Permanent Data Channel Closure

Josh · Secure Sovereign

Pre-Proposal · Standards Track · Created August 12, 2026

CONTENTS

Abstract

This BIP permanently closes the consensus-layer data embedding channels that have been used to bloat the Bitcoin blockchain with arbitrary non-monetary content. It is a permanent soft fork with no expiry.

The proposal invalidates the dedicated and unenforced embedding channels used for bulk non-monetary content: large `OP_RETURN` outputs, non-template scriptPubKeys, Tapleaf and P2WSH dead pushes, annexes, unconstrained witness versions, and oversized control blocks. It restores the long-established 83-byte `scriptPubKey` limit on OP_RETURN at the consensus layer, caps individual script-argument items at 256 bytes, and adds aggregate witness byte limits per input and per transaction that close multi-push fragmentation inside a transaction. It does not close payment-necessary fields (hashes, amounts, `nSequence`, ordering) or coinbase `scriptSig`. BIP141 witness weight is unchanged. A pre-activation UTXO remains spendable, including one reveal of data already committed in that coin. That spend does not authorize new envelopes.

This BIP is designed to work in conjunction with the Static Per-Output Miner Fee and Dynamic Escalation of the Per-Output Miner Fee pre-proposals. Those BIPs price UTXO creation. This BIP closes the data channels that would otherwise let spammers pack payload into witness fields of a few transactions. The three proposals together form a coherent anti-spam stack: the fee BIPs price output count; this BIP closes the dedicated channels.

Motivation

The Problem

Bitcoin has no consensus rule preventing arbitrary data from being embedded in transactions. Policy filters and relay rules can be bypassed by routing through a miner who does not enforce them. That is why data embedding debates have continued without resolution. Policy binds willing participants. Consensus binds everyone.

Starting with the inscription technique first exploited in 2022, a sustained trend has emerged around embedding arbitrary data into Bitcoin transactions. Ordinals, BRC-20 tokens, Runes, and similar protocols use Bitcoin as a data store rather than a monetary network. This creates significant and permanent burdens on node operators: storage, bandwidth, and RAM costs that fall on roughly 60,000 full nodes, none of whom were compensated for accepting that burden.

Pruning can drop old blocks from disk. It does not skip the download at first sync. *Full Cost of Running a Bitcoin Node* (v2.4, July 2026) estimates the aggregate non-monetary burden on the node network at approximately $4 million per year.

The core externality is structural. The actor embedding data pays a miner once. The burden of storing and serving that data falls on every node operator indefinitely. No market mechanism corrects this because node operators receive no part of the miner fee and have no mechanism to refuse specific content at the consensus layer.

The Two Embedding Surfaces

Data embedding operates on two surfaces that require different consensus tools to close.

The first is the per-item surface: individual data fields large enough to carry meaningful payloads. Output scriptPubKeys, OP_PUSHDATA payloads, witness stack items, Taproot annexes, and control blocks have all been exploited as single-field embedding vectors. Capping each field at a size that is sufficient for legitimate cryptographic use but insufficient for image-scale payloads closes this surface.

The second is the fragmentation surface: multiple sub-cap items assembled by an off-chain indexer into a larger payload. A per-item cap creates a window of permitted size per push. A transaction can contain many inputs, each with many witness stack items. An image or arbitrary file can be reconstructed from a sequence of small chunks distributed across witness fields in one transaction or spread across multiple transactions with a reassembly index maintained off-chain. Per-item caps raise the cost and complexity of embedding; they do not close the channel.

Closing both surfaces requires two complementary rule types: per-item caps on individual fields, and aggregate caps on total witness data per input and per transaction.

Why Permanent Rules Are the Right Mechanism

A temporary consensus rule creates an expiry event that the data-embedding ecosystem can plan around. Tooling can be built for the post-expiry environment, and activity resumes the moment the rule lapses. A permanent rule removes that planning horizon entirely and eliminates the repeated soft-fork cost of re-litigating the same question on a cycle.

The underlying principle does not weaken over time: node operators should not bear the indefinite cost of arbitrary data storage initiated by parties who paid miners once and never compensated them. The rule should match the principle.

The Fragmentation Surface in Detail

Two fragmentation paths must be closed by this proposal.

The within-transaction path uses multiple witness stack items, each within the per-item cap, within a single transaction. No existing consensus rule limits the count of such items or their aggregate byte total per input or per transaction. A single transaction can carry an arbitrarily large payload distributed across many small pushes.

The cross-transaction path splits a payload across multiple transactions, each carrying one or more small chunks, with a reassembly index maintained off-chain. Each individual transaction complies with per-item caps. The payload is reconstructed by an indexer that reads the chain as a data store rather than as a ledger of monetary transfers. The Ordinals developer community published fragmentation tooling confirming this path is viable and ready to deploy whenever per-item caps are enforced.

Closing both paths requires aggregate limits: a maximum total witness byte count per input and per transaction that makes large-payload fragmentation invalid regardless of how finely the payload is divided.

The Interaction With the Per-Output Fee BIPs

The Static Per-Output Miner Fee prices every new non-coinbase output permanently. The Dynamic Escalation layer keeps that price from decaying. Together they make UTXO creation costly in sats, permanently and irrecoverably.

Without data channel closure, a spammer facing the per-output fee can minimize output count while maximizing payload per output by fragmenting data within the witness fields of a small number of transactions. The fee BIPs price outputs; they do not directly price witness bytes. This BIP closes that gap. With all three proposals active, the available surface for data embedding is narrow, the per-output cost is permanent, and fragmentation increases the transaction count and thus the aggregate fee burden rather than routing around it.

