LNP/BP Standards Association

Single-use-seal changes due to Lightning network (Bifrost) & Taproot

Dr Maxim Orlovsky,
Pandora Core AG

* Single-use-seals standards and reference implementation are maintained by
LNP/BP Standards Association

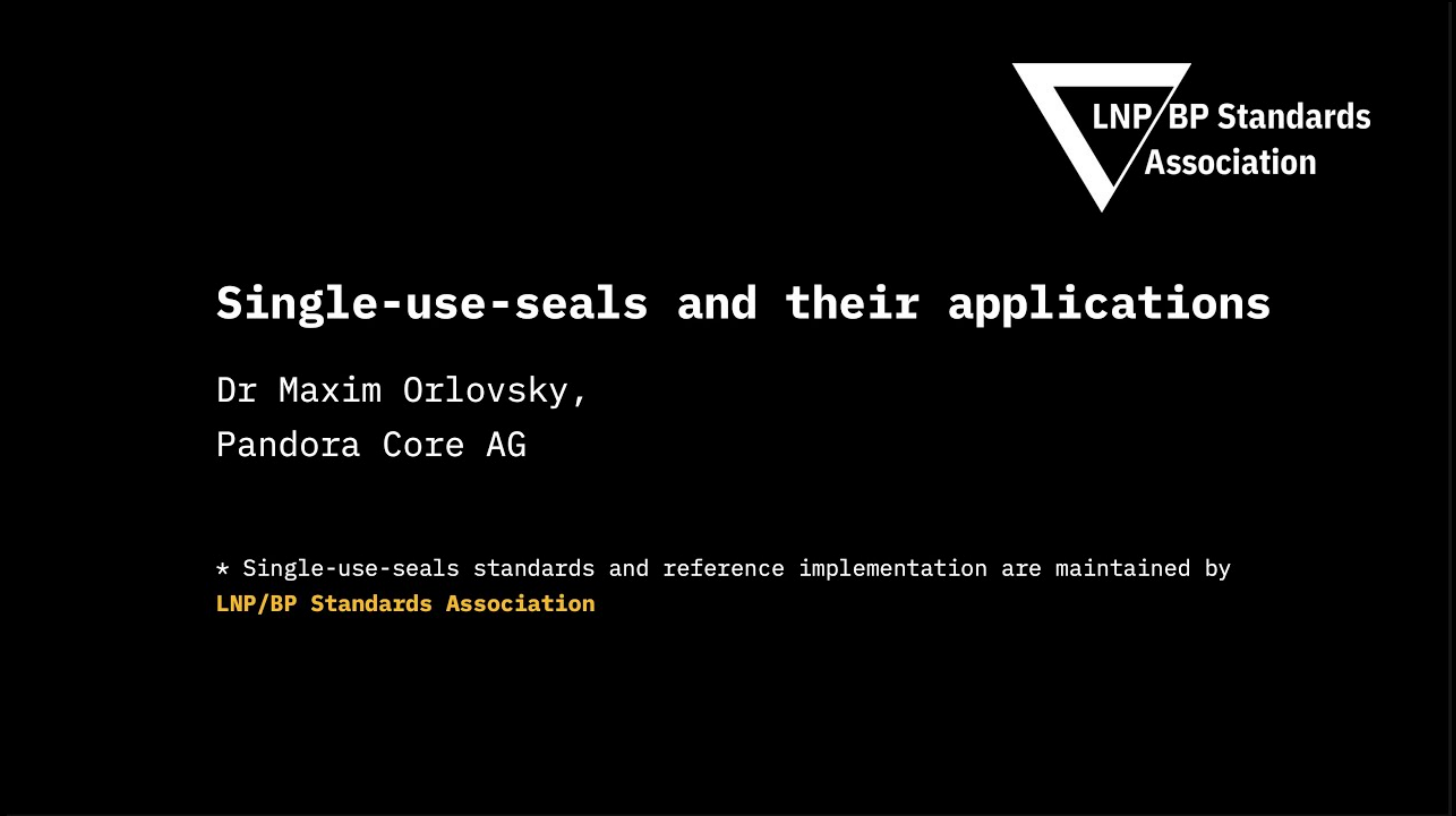
Target audiences for the call

- Peer reviewers of client-side-validation, single-use-seals & RGB
- Contributors to the reference rust implementation of the above
- Alternative/independent implementation devs (python, C/C++, etc)
- Technical writers

Background

What are single-use-seals?

