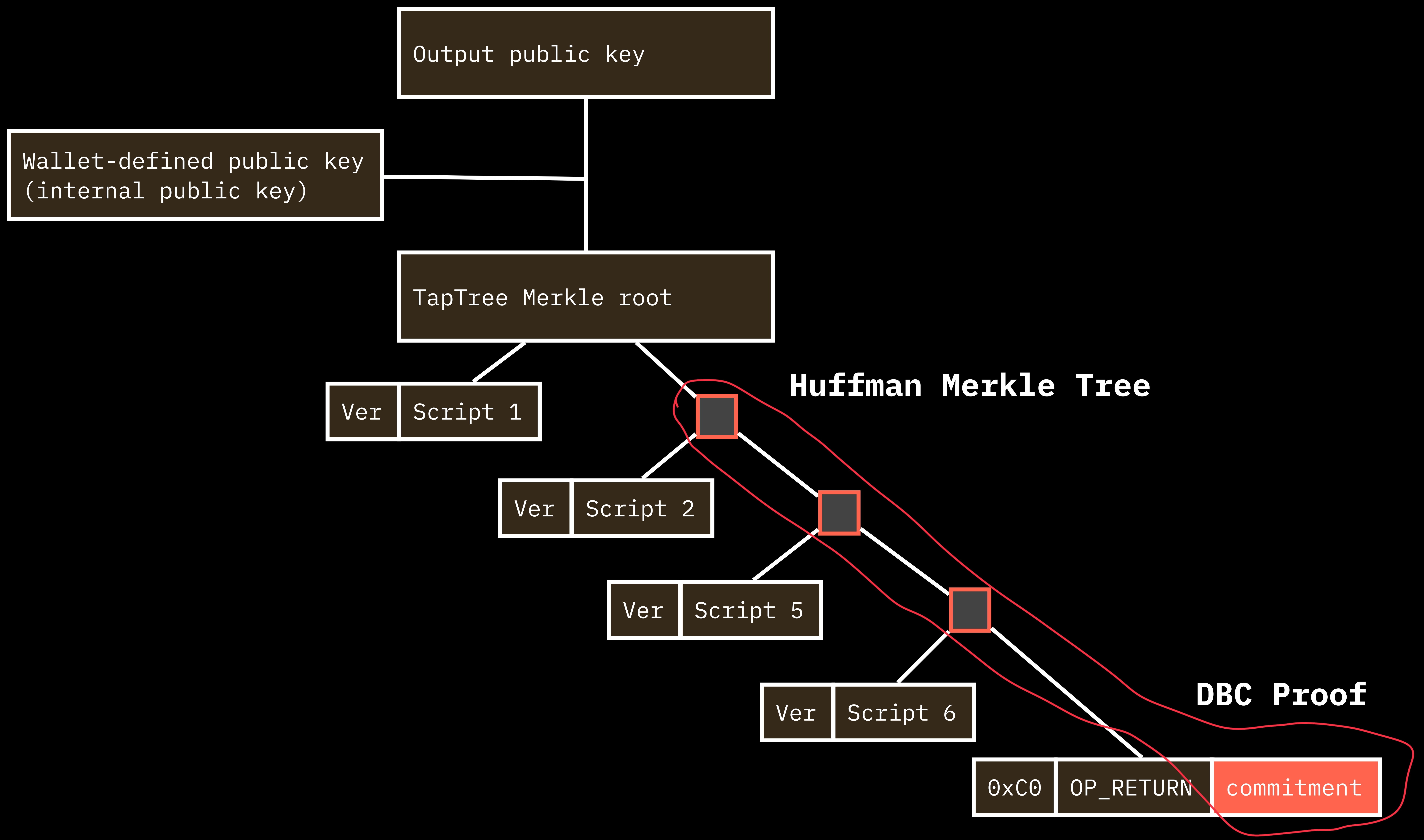
Determining single transaction output containing commitment:

nseq	**nlocktime**	**1st output**
		last output

Designing tap-return (“tapret”) DBCs



Designing tap-return (“tapret”) DBCs



Tap-return (“tapret”) DBCs

- Commitment:
  - add rightmost tapscript `` `OP_RETURN` `` to TapTree
    if the previously rightmost script hash < commitment hash,
    add two nodes, where one is hidden with hash = hash - 1
  - update rightmost merkle path, merkle root and output key
  - keep/store internal key and new merkle path to the rightmost commitment node
- Verify
  - take commitment value, save merkle path and internal key
  - create OP_RETURN tap script with commitment and merkle-proof it toproduce the same scriptPubkey/address as in the mined tx