Specification

Permanent Deployment

The rules in this BIP have no expiry. They take effect at `activation_height` and remain in effect for all subsequent blocks. There is no `active_duration` parameter and no EXPIRED state. The deployment transitions from ACTIVE to a permanent terminal state with continued enforcement, not expiry.

UTXO Grandfathering

Spend-side rules and output-side rules are not the same exemption.

Output rules (Rule 1). Apply to every transaction output created in a block at or above `activation_height`, including coinbase outputs. A transaction that spends only pre-activation UTXOs still may not create a non-template scriptPubKey. One old input does not exempt new outputs.

Spend rules (Rules 2 through 13). Apply only to inputs whose prevout was created at or after `activation_height`. Inputs spending UTXOs created before `activation_height` are exempt from Rules 2 through 13 so those coins are not frozen. That exemption is per input and covers only the spend of that already-created UTXO. It is not a license to mint new inscriptions.

A pre-activation envelope UTXO may still reveal its committed witness when it is spent. The next outputs in that same transaction are new. They must satisfy Rule 1. When those new outputs are later spent, Rules 2 through 13 apply. Funding a new P2TR from old coins does not grandfather the new output's spend: P2TR is a valid template, but an inscription-style witness on that later spend is invalid.

Mixed transactions are valid if every new output satisfies Rule 1 and every post-activation input satisfies Rules 2 through 13.

Coinbase `scriptSig` / extraNonce is out of scope. The coinbase transaction has no prevout, so Rules 2 through 13 do not apply to it. Rule 1 still applies to coinbase outputs.

Rule Set

The following rules apply to all blocks at or above `activation_height`.

Rule 1. Every output `scriptPubKey` created at or after `activation_height` must be exactly one of the templates below, or a valid OP_RETURN output as defined here. Any other `scriptPubKey` is invalid. Templates are byte-exact (push opcodes must be the minimal opcode for that length; `OP_PUSHDATA1` for a 20-byte hash is not P2PKH).

Name	`scriptPubKey`
P2PK compressed	`OP_PUSHBYTES_33 {33-byte pubkey} OP_CHECKSIG`
P2PK uncompressed	`OP_PUSHBYTES_65 {65-byte pubkey} OP_CHECKSIG`
P2PKH	`OP_DUP OP_HASH160 OP_PUSHBYTES_20 {20-byte hash} OP_EQUALVERIFY OP_CHECKSIG`
P2SH	`OP_HASH160 OP_PUSHBYTES_20 {20-byte hash} OP_EQUAL`
P2WPKH	`OP_0 OP_PUSHBYTES_20 {20-byte hash}`
P2WSH	`OP_0 OP_PUSHBYTES_32 {32-byte hash}`
P2TR	`OP_1 OP_PUSHBYTES_32 {32-byte x-only pubkey}`
P2A	`OP_1 OP_PUSHBYTES_2 0x4e73` (BIP 433)

OP_RETURN. A valid OP_RETURN output has `len(scriptPubKey) <= 83` bytes, the first opcode is `OP_RETURN` (0x6a), and the remainder is at most one data push (opcodes `OP_0`, `OP_1NEGATE`, `OP_1` – `OP_16`, or a single `OP_PUSHBYTES` / `OP_PUSHDATA1` / `OP_PUSHDATA2` with its payload). No further opcodes. 83 bytes is the `scriptPubKey` length (historical Core datacarrier), not 83 bytes of payload. Multiple OP_RETURN outputs in one transaction are each subject to this cap.

Bare multisig and all other non-template forms are invalid.

Rule 2. On inputs whose prevout was created at or after `activation_height`, individual OP_PUSHDATA payloads in that input's `scriptSig` and script-argument witness items exceeding 256 bytes are invalid, except for the redeemScript push in BIP16 scriptSigs. Script-argument witness items are witness stack elements passed to the script interpreter as inputs, excluding witness scripts, Tapleaf scripts, control blocks, annexes, and Taproot key-path signatures. Rule 2 does not apply to `scriptPubKey` (Rule 1) or to grandfathered inputs.

Rule 3. Spending undefined witness versions or Tapleaf versions is invalid. Creating outputs with undefined witness versions is already invalid under Rule 1: those scriptPubKeys are not a defined template. A future soft fork that defines a new witness version must add the corresponding template to the Rule 1 whitelist in the same deployment that defines spending rules. There is no pre-creation window.

Rule 4. Witness stacks with a Taproot annex are invalid.

Rule 5. Taproot control blocks larger than 257 bytes are invalid.

Rule 6. Tapscripts including OP_SUCCESS opcodes anywhere, even unexecuted, are invalid.

Rule 7. Tapscripts containing OP_IF or OP_NOTIF anywhere, even unexecuted, are invalid.

Rule 8. The aggregate byte count of all script-argument witness items across **post-activation inputs only** in a single transaction must not exceed `max_witness_bytes_per_tx`. Grandfathered inputs are omitted. The same exclusions as Rule 2 apply. A transaction where that aggregate exceeds `max_witness_bytes_per_tx` is invalid.

Rule 9. The aggregate byte count of all script-argument witness items within a single **post-activation** input must not exceed `max_witness_bytes_per_input`. Grandfathered inputs are omitted. The same exclusions as Rule 2 apply. An input where the aggregate exceeds `max_witness_bytes_per_input` is invalid.