- https://www.youtube.com/watch?v=gGPLYfWOb_8



LNP/BP Standards Association

Single-use-seals and their applications

Dr Maxim Orlovsky,
Pandora Core AG

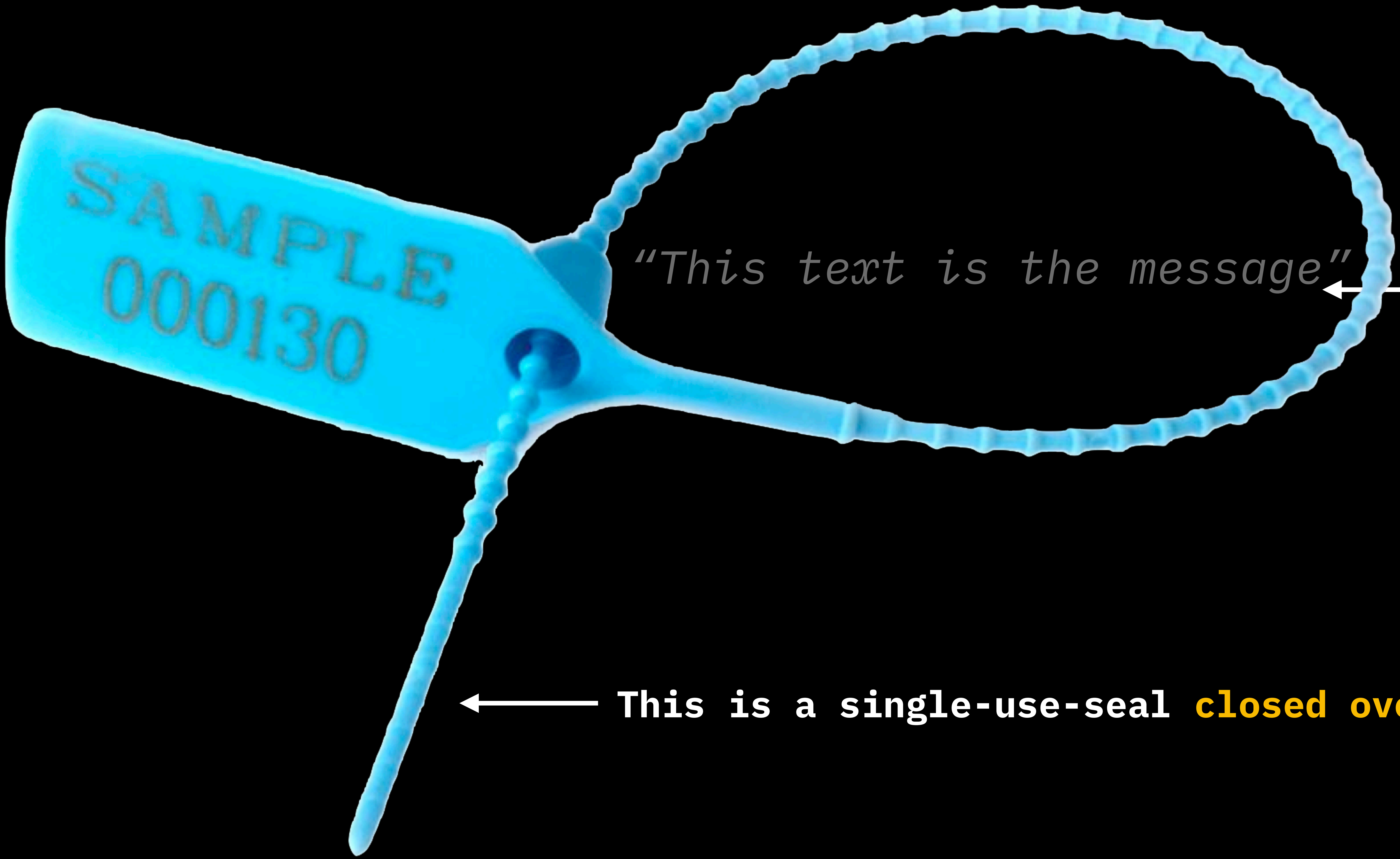
* Single-use-seals standards and reference implementation are maintained by
LNP/BP Standards Association

What is a single-use-seal?

- Form of applied cryptographical commitment
- more advanced than
  - simple commitments
  - timestamps
- Proposed as a cryptographic primitive by Peter Todd, following his development of timestamps

Single-use-seals vs other commitment schemes

	Simple commitment (digest/hash)	Timestamps	Single-use-seals
Commitment publication does not reveal the message	Yes	Yes	Yes
Proof of the commitment time / message existence before certain date	Not possible	Possible	Possible
Prove that no alternative commitment can exist	Not possible	Not possible	Possible



“This text is the message”

This is a **message**

This is a single-use-seal **closed over the message**

Three single-use-seals procedures

- **Generate:**
  define a single-use-seal = do a promise of future commitment
- **Close over message:**
  fulfill the promise and create a commitment
- **Verify:**
  verify that the seal was indeed closed, once and for all

Bitcoin single-use-seals

TxO-based bitcoin single-use-seal (LNPBP-8)

Transaction A:

Seal close witnesses:

1. Witness transaction:

Client-side-validated data:

Seal definition:

