BitVM-448: Covenant-Based BitVM3 Bridges

Robin Linus
Glass Coins

Abstract

This note sketches BitVM-448, a BIP–448-based BitVM3 bridge using `OP_TEMPLATEHASH` covenants and `OP_CHECKSIGFROMSTACK` for deleted-key recursion. Native template covenants replace per-deposit pre-signed transactions, removing the per-deposit cosigner set from bridge safety and allowing non-interactive deposits into prepared slots. The operator's per-attempt bond is moved into the asserted UTXO, so an invalid assertion can return the reserve to the bridge while burning the bond, without a sibling-input covenant.

1 Goal

In BitVM3, an operator first completes a peg-out using its own liquidity and then opens a Bitcoin-side assertion that proves the peg-out was valid. Challengers check the assertion off-chain using BitVM3-core. If the assertion is false, the check reveals a false-output label L_i^*, which can be used on-chain to stop the operator's withdrawal [1].

In this note, TH denotes the `OP_TEMPLATEHASH` covenant component, while CSFS denotes `OP_CHECKSIGFROMSTACK`. The construction uses TH for both the deposit and assertion covenants; CSFS is needed only for the deleted-key recursive variant [2, 3, 4]. TH is not a sibling-input covenant: it omits input prevouts, amounts, and scripts, so it cannot require a particular bond input. The transaction graph instead moves the reserve and bond into one asserted UTXO.

2 Remodeled state machine

Let D be the bridge reserve amount, B the operator bond, and Δ the challenge delay. Each operator $\mathcal{O}_i$ has one global slashing secret

$$s_i := L_i^*, \qquad h_i := \mathsf{H}(s_i).$$

The reuse of s_i is intentional: if $\mathcal{O}_i$ lies once, s_i becomes public, and every future attempt by $\mathcal{O}_i$ to touch bridge funds can be cancelled. The operator is permanently poisoned.

A reserve state is denoted $\mathsf{Funds}_S(D)$, where S is the set of operators still syntactically live in that state. The reserve output is a Taproot covenant Taptree with one separate leaf/contract for each operator $i \in S$. The leaf for $\mathcal{O}_i$ lets that operator start a withdrawal assertion, but only via a TH-committed transaction template of the form

$$\mathsf{Funds}_S(D) + I_i(B + f_{\text{kick}}) \longrightarrow A_{S,i}(D + B).$$

Here I_i is any input supplied by the operator. It funds both the bond B and the kickoff/assertion transaction fee f_{kick}. Since the asserted output has value $D + B$ while the reserve contributes only D, the operator must supply the additional value. The covenant does not identify I_i: the bond

is a per-attempt bond enforced by output value, not a separately registered stake UTXO. Once absorbed into $A_{S,i}(D+B)$, however, the bond is controlled by the assertion covenant and becomes slashable on fraud. Figure 1 shows one operator branch.

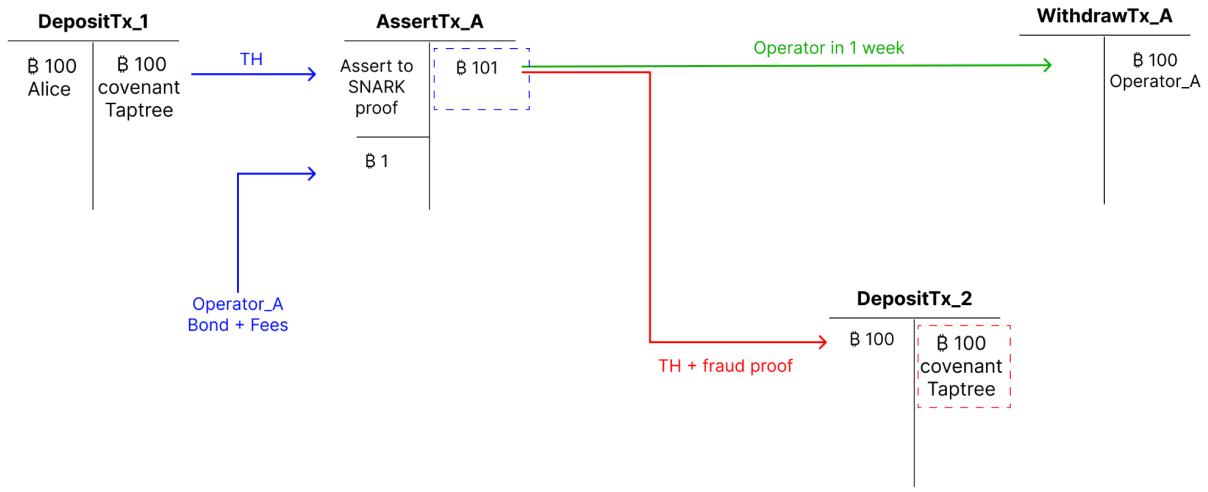

Figure 1: Transaction graph for one operator branch. The right-hand output of DepositTx_1 is a covenant Taptree with one leaf/contract per operator; the depicted leaf is for $\mathcal{O}_A$ and commits via TH to AssertTx_A. The operator adds a bond input, creating an asserted output of value $D+B$. If no fraud proof appears for Δ, the operator withdraws. If the false-output label is revealed, anyone can redeposit D into the next covenant Taptree and the bond is paid as fees.