Rule 10. Tapleaf scripts containing unreferenced push data are invalid. The determination is static, without execution, using the algorithm in Unreferenced push analysis with `kind = TAPSCRIPT` (Appendix A).

Rule 11. Tapscripts containing push constants whose aggregate byte count exceeds `max_tapleaf_push_bytes` are invalid, as a secondary enforcement layer on Tapleaf script body size independent of the unreferenced push analysis in Rule 10. This cap is load-bearing for consume-and-ignore patterns (for example `OP_PUSH` followed by `OP_DROP`) that Rule 10 treats as referenced.

Rule 12. Segwit v0 witness scripts (the last witness stack item on P2WSH spends) containing unreferenced push data are invalid. The static analysis is the same as Rule 10 with `kind = WITNESS_V0`.

Rule 13. Segwit v0 witness scripts containing push constants whose aggregate byte count exceeds `max_witness_script_push_bytes` are invalid. This is the P2WSH counterpart to Rule 11. Witness scripts remain excluded from Rules 2, 8, and 9: those rules cap script-argument items, not script bodies.

Unreferenced push analysis

Used by Rules 10 and 12. Walk the script as a sequence of opcodes plus immediate payloads. A **push** is any of: `OP_0`, `OP_1NEGATE`, `OP_1` through `OP_16`, a direct push `0x01`–`0x4b`, `OP_PUSHDATA1`, `OP_PUSHDATA2`, or `OP_PUSHDATA4`.

For each push, let `next` be the following opcode after that push and its payload.

- If there is no `next` (the push is last), the push is unreferenced and the script is invalid.
- If `next` is itself a push, this push is **not** classified by that fact. Consecutive pushes are how `OP_2 <key> <key> OP_3 OP_CHECKMULTISIG` and other multi-argument scripts work. Multi-chunk dumps are invalid because the last chunk is followed by a non-consuming opcode (`OP_ENDIF`, `OP_NOP`, ...) or is last; Rule 11 and Rule 13 cap consume-and-ignore patterns (`OP_PUSH` / `OP_DROP`).

- If `next` is not a push and `stack_items_read(next, kind) == 0`, the push is unreferenced and the script is invalid.
- If `next` is not a push and `stack_items_read(next, kind) >= 1`, the push is referenced.

`stack_items_read(opcode, kind)` is defined for every byte `0x00` – `0xFF` in Appendix A. Rule 10 uses `kind = TAPSCRIPT`. Rule 12 uses `kind = WITNESS_V0`. The value is the data-independent prefix of main-stack items Bitcoin Core's `EvalScript` requires from the top: items the opcode always inspects or pops before any data-dependent extra pops. Replacement versus deletion does not matter. `OP_DUP`, `OP_DROP`, `OP_IF`, `OP_SIZE`, and `OP_CHECKLOCKTIMEVERIFY` are all `>= 1`. `OP_NOP`, `OP_DEPTH`, `OP_FROMALTSTACK`, `OP_ELSE`, `OP_CODESEPARATOR`, disabled opcodes, `OP_RETURN`, and unknown bytes are `0`.

`OP_PUSH` followed by `OP_DROP` is referenced under this test. Rule 11 (Tapleaf) and Rule 13 (P2WSH) exist to cap that consume-and-ignore pattern by aggregate push-constant bytes. `OP_FALSE OP_IF <data> OP_ENDIF` is invalid because `<data>` is followed by `OP_ENDIF` (`stack_items_read = 0`). Standard P2WSH multisig remains valid because each key push is followed by another push or by `OP_CHECKMULTISIG`.

Tapscript `OP_SUCCESS` bytes have `stack_items_read = 0` and are independently invalid under Rule 6.

Parameters

Parameter	Description	Value
`max_witness_bytes_per_input`	Aggregate script-argument witness byte cap per input	TBD via calibration
`max_witness_bytes_per_tx`	Aggregate script-argument witness byte cap per transaction	TBD via calibration
`max_tapleaf_push_bytes`	Aggregate push constant byte cap per Tapleaf script body	TBD via calibration
`max_witness_script_push_bytes`	Aggregate push constant byte cap per P2WSH witness script body	TBD via calibration
`activation_height`	Set by signaling process	Determined at deployment

Provisional anchors for calibration: `max_witness_bytes_per_input` should be set high enough that legitimate complex multisig and Lightning channel constructions are unaffected and low enough that fragmented image-scale payloads are blocked within a single input. `max_witness_bytes_per_tx` should be set to a multiple of `max_witness_bytes_per_input` sufficient for high-input-count CoinJoin and batch transactions while still blocking cross-input fragmentation at payload scales that impose meaningful node burden.

`max_witness_script_push_bytes` may equal `max_tapleaf_push_bytes` after calibration; it is listed separately because segwit v0 scripts have a 10,000-byte historical ceiling and legitimate P2WSH multisig paths may need a larger push-constant budget than a Tapleaf.

Defined Witness Versions

Rules 1 and 3 apply to all witness versions not defined as of this BIP. Creation of such outputs is invalid under Rule 1. Spending them is invalid under Rule 3. Pre-activation UTXOs remain grandfathered. The defined versions are:

- Witness v0 with a 20-byte program (P2WPKH) or a 32-byte program (P2WSH), per BIP 141.
- Witness v1 with a 32-byte program (Taproot/P2TR), per BIP 341. The only defined Tapleaf version is 0xc0 (Tapscript), per BIP 342.
- Witness v1 with the 2-byte program 0x4e73 (P2A, per BIP 433), spent with an empty witness stack only.

