Capability Parity of CSFS+CAT with CTV and ANYPREVOUT for Common Covenant Patterns: On-Chain Evidence from Inquisition Signet

Aaron Zhang

Independent Researcher · aaron@btcdevlab.org

Abstract

We show, with confirmed on-chain transactions on Bitcoin Inquisition signet, that the combination of `OP_CHECKSIGFROMSTACK` (BIP 348) and `OP_CAT` (BIP 347) **reproduces the covenant capabilities of `OP_CHECKTEMPLATEVERIFY` (BIP 119) and BIP 118 ANYPREVOUT for three common patterns** — output binding and input binding (built directly from CSFS+CAT), and the eltoo state-replacement pattern (realized through a CSFS delegation ladder with CTV settlement) — for spend patterns where the spending signer participates. We do not claim sufficiency for every conceivable covenant, nor that CSFS+CAT should replace either opcode as a deployment optimization; the claim is parity on the specific patterns we construct and confirm on chain. Our approach rests on two named techniques. First, *same-signature binding* (§3.2): a single Schnorr signature must satisfy both `OP_CHECKSIG` (against the node-computed sighash) and `OP_CHECKSIGFROMSTACK` (against a witness-provided tapscript signature-message preimage). Because one signature cannot stand for two different transactions, this forces the witness-supplied preimage to be byte-identical to the node-computed one (§3.2). Second, *preimage decomposition* (§3.3): a construction emulating the disabled `OP_SUBSTR` via `OP_CAT` + `OP_SIZE` + witness-provided padding, letting Script extract and verify arbitrary fields of the preimage. Together the two techniques recover `sha_outputs` (output binding, reproducing CTV's output template), `sha_prevouts` (input binding, beyond CTV's template), and `sha_amounts` (amount constraints).

We validate the constructions with confirmed transactions on Bitcoin Inquisition signet. To our knowledge, and after searching Delving Bitcoin, the Inquisition issue tracker, and public signet block explorers prior to this work, we are not aware of prior on-chain demonstrations of field-level tapscript-sighash introspection via CSFS+CAT under the BIP-347/348 environment. BIP-347 (CAT) and BIP-348 (CSFS) are themselves activated by redefining `OP_SUCCESS_x` opcode positions — `OP_SUCCESS_126` and `OP_SUCCESS_204` respectively — so the construction adds capability through the activation path the BIPs themselves use, without architectural surface beyond what those activations already entail. We discuss capability coverage and script-size / resource trade-offs. In §8 we observe, informally, that the security objections commonly raised specifically against CAT (fungibility, recursion, MEV) apply at comparable construction cost to other covenant proposals as well; this is a discussion, not a formal claim. We conclude by discussing what the empirical evidence says about the *shape* of the activation question, leaving the activation decision itself to the community.

Keywords: Bitcoin covenants; OP_CAT; OP_CHECKSIGFROMSTACK; Tapscript; sighash-preimage introspection; CHECKTEMPLATEVERIFY (CTV); ANYPREVOUT (APO); eltoo; Bitcoin Inquisition signet.

1. Introduction

1.1 The covenant fragmentation problem

Currently, five competing covenant proposals are in circulation in the Bitcoin community: BIP 118 (`SIGHASH_ANYPREVOUT`), BIP 119 (`OP_CHECKTEMPLATEVERIFY`), BIP 345 (`OP_VAULT`), BIP 347 (`OP_CAT`), and BIP 348 (`OP_CHECKSIGFROMSTACK`). Each addresses a specific use case — eltoo channels, congestion control, vault patterns, data attestation — and each has its own constituency. After six years of debate, the community has not agreed on which to activate, in what order, or whether they should be activated at all.

Within this landscape, a recurring sub-question has appeared in research forums: *which combinations of opcodes are jointly sufficient for which capability classes*? Andrew Poelstra's early analysis of `OP_CAT + OP_CHECKSIG` argued for theoretical equivalence to a broad class of covenants. Russell O'Connor [O'Connor 2022] proposed that `OP_TXHASH + OP_CHECKSIGFROMSTACKVERIFY` would replace both `CTV` and `ANYPREVOUT` while staying simpler. Jeremy Rubin [Rubin 2025] sketched several CSFS-based constructions, including an LN-Symmetry variant. These works argue at the level of theoretical capability; on-chain evidence under the already-activated `BIP-347` + `BIP-348` environment has been sparse.

1.2 Direct motivation: Towns's observation of empirical gaps

In March 2025, Bitcoin Inquisition maintainer Anthony Towns observed in a public thread on CTV+CSFS [Towns 2025, Delving Bitcoin /t/1509/14]:

> "the exploration into ln-symmetry has been both fairly preliminary (to the best of my knowledge, nobody has been able to reproduce Greg's results [...]). It also used APO, the annex, and some custom relay rules, **not CTV or CSFS** [...] so even if it did provide good evidence that eltoo was possible with APO, **someone would still need to do the work to redesign it for CTV/TXHASH/CSFS** before it was good evidence for that approach."

Two distinct gaps in the empirical record are identified in this paragraph: (i) Sanders' APO-based eltoo experiment had not been independently reproduced under the current Inquisition software stack and contemporary relay features, and (ii) no port of eltoo to a CTV- or CSFS-based opcode set existed. The community had access to a theoretical literature on covenant capability, but lacked confirmed-on-chain artifacts under the already-activated BIP-347/348 environment.

The present paper consolidates the on-chain results that close these gaps and analyses the underlying technique — same-signature binding combined with sighash-preimage decomposition — that makes the CSFS-based reconstructions possible. The complete TxID inventory is given in §1.7.

1.3 Historical context: CSFS+CAT feasibility as an open 2023 question

During September–October 2023, several covenant researchers publicly judged CSFS+CAT as a covenant construction not reasonably feasible [Harding 2023; O'Beirne 2023b; Towns 2023; Roose

2023]. The discussions did not subsequently produce on-chain constructions, and the direction effectively closed. §10.8–§10.11 trace this lineage in detail. We mark it here as the open question this paper addresses by construction rather than by argument.

1.4 Design context: composable primitives in Tapscript

A longer architectural arc in Tapscript provides context. BIP-342 replaced `OP_CHECKMULTISIG` — a packed verification pattern that combined batch logic, public key handling, and signature checking — with `OP_CHECKSIGADD`, a single-signature primitive that scripts assemble themselves. The replacement was driven primarily by enabling batch validation of Schnorr signatures (BIP-342).

Independently, BIP-342 reserved unused opcode positions as `OP_SUCCESS_x` — opcodes that succeed unconditionally in Tapscript v1 and can be assigned semantics by future soft-forks. BIP-347 (CAT) is activated by redefining `OP_SUCCESS_126`; BIP-348 (CSFS) by redefining `OP_SUCCESS_204`. Activating CSFS+CAT therefore uses the same extension mechanism that BIP-347 and BIP-348 themselves use — independently of which specific opcodes were anticipated when those slots were originally reserved.

Whether this mechanism continues to be the right path for future capability additions is a question for the community, not one this paper takes a position on.

1.5 Research question

> *For the covenant capability patterns that CTV and APO are commonly cited to provide, can already-activated tapscript opcodes (CSFS + CAT) reach capability parity — reproducing the same enforced relation on the spending transaction — and can this parity be demonstrated with on-chain evidence rather than theoretical argument alone?*

1.6 Contributions

1. **Same-signature binding** (§3.2) — A construction in which one 64-byte Schnorr signature must satisfy both `OP_CHECKSIG` (against the real tapscript-sighash) and `OP_CHECKSIGFROMSTACK` (against a witness-provided preimage); because one signature cannot stand for two different messages, this forces the witness-supplied preimage to equal the real one.
2. **Preimage decomposition — OP_SUBSTR emulation via OP_CAT + OP_SIZE** (§3.3) — A method for extracting any contiguous field of the tapscript signature-message (BIP-341 common SigMsg with BIP-342 tapscript extension, 212 bytes in the no-annex default case) via a chunked witness, `OP_SIZE` boundary locks, `OP_CAT` reassembly, and tagged-hash verification. `OP_SUBSTR` was disabled in Bitcoin Script's early consolidation; this construction recovers its functionality by composition of currently-available opcodes.
3. **Three on-chain validations** — Demonstrations of (a) output binding (reproducing CTV's output template), (b) input binding (beyond CTV's template), (c) the eltoo state-replacement pattern, realized through a CSFS delegation ladder with CTV settlement — matching APO's

eltoo topology, though via an O(n) delegation chain rather than APO's constant-size rebinding. All on Bitcoin Inquisition signet.[1]

4. **Capability comparison** — A side-by-side comparison showing that, for the patterns examined, CSFS+CAT covers the capability surface of CTV and the rebinding pattern of APO, with explicit script-size and sigops trade-offs.
5. **An informal comparison of the security objections** (§8) — We take the three objections raised specifically against CAT (fungibility, recursion, MEV) and observe that each applies, at comparable or lower construction cost, to other covenant proposals as well (CTV, APO, TXHASH). This is a qualitative discussion meant to inform the activation debate, not a formal result.
6. **Activation-question framing** — A discussion of how empirical capability evidence changes the *shape* of the activation question, without advocating any particular activation ordering.