Txid:vout

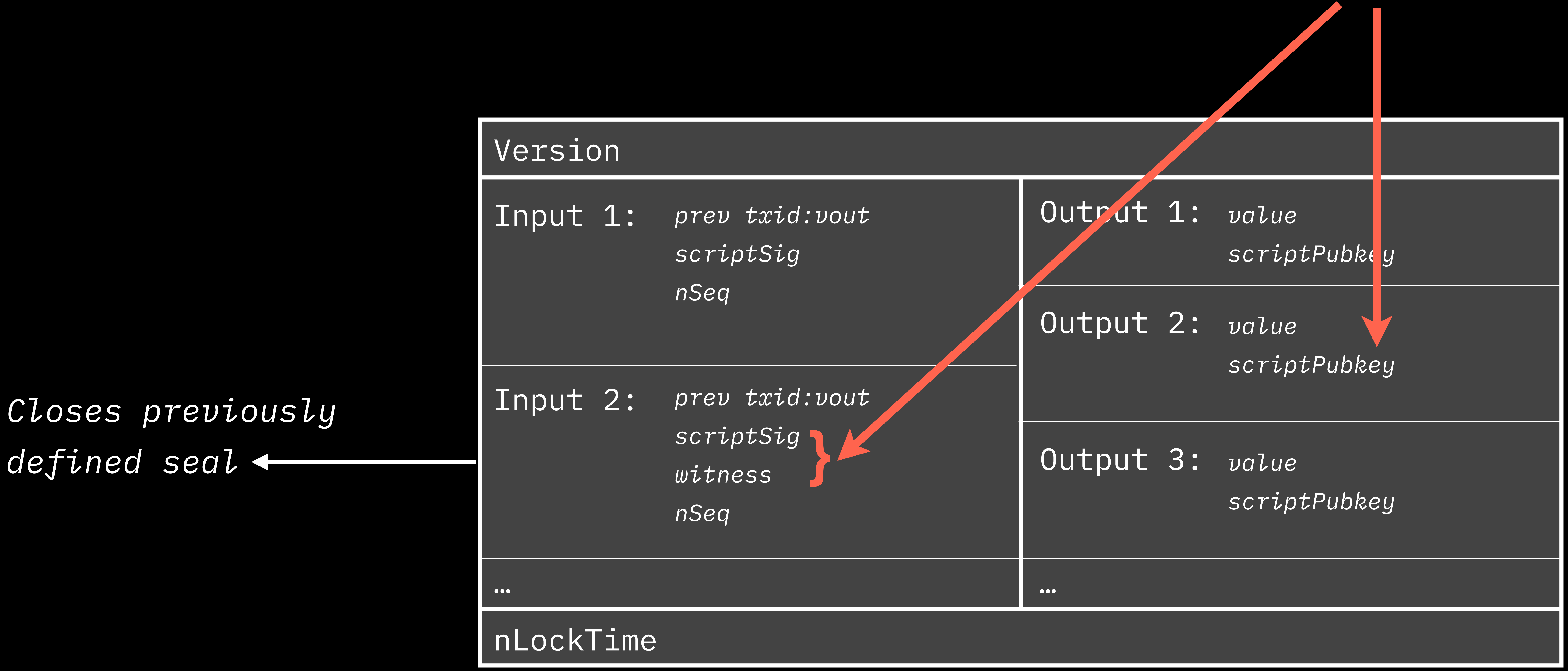
2. Extra-transaction witness:

Proofs

Commitment data

Closing seals with bitcoin transactions

Single-use-seal closing message commitment can go in here



Closes previously defined seal

Version

Input 1: *prev txid:vout*
scriptSig
nSeq

Input 2: *prev txid:vout*
scriptSig
witness
nSeq

…

Output 1: *value*
scriptPubkey

Output 2: *value*
scriptPubkey

Output 3: *value*
scriptPubkey

…

nLockTime

Pay-to-contract & sign-to-contract commitments

Pay-to-contract: Q = P + Hash(*message* || P) * G, where

- P is public key corresponding to some private key *p*
- Q is the “tweaked public key”
- This is how Taproot also works

Sign-to-contract: (R + Hash(*message* || R) * G, S), where

- (R, S) is the normal signature
- R is some nonce *r* * G created for each signature & independent from P
- S is *f(p,* G*)*; *f* is different for ECDSA and BIP-340 signatures

Pay-to-contract & sign-to-contract commitments

Pay-to-contract: $Q = P + \mathrm{Hash}(\mathit{message} \mathbin{||} P) * G$, where

- requires keeping info client-side to spend output later
- very similar & easily combined with Taproot

Sign-to-contract: $(R + \mathrm{Hash}(\mathit{message} \mathbin{||} R) * G,\ S)$, where

- less compatible with hardware wallets

> Pay-to-contract commitments are more compatible with HW wallets, so let’s start from them
>
> *— Giacomo Zucco in 2019, approved by RGB community :)*

TxO-P2C bitcoin single-use-seal

Transaction A:

Seal close witnesses:

1. Witness transaction:

fee
value

Pay-to-contract
commitment
(LNPBP-1 & 2)

LNPBP-3

protocol
id

Client-side-validated data:

Seal definition: Txid:vout

2. Extra-transaction witness:

Proofs

Commitment data

“Upgraded” pay-to-contract (late 2019)

- **LNPBP-1**: “public key commitments” – how to deterministically put a pay-to-contract style commitment into a set of public keys

- **LNPBP-2**: “scriptPubkey commitments” - how to extract a set of public keys used in commitments from any type of tx output

- **LNPBP-3**: “transaction commitments” - how to detect output that must contain a commitment

  vout = (fee_sat + protocol_id) mod output_count

  - Uses transaction indirect data (fee)

  - Uses extra-transaction data
    (256-bit id of protocol creating commitment)

Problem: nothing prevents from defining the same single-use-seal multiple times – but