Output-based DBCs

Specific forms of commitment:

Return-to-contract (R2C)

op_return **tap-return**

OP_RETURN P2TR

Determining single transaction output containing commitment:

nseq **nlocktime** **1st output**

last output

nSeq/nLockTime

- Collisions with possible future consensus soft forks
- Hard to design in a way which will prevent unchain analysis and not interfere with LN features

First/last output

- Hard to mitigate chainanalysis: first/last output will be change output
  - signal output number in single-use-seal definition
  - use multiple change outputs

Output-based DBCs

Specific forms of commitment:

Return-to-contract (R2C)

tap-return

P2TR

Determining single transaction output containing commitment:

nseq **nlocktime** **1st output**

last output

Feasibility analysis

Current P2C DBC (LNPBP1-3)

- Many (most of) outputs in transaction are tweaked, making hard to pay to an address
- Compatibility problems with hardware wallets
- Complexity in wallet implementation
- Complexity in transaction construction due to fee adjustment
- Lot of complex code to handle different type of transaction outputs

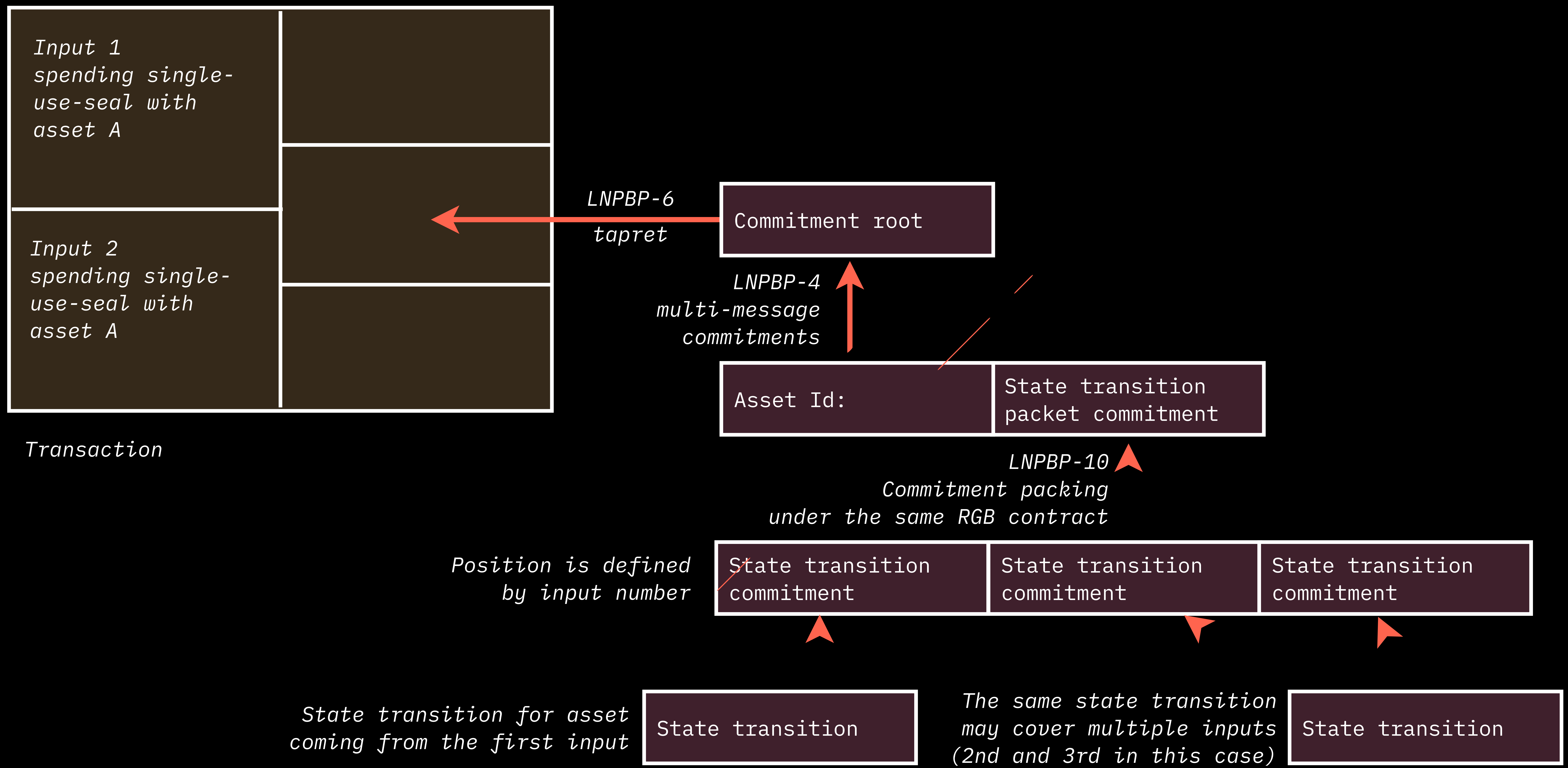
Switch to Tap-return (LNPBP-6)