1.7 Empirical Evidence Summary

All on-chain validations in this paper are documented in public Delving Bitcoin posts by the author (with complete witness layouts, ASM step traces, and explorer links). The TxIDs below are reachable via standard signet explorers (e.g., `https://mempool.space/signet/tx/<txid>`):

Construction	Series Post	TxIDs (truncated)	Delving URL
APO+CTV eltoo state chain, 3 rounds, 6 txs	Post #3	`386dbb6a...`, `096e31cc...`, `091309b7...`, `13957f49...`, `03c0577c...`, `46091190...`	/t/2413
CSFS+CTV eltoo via key-delegation ladder, 2 txs	Post #5	`92efc475...`, `b96324da...`	/t/2430
sha_prevouts (input binding), positive	Post #6	`b7959a71...`, `7311da6e...`	/t/2483
sha_prevouts (input binding), attack	Post #6	rejected by `testmempoolaccept`	/t/2483
sha_outputs (output binding, CTV-equivalent), positive	Post #7	`2f345180...` (block 300379)	/t/2483/3
sha_outputs (output binding, CTV-equivalent), attack	Post #7	rejected by `testmempoolaccept`	/t/2483/3

The paper's central result is self-contained. The two core techniques are presented in full in §3 (same-signature binding, with a plain-language justification, in §3.2; preimage decomposition in

[1] We searched Delving Bitcoin, the Bitcoin Inquisition issue tracker, public signet block explorers, the bitcoin-dev mailing list archive, and Jeremy Rubin's blog prior to this work; we are not aware of prior on-chain demonstrations of this specific construction on Inquisition signet under BIP-347/348.

§3.3), and the output-binding construction is reproduced inline at the byte level in §4.4 — full annotated tapscript ASM, witness layout, hardcoded values, and the boundary-shift attack analysis — so the central claim can be checked without any external resource. The detailed byte-level records for the input-binding (§5) and rebinding (§6) constructions are in the public Delving Bitcoin posts cited in the table above; those two sections reuse the §4 technique, so the posts are reproduction aids, not prerequisites for the argument.

1.8 Scope and non-claims

To prevent misreading, we state explicitly what this paper does *not* claim:

- **Not a supersession claim.** BIP-118 APO and BIP-119 CTV each offer properties this construction does not replicate — APO provides natural signature reuse across compatible prevouts with constant per-state witness size; CTV provides a minimal single-opcode template check. Each is optimized for its target capability, and each may remain the preferred deployment choice for its intended use cases.
- **Not a minimum-size claim.** The constructions presented use witness chunking, chunk-size locks via `OP_SIZE`, chunk reassembly via `OP_CAT`, and (for rebinding) a CSFS key-delegation ladder whose witness scales linearly with state count. Witness footprints exceed those of CTV or APO for equivalent functionality. This approach explicitly trades computational and witness footprint for primitive generality; we claim only that the constructions execute on-chain within the existing Inquisition signet standardness rules.
- **Not a deployment-ordering claim.** The empirical evidence that CSFS+CAT reaches capability parity with CTV and APO for the patterns examined changes the *shape* of the activation question; it does not by itself select an activation ordering among CSFS, CAT, CTV, APO, or other proposals. That choice remains a community decision informed by many factors (review maturity, footprint optimization, deployment infrastructure, political tractability) that exceed the scope of this paper.
- **Not a generality claim.** We establish parity on three specific patterns we construct and confirm on chain (§4–§6) and indicate, in §7, how the same technique extends field-by-field. We do not claim a closed-form proof that *every* covenant expressible with CTV or APO is reproducible; that universal statement would require a formalization of "covenant" as a relation over transactions, which we leave to future work (§7.3).

The paper's contribution is on-chain-confirmed capability parity for the three patterns examined, reached with the already-activated CSFS + CAT pair, together with the technique that makes the parity extend field-by-field.

1.9 Independence from companion paper

This paper stands on its own: the construction, security argument, and experimental evidence do not depend on the companion paper [Zhang 2026e], which develops a broader historical framing (Script's evolving relationship with sighash visibility) that readers may consult at their preference.

2. Background

2.1 Bitcoin covenant proposals

We summarize the five covenant-related BIPs by category:

BIP	Opcode / mode	Mechanism	Activated on Inquisition?
118	`SIGHASH_ANYPREVOUT` (sighash mode)	Signature does not commit to prevout; rebinding	Yes
119	`OP_CHECKTEMPLATEVERIFY`	Commit to a 32-byte template hash	Yes
345	`OP_VAULT`, `OP_VAULT_RECOVER`	Specialized vault flow	No
346	`OP_TXHASH`, `OP_CHECKSIGFROMSTACKVERIFY`	General transaction-field hashing	No
347	`OP_CAT`	Stack concatenation, 520-byte limit	Yes
348	`OP_CHECKSIGFROMSTACK`	Schnorr signature verification over stack-provided message	Yes

The proposals fall into three categories: *signature-mode changes* (BIP 118), *specialized verification opcodes* (BIPs 119, 345, 346), and *primitive opcodes* (BIPs 347, 348). The first two add capability through dedicated machinery for specific patterns. The third does not target any specific pattern but provides general building blocks.

2.2 Tapscript signature-message preimage

For tapscript script-path spends, the message a Schnorr signature actually signs is computed from the **tapscript signature-message preimage**. This preimage is constructed as `0x00 || SigMsg(hash_type, ext_flag=1) || ext`, where:

- `0x00` is the epoch byte;
- `SigMsg(hash_type, ext_flag)` is the **BIP-341 common SigMsg** (whose length depends on `hash_type` and the presence of an annex); and
- `ext` is the **BIP-342 tapscript extension**, containing `tapleaf_hash`, `key_version`, and `codesep_pos`.

For the canonical case in this paper — script-path spend, default sighash (`SIGHASH_DEFAULT`), no annex, no `SINGLE`/`NONE`/`ANYONECANPAY` — the layout is:

Offset (bytes)	Length	Field	Source
0	1	`epoch` (currently 0x00)	BIP-341
1	1	`hash_type`	common SigMsg

Offset (bytes)	Length	Field	Source
2	4	`nVersion`	common SigMsg
6	4	`nLockTime`	common SigMsg
10	32	`sha_prevouts`	common SigMsg
42	32	`sha_amounts`	common SigMsg
74	32	`sha_scriptpubkeys`	common SigMsg
106	32	`sha_sequences`	common SigMsg
138	32	`sha_outputs`	common SigMsg
170	1	`spend_type`	common SigMsg
171	4	`input_index`	common SigMsg
175	32	`tapleaf_hash`	tapscript ext (BIP-342)
207	1	`key_version`	tapscript ext (BIP-342)
208	4	`codesep_pos`	tapscript ext (BIP-342)

Total: 212 bytes.

The full 64-byte tag prefix used in BIP-340 tagged hashing of `TapSighash` is `SHA256("TapSighash") || SHA256("TapSighash")`. The Schnorr signature commits to `SHA256(tag_prefix || preimage)` — a 276-byte hash input.

This structure is central to our construction. Because the preimage is field-aligned and fixed-length in the canonical case, Script can extract any field by chunking the witness at field boundaries.[2]

2.3 Schnorr signature properties used

Three properties of BIP-340 Schnorr signatures matter for what follows:

1. **Existential unforgeability under chosen-message attack (EU-CMA)** [BIP 340, security argument]. Without the secret key, an adversary cannot produce a valid `(m, sig)` pair for any `m` not previously signed.
2. **Verification determinism**. Given `(P, m, R, s)`, validity is a deterministic predicate computed via `R = sG - eP` with `e = H(R || P || m)`.
3. **Two-message binding**. Given a fixed `(P, R, s)`, the challenge scalar `e` with `eP = sG - R` is uniquely determined, so `Schnorr_Verify(P, m, (R, s)) = true` holds exactly for messages `m` with `H_chal(R || P || m) = e (mod n)`. Making one signature verify on two distinct messages therefore requires a collision in the BIP-340 challenge hash `H_chal` — independent of whether the secret key is known. This is the property we use (it survives a key-holding spender), explained in plain terms in §3.2.

[2] A practical note for implementers: BIP-341 introduces the `epoch` byte before the common SigMsg, shifting subsequent offsets by one byte relative to a naive reading of the common SigMsg layout in isolation. The corrected combined offset table (epoch + common SigMsg + BIP-342 tapscript extension) is the one given inline in §2.2; the canonical specification is BIP-341 §4.1.7 combined with BIP-342 §"Common signature message extension". All offsets in this paper assume `SIGHASH_DEFAULT`, no annex, and tapscript script-path spend (`ext_flag = 1`).

2.4 Bitcoin Inquisition

Bitcoin Inquisition is a fork of Bitcoin Core maintained by ajtowns as the community's experimental platform for covenant proposals [Bitcoin Inquisition repository]. It activates BIP 118, 119, 347, 348, 349 and other proposals on its own signet, allowing covenant constructions to be tested before mainline activation. Inquisition signet blocks are signed by a designated challenge key; transactions on it have no monetary value, but the validation rules — including all consensus checks for the activated BIPs — are exactly those any future mainline activation would enforce.

All on-chain evidence in this paper is from Inquisition signet. Each cited TxID is independently inspectable.