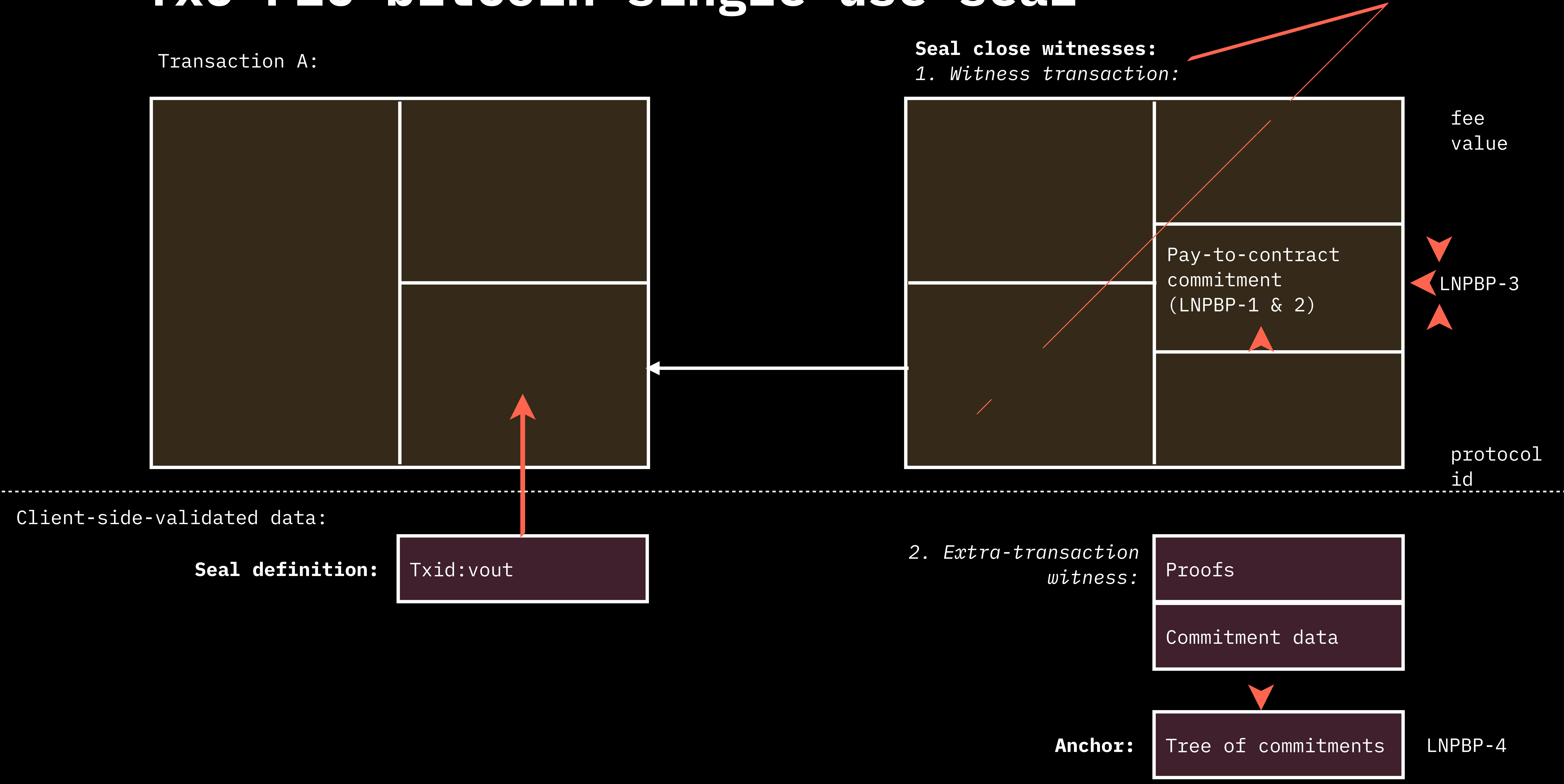
this actually is a **feature**: reduction of UTXO set size

But how to commit to multiple messages in a single seal closing?

Answer is **LNPBP-4** (early 2020): compact & deterministic merklization of multiple messages under multiple protocols into a single commitment

- Allows using multiple protocols under the same single-use-seal
  - Multiple RGB assets
  - and/or timestamps
  - and/or future client-side-validation protocols

TxO-P2C bitcoin single-use-seal



Transaction A:

Seal close witnesses:
1. Witness transaction:

fee value

Pay-to-contract commitment (LNPBP-1 & 2)

LNPBP-3

protocol id

Client-side-validated data:

Seal definition: Txid:vout

2. Extra-transaction witness:

Proofs

Commitment data

Anchor: Tree of commitments

LNPBP-4

LNPBP-4 pending upgrade (late 2021)

- Current LNPBP-4 uses linear serialization of per-protocol commitments:
  - **O(N)** data storage on the client-side
  - **O(const)** verification time
- New LNPBP-4 will use Merkle trees:
  - **O(log(N))** data storage
    (just 512 bytes per 65536 assets on the same single-use-seal)
  - **O(log(N))** verification time
  - Better privacy
    (by default we use more fake protocols to hide number of assets)

And we were happy and made an implementation, currently used in RGB

Bitcoin TxO single-use-seals v1 (TxO/P2C)

- “Upgraded **pay-to-contract**”: LNPBP-1,2,3 (RGB community, Nov 2019)
- **Multi-message commitments**: LNPBP-4 (Maxim Orlovsky, Jan 2020)
- **Single-use-seals** paradigm & API: LNPBP-8 (Peter Todd, 2016)
- Bitcoin **TxO-P2C single-use-seals** using all of the above: not yet completed (fortunately) LNPBP-10 standard

Today, we need amendments to it because of LN + Taproot (late 2021)

Lightning is, in fact, thunderstorm

Pay-to-contract-specific drawbacks

During 2021, with work of RGB in lightning network and first wallets, it became apparent that TxO-P2C seals:

- Require other tx participants to modify their private keys and keep that information. Relevant for:
  - Lightning (especially multi-peer)
  - CoinJoin, PayJoin
  - PSBT multisigs with change addresses
- For channel funding transactions an interactive protocol is required to create anchor data, for instance to join RGB state transitions, which will expose private asset data
- Make “pay to my address” sometimes (non-predictably) impossible, when multiple assets on the same UTXO will require tweaking of all transaction outputs

Lightning network problems with P2C

Mitigable by breaking LN protocol in channel – but not network-wise (discovered in 2019-2020)

- Lexicographic output ordering (BIP-69)
- Deterministic key derivation for outputs
- Need to adjust fees ends up in multiple additional interactive communication rounds and not supported by the protocol

Non-mitigable (discovered in 2021)

- Need to interactively create state transitions and expose assets

Can we fix that with Bifrost?

wait, what is Bifrost?