Witness item classification

Rules 2, 8, and 9 count only **script-argument** witness items:

- **P2WPKH:** both items (signature, pubkey) are script-argument.
- **P2WSH:** every item except the last. The last item is the witness script (Rules 12–13).
- **P2TR key-path:** the signature item is excluded. BIP 341 already makes extra items invalid.
- **P2TR script-path:** every item except the last two (Tapleaf script, control block). An annex is invalid under Rule 4.
- **P2A:** empty witness only.

Byte counts are `len(item)` for each counted item, summed as 64-bit integers.

Activation

This BIP deploys via BIP-8 or BIP-9 style signaling with a minimum activation window of one year and no mandatory lock-in fallback. The threshold and signaling parameters are set at deployment. Miners who do not signal are not penalized. If signaling does not reach threshold within the window, the process restarts with revised parameters or renewed community discussion.

BIP-8/9 without lock-in-on-timeout is miner signaling. Node operators absorbing non-monetary burden are a political constituency for this rule; they are not the signaling threshold. If miners do not signal, this BIP does not activate under the specified mechanism. User-activated soft fork deployment is a separate choice and is not specified here.

Rationale

Why Permanent Rather Than Temporary

A temporary rule creates an expiry date. Expiry gives the data-embedding ecosystem a planning horizon: tooling can be built for the post-expiry environment, and activity can resume the moment the rule lapses. A permanent rule removes that horizon entirely. The case for permanence rests on the same foundation as the case for the permanent per-output fee: the cost imposed on node operators by arbitrary data storage is not a temporary problem. The externality is structural. The rule should be structural too.

Why Per-Item Caps Alone Are Insufficient

A 256-byte cap per individual witness item is the correct tool for closing single-field contiguous data embedding. It does not close fragmentation because it imposes no constraint on the number of items or on their aggregate size. Rules 8 and 9 are the missing constraint. They place a ceiling on total witness data per input and per transaction that makes large-payload fragmentation invalid regardless of how finely the payload is divided. A 50 kilobyte image split into 256-byte chunks requires roughly 200 chunks. With appropriate aggregate limits, that transaction or that input is invalid. With cross-transaction fragmentation, the aggregate limit per transaction is reached before the payload fits, so the payload requires more transactions, each paying the per-output fee, making the total cost of data embedding proportional to data volume. That is the correct pricing signal.

Why the Aggregate Limits Are Scoped to Script-Argument Witness Items

Witness scripts and Tapleaf scripts are excluded from the aggregate counts because they are scripts, not data. Their size is governed by legitimate cryptographic and contract requirements. Taproot key-path signatures are fixed-length by consensus. Control blocks encode the Merkle path to a Tapscript and are governed by Rule 5. Annexes are invalidated by Rule 4. The aggregate limits target the data surface, not the script surface, consistent with this BIP's purpose.

Why OP_RETURN Is Capped at 83 Bytes

OP_RETURN outputs are provably unspendable and do not enter the UTXO set. Historically, up to 83 bytes of `scriptPubKey` have been tolerated to avoid unprovably unspendable spam in other output scripts. This BIP restores that cap at consensus. It does not replace the per-output fee: `OP_RETURN` still pays *Static Per-Output Miner Fee* when that BIP is active. At a correctly calibrated per-output fee, the cost per OP_RETURN output is negligible for legitimate low-volume use and material only for high-volume data embedding. The 83-byte figure is total script length, matching historical `-datacarriersize`, not 83 bytes of payload after `OP_RETURN`. Capping OP_RETURN at consensus restores a principle that Core relay policy previously enforced but that miners could bypass.

Why OP_IF and OP_NOTIF Are Invalidated in Tapscript

With Taproot, branching conditions can be evaluated off-chain, revealing only the intended execution path. OP_IF is redundant in Tapscript for well-designed monetary constructions and has been commonly used to inject data that is skipped at execution. Closing this gap eliminates a common embedding abuse. Current Lightning is mostly P2WSH and is not affected. Taproot Miniscript and taproot-channel HTLC leaves that still use OP_IF must split those branches into separate leaves before activation.

Why Undefined Witness Versions Are Closed to Creation and Spending

Undefined witness versions have unlimited witness stack sizes, creating an unconstrained embedding surface. Rule 3 closes spending. Rule 1 already closes creation: an undefined-version scriptPubKey is not a defined template. Leaving creation valid would park a new class of hash-channel UTXOs and is not required for upgrades.

A future witness version is added the same way P2TR was added: one soft fork that both whitelists the output template and defines spending rules. Senders do not need a pre-creation window. Pre-activation UTXOs, including any rare undefined-version outputs that already exist, remain spendable under grandfathering.

Why BIP141 Weight Is Unchanged

This BIP does not reprice witness bytes. After Rules 2, 7, and 10 through 13, bulk JPEG payload is invalid, so the BIP141 discount no longer subsidizes dedicated embedding. Remaining witness bytes are signatures, scripts, and control blocks.

Repricing all witness data is a different proposal and would mix a Lightning and batching design fight into this activation. It is not required for channel closure. A later BIP could, if desired, apply full weight to script-argument witness items only (the Rule 2 set) without touching signature or script weight. That change is out of scope here.