3. Two Techniques: Same-Signature Binding and Preimage Decomposition

This section presents the core technical contribution. The construction rests on three components: (1) a threat model identifying the attack `OP_CHECKSIGFROMSTACK` alone does not block; (2) **same-signature binding** (§3.2), closing that attack by forcing the witness-supplied preimage to equal the node-computed one; (3) **preimage decomposition** (§3.3), an emulation of the disabled `OP_SUBSTR` via `OP_CAT` + `OP_SIZE` + witness-provided padding, letting Script extract arbitrary fields from the bound preimage.

3.1 Threat model

`OP_CHECKSIGFROMSTACK` verifies `Schnorr_Verify(pubkey, message, signature)` for a triple all sourced from the witness stack. Used in isolation, it answers *"did the holder of this public key sign this message?"* — but the holder controls what the message is. Nothing in the opcode requires `message` to be a real sighash preimage of the spending transaction.

A naive use of CSFS for transaction introspection — *"the witness contains the preimage; CSFS verifies the signature is over it; therefore Script knows the transaction's outputs"* — fails to a trivial attack: the key holder can sign an *arbitrary* preimage with chosen field values and supply that as the witness. CSFS will validate, because CSFS only sees the stack.

The challenge is to *force* the witness preimage to be the same bytes the node actually computes when validating the transaction's signatures.

3.2 Same-signature binding

We construct a script that uses a single 64-byte Schnorr signature in two checks:

```
Witness: [ sig_64, ..., preimage_chunks, ..., pubkey ]

Script:
  ... (reassemble preimage from chunks) ...
  ... (compute tagged hash) ...
  <pubkey_32>
```

```
OP_CHECKSIGFROMSTACK          # CSFS check: sig over witness-derived hash
OP_VERIFY
<pubkey_32>
OP_CHECKSIG                   # CHECKSIG: sig over node-computed sighash
```

The CSFS check verifies `sig` against the tagged hash of the witness-supplied preimage. The `OP_CHECKSIG` check verifies the *same* `sig` against the tapscript-sighash the node computes for this transaction.

Why the two checks force the preimage to be honest. The two checks share one signature, and that shared signature is what does the work. Suppose the spender tries to cheat: it feeds a *fabricated* preimage to the CSFS check (say, one with a different `sha_outputs`), while the node, validating the transaction, computes the *real* preimage for the `OP_CHECKSIG` check. For the spend to succeed, the *same* signature would have to verify against two different messages — the tagged hash of the fabricated preimage and the tagged hash of the real one. A Schnorr signature does not allow that: a given signature is tied to a single message. Making one signature pass for two different byte-strings would require colliding the hashes involved (the TapSighash hash, or the hash computed inside Schnorr verification) — a SHA-256 collision, which is computationally infeasible ($\sim 2^{128}$ work).

We state this as the property the rest of the paper relies on:

> **Same-signature binding (informal).** If one Schnorr signature passes *both* the CSFS check (against a witness-supplied preimage) and the `OP_CHECKSIG` check (against the node-computed sighash), then the two preimages are the same bytes — barring a SHA-256 collision.

A point worth making explicit: this does *not* rely on the spender being unable to forge signatures. The spender holds the key and can sign anything it likes. What it cannot do is make *one* signature stand for *two* different transactions. The intuition is an exam analogy: the node holds the official answer sheet (the real preimage) and will not show it; the spender may submit its own answer sheet (the witness preimage); a single red pen — the one signature under one public key — grades both, and if it marks both "correct," the two sheets must be character-for-character identical.

In practice, for the purposes of script analysis: *if both checks pass on chain, the witness-supplied preimage is byte-identical to the real one.* The signer cannot lie about the preimage.

This is the central building block. Once witness bytes equal real bytes, Script can read transaction fields by reading bytes from the witness.

3.3 Preimage decomposition (OP_SUBSTR emulation via CAT + OP_SIZE)

`OP_SUBSTR` was disabled in Bitcoin Script's early consolidation, along with several other opcodes, as part of the 2010 hardening against adversarial script execution. The functionality it provided — extracting a substring at a specified position and length — turns out to be exactly what on-chain

covenant introspection requires: given a signature-message preimage, extract a specific field. We recover this functionality by composition of currently-available opcodes.

To verify a specific field — say, `sha_outputs` at byte offset 138 — Script needs to confirm not just *"the preimage is real"* but also *"the bytes at this position match an expected value"*. We chunk the witness:

```
preimage = pre_a || pre_b || pre_c || sha_outputs || post
              46       46        46          32          42       (bytes)
```

The witness presents these as separate stack items. The script:

1. Verifies `OP_SIZE` of each chunk, locking the boundaries (an attacker cannot shift `sha_outputs` left or right without changing some chunk's size).
2. Compares the `sha_outputs` chunk against an expected hardcoded value with `OP_EQUALVERIFY`.
3. Reassembles the full preimage with `OP_CAT` (preserving order).
4. **Global length assertion**: verifies the reassembled preimage is exactly 212 bytes. This rules out length-extension and truncation attacks against the hash preimage, and restricts the construction to the canonical sighash layout (no `SIGHASH_ANYONECANPAY` / `SIGHASH_NONE` / `SIGHASH_SINGLE` variants, no annex) for which the offset table holds.
5. Computes the tagged hash and runs the same-signature binding from §3.2.

Step 1 is the critical defense against position manipulation. Without `OP_SIZE` locks, an attacker could shift chunk boundaries (e.g., `pre_a` of length 47, "sha_outputs" chunk actually starting at byte 139 of the real preimage). With `OP_SIZE` locks combined with the global length assertion in step 4, every byte occupies a cryptographically pinned position within the 212-byte layout.

In practice, the script implementation may optimize by locking only one chunk's size with `OP_SIZE` and relying on same-signature binding to anchor the rest — any shifted reassembly would require a SHA-256 collision against the hardcoded field value, which is negligible. Both forms are valid; the formulation here is the maximally explicit one.

3.4 Generalization

Observation. *Any contiguous field of the canonical tapscript signature-message preimage can be extracted and verified using same-signature binding with O(1) additional opcodes per chunk boundary.*

The same construction reads any field — `sha_prevouts`, `sha_amounts`, `sha_scriptpubkeys`, `sha_outputs`, etc. — by adjusting chunking. The additional non-push opcode cost is approximately `2k` (one `OP_SIZE` + one `OP_CAT` per chunk), where `k` is the number of chunks.

More generally, CSFS+CAT gives Script field-level read access to any part of the tapscript signature-message preimage — the same read access a dedicated introspection opcode (`OP_TXHASH`-style) would provide — without activating one.

The remaining sections demonstrate this on chain.

4. Output Binding: Capability-Equivalent to CTV

4.1 Construction

`OP_CHECKTEMPLATEVERIFY` (BIP 119) commits the spending transaction to a 32-byte template hash covering outputs, sequences, locktime, version, and vin/vout counts. Its core function — *making the spending transaction's outputs match a precommitted template* — corresponds to constraining the `sha_outputs` field of the tapscript signature-message preimage.

Using §3, we extract `sha_outputs` (bytes 138–169) and verify it equals an expected value:

```
preimage chunks:  pre_a(46) + pre_b(46) + pre_c(46) + sha_outputs(32) + post(42)
                                                                          \
                                                          sha_outputs == hardcoded
```

The script has 26 non-push opcodes in 188 bytes. All witness items are under 80 bytes (Inquisition standardness limit). The full line-by-line annotated ASM, the witness layout, the hardcoded values, and the boundary-shift attack analysis are reproduced inline in §4.4 below; additional step-by-step stack traces appear in the supplementary Delving Bitcoin post [Zhang 2026d, /t/2483/3].

Goal / witness layout / enforcement target / TxID:

Goal	Force spending transaction's `sha_outputs` to equal a hardcoded value (CTV-equivalent template enforcement).
Witness layout	`[sig, post, sha_outputs, pre_c, pre_b, pre_a, pubkey]` (top to bottom).
Enforcement target	`sha_outputs` field at preimage byte offset 138.
Confirmed positive TxID	`2f345180bc6654353a247a50559c0be42153707b2ca29b621e57671ed41f37e6` (block 300379, Inquisition signet).
Negative-test result	`testmempoolaccept` rejection on different output address (`OP_EQUALVERIFY` failure).

4.2 Script design

The script has six execution phases:

1. **SIZE checks**: each chunk's length is verified, locking the field boundary.
2. **`sha_outputs` equality**: the chunk is compared against the hardcoded expected value via `OP_EQUALVERIFY`.
3. **CAT reassembly**: chunks are concatenated in order to reconstruct the 212-byte preimage.
4. **Tagged hash**: `tag_prefix || preimage` is hashed to produce the message used by CSFS.
5. **CSFS check**: the witness signature is verified against the witness-derived message under the hardcoded public key.
6. **CHECKSIG**: the same signature is verified by the node against the real tapscript-sighash. If both pass, the witness preimage equals the real preimage (same-signature binding, §3.2), and

the `sha_outputs` constraint is authenticated.

What this proves. A specific output template (one fixed by the hardcoded `sha_outputs` value) is enforced at the consensus layer for any spend of this UTXO.

What this does not prove. It does not prove CTV is unnecessary, nor that this construction is preferable to CTV in deployment. It proves the *capability* of constraining `sha_outputs` is available with already-activated opcodes.