LNP/BP Standards Association

Bifrost

generalized lightning network

Can we fix that with Bifrost?

- Lexicographic output ordering (BIP-69) – yes
- Deterministic key derivation for outputs – yes
- Need to interactively create state transitions and expose asset data – no
- Fee adjustments - only partially

Plus, with Bifrost, we’ve got other problems:

- No-signature-spendings of anchor outputs – but this is addressable with changing “anyone-can-spend” condition to “any channel peers may spend” condition
- More parties in channel factories/multi-peer channels, so it’s harder to orchestrate P2C commitments

So what to do?

— me two weeks ago

And when RGB / Bifrost?

— *RGB community*

Let’s use sign-to-contract!

my original proposal for RGB v2

Not that easy: sign-to-contract drawbacks

- Incompatible with **P2TR script path spendings** (worse than P2SH in P2C, since we can future tap leaf versions without Secp256k1 signatures)
- Problems with **MuSig2** compatibility – at least not yet solved
- Hardly compatible with **hardware wallets** (may be a different subset than for P2C compatibility)
- **No implementations** for BIP-340 signatures

Sign-to-contract with Bifrost (LN upgrade):

- Can’t be used with Taproot-based lightning anchor outputs, since they imply non-signature spendings (potentially fixable)
- No ready-to-go S2C scheme compatible/safe with MuSig2
- Non-compatible if used with hardware-based key signing

Other alternatives which were considered

- Using Taproot Annex (signed part of the witness)
  - may render existing UTXO unspendable with after some future soft fork
  - may conflict with future soft forks on consensus level
- Using “dumb” extra witness stack item before control block
  - May conflict with some scripts logic
  - Will consume 32 bytes of block space
  - May be removed - or replaced - by miners (since witness data are not signed)

TxO-P2C can’t be used

- With some hardware wallets
  (mitigated by OP_RETURN P2C fallback)
- With transactions having multiple receivers
- When it is impossibility to adjust fees
- With protocols based on lexicographic output ordering (BIP-69)
- When outputs has deterministically derived scripts
- Pay-to-address requests & bitcoin URLs
  (may potentially, but not always, be mitigated with adding a lot of change UTXOs)

Takeaway: no guarantees that P2C may be applied in a reasonable manner

TxO-S2C can’t be used

- Some hardware wallets (no fallback)
- Taproot MuSig2 spendings (temporarily)
- Taproot script path spendings when no signatures are used
  (lightning anchor outputs)
- Taproot with future script versions
  (post-BIP-342/TapScript)

TxO-P2C can’t be used

- Lightning network (and some Bifrost transactions)
- PayJoin / CoinJoin
- Complex multisig setups
- Bitcoin URLs and pay-to-address requests / donations

TxO-S2C can’t be used

- Some of transactions in Bifrost channels
- Hardware wallets
- Bitcoin bounties when output may be spent without signature

Comparing bitcoin commitment schemes

	Pay-to-contract (P2C)	Sign-to-contract (S2C)	OP_RETURN-to-contract (R2C)
Onchain space consumption	0 bytes	0 bytes	42 bytes
Client-side space consumption	33 bytes	32 bytes	0 bytes
Can be combined with SegWit v0 outputs	Yes	Yes	No
Can be combined with SegWit v1 (Taproot) outputs	Yes, with modifications (T2C)	Yes, if TapScript + signatures are used	No
Can be used by hardware wallets	Sometimes	Mostly not	Yes
Multisig vulnerability	No	Yes	No

How we can use both P2C and S2C?

Only if we signal at tx level which scheme is used
(otherwise we have a double-spend problem)

… and we need to signal on a per-spent-output bases
(ie inside spending transaction input)

… signalling can’t be part of the transaction witness:

- the same set of problems as S2C itself
- witness parts without consensus-required signatures may be modified by miners

➡ Thus, the only way to signal is to use **nSequence**

nSequence encumbers

- Replace-by-fee (BIP-125)
- CheckSequenceVerify (BIP-68)
- Chain analysis (need to avoid)

How replace-by-fee (BIP-125) encumbers nSeq

Transactions with nSeq (in all inputs) **0xFFFFFFFF** or **0xFFFFFFFE** are not participating RBF

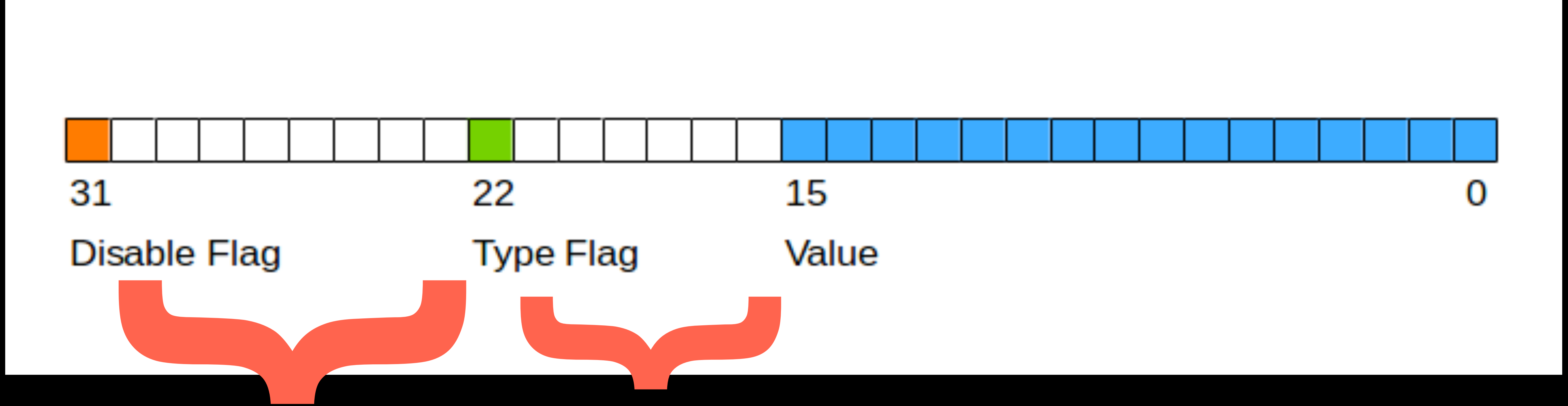
- We have at least two options for non-RBF nSeq number to distinguish P2C from S2C commitments