Why Bare Multisig Is Closed at the Output Template Layer

Bare multisig scriptPubKeys encode public keys directly in the output script. The Stamps protocol exploits this by substituting arbitrary data for valid public key bytes. Because the output script is stored permanently in the UTXO set until spent, this creates indefinite node storage burden with no monetary justification. A length cap on scriptPubKeys does not cleanly close this channel because a three-of-three bare multisig output can be constructed within plausible byte limits while still carrying meaningful data payloads in its fake keys. The output template whitelist in Rule 1 closes the channel completely: bare multisig is not a defined template and is invalid at output creation, before any data reaches the UTXO set. Legitimate monetary use cases that historically relied on bare multisig are fully served by P2SH, P2WSH, and P2TR, all of which are on the whitelist.

Why Tapleaf Scripts Are Analyzed for Unreferenced Push Data

The exclusion of Tapleaf scripts from per-item witness caps in Rule 2 is correct for legitimate contract logic: push constants in a Tapleaf script serve as inputs to opcodes that consume them. The Taproot envelope embedding technique abuses this exemption by placing arbitrary data as push constants that are never consumed: typically `OP_FALSE OP_IF <data> OP_ENDIF`, where `<data>` is followed by `OP_ENDIF` (`stack_items_read = 0`). These dead pushes are present in the witness at spend time, visible to every node, and stored indefinitely. Rule 10 invalidates a script-body push whose immediate next non-push opcode consumes nothing, or a push that is last. Consecutive pushes are not, by themselves, unreferenced: that pattern is monetary (`OP_2 <keys> OP_3 OP_CHECKMULTISIG`). Rule 11 caps consume-and-ignore patterns such as `OP_PUSH` followed by `OP_DROP` that Rule 10 treats as referenced.

Why P2WSH Witness Scripts Get the Same Treatment

Rules 2, 8, and 9 correctly exclude witness scripts from per-item and aggregate witness-item caps: those scripts are program bodies, not data fields. Segwit v0 witness scripts can still be up to 10,000 bytes. Dead pushes inside a P2WSH script are therefore a remaining dedicated channel if only Tapleaf bodies are analyzed.

Rules 12 and 13 close that channel. Rule 12 applies the Rule 10 unreferenced-push analysis to the last witness stack item on a P2WSH spend. Rule 13 applies the Rule 11 aggregate push-constant cap via `max_witness_script_push_bytes`. BIP16 redeemScripts are not given this treatment: P2SH script-path dumps remain a leftover by scope, not a recommended embedding path. RedeemScript pushes remain excepted from the 256-byte item cap in Rule 2. P2SH is not the high-bandwidth inscription channel; P2WSH was. A later BIP could close P2SH without changing this one.

Interaction With the Per-Output Fee BIPs

The Static Per-Output Miner Fee and Dynamic Escalation BIPs price UTXO creation. This BIP closes the data channels. The data-channel BIP and the static fee BIP have no consensus dependency on each other. Dynamic escalation still requires the static fee BIP. Coordinated activation is stronger: without this BIP, an attacker facing only the fee BIPs can minimize UTXO count and pack data into witness fields; without the fee BIPs, an attacker facing only this BIP can spread data across many cheap transactions.

Backwards Compatibility

Existing coins are never frozen. A pre-activation UTXO can be spent under pre-activation spend rules, including one inscription reveal if that coin already committed an envelope. That spend does not create a new grandfathered inscription UTXO. New outputs created at or after `activation_height` must match Rule 1 even when funded entirely by pre-activation inputs, and their later spends are fully bound by Rules 2 through 13. Coinbase outputs created at or after activation must also be a defined template or a valid 83-byte OP_RETURN. Coinbase `scriptSig` is unchanged.

Some wallet software, including certain Miniscript compilers, habitually creates Tapleaves containing OP_IF and may place required scripts deeper than 7 Taptree levels. Lightning taproot-channel designs that keep OP_IF in a Tapscript HTLC leaf must split those branches. Current P2WSH Lightning funding scripts are unaffected by Rule 7. The mitigations are: split OP_IF branches into separate Tapleaves and keep every required script-path leaf at depth 7 or less. Taproot control blocks are capped at 257 bytes, limiting Taptrees to 128 script leaves, which is sufficient for modern complex transactions but may constrain advanced off-chain contract schemes that rely on very large script trees.

Rules 8 and 9 require wallets and transaction construction libraries to track aggregate witness byte consumption per input and per transaction. Rules 12 and 13 require P2WSH constructors to avoid unreferenced pushes and to stay within `max_witness_script_push_bytes`. The calibration checklist must confirm that no legitimate high-witness-count use case (Lightning channel opens, complex multisig, CoinJoin, large P2WSH) is invalidated at correctly calibrated parameter values.

Wallets that do not update will produce transactions that enforcing nodes reject at and after `activation_height`. The minimum one-year signaling window provides time for updates. As with the per-output fee BIPs, updates should be complete before activation.