4.3 Comparison with CTV

Property	`OP_CHECKTEMPLATEVERIFY`	CSFS+CAT output binding
Functional outcome	Enforce 32-byte template hash	Enforce 32-byte `sha_outputs`
Script length	~34 bytes	~188 bytes (5.5×)
Non-push opcodes used	1	26
Needs signing key?	No	Yes
Needs new opcode?	Yes (BIP 119)	No (CSFS, CAT already activated on Inquisition)

For the standard CTV use case (committing to a fixed output template), the two are functionally equivalent. CTV is dramatically cheaper in script size and in non-push opcode count, which is exactly the point of having a specialized opcode. CSFS+CAT requires the spender to provide a signature, which CTV does not.

We do **not** argue CSFS+CAT *replaces* CTV in deployment. We argue CSFS+CAT *covers the same capability surface* at a known cost. The deployment question — "is the size cost worth not adding a specialized opcode?" — is a separate matter touched in §9.

A note on resource limits: BIP-342 explicitly removed the legacy 201 non-push opcode limit and the 10,000-byte script size limit for tapscript [BIP-342, "Resource limits"]. The relevant limit that remains is the per-script sigops budget, which scales with witness size. Our constructions consume small sigops budgets; the relevant scarcity in this work is script bytes and witness bytes, not opcode count.

4.4 The on-chain construction, byte for byte

This subsection reproduces the confirmed output-binding construction at the level of detail needed to verify it without consulting any external resource. The transaction is `2f345180bc6654353a247a50559c0be42153707b2ca29b621e57671ed41f37e6` (Inquisition signet, block 300379), spending a 50,000-sat UTXO created by `05f9372dff7184ad736373165bc617ba100315bd2c187b0a4c893d796b2ca0cc`:1.

Hardcoded values in the tapleaf.

Constant	Value	Role
Expected `sha_outputs`	`eb0fdcd005abb92861dfa7c7680c 8a7b417e75c62c823b425615a0be e9082d7d`	the enforced 32-byte output template (offset 138)
`pubkey`	`ff1f9f...86b8` (x-only)	key for both CSFS and CHECKSIG
`SHA256("TapSighash")`	`f40a48df4b2a70c8b4924bf26546 61ed3d95fd66a313eb87237597c6 28e4a031`	pushed twice and `OP_CAT`-ed into the 64-byte BIP-340 tag prefix

Witness layout (top to bottom): `[ sig_64, post(42), sha_outputs(32), pre_c(46), pre_b(46), pre_a(46), control_block ]`. Each item is ≤ 80 bytes, satisfying Inquisition standardness. The five preimage chunks partition the 212-byte preimage at byte offsets `[0,46) [46,92) [92,138) [138,170) [170,212)`; only the fourth chunk coincides with a BIP-341 field boundary (`sha_outputs`, bytes 138–169) — the `pre_*` chunks are length-driven padding, not field-aligned.

Annotated tapscript ASM (the actual witness script of the spend):

```
OP_SWAP                              (1) bring the sha_outputs chunk to the top
OP_DUP                               (2) duplicate it
OP_PUSHBYTES_32 eb0fdcd0...082d7d    (3) push the expected sha_outputs
OP_EQUALVERIFY                       (4) require: witness sha_outputs == expected  (template)
OP_ROT                               (5) rotate pre_c up
OP_SIZE
OP_PUSHBYTES_1 2e                    (6) push 0x2e = 46
OP_EQUALVERIFY                       (7) require: |pre_c| == 46   (boundary lock)
OP_SWAP / OP_CAT x4                  (8) CAT-reassemble 5 chunks -> 212-byte preimage
OP_PUSHBYTES_32 f40a48df...4a031     (9) push SHA256("TapSighash")  (tag, upper)
OP_PUSHBYTES_32 f40a48df...4a031     (10) push SHA256("TapSighash")  (tag, lower, identical)
OP_CAT                               (11) form the 64-byte tag prefix
OP_SWAP
OP_CAT                               (12) tag_prefix || preimage   (64 + 212 = 276 bytes)
OP_SHA256                            (13) -> witness_SigMsg (32 bytes)
OP_OVER / OP_SWAP                    (14) bring sig up
OP_PUSHBYTES_32 ff1f9f...86b8        (15) push pubkey
OP_CHECKSIGFROMSTACK                 (16) (OP_SUCCESS_204) verify sig over witness_SigMsg
OP_VERIFY                            (17) CSFS must return 1
OP_PUSHBYTES_32 ff1f9f...86b8        (18) push the same pubkey
OP_CHECKSIG                          (19) verify the same sig over the node-computed SigMsg
```

Step (4) enforces the template; steps (16) and (19) are the two arms of same-signature binding (§3.2). Because one signature cannot pass for two different preimages, (16) + (19) on a single `sig` force the reassembled 212-byte witness preimage to equal the node's real preimage; step (4) then

pins the real `sha_outputs` to the hardcoded value — exactly CTV's enforced relation.

Why a boundary shift fails. A spender wishing to send funds elsewhere might try to misalign the chunks — e.g. make `pre_a` 47 bytes so the fourth chunk reads preimage bytes `[137,169)` instead of `[138,170)`, hoping `OP_EQUALVERIFY` at (4) still matches. For step (4) to pass, the shifted slice would have to equal the hardcoded constant; for (16)+(19) to pass, the reassembled preimage must still equal the real one. Concretely the attacker would need a transaction whose real preimage satisfies `preimage[137]=0xeb` and `preimage[138..169]` equal to the first 31 bytes of the constant — i.e. forcing a SHA-256 image (`sha_outputs`) to match a *fixed* 248-bit target. Because the target is fixed, there is no birthday shortcut; this is a second-preimage-style search far beyond any feasible work (well past 2^{128}). The same argument rules out every other shift. Hence the *only* feasible way to satisfy (4) together with (16)+(19) is the aligned case, in which the fourth chunk *is* the transaction's real `sha_outputs`. (The single `OP_SIZE` lock at (7) is a belt-and-suspenders boundary pin; the cryptographic anchor is (4)+(16)+(19), consistent with the "lock one chunk" optimization noted in §3.3.)

Negative test. Re-running the spend with an output paid to a different address, with all else held fixed, is rejected by `testmempoolaccept` with an `OP_EQUALVERIFY` failure at step (4) — the witnessed `sha_outputs` no longer matches the constant, and no aligned chunking can make it match. This is the on-chain analogue of a CTV template-mismatch rejection.

5. Input Binding: Capability Beyond CTV

5.1 Construction

`OP_CHECKTEMPLATEVERIFY` commits to outputs but *not* to inputs — its template does not include `sha_prevouts`. Some applications need to enforce that a spending transaction consumes a specific UTXO (or set of UTXOs), and CTV alone cannot do this.

CSFS+CAT can. Using §3, we extract `sha_prevouts` (bytes 10–41) instead of `sha_outputs`:

```
preimage chunks:  pre_a + sha_prevouts(32) + post
                                \
                          sha_prevouts == SHA256(prevout_A || prevout_B)
```

The expected value is hardcoded as `SHA256(serialized_outpoint_A || serialized_outpoint_B)`.

Goal / witness layout / enforcement target / TxID:

Goal	Force the spending transaction's input set to match a precommitted set of outpoints.
Witness layout	`[sig, post, sha_prevouts, pre_a, pubkey]`.
Enforcement target	`sha_prevouts` field at preimage byte offset 10.
Confirmed positive TxID	7311da6e30886fab6594b633bcb97e4436d49f02ecc244d4bc7c59cb1ea83558

Negative-test result	`testmempoolaccept` rejection when input set is changed.

5.2 Relation to BitVM bridge discussion

Robin Linus [Linus 2025b, Delving Bitcoin /t/1591] proposed a BitVM bridge construction relying on a "scriptSig trick" to bind the operator's onchain step to a specific UTXO. ajtowns observed in the same thread that CTV alone does not provide this binding: CTV commits to the *content* of `scriptSig` but not to *which UTXO that scriptSig comes from*. Jeremy Rubin proposed an `OP_CODESEPARATOR` workaround at the legacy script layer.

CSFS+CAT input binding addresses the same problem natively in tapscript: by directly verifying `sha_prevouts`, the construction enforces that the spending transaction has the precommitted input outpoints, regardless of how the inputs are constructed.

5.3 Scope

This shows that constraining a transaction's input set — not just its outputs — is available with already-activated CSFS+CAT, a capability missing from CTV's template. It does not establish that input binding is the right primitive for any particular application; the BitVM bridge is one motivating example, not the only design choice for that protocol.

The evidence is direct: the same script structure as §4 — only the chunk position and the expected value change. CSFS+CAT covers a strictly larger capability surface than CTV with respect to fields not present in CTV's template (which includes `sha_prevouts` and `sha_amounts`).

6. Eltoo State Replacement via a CSFS Delegation Ladder (CTV Settlement)

6.1 CSFS key delegation ladder

`SIGHASH_ANYPREVOUT` (BIP 118) provides *signature rebinding*: a signature signed under APO does not commit to the spending transaction's prevouts, allowing the same signature to authorize spends from multiple compatible UTXOs. This is the rebinding primitive eltoo channels rely on for state replacement.