There are two distinct covenants in this graph. The first is the *deposit covenant*, in $\mathsf{Funds}_S(D)$. It is the deposit Taptree and commits the reserve output to the allowed kickoff/assertion templates for the currently live operators. The second is the *assertion covenant*, in $A_{S,i}(D+B)$. It commits the asserted output to either the timeout withdrawal path or the fraud path that recreates the reserve covenant Taptree. The deposit covenant gives non-interactive deposits into prepared slots; the assertion covenant ensures that a failed assertion can only return the funds to the bridge state rather than spend them elsewhere.

The asserted output $A_{S,i}(D+B)$ has two spending paths:

Cancel/redeposit.

Anyone knowing $s_i = L_i^*$ may spend immediately:

$$A_{S,i}(D+B) \xrightarrow{s_i} \mathsf{Funds}_{S\setminus\{i\}}(D).$$

The missing value B is paid as the cancellation/redeposit transaction fee. Thus a false assertion pays for its own recovery, and in the syntactic enumeration variant the cheating operator is removed from that deposit's next Taptree.

Timeout/withdrawal.

If no valid cancellation appears for Δ blocks, $\mathcal{O}_i$ may withdraw:

$$A_{S,i}(D+B) \xrightarrow{\Delta} \mathsf{Withdraw}_i(D+B),$$

receiving the bridge amount and recovering its bond. The assertion statement must bind to the particular peg-out and reserve slot, so that one valid burn proof cannot be replayed against multiple deposits.

In variants that do not syntactically remove slashed operators, a malicious operator whose secret is already public may still create nuisance assertions if its branch remains in some Taptree. This does not affect safety: each attempt requires a fresh bond and can be cancelled with the already-known secret. In the strict enumeration variant above, the operator is removed from the next state of that deposit, so it cannot reattempt on that deposit.

3 Prepared deposit slots

A subtle but important point is that the covenant graph is per prepared deposit slot. The relevant TH values commit to exact output scripts and amounts. Since a deposit output contains a Taptree of operator-specific contracts and BitVM3 assertion material, changing the amount, slot identifier, operator set, keys, or garbled-circuit/adaptor-signature material changes the next output template.

Therefore BitVM-448 should be understood as follows. During setup, the parties pre-generate a large catalog of deposit slots

$$\mathsf{slot}_1, \ldots, \mathsf{slot}_M.$$

Each slot has a fixed amount or denomination, a unique slot identifier, a root for its deposit covenant graph, and the corresponding public assertion material. A user deposits non-interactively by paying to an unused slot root. The side system accepts deposits only into valid unused slots, for example by tracking a slot sequence or an allocation set. This avoids collisions and prevents users from depositing into a wrong or stale covenant root.

This model gives a finite number of prepared deposits and a finite prepared TVL, but the catalog can be made large. Setup is once for the catalog, not once for arbitrary future templates. If the bridge later needs more capacity, it can run another setup round that publishes an additional catalog.

4 Recursive instantiations

4.1 Enumeration recursion

Enumeration recursion requires no deleted-key setup and does not use CSFS. One pre-constructs a finite graph of TH values for all allowed slashing histories of a deposit slot. A state $\mathsf{Funds}_S(D)$ has one assertion edge for each $i \in S$, and a failed assertion redeposits into $\mathsf{Funds}_{S\setminus\{i\}}(D)$, thereby removing the malicious operator from that slot's next Taptree.

After merging histories that lead to the same surviving operator set, the full strict graph has exactly

$$N_{\text{state}} = 2^n$$

reserve states for n operators. The number of assertion/cancel transitions is

$$N_{\text{edge}} = \sum_{S \subseteq \{1,\ldots,n\}} |S| = n2^{n-1}.$$

This is the fully general syntactic-removal graph for a single prepared slot. A catalog of M independent slots multiplies these setup counts by M.