Reference Implementation

High-level pseudocode:

```
ACTIVATION_HEIGHT = <determined by signaling process>
MAX_WITNESS_BYTES_PER_INPUT = <value to be set at activation>
MAX_WITNESS_BYTES_PER_TX = <value to be set at activation>

def script_argument_witness_items(input_witness):
    # Exclude: witness scripts, Tapleaf scripts, control blocks,
    # annexes, and Taproot key-path signatures.
    # Return the remaining witness stack elements.
    ...

def is_valid_transaction(tx, block_height):
    if block_height < ACTIVATION_HEIGHT:
        return True

    # Rule 1: every new output, including coinbase outputs
    for out in tx.outputs:
        if not is_allowed_scriptpubkey(out.scriptPubKey):
            return False

    if tx.is_coinbase:
        return True

    tx_aggregate = 0

    for inp in tx.inputs:
        if inp.utxo_height < ACTIVATION_HEIGHT:
            continue  # spend rules only; Rule 1 already applied to outputs

        items = script_argument_witness_items(inp.witness)

        # Rule 2: per-item cap
        for item in items:
            if len(item) > 256:
                return False

        # Rule 9: per-input aggregate cap
        input_aggregate = sum(len(item) for item in items)
        if input_aggregate > MAX_WITNESS_BYTES_PER_INPUT:
            return False

        tx_aggregate += input_aggregate

    # Rule 8: per-transaction aggregate cap (post-activation inputs only)
    if tx_aggregate > MAX_WITNESS_BYTES_PER_TX:
        return False

    # Rules 3-7, 10-13: omitted here for brevity; see Specification
    ...
```

```
    return True
```

Detailed test vectors, integer arithmetic requirements, precise witness item classification, and edge cases will be provided in a future numbered BIP submission.

Calibration Checklist

A. Aggregate Limit Determination (Hard Gate)

- Establish the maximum script-argument witness byte count per input required by legitimate high-witness-count constructions: Lightning channel opens, complex multisig (5-of-7 and above), CoinJoin inputs at realistic participant counts, and Taproot script-path spends with deep trees up to the 128-leaf limit imposed by Rule 5.
- Set `max_witness_bytes_per_input` above the 99th percentile of legitimate per-input witness consumption observed across full chain history.
- Confirm that an image-scale fragmented payload (50 kilobytes, split into 256-byte chunks) requires per-input or per-transaction witness aggregates that exceed the calibrated limits and is therefore invalid.
- Set `max_witness_bytes_per_tx` to accommodate high-input-count CoinJoin transactions at realistic participant counts while blocking cross-input fragmentation at image scale.
- Lock in both values or document required adjustments with full reasoning before the proposal moves forward.
- Set `max_tapleaf_push_bytes` and `max_witness_script_push_bytes` above legitimate Miniscript, Lightning, and P2WSH multisig push-constant totals and below image-scale script-body dumps. Confirm Rule 11 and Rule 13 catch consume-and-ignore patterns that Rule 10 and Rule 12 treat as referenced.

B. Collateral Damage (Must Pass)

- Lightning channel opens: confirm per-input and per-transaction aggregate witness bytes remain below limits at all relevant funding configurations.
- CoinJoin transactions at 50, 100, and 200 participants: confirm aggregate limits are not reached.
- Complex multisig up to 5-of-7 and above: confirm script-path spend witness aggregates remain within limits.
- Taproot script-path spends at maximum allowed tree depth (128 leaves, Rule 5): confirm aggregate limits not reached.
- Ordinary payments (P2WPKH, P2TR keypath): confirm negligible witness aggregates relative to limits.
- P2WSH multisig spends: confirm Rule 12 and Rule 13 do not invalidate legitimate witness scripts at calibrated `max_witness_script_push_bytes`.

C. Spam Efficacy

- Confirm that image-scale payloads (at 50 kilobytes and above) cannot be embedded in a single transaction without exceeding Rule 8 or Rule 9 limits, and cannot be embedded in a P2WSH witness script or Tapleaf body without exceeding Rule 11 or Rule 13.
- Confirm that cross-transaction fragmentation at image scale, priced through the per-output fee BIPs, produces a per-byte cost that makes sustained spam economically irrational at calibrated fee levels.
- Model the combined cost of the three-BIP stack (per-output fee plus aggregate witness limits) against realistic attacker budgets for inscription-wave-scale activity.

D. Consensus Edge Cases

- Confirm the witness item classification in Specification (P2WPKH, P2WSH, P2TR key-path, P2TR script-path, P2A).
- Confirm integer arithmetic for aggregate byte counting so all implementations agree bit-for-bit.
- Confirm grandfathering: Rule 1 on every output created at or after `activation_height`; Rules 2–13 only on inputs whose prevout was created at or after that height.
- Confirm reorg behavior at `activation_height`: blocks below the activation height are never subject to any rule in this BIP regardless of reorg depth.
- Confirm that the witness item classification algorithm is reproducible from block data alone without mempool history.
- Confirm implementations match the Appendix A grids bit-for-bit for both `WITNESS_V0` and `TAPSCRIPT`.
- Confirm Rule 12 does not invalidate standard P2WSH `OP_2 <keys> OP_3 OP_CHECKMULTISIG`: consecutive pushes are not unreferenced.

E. Wallet and Exchange Validation

- Share calibration results with at least one Lightning implementation developer, one exchange wallet engineer, and one CoinJoin implementation developer before proceeding.
- Confirm that fee estimation and transaction construction tooling can enforce aggregate witness limits internally before broadcasting.

F. Exit Criteria

- Both aggregate limit values confirmed and gate A passes.
- Collateral damage checklist B passes at calibrated values.
- Spam efficacy checklist C confirms image-scale payloads are blocked or prohibitively expensive under the three-BIP stack.
- Consensus edge cases in checklist D resolved and written into Specification.

Appendix A. `stack_items_read`

Normative. Every opcode byte has an assigned integer. Implementations MUST NOT treat any byte as unspecified.