We construct comparable rebinding semantics through CSFS itself, without a rebinding-specialized opcode. The construction is a *key-delegation chain* in the broader sense — each step's signing authority is forwarded to the next step's public key via a stack-verified signature, with the spending transaction's binding asserted at the chain's terminal step. This composition family includes related constructions in the literature on successive-signature delegation; we do not claim equivalence to any specific prior protocol but locate this work within that family for the reader's orientation.

```
K signs R_0's pubkey          (off-chain: K -> R_0 delegation)
R_0 signs R_1's pubkey        (off-chain: R_0 -> R_1 delegation)
R_1 signs R_2's pubkey        (off-chain: R_1 -> R_2 delegation)
```

```
R_2 signs CTV template hash      (off-chain: terminal authorization)

Tapscript leaf (42 bytes):
<K_pub_32> CSFS V CSFS V CSFS V CSFS V CTV
```

The script chains four `OP_CHECKSIGFROMSTACK` verifications, each consuming a (signature, message, pubkey) triple from the witness. The first verification's pubkey is hardcoded as the channel root `K`; subsequent verifications consume pubkeys from the witness, with each intermediate pubkey playing two roles: as the *message* verified by the previous CSFS (i.e., the previous key delegated to this one), and as the *pubkey* for the next CSFS. The fourth CSFS's message is the CTV template hash, consumed at the final `OP_CHECKTEMPLATEVERIFY` to enforce the settlement output template.

The construction realizes the eltoo state-replacement pattern through key delegation — each state has its own deterministically derived key `R_i = SHA256(K_secret || "state_i")`, and the ladder chain of CSFS signatures proves authority has been transferred from `K` through the state sequence to `R_i`. No opcode specialized for rebinding is required.

6.2 Comparison with Sanders' approach

Sanders [2026, Delving Bitcoin /t/359/17] independently implements eltoo on Inquisition using `OP_TEMPLATEHASH + OP_CHECKSIGFROMSTACK + OP_INTERNALKEY` — a regtest-only construction awaiting `OP_TEMPLATEHASH` activation. Sanders' construction uses `OP_TEMPLATEHASH` to provide rebinding semantics, with CSFS serving as a signature-verification utility.

The two constructions make different architectural choices: in Sanders' construction, the rebinding primitive is a specialized opcode (`OP_TEMPLATEHASH`); in ours, the rebinding primitive emerges from composing CSFS with a key-derivation discipline. Both approaches realize the same pattern at the protocol layer; they differ in which opcode carries the rebinding semantics. The choice has practical consequences: our construction requires no opcode beyond what is already activated on Inquisition signet, while Sanders' awaits `OP_TEMPLATEHASH` activation.

6.3 Structural comparison with APO+CTV

Property	APO+CTV ([Zhang 2026, Post #3])	CSFS+CTV ladder (this work, [Zhang 2026, Post #5])
State count	3	3
On-chain transactions	6	2
State mechanism	APO signature rebinding	CSFS ladder delegation
Settlement enforcement	CTV (3 outputs)	CTV (3 outputs)
Witness per spend	~150 B	512 B (carries full state history)
Opcodes required	APO + CTV	CSFS + CTV
Soft-fork dependencies (mainnet)	BIP 118 + BIP 119	BIP 347 + BIP 348 + BIP 119

Both implementations achieve the eltoo state-update topology under the same scenario. As ajtowns [Towns 2025] noted, CTV+CSFS is "somewhat more costly" than APO; our witness-size figures

quantify what "more costly" looks like for the rekey-ladder pattern. The trade-off is real and should not be obscured.

6.4 On-chain validation

Phase	TxID
Fund	92efc47554d25d74d9f567594d37375d8c8a1a2ea6bd1864600d5a49531fda34
Spend (3-output settlement, CTV-enforced)	b96324da612950339f564fbc88fe1d5c5751070e57bf796d32ee7171238382de

Negative tests (via `testmempoolaccept`, not broadcast):

Test	Intent	Result
`stale_state`	`R_0` signs state-2's CTV hash	Rejected (Schnorr verification failure)
`broken_chain`	Skip `R_1` (K -> R_0 -> R_2)	Rejected (Schnorr verification failure)

This shows the eltoo state-replacement pattern can be realized using only opcodes activated on Inquisition signet (CSFS + CTV), via a delegation ladder, with consensus-enforced settlement. It does not show the rekey ladder is optimal for production Lightning channels (the witness-size cost is real), nor that CSFS+CTV equals APO on every metric — it does not (see §6.3); APO's rebinding is constant-size, the ladder is O(n). What it establishes is that the eltoo topology is reachable today without APO.

7. Capability Comparison

7.1 Coverage table (for the patterns examined)

Capability	CTV (BIP 119)	APO (BIP 118)	TXHASH (BIP 346)	CSFS+CAT (this work)
Output binding (template)	✓	(via signing)	✓	✓ (§4)
Input binding (`sha_prevouts`)	×	× (only "any prevout")	✓	✓ (§5)
Signature rebinding (eltoo state replacement)	×	✓	(composable with CSFS)	✓ (§6)
Amount constraints (`sha_amounts`)	×	×	✓	✓ (§3 extension)

Capability	CTV (BIP 119)	APO (BIP 118)	TXHASH (BIP 346)	CSFS+CAT (this work)
Sequence constraints (`sha_sequences`)	partial (vin)	×	✓	✓ (§3 extension)
Sighash flag detection	×	×	×	✓ (preimage SIZE check; cf. Rubin 2025)
Activated on Inquisition (April 2026)?	✓	✓	×	✓

This table is scoped to the patterns examined in this paper and the standard reference patterns associated with each opcode. It does not claim coverage of every conceivable covenant.

7.2 Extensibility

Beyond the three on-chain validations, the same construction extends to:

- **Cross-input amount validation** via `sha_amounts` (bytes 42–73): hardcode the expected sum-of-inputs.
- **Co-input scriptpubkey constraints** via `sha_scriptpubkeys` (bytes 74–105): require all inputs to be of a particular type.
- **Position enforcement** via `input_index` (bytes 171–174): ensure the spending input is at a particular index.
- **Sighash flag detection** via the total preimage `OP_SIZE`: 212 vs other lengths distinguishes default vs `ANYONECANPAY` mode (Rubin's [2025] Conjecture #3).
- **Other proposed opcodes, by composition.** Each is ultimately a commitment to, or a read of, transaction fields, so the same technique reaches them: `OP_TEMPLATEHASH` (a narrow template-commitment hash — the §4 output-binding pattern at a different field scope); the output-restriction core of `OP_VAULT` (output binding plus native `OP_CHECKSEQUENCEVERIFY` and an alternate recovery tapleaf); and `OP_PAIRCOMMIT`, a restricted form of `OP_CAT` (pair-and-hash) that `OP_CAT` directly subsumes (cf. §10.13).

We have not demonstrated these extensions on chain — they are reasoned from the same construction, not separate experiments — but the script-level pattern is identical to §4–§6.

7.3 Capability statements (empirical observations)

We summarize the capability relation as two empirical observations. They are scoped to the patterns demonstrated in §4–§6 and the field-extraction technique of §3, and we state them as observations supported by on-chain evidence rather than as theorems; the universal-quantifier versions would require formalizing "covenant" as a relation over transactions and "constructible" as an inductive predicate over scripts, which we leave to follow-up work.

Observation 1 (Output coverage). *Each output-template constraint that* `OP_CHECKTEMPLATEVERIFY` *enforces — fixing* `sha_outputs` *to a 32-byte value — is reproduced by the CSFS+CAT construction of §4 at the cost of $O(n)$ additional script bytes, where n is the number of constrained fields. The output-binding instance is confirmed on chain (§4.4).*

Observation 2 (Input-binding gap). *The CSFS+CAT construction of §5 enforces a constraint on* `sha_prevouts` *(the spending transaction's input set) that CTV's template cannot express, since* `sha_prevouts` *is not a field of CTV's template. This input-binding instance is confirmed on chain (§5).*

Together they bracket the parity claim of this paper: on the output axis CSFS+CAT matches CTV (Observation 1), and on the input axis it strictly exceeds CTV's template (Observation 2). Neither asserts coverage of every conceivable covenant; both rest on the field-level extraction technique of §3, whose per-field cost is given in §3.4.

7.4 Limitations

Limitations:

1. **Script size and witness size**. CSFS+CAT output binding is ~5.5× larger than CTV in script bytes. The CSFS rebinding ladder produces a witness linear in state count. For high-throughput or size-sensitive deployments, this matters operationally even though it does not matter capability-wise.
2. **Signer-participation requirement**. CSFS+CAT constructions require a Schnorr signature from the spending party; the same-signature binding fundamentally needs a signer. For trustless covenants (those that should execute without any signer), this is a real limitation. A NUMS-key construction may circumvent this; we leave it to future work.
3. **Resource budget**. BIP-342 removed the 201 non-push opcode limit, but sigops budget remains (50 + witness bytes, with each CSFS or CHECKSIG consuming 50 sigops). Multiple field constraints composed in a single script can become sigops-budget-constrained; witness sizes scale accordingly.

The paper's claim is *capability parity* for the patterns examined, *not optimality*. CSFS+CAT covers what CTV and APO's rebinding pattern cover; whether it should *replace* either in any specific deployment is a separate question.

7.5 Relationship to Great Script Restoration (GSR)

The constructions presented here operate under the current tapscript sigops-budget framework. The Great Script Restoration proposal [Julian 2025, Delving Bitcoin /t/2094] generalizes this to a **varops budget** — assigning computational cost to all operations uniformly at 5,200 units per transaction weight unit — and separately re-enables several opcodes disabled in 2010, including `OP_SUBSTR` itself. Rusty Russell and Antoine Poinsot are active contributors to the technical discussion; a prototype branch and benchmarking harness are public.