- We can use parity of nSeq for RBF-opt-in transactions

Thus, the latest bit of nSeq (**nSeq & 0x00000001 = nSeq mod 2**) may be used to distinguish P2C from P2S single-use-seal spends

CHECK: For nLockTime activation conflict

How relative locktime (BIP-68) encumbers nSeq



bits that we can use

- Requires tx to be RBF, so orthogonal with RBF compatibility (we still do not need to change algorithm for RBF-opt-out case)
- We can use bits no 30-23 and 21-16, but they may be assigned future consensus meaning
- So we will not use individual bits, but count number of non-BIP-68 bits set for distinguishing P2C from S2C commitments
  - this will maximize compatibility with up to ~6 future soft forks touching nSeq

How lightning network uses nSeq

- It uses RBF with random 24 lower nSeq bits in funding->commitment spending
  ** this is indeterministic under BIP-68, opened an issue*

  ➡ we need to use only upper bits 30-23 if we'd like to be compatible with legacy LN in single-use-seals

- It uses RBF with nSeq set to 1 in HTLC timeout tx and
  when *to_remote* output is spent
  but only if *option_anchors* is used;
  otherwise it will be either non-RBF or RBF with 0x00

- It uses RBF with nSeq set to *to_self_delay* in spending HTLC transactions (used on non-cooperative closings)

- Everywhere else it uses RBF with nSeq set to 0

How lightning network uses nSeq: Analysis

- First: randomly
  (sometimes using (pseudo-)random numbers, sometimes negotiated values, sometimes fixed constants)

- Second: constantly changing rules

Conclusion:

- If we never use single-use-seals with LN (and use Bifrost instead), we do not give a fuck about these peculiarities
- If we plan to do otherwise, we have to limit RBF-opt-in version of single-use-seals with just 8 bits (and restrict future soft-fork resistance) + be ready for on-chain analysis tracing

Since even P2C single-use-seals are practically incompatible with existing “legacy” LN, I propose the first option

— So let’s move forward with Bifrost

Final TxO s-u-s algorithm (part of LNPBP-10)

- For non-RBF tx (nSeq < 0xFFFFFFFE)
  - Test for nSeq parity (nSeq & 0x1)
- Else
  - Count set bits in positions 30-23 and 21-16
  - Test for this count parity

Algorithm properties

- For non-RBF transactions on-chain analysis can’t detect use of single-use-seals
- For RBF non-lightning transactions wallets may construct them in such way that no on-chain analysis on single-use-seal presence is possible
- We do not support lightning transactions
- But we support Bifrost

Bifrost specifics

- All Bifrost transactions, independently of the use of RGB/single-use-seals, must use pseudo-random nSeq bits in position 21-16, making it indistinguishable from Lightning transactions
- Use of P2C/S2C commitments can be controlled by changing the number of set bits in positions 21-16

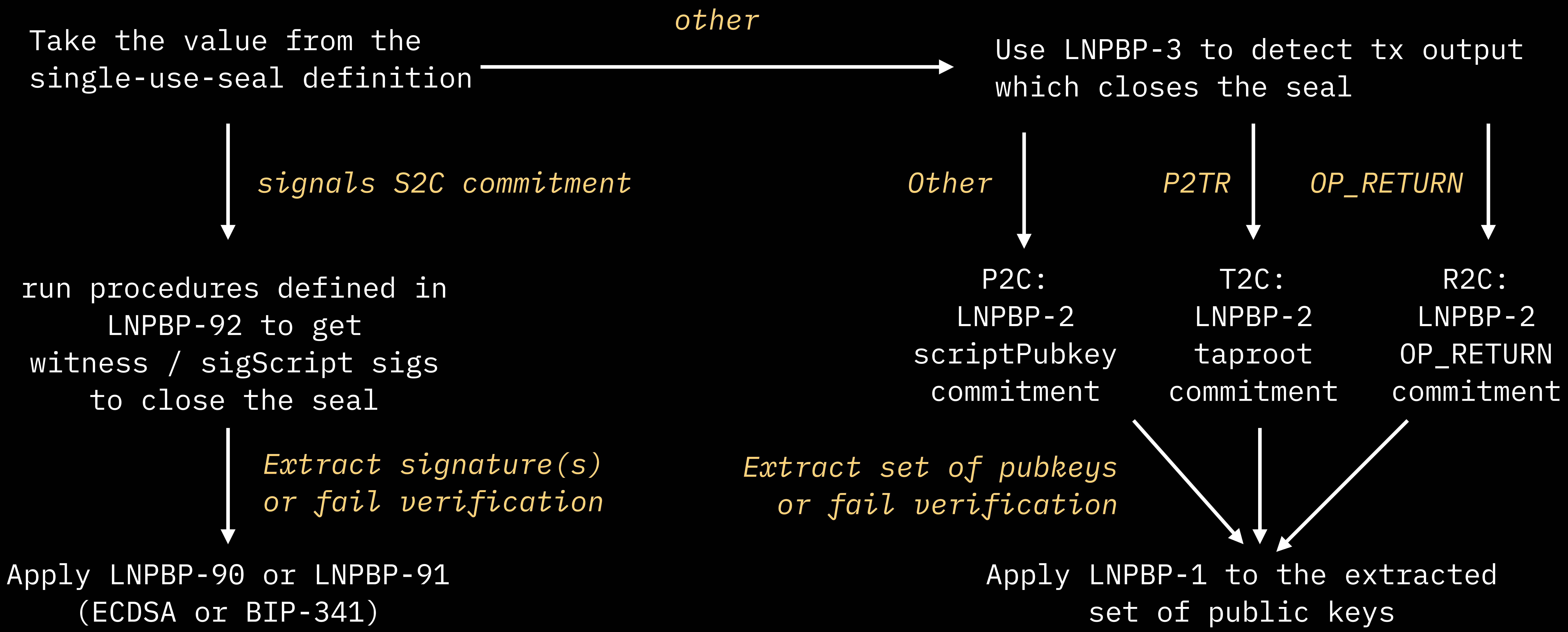
Odd or even?