Source of the counts: Bitcoin Core `EvalScript` in `src/script/interpreter.cpp` and `IsOpSuccess` in `src/script/script.cpp` as of commit `b2c45888` (2026-08-12). A later Core change that alters stack effects of an existing opcode requires a new consensus discussion; it does not silently amend this appendix.

Definition

`stack_items_read(opcode, kind) -> integer >= 0`

`kind` is `WITNESS_V0` (Rule 12) or `TAPSCRIPT` (Rule 10).

The return value is how many main-stack items, counted from the top, the opcode **always** requires before any data-dependent extra pops. It is not the eventual total for `OP_PICK`, `OP_ROLL`, or `OP_CHECKMULTISIG`. Those opcodes always consume the top item first (`n` or `nKeys`), so a push immediately before them is referenced whenever the V0 count is `>= 1`.

Unexecuted `OP_IF` / `OP_NOTIF` do not pop in `EvalScript`. This analysis is static and does not track `vfExec`. Both opcodes are `1` in both kinds. Tapscript still invalidates them under Rule 7.

`OP_CHECKLOCKTIMEVERIFY` and `OP_CHECKSEQUENCEVERIFY` inspect the top item and leave it in place. They are `1`. This BIP assumes the consensus flags that activate those opcodes; it does not model the historical NOP2/NOP3 fallback.

Assignment (apply in order)

1. If `kind == TAPSCRIPT` and `IsOpSuccess(opcode)` (BIP 342): return `0`. The SUCCESS bytes are `0x50`, `0x62`, `0x7e` – `0x81`, `0x83` – `0x86`, `0x89` – `0x8a`, `0x8d` – `0x8e`, `0x95` – `0x99`, and `0xbb` – `0xfe`.
2. If `kind == TAPSCRIPT` and opcode is `OP_CHECKMULTISIG` (`0xae`) or `OP_CHECKMULTISIGVERIFY` (`0xaf`): return `0` (Core fails with `SCRIPT_ERR_TAPSCRIPT_CHECKMULTISIG` before popping).
3. If opcode is `OP_CHECKSIGADD` (`0xba`): return `3` if `kind == TAPSCRIPT`, else `0` (`SCRIPT_ERR_BAD_OPCODE` in v0).
4. If `kind == WITNESS_V0` and opcode is a disabled splice/bitwise/numeric opcode (`OP_CAT`, `OP_SUBSTR`, `OP_LEFT`, `OP_RIGHT`, `OP_INVERT`, `OP_AND`, `OP_OR`, `OP_XOR`, `OP_2MUL`, `OP_2DIV`, `OP_MUL`, `OP_DIV`, `OP_MOD`, `OP_LSHIFT`, `OP_RSHIFT`): return `0` (fail before the `EvalScript` switch).
5. If opcode is a push (`0x00` – `0x4e`, `OP_1NEGATE`, `OP_1` – `OP_16`): return `0`.
6. Otherwise use the named table below. Bytes with no named row, including `OP_RESERVED`, `OP_VER`, `OP_VERIF`, `OP_VERNOTIF`, `OP_NOP1` / `OP_NOP4` – `OP_NOP10`, and `0xff`, return `0`.

Opcode atlas

The live map is the 256-byte `stack_items_read` table: row `R`, column `F` is byte `0xRF`. Copy-out tables are under the atlas.

Dark cell = 0. A push immediately before that opcode is unreferenced. Bright cell = n ≥ 1. The preceding push is consumed. Byte `0xRF` sits at row `R`, column `F`.

push | reads 0 | 1 | 2 | 3 | 4 | 6 | SUCCESS | V0 ≠ Tap

SELECT AN OPCODE

Tap a map cell or a named chip.

Color is the data-independent prefix `EvalScript` always requires from the top of the main stack.

WITNESS_V0 · RULE 12

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6_	0	0	0	1	1	0	0	0	0	1	0	1	0	2	2	3
7_	4	6	4	1	0	1	1	2	2	1	1	3	2	2	0	0
8_	0	0	1	0	0	0	0	2	2	0	0	1	1	0	0	1
9_	1	1	1	2	2	0	0	0	0	0	2	2	2	2	2	2
A_	2	2	2	2	2	3	1	1	1	1	1	0	2	2	1	1
B_	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
C_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

TAPSCRIPT · RULE 10

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6_	0	0	0	1	1	0	0	0	0	1	0	1	0	2	2	3
7_	4	6	4	1	0	1	1	2	2	1	1	3	2	2	0	0
8_	0	0	1	0	0	0	0	2	2	0	0	1	1	0	0	1
9_	1	1	1	2	2	0	0	0	0	0	2	2	2	2	2	2
A_	2	2	2	2	2	3	1	1	1	1	1	0	2	2	0	0
B_	0	1	1	0	0	0	0	0	0	0	3	0	0	0	0	0
C_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
E_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F_	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

WORKED NEXT-OPCODE TESTS

envelope · invalid	OP_FALSE OP_IF 1 <jpeg> OP_ENDIF 0	jpeg is followed by ENDIF
dump · referenced, then capped	<jpeg> OP_DROP 1	Rule 11 / 13 cap PUSH/DROP
multisig · valid	OP_2 <k1> <k2> OP_3 CHECKMULTISIG 1	consecutive pushes are not dead

---

NAMED OPCODES

Spec callouts: every `≥ 1` consumer, plus zeros that are easy to get wrong. Chip count is V0, or `V0/Tap` when they differ.