Our work relates to GSR in two directions:

1. **Under varops accounting**: the witness-chunking costs in §3.3 and the rebinding-ladder witness scaling in §6 translate directly into varops consumption. The construction's feasibility does not depend on the choice between sigops and varops; only the accounting changes.
2. **If OP_SUBSTR is re-enabled under GSR**: the "preimage decomposition via CAT + OP_SIZE" construction in §3.3 becomes a historical artifact of the SUBSTR-absent period. The conceptual structure — same-signature binding combined with field extraction — remains identical, but field extraction collapses from a chunked-witness composition to a single opcode call. This is a deployment optimization, not a capability gain.

The empirical feasibility demonstrated in this paper therefore holds across both the current opcode environment and the proposed GSR environment, with distinct resource profiles. Whether GSR activates, and on what timeline, is independent of the question this paper addresses.

8. Putting the CAT-specific security objections in context

The cryptographic security of the constructions is covered by same-signature binding (§3.2). A separate layer of objection is raised against CAT and CSFS at the *systemic* level — fungibility, recursion, and MEV. This section is a short, qualitative discussion of those three; it is not a formal result. The observation is simple: when you actually try to build each feared construction, it turns out to be available — at comparable or lower cost — under other covenant proposals too, so the concern is not specific to CAT.

One clarification first. It is sometimes said that "CAT and CSFS leak transaction information." They do not. The signature-message preimage already encodes `sha_outputs`, `sha_prevouts`, `sha_amounts`, and the rest; the node computes these during validation regardless of the script. What CAT and CSFS add is the ability to *verify* a witness-supplied value against that node-known preimage (via same-signature binding, §3.2) — not to *reveal* anything new. So the real question is about the expanded *capability* to constrain transaction structure, which the next three paragraphs discuss.

Fungibility. A fungibility-breaking covenant restricts a transaction's outputs to a whitelist. The cost of building one runs roughly TXHASH (cheapest — a direct output read) < CAT+CSFS (a hardcoded whitelist Merkle root plus a witness Merkle proof, ~30–50 script-bytes) ≤ APO (the issuer pre-signs each whitelisted address) < CTV (one tapleaf template per whitelist entry, growing with the list). Any opcode that lets Script see output scriptpubkeys is enough; CAT is not the cheapest enabler.

Recursion. A recursive (quine) covenant needs Script to reference its own identity. CAT can reassemble its own tapleaf bytes and close the loop. But CTV alone cannot — and CTV *plus* a self-introspection opcode (e.g. a hypothetical `OP_TAPLEAFHASH` with the already-proposed `OP_INTERNALKEY`) can, as can APO or TXHASH plus such an opcode. Quine is an emergent property of self-introspection combined with template matching, not something unique to CAT; a community worried about recursion should specify which self-introspection capabilities it finds acceptable rather

than single out the assembly primitive.

MEV. Dynamic on-chain financial logic (AMMs, auctions, liquidations) is easiest to express under TXHASH (direct field reads), moderate under CAT+CSFS and APO, and hardest under CTV — whose pre-commitment requirement actually *limits* dynamic logic. So by construction cost the ordering roughly inverts the fungibility one: CTV < CAT+CSFS ≈ APO < TXHASH. CTV, often called the "safest" proposal, contributes least to MEV, and TXHASH the most; CAT sits in the middle.

Across all three objections the pattern is the same: the feared capability follows from transaction/output introspection in general, not from CAT specifically, and CAT sits in the middle of the cost ordering rather than at the dangerous end. We take no position on whether CAT — or any other opcode — should be activated; that is a community decision. We note only that an objection aimed at CAT would, to be consistent, have to apply equally to the other proposals that reach the same capability at similar or lower cost.

9. Discussion: What the Empirical Evidence Says About the Activation Question

We do not advocate for any particular activation ordering. The on-chain evidence in §4–6 changes the *shape* of the activation question in a way worth stating explicitly.

9.1 From "which capability" to "which optimization"

A common framing of Bitcoin's covenant debate is: *which dedicated opcode provides which uniquely needed capability*? This framing implicitly assumes that capability is what's at stake.

The empirical evidence in this paper suggests that, for the patterns examined (output binding, input binding, eltoo-style rebinding), **the capability is already present with already-activated opcodes (CSFS, CAT) on Inquisition signet**. What dedicated opcodes (CTV, APO, TXHASH) provide is *optimization*: smaller scripts, smaller witnesses, no signer requirement, simpler analysis.

Reframing the activation question as *"which optimizations to add, in what order"* — rather than *"which capabilities to enable"* — has at least three consequences worth considering:

- **Specialization timing**. Optimization decisions can be informed by observed deployment patterns rather than by speculation about future use.
- **Review surface**. Two general-purpose primitives (already activated on Inquisition for testing) versus several pattern-specific opcodes presents a different review surface for any subsequent mainnet activation discussion.
- **Engineering optionality**. Builders working under existing CSFS+CAT activation on Inquisition signet can prototype today; a future specialized opcode becomes an optional optimization rather than a blocker.

The table below maps the major covenant proposals onto how CSFS+CAT reaches the same capability, and whether we demonstrate it on chain or reach it by reasoning from the same technique:

Proposal	Capability, in essence	How CSFS+CAT reaches it	In this paper
CTV (BIP 119)	commit an output template	output binding (§4)	demonstrated on chain
TXHASH (BIP 346)	read selected transaction fields	field-level preimage read (§3.4)	shown in general
APO (BIP 118)	rebind a signature across prevouts	CSFS delegation ladder (§6) — eltoo *topology* (O(n)), not APO's O(1)	demonstrated on chain (eltoo)
OP_VAULT (BIP 345)	output restriction + delay + recovery	output binding + native CSV + recovery tapleaf	by composition
OP_TEMPLATEHASH (LNhance)	a narrow template-commitment hash	same field-hash technique as §4	by composition
OP_PAIRCOMMIT (LNhance)	pair-and-hash (a restricted `OP_CAT`)	subsumed by `OP_CAT` (cf. §10.13)	by composition

We state this as an *observation*, not a theorem. That CAT-based composition is broadly powerful has been argued before at the level of theory (e.g. [Poelstra 2021; Harding 2023]); what this paper adds is to make three instances concrete with confirmed on-chain transactions rather than by assertion. On the introspection axis — committing to or reading transaction fields — the dedicated-opcode family is, capability-wise, largely reproducible on top of already-activated CSFS+CAT. Three caveats bound the observation, and they are exactly where dedicated opcodes earn their place: (i) it is *capability*, not *efficiency* — our constructions are larger in script and witness; (ii) it covers *signer-participating* spends, since same-signature binding needs a spending signature, so keyless covenants are out of scope (§7.4); and (iii) the eltoo result reaches APO's topology through O(n) delegation, not APO's constant-size rebinding. Within those bounds, the differences among the proposals are matters of optimization, ergonomics, and — for the narrow opcodes — a *deliberately* bounded capability surface (a safety and review feature, not a deficiency), rather than of reachable capability.

9.2 Alignment with Tapscript's extension mechanism

BIP-347 (CAT) and BIP-348 (CSFS) are themselves activations of `OP_SUCCESS_x` opcode positions — `OP_SUCCESS_126` and `OP_SUCCESS_204` respectively. Whether or not those specific redefinitions were anticipated when BIP-342 reserved these slots, the activation path is the same one BIP-347 and BIP-348 themselves use. Our constructions are downstream of those activations and add no further architectural surface.

9.3 Preservation of batch validation

BIP-342 replaced `OP_CHECKMULTISIG` with `OP_CHECKSIGADD` to enable batch validation of Schnorr signatures. CSFS (BIP 348) uses Schnorr signatures with per-signature verification, consistent with the `OP_CHECKSIGADD` model. CAT (BIP 347) does not affect signature verification. Constructions

using CSFS+CAT therefore do not break batch validation, which is the operational property the BIP-342 redesign enabled.

9.4 What we are not arguing

We take no position on whether any opcode (CTV, APO, OP_VAULT, TXHASH) should be activated, nor do we claim CSFS+CAT is optimal for any protocol: review effort, ecosystem coordination, and specialization trade-offs all enter that decision. The empirical evidence only narrows the part of the question that is technical; the rest belongs with the community.

9.5 Answering by construction, not by argument

We set out (§1.3) to answer a question by construction rather than by argument. The on-chain transactions in §4–§6 present the facts: we do not claim CSFS+CAT supersedes CTV or APO, nor that it is optimal in any deployment. Activation decisions remain with the community.

10. Related Work

Ordered by topical proximity to this paper.

10.1 Poelstra: CAT + CHECKSIG analysis

Andrew Poelstra's early theoretical analysis of `OP_CAT + OP_CHECKSIG` argued for theoretical equivalence of the combination to a broad class of covenants. We extend this in two directions: (1) we use CSFS rather than CHECKSIG-only, which avoids needing the sighash to be predictable in advance; (2) we provide on-chain evidence with concrete TxIDs, where Poelstra's analysis was theoretical.