- Compatible only with taproot-enabled wallets
  - RGB assets still can be allocated to non-taproot outputs
  - Will stimulate Taproot adoption and speed up LN update to Taproot (Bifrost)
- Just a single output is tweaked, it can be change or specially created output
- No interactive protocols
- No need to change hardware wallet signers
- Simple streamlined code
- No recursive fee adjustments

The last problem to solve:

How to construct multi-party transactions and avoid interactive protocols and RGB asset information leaks?

New way


Input 1
spending single-use-seal with asset A
Input 2
spending single-use-seal with asset A
Transaction
LNPBP-6
tapret
Commitment root
LNPBP-4
multi-message commitments
Asset Id:
State transition packet commitment
LNPBP-10
Commitment packing under the same RGB contract
Position is defined by input number
State transition commitment
State transition commitment
State transition commitment
State transition for asset coming from the first input
State transition
The same state transition may cover multiple inputs (2nd and 3rd in this case)
State transition

DBC protocols:

- DBC proofs (LNPBP-9)

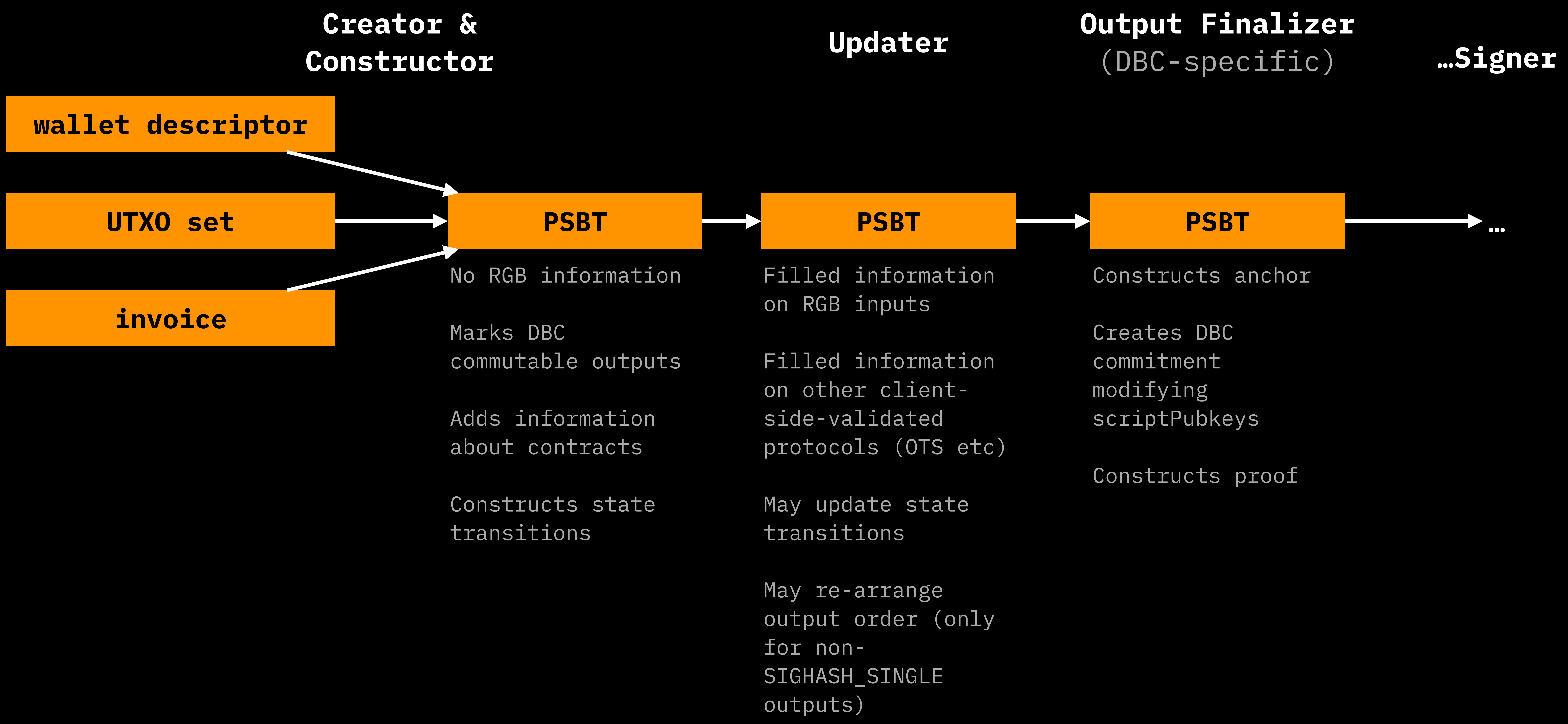
Output-based schemata

- Integral (whole transaction)
  - Fee-based any type of output (LNPBP-3)