CONTROL

IF 1 · NOTIF 1 · ELSE 0 · ENDIF 0 · VERIFY 1 · RETURN 0

STACK

TOALTSTACK 1 · FROMALTSTACK 0 · 2DROP 2 · 2DUP 2 · 3DUP 3 · 2OVER 4 · 2ROT 6 · 2SWAP 4 · IFDUP 1 · DEPTH 0 · DROP 1 · DUP 1 · NIP 2 · OVER 2 · PICK 1 · ROLL 1 · ROT 3 · SWAP 2 · TUCK 2

SPLICE

SIZE 1

BITWISE / EQUAL

EQUAL 2 · EQUALVERIFY 2

NUMERIC

1ADD 1 · 1SUB 1 · NEGATE 1 · ABS 1 · NOT 1 · 0NOTEQUAL 1 · ADD 2 · SUB 2 · BOOLAND 2 · BOOLOR 2 · NUMEQUAL 2 · NUMEQUALVERIFY 2 · NUMNOTEQUAL 2 · LESSTHAN 2 · GREATERTHAN 2 · LESSTHANOREQUAL 2 · GREATERTHANOREQUAL 2 · MIN 2 · MAX 2 · WITHIN 3

CRYPTO

RIPEMD160 1 · SHA1 1 · SHA256 1 · HASH160 1 · HASH256 1 · CODESEPARATOR 0 · CHECKSIG 2 · CHECKSIGVERIFY 2 · CHECKMULTISIG 1/0 · CHECKMULTISIGVERIFY 1/0 · CHECKSIGADD 0/3

LOCKTIME

CHECKLOCKTIMEVERIFY 1 · CHECKSEQUENCEVERIFY 1

▶ Normative copy-out tables

Security Considerations

No expiry surface. Because this BIP is permanent, there is no expiry date for the data-embedding ecosystem to plan around. The rule is stable indefinitely.

Grandfathering boundary. Rule 1 uses the creating block's height. Rules 2 through 13 use the prevout's creating height. A mixed transaction is not a blanket exemption. Implementations must not treat “any grandfathered input” as exempting new outputs, and must not treat a grandfathered spend as authorizing inscription witness on later spends of the new outputs. A pre-activation envelope UTXO may still reveal its committed witness once, when that coin is spent. New envelopes cannot be created: a post-activation P2TR funded from old coins is a new prevout, and Rules 2 through 13 reject inscription-style witness when it is spent.

Undefined-version creation. Rule 1 and Rule 3 must agree: undefined witness-version outputs cannot be created after activation and cannot be spent except via grandfathered pre-activation UTXOs. Implementations that allow creation while rejecting spends, or the reverse, will split the chain.

Aggregate limit manipulation. An attacker who knows the `max_witness_bytes_per_tx` limit could construct transactions that stay just below it across many transactions, accumulating a fragmented payload. The per-output fee BIPs price each such transaction through its output count, so aggregate data embedding volume remains proportional to aggregate fee spend. This is the intended outcome. Fragmentation across many transactions in one block remains possible up to the weight limit. A per-block data cap is rejected: it collides with high-throughput monetary blocks (batch payouts, many channel closes, several CoinJoins). That path is priced by the per-output fee, not by a fourth aggregate.

Witness item classification. The exclusion of witness scripts, Tapleaf scripts, control blocks, annexes, and key-path signatures from the aggregate counts must be specified precisely and implemented identically across all compliant implementations. Classification errors that treat script material as data (or vice versa) produce consensus failures. Witness scripts and Tapleaf scripts are instead governed by Rules 10 through 13. The calibration checklist requires a shared witness item classification algorithm before submission.

Private mempool arrangements. Miners accepting non-compliant transactions through private arrangements produce blocks that enforcing nodes reject. Orphan risk limits sustained defection, as with the per-output fee BIPs.

References

- *BIP Pre-Proposal: Static Per-Output Miner Fee* (Josh / Secure Sovereign, July 2026).
- *BIP Pre-Proposal: Dynamic Escalation of the Per-Output Miner Fee* (Josh / Secure Sovereign, July 2026).
- BIP 141: Segregated Witness.
- BIP 341: Taproot.
- BIP 342: Tapscript (`IsOpSuccess` ranges).
- Bitcoin Core `src/script/interpreter.cpp` (`EvalScript`) and `src/script/script.cpp` (`IsOpSuccess`), commit `b2c45888`.
- BIP 433: Pay-to-Anchor (P2A).
- BIP 16: Pay to Script Hash.
- *The Achievable Floor* (secsov.com, 2026).
- *Bitcoin Is Not a Hard Drive* (secsov.com, 2026).
- *Full Cost of Running a Bitcoin Node*, v2.4 (secsov.com, July 2026).
- Various Delving Bitcoin and bitcoin-dev threads on UTXO bloat and data embedding (2023–2026).

Copyright

This document is licensed under the BSD 2-Clause License.

RELATED

Works with

Static Per-Output Miner Fee · Dynamic Escalation of the Per-Output Miner Fee

Related writing

The Achievable Floor · Bitcoin Is Not a Hard Drive · Full Cost of Running a Bitcoin Node

SECURESOVEREIGN

Markdown is the canonical source: https://secsov.com/bips/permanent-data-channel-closure/index.md

Formatted snapshot of https://secsov.com/bips/permanent-data-channel-closure

Josh · Secure Sovereign