10.2 O'Connor 2022: TXHASH + CSFS

Russell O'Connor [O'Connor 2022] proposed `OP_TXHASH + OP_CHECKSIGFROMSTACKVERIFY` as a replacement for both CTV and APO, arguing for "primitives composition over packed patterns." We share the philosophical orientation. We diverge on the specific primitives: O'Connor's proposal requires `OP_TXHASH` (BIP 346) activation, which has not happened and is not on Inquisition; ours uses already-activated `OP_CAT` and `OP_CHECKSIGFROMSTACK`. Our rebinding mechanism (rekey ladder, §6) is structurally distinct: CSFS itself constructs the rebinding semantics, rather than CSFS verifying a hash from a new opcode.

10.3 Rubin 2024 (with Rearden): CSFS Re-Keying and Laddering

Rubin and Rearden [Rubin 2024] sketched the rekey delegation + N-hop laddering pattern in a December 2024 blog post, with stack-manipulation scripts (DUP TOALT shuffling) and BIP32-derived internal keys. The concept-level construction for §6 originates with them; we credit them explicitly

in the construction notes for the on-chain implementation [Zhang 2026 Post #5]. The implementation here departs in engineering surface — witness duplication instead of OP_DUP/ALTSTACK shuffling, deterministic SHA256 derivation instead of BIP32, and an Inquisition signet broadcast.

10.4 Rubin 2025: Fun with CSFS

Rubin [2025] sketches three CSFS constructions — equivocation penalty, lookup tables, and sighash flag detection — as exploratory exercises. The sighash flag detection conjecture (Conjecture #3: detecting `ANYONECANPAY` via preimage `OP_SIZE`) is a precursor to our full preimage decomposition. We extend Rubin's "detect flag" to "read arbitrary field," with on-chain validation Rubin did not provide.

10.5 Sanders 2026: TEMPLATEHASH + CSFS for LN-Symmetry

Greg Sanders [Sanders 2026] is the engineering lead implementing LN-Symmetry (eltoo) for Core Lightning. The current state is two variants: an APO-based variant runnable on Inquisition signet, and a `OP_TEMPLATEHASH + OP_CHECKSIGFROMSTACK + OP_INTERNALKEY`-based variant running in regtest only, awaiting `OP_TEMPLATEHASH` activation. As discussed in §6.2, Sanders' construction uses `OP_TEMPLATEHASH` to provide rebinding semantics; our construction uses CSFS itself. The two are different architectural choices, both valid; they differ in which opcode carries the rebinding primitive.

10.6 Decker, Russell, Osuntokun 2018: eltoo

Christian Decker et al. [Decker 2018] introduced eltoo as a *protocol* for symmetric Lightning channel updates — explicitly opcode-agnostic, with APO discussed only as the rebinding primitive available at the time. Our APO+CTV [Zhang 2026, Post #3] and CSFS+CTV [Zhang 2026, Post #5] implementations on the same Alice/Bob/Carol scenario are direct evidence that the protocol's opcode-agnostic claim holds in practice.

10.7 Linus 2025: Binohash

Robin Linus [Linus 2025a] demonstrates transaction introspection without softfork by exploiting `FindAndDelete` removal of script content from sighash calculation, combined with off-chain hash grinding (~84-bit collision resistance, ~$50 GPU cost). Binohash and our construction sit at different architectural layers: Binohash exploits a legacy sighash quirk to crack visibility at one layer; CSFS+CAT reads the preimage directly via the verified-copy mechanism. Both are paths through the same architectural barrier; ours requires CSFS and CAT activation, Binohash does not.

10.8 O'Beirne 2023: bundled covenant softfork proposal

James O'Beirne [O'Beirne 2023a, Delving Bitcoin /t/98] proposed activating BIP-118, BIP-119, and BIP-345 together as a bundled softfork, citing community use-case momentum and consolidated review effort. The thread is the source of the 2023 discussion of CSFS+CAT feasibility cited in §1.3.

O'Beirne's subsequent technical reply [O'Beirne 2023b, /t/98/14] articulated the capability objection this paper addresses empirically. Earlier, in a separate thread on TXHASH+CSFS unification [O'Beirne 2023c, Delving Bitcoin /t/60/4], O'Beirne requested "concrete CSFS use cases beyond CTV/APO unification" — a request to which the constructions in §4–§6 of this paper can be read as a partial response, 2.5 years later.

10.9 Harding 2023: CSFS+CAT as permissive alternative

David Harding's September 2023 response [Harding 2023, Delving Bitcoin /t/98/11] mentioned `OP_CSFS + OP_CAT` in passing as a "stub proposal ... a really small consensus change and yet it allows implementation of any sort of transaction introspection we might want." The remark was a throwaway aside within a broader skeptical post and did not develop the construction. The present paper does not claim to implement Harding's suggestion — the constructions, security argument, and on-chain validations are developed independently — but we note the remark for historical completeness, and we acknowledge it as the earliest public suggestion we have located that CSFS+CAT might be sufficient for general introspection.

10.10 Roose 2023: CATT toolkit proposal

Steven Roose [Roose 2023, Delving Bitcoin /t/125] proposed the CATT ("Covenant All The Things") toolkit combining `OP_TXHASH`, `OP_CHECKTXHASHVERIFY`, `OP_CHECKSIGFROMSTACK`, `OP_CAT`, `OP_TX`, 64-bit arithmetic, and `OP_TWEAKADD` as a general-purpose covenant kit. Within that thread Roose asserted that "building a covenant with just CAT+CSFS is indeed not reasonably feasible" — a claim predicated on the then-available understanding; the present paper's §4–§6 provide empirical clarification under the BIP-347/348 activation now present on Inquisition signet. We remain interested in the broader CATT design and see the present work as occupying the specific subset that already-activated opcodes can cover.

10.11 Towns 2023: CATT-thread skepticism

Anthony Towns' [Towns 2023, Delving Bitcoin /t/125/2] response in the CATT thread expressed skepticism that "it'll be reasonably feasible to do anything useful with [CAT+CSFS] in script." As with §10.10, we read this as calibration against the information available in late 2023. Towns' later observation in March 2025 of specific empirical gaps [Towns 2025] — quoted in §1.2 — shifted the same author's framing from "skeptical it is feasible" to "someone needs to do the work to demonstrate it." The present paper is an attempt at that demonstration.

10.12 Reardencode 2023: TXHASH+CSFS -> TEMPLATEHASH evolution

Ethan Heilman writing as Reardencode [Reardencode 2023, Delving Bitcoin /t/60] took Russell O'Connor's TXHASH+CSFS concept and developed it across a sequence of posts into what eventually became the LNHANCE `OP_TEMPLATEHASH` design — a narrower commitment-producing opcode intended to provide the "bridge" function of `OP_CAT` with a controlled capability boundary. Reardencode's design trajectory and ours answer different questions: theirs, "what is the smallest

new opcode that safely provides the bridge function"; ours, "can the bridge function be realized from already-activated opcodes at all." Both questions remain open as design choices for the broader community.

10.13 Josh & Reardencode 2025: CSFS+PAIRCOMMIT mass delegated introspection

Josh (with script credit to Reardencode) [Josh 2025, Delving Bitcoin /t/1599] described a CSFS + `OP_PAIRCOMMIT` construction providing "mass delegated introspection" — authorizing a large set of possible transactions with a single signature, under a Merkle-tree commitment. The construction is theoretical (regtest prototypes, no on-chain demonstration); it requires `OP_PAIRCOMMIT` (BIP 442), which is not currently activated on Inquisition. Our CSFS+CAT construction occupies the same conceptual neighborhood using `OP_CAT` as the merkle/aggregation primitive instead of `OP_PAIRCOMMIT`; the trade-off — ~9 additional bytes per Merkle hash node under `OP_CAT` + `OP_SIZE` boundary checks versus 1 byte under `OP_PAIRCOMMIT` — was analyzed in the same thread by Steven Roose [Roose 2025, Delving Bitcoin /t/1599/3].

10.14 Arshbot 2025: CTV+CSFS as first step

Arshbot [Arshbot 2025, Delving Bitcoin /t/1781] argued for CTV+CSFS activation as an incremental first step before broader TXHASH-style generality, citing narrower scope and higher consensus tractability. The framing is aligned with this paper's §1.8 non-supersession stance: the "first step" question and the "maximal already-activated capability" question are separable, and evidence on the latter does not dictate an answer on the former.

10.15 Julian 2025: Great Script Restoration and the varops budget

Julian [Julian 2025, Delving Bitcoin /t/2094] proposes the Great Script Restoration, a Tapscript leaf-version proposal that re-enables approximately fifteen opcodes disabled in Bitcoin v0.3.1 (including `OP_SUBSTR`, `OP_MUL`, `OP_DIV`, and related arithmetic and string primitives) and introduces a *varops budget* — a generalization of the current sigops budget that assigns computational cost to all operations uniformly at 5,200 units per transaction weight unit. Rusty Russell and Antoine Poinsot are active contributors to the technical discussion; a prototype branch and benchmarking harness are public.

The detailed relationship of this paper's work to GSR — under both the varops-accounting axis and the `OP_SUBSTR` re-enablement scenario — is treated in §7.5.

11. Conclusion

`OP_CHECKSIGFROMSTACK` and `OP_CAT` — both already activated on Bitcoin Inquisition signet — reach capability parity with CTV and APO for the covenant patterns examined in this paper: output binding (CTV-equivalent), input binding (beyond CTV), and signature rebinding for the eltoo state-replacement pattern (APO-equivalent for that pattern). We have shown this with three on-chain validations on Inquisition signet, supported by same-signature binding (§3.2): a single

signature cannot stand for two different transactions, so a witness-supplied preimage that passes both checks must be the real one.