  - First or last taproot output (LNPBP-9)
- Output-type standards
  - Script-based P2C commitments (LNPBP-2)
  - Tap-return commitments (LNPBP-6)
- Data tweaking
  - Pubkey-based P2C commitments (LNPBP-1)

Input-based schemata

- Input-type standards
  - ECDSA S2C commitments (LNPBP-39)
  - Schnorr S2C commitments (LNPBP-??)
  - Music S2C commitments (LNPBP-??)
  - Taproot S2C commitments (LNPBP-??)

PSBT workflow with DBCs



	Vendor type	Key	Value	Added by / *Modified by	Comments
Global					
RGB_CONTRACT	LSA	ContractId	Genesis	Creator, Constructor *Updater	*Provides signing devices with contract information to display to users*
RGB_STATE_TRANSITION	LSA	ContractId	Transition	Creator, Constructor *Updater	*(partially) constructed state transition*
Input					
RGB_INPUT	LSA	ContractId	Set<Transition>	Constructor, Updater	*Device must maintain client-validated state checksum + "USTOs" and validate*
Output					
DBC_COMMITABLE	LSA	–	–	Creator, Constructor	*Marks outputs which are allowed to host DBC commitments*
OTS_ROOT_COMMITMENT	LSA	–	32 bytes	Creator, Constructor	
LNPBP4_MULTI_COMMITMENT	LSA	–	MultiCommitment	Output finalizer	*Anchor data from which the tweak value is generated*
DBC_PROOF	LSA	–	Proof	Output finalizer	*DBC extra-transaction proof*