- We use parity of nSeq value (non-RBF) – or parity of the count of bits set in certain positions (RBF, Bifrost) – to signal P2C or S2C version of single-use-seal closing schema

- Odd: S2C, since it will match 0xFFFFFFFF, the most frequent nSeq in history. We'd like it to be the main commitment scheme in the future.

- Even: P2C

Summary of LNPBP-10 single-use-seal verification



Take the value from the single-use-seal definition

other

Use LNPBP-3 to detect tx output which closes the seal

signals S2C commitment

run procedures defined in LNPBP-92 to get witness / sigScript sigs to close the seal

Extract signature(s) or fail verification

Apply LNPBP-90 or LNPBP-91 (ECDSA or BIP-341)

Other

P2TR

OP_RETURN

P2C: LNPBP-2 scriptPubkey commitment

T2C: LNPBP-2 taproot commitment

R2C: LNPBP-2 OP_RETURN commitment

Extract set of pubkeys or fail verification

Apply LNPBP-1 to the extracted set of public keys

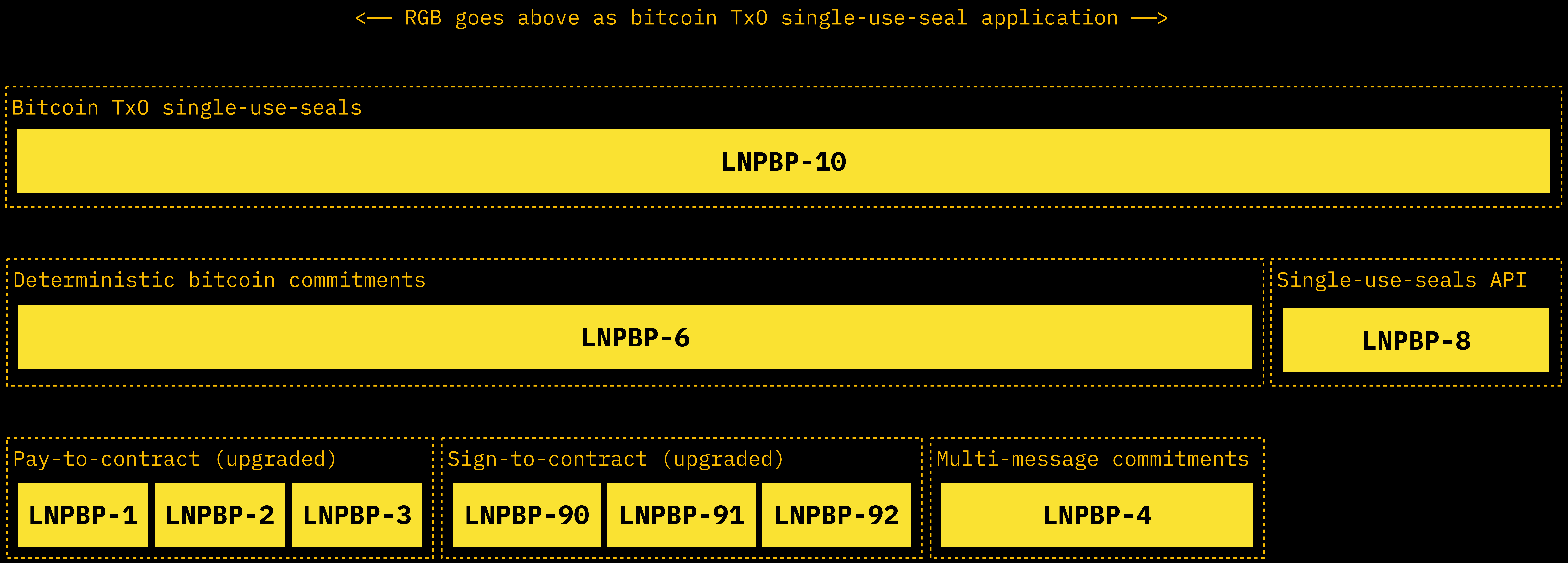
Designing sign-to-contract: LNPBP-90-92

Will be the topic of our next dev call

Action points

- Help writing and designing sign-to-contract standards (LNPBP-90, 91 & 92): github.com/LNP-BP/LNPBPs/pull/118/files
- Review & contribute to finalization of TxO single-use-seals (LNPBP-10): github.com/LNP-BP/LNPBPs/pull/117/files
- Review previous work on deterministic bitcoin commitments: github.com/LNP-BP/LNPBPs (see LNPBP-1, 2, 3, 4 standards)
- Help implementing sign-to-contract stack
  - secp256k1 – github.com/bitcoin-core/secp256k1/pull/1018
  - rust ports to rust-secp256k1(-zpk), once the above will be completed & merged
  - BP core library PR to use these implementations