The trade-off is script size and, for the rebinding ladder, witness size — not capability. Whether CSFS+CAT *should* replace CTV or APO in any specific deployment is a separate question, depending on optimization considerations not addressed here.

The empirical evidence narrows the technical part of Bitcoin's activation question. The remaining decisions — review effort, specialization timing, ecosystem coordination — are not technical and are appropriately the community's to make.

Every construction in this paper is confirmed on Inquisition signet and can be re-derived from the transactions cited here.

Appendices: Supplementary Reproduction Material

The paper's central result is self-contained: the same-signature binding argument is given in full (in plain language) in §3.2, and the output-binding construction is reproduced byte-for-byte in §4.4. The items below are **supplementary** — additional traces and the full records for the input-binding and rebinding constructions — provided for reproduction, not required to follow the argument.

- **A (Output binding — extended trace)**: the byte-level construction is inline in §4.4; Zhang 2026d, *Taproot-native output binding via sighash preimage decomposition — does this replace CTV?*, Delving Bitcoin /t/2483/3, 2026-05-13, adds the full per-step stack trace and the failed-substitution attack record (rejected by `testmempoolaccept` with `Script failed an OP_EQUALVERIFY operation`).
- **B (Input binding — full construction)**: Zhang 2026c, *Taproot-native prevout binding via sighash preimage decomposition*, Delving Bitcoin /t/2483, 2026-05-06. Contains tapscript design (196 bytes, hardcoded B prevout, dynamic SHA256 reconstruction), witness layout, on-chain TxIDs `b7959a71...` and `7311da6e...`, and the BitVM-bridge use-case discussion.
- **C (CSFS ladder construction — full witness layout)**: Zhang 2026b, *Eltoo State Chain on Signet Again: Three Rounds, Two Transactions (CSFS+CTV with rekey and ladder, no CAT)*, Delving Bitcoin /t/2430, 2026-04-21. Contains the rekey ladder structure, terminal CTV settlement, and on-chain TxIDs `92efc475...`, `b96324da...`.
- **D (Combined sighash preimage offset table)**: the canonical 212-byte `epoch || common SigMsg || BIP-342 extension` layout is given inline in §2.2 (Table). The canonical specification is BIP-341 §4.1.7 + BIP-342 §"Common signature message extension".

Additional companion material:

- Eltoo state chain APO+CTV variant (Post #3, Delving Bitcoin /t/2413, 2026-04): contains 6-tx confirmed chain.

References

1. **BIP 118** — Decker, C., Towns, A. *SIGHASH_ANYPREVOUT for Taproot Scripts*. https://github.com/bitcoin/bips/blob/master/bip-0118.mediawiki
2. **BIP 119** — Rubin, J. *CHECKTEMPLATEVERIFY*. https://github.com/bitcoin/bips/blob/master/bip-0119.mediawiki
3. **BIP 340** — Wuille, P., Nick, J., Ruffing, T. *Schnorr Signatures for secp256k1*. https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki
4. **BIP 341** — Wuille, P., Nick, J., Towns, A. *Taproot: SegWit version 1 spending rules*. https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki
5. **BIP 342** — Wuille, P., Nick, J., Towns, A. *Validation of Taproot Scripts*. https://github.com/bitcoin/bips/blob/master/bip-0342.mediawiki
6. **BIP 346** — O'Connor, R. *OP_TXHASH and OP_CHECKTXHASHVERIFY*. https://github.com/bitcoin/bips/pull/1500
7. **BIP 347** — Heilman, E., Sabouri, A. *OP_CAT in Tapscript*. https://github.com/bitcoin/bips/blob/master/bip-0347.md
8. **BIP 348** — Black, B., Rubin, J. *OP_CHECKSIGFROMSTACK*. https://github.com/bitcoin/bips/pull/1535
9. **Decker, C., Russell, R., Osuntokun, O.** (2018). *eltoo: A Simple Layer 2 Protocol for Bitcoin*. https://blockstream.com/eltoo.pdf
10. **Linus, R.** (2025a). *Binohash: Transaction Introspection Without Softforks*. https://github.com/RobinLinus/binohash
11. **Linus, R.** (2025b). *How CTV+CSFS improves BitVM bridges*. Delving Bitcoin. https://delvingbitcoin.org/t/1591
12. **O'Connor, R.** (2022). *TXHASH + CHECKSIGFROMSTACKVERIFY in lieu of CTV and ANYPREVOUT*. bitcoin-dev mailing list, February 2022. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2022-February/019872.html
13. **Rubin, J., Rearden** (2024). *CSFS Re-Keying and Laddering, Deterministic Update Rekeying, & Applications to LN-Symmetry*. https://rubin.io/bitcoin/2024/12/02/csfs-ctv-rekey-symmetry/
14. **Rubin, J.** (2025). *Fun with CSFS I*. https://rubin.io/bitcoin/2025/03/05/csfs-fun/
15. **Sanders, G. (instagibbs)** (2026). *LN-symmetry project recap*. Delving Bitcoin, 2026-04-14. https://delvingbitcoin.org/t/359/17
16. **Towns, A.** (2025). Reply on "CTV+CSFS: Can we reach consensus on a first step towards covenants?" Delving Bitcoin /t/1509/14, 2025-03-12. https://delvingbitcoin.org/t/ctv-csfs-can-we-reach-consensus-on-a-first-step-towards-covenants/1509/14
17. **Bitcoin Inquisition** repository. https://github.com/bitcoin-inquisition/bitcoin
18. **Zhang, A.** (2026a). *Eltoo State Chain on Signet: Three Rounds, Six Transactions (APO+CTV)*. Delving Bitcoin, 2026-04. https://delvingbitcoin.org/t/eltoo-state-chain-on-signet-three-rounds-six-transactions-apo-ctv/2413. (Post #3 in the series.)
19. **Zhang, A.** (2026b). *Eltoo State Chain on Signet Again: Three Rounds, Two Transactions (CSFS+CTV with rekey and ladder, no CAT)*. Delving Bitcoin, 2026-04-

21. https://delvingbitcoin.org/t/eltoo-state-chain-on-signet-again-three-rounds-two-transactions-csfs-ctv-with-rekey-and-ladder-no-cat/2430. (Post #5 in the series.)
20. **Zhang, A.** (2026c). *Taproot-native prevout binding via sighash preimage decomposition*. Delving Bitcoin, 2026-05-06. https://delvingbitcoin.org/t/taproot-native-prevout-binding-via-sighash-preimage-decomposition/2483. (Post #6 in the series.)
21. **Zhang, A.** (2026d). *Taproot-native output binding via sighash preimage decomposition — does this replace CTV?* Delving Bitcoin, 2026-05-13. https://delvingbitcoin.org/t/taproot-native-prevout-binding-via-sighash-preimage-decomposition/2483/3. (Post #7 in the series, posted as reply to /t/2483.)
22. **Zhang, A.** (2026e). *Three Layers of Commit-Reveal: A Unifying View of Bitcoin Script Evolution*. Manuscript in preparation.
23. **Poelstra, A.** (2021). *CAT and Schnorr Tricks I & II* — emulating covenants via OP_CAT + OP_CHECKSIG. Blockstream Engineering Blog. https://medium.com/blockstream/cat-and-schnorr-tricks-i-faf1b59bd298
24. **O'Beirne, J.** (2023a). *Bundled covenant soft fork (BIP-118, BIP-119, BIP-345)*. Delving Bitcoin. https://delvingbitcoin.org/t/98
25. **O'Beirne, J.** (2023b). *Reply on the covenant-bundle capability objection*. Delving Bitcoin. https://delvingbitcoin.org/t/98/14
26. **O'Beirne, J.** (2023c). *Reply requesting concrete CSFS use cases beyond CTV/APO unification*. Delving Bitcoin. https://delvingbitcoin.org/t/60/4
27. **Harding, D.** (2023). *Reply: OP_CSFS + OP_CAT as a permissive introspection stub*. Delving Bitcoin. https://delvingbitcoin.org/t/98/11
28. **Roose, S.** (2023). *CATT: Covenant All The Things toolkit proposal*. Delving Bitcoin. https://delvingbitcoin.org/t/125
29. **Towns, A.** (2023). *Reply: skepticism on CAT+CSFS feasibility (CATT thread)*. Delving Bitcoin. https://delvingbitcoin.org/t/125/2
30. **Heilman, E. (Reardencode)** (2023). *TXHASH + CSFS toward OP_TEMPLATEHASH*. Delving Bitcoin. https://delvingbitcoin.org/t/60
31. **Josh** (2025). *How CSFS+PAIRCOMMIT enables mass delegated introspection*. Delving Bitcoin. https://delvingbitcoin.org/t/1599
32. **Roose, S.** (2025). *Reply: OP_CAT vs OP_PAIRCOMMIT cost analysis*. Delving Bitcoin. https://delvingbitcoin.org/t/1599/3
33. **Arshbot** (2025). *CTV+CSFS as an incremental first step toward covenants*. Delving Bitcoin. https://delvingbitcoin.org/t/1781
34. **Julian** (2025). *Great Script Restoration (varops budget)*. Delving Bitcoin. https://delvingbitcoin.org/t/2094