Standards hierarchy



<—— RGB goes above as bitcoin TxO single-use-seal application ——>

Bitcoin TxO single-use-seals

LNPBP-10

Deterministic bitcoin commitments

LNPBP-6

Single-use-seals API

LNPBP-8

Pay-to-contract (upgraded)

LNPBP-1 LNPBP-2 LNPBP-3

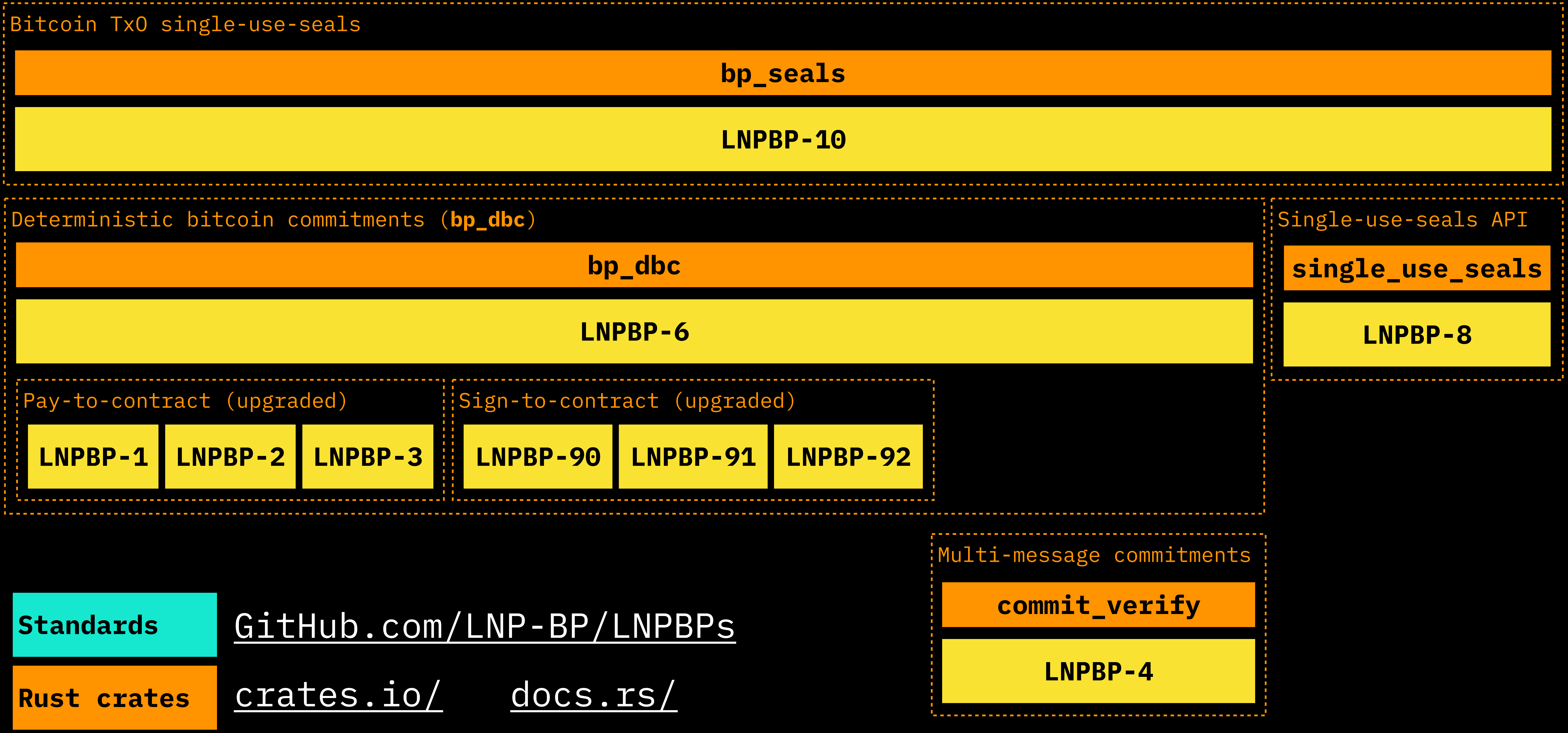
Sign-to-contract (upgraded)

LNPBP-90 LNPBP-91 LNPBP-92

Multi-message commitments

LNPBP-4

Standards & reference implementations



Bitcoin TxO single-use-seals

bp_seals

LNPBP-10

Deterministic bitcoin commitments (**bp_dbc**)

bp_dbc

LNPBP-6

Pay-to-contract (upgraded)

LNPBP-1 LNPBP-2 LNPBP-3

Sign-to-contract (upgraded)

LNPBP-90 LNPBP-91 LNPBP-92

Single-use-seals API

single_use_seals

LNPBP-8

Multi-message commitments

commit_verify

LNPBP-4

Standards GitHub.com/LNP-BP/LNPBPs

Rust crates crates.io/ docs.rs/