A bounded-depth version is often more realistic. If the graph syntactically removes cheating operators but only supports at most T failed assertions before falling back to migration or recovery, then the number of reserve states is

$$N_{\text{state}}^{(T)} = \sum_{k=0}^{T} \binom{n}{k},$$

where k is the number of omitted operators. The corresponding number of assertion edges is

$$N_{\text{edge}}^{(T)} = \sum_{k=0}^{T-1} (n-k) \binom{n}{k}.$$

For $T = n$ this recovers the full 2^n graph. For small T, it gives syntactic operator removal without the full exponential blow-up.

4.1.1 Setup and storage cost

The full graph for each prepared slot must be computed at least once during setup, because the initial root commits recursively to all child states. A bottom-up dynamic program over the subset lattice computes the state roots and checks all allowed transitions. This is an exponential setup cost for the full 2^n graph, or a polynomial-in-n setup cost for fixed bounded depth T.

Verifiers compute the catalog once during setup to check that the advertised slot roots are correct. After that, they can store only the public catalog root or list of slot roots, the public parameters, and the slot-allocation rule; the intermediate graph data can be deleted. Depositors only need the selected unused slot root and parameters.

Operators have the meaningful online storage requirement, because they must later produce the Taproot branch and template data needed to start their own withdrawals. In the fully materialized worst case for one slot, operator $\mathcal{O}_i$ stores paths for all states in which it has not been slashed, i.e. all subsets S with $i \in S$, giving 2^{n-1} own-operator paths. This worst case is usually unnecessary. An implementation can use an optimistic cache: store only the initial path, or store paths up to a small slash depth h, and recompute deeper states on demand.

If h is the number of other-operator slashes precomputed, the number of paths cached by one operator per slot is

$$N_{\text{path}}^{(h)} = \sum_{k=0}^{h} \binom{n-1}{k}.$$

Thus $h = 0$ means storing only the initial Taptree path; $h = 1$ stores the initial path and all one-slash continuations; $h = 2$ stores all states after at most two other operators have been removed. If more slashes occur, the relevant Taptree and path are recomputed from the deterministic public construction during the challenge/withdrawal window.

n	full 2^n states	full edges $n2^{n-1}$	$T = 2$ states	$T = 3$ states	$h = 2$ own paths
20	1.05M	10.5M	211	1,351	191
24	16.8M	201M	301	2,325	277
30	1.07B	16.1B	466	4,526	436
100	$\approx 1.27 \cdot 10^{30}$	$\approx 6.34 \cdot 10^{31}$	5,051	166,751	4,951

The practical conclusion is that full syntactic enumeration is comfortable for a single or small number of slots up to roughly $n \leq 24$, engineering-heavy around $n = 26$–28, and not attractive

around $n \approx 30$ or for large catalogs. For many prepared deposit slots or large operator sets, one should either bound the number of syntactic slashes T, shard the operator set, or use deleted-key recursion. Long-term verifier storage is not the bottleneck; the bottleneck is the one-time setup computation and the engineering/audit burden of generating a large catalog correctly.

4.1.2 Semantic-slashing unrolling

The subset graph is needed only if the covenant state syntactically removes slashed operators. If one accepts semantic slashing–a slashed operator remains in the Taptree but its secret s_i is public–then the trustless enumeration can be unrolled by attempt depth instead of by surviving operator subsets:

$$\mathsf{Funds}_t(D) \xrightarrow{i} A_{t,i}(D+B) \xrightarrow{s_i} \mathsf{Funds}_{t+1}(D).$$

For n operators and a maximum of T failed attempts, this has $T+1$ reserve states and nT assertion branches. It still requires no deleted-key setup, but it is finite and does not syntactically kick out cheaters. The strict syntactic-removal graph above is preferable when the next Taptree should omit cheating operators.

4.2 Deleted-key recursion

The compact recursive instantiation uses both TH and CSFS. This is the BitVM-448 analogue of the CTV+CSFS recursive covenant pattern first pointed out by Towns [7]: a throwaway Schnorr key signs the template hash of an output that recreates the same script, and the key is then deleted. TH plays the template-hash role that BIP–119 CTV plays in that discussion [6, 3]; CSFS is the opcode that checks the signature over the template hash inside the script [4].

Concretely, a setup key signs the template hash of a self-replicating covenant, and the script uses CSFS to verify that signature against the TH value. After setup, the key is deleted. The setup can be implemented as a large one-time n-of-n ceremony, for example via a MuSig-style aggregate signature. This avoids preconstructing the enumeration graph, at the price of a deleted-key trusted setup.

The trusted setup is not universal for arbitrary future deposits. It is per prepared output template: the signature commits to the next output script and amount through the template hash. In the bridge setting, this means per prepared deposit slot, or per fixed template class if the exact same output template can be reused safely. Thus deleted-key recursion is linear and compact for a large prepared catalog, but the catalog must still be generated and published during setup.

In this compact version, slashed operators need not be removed from the Taproot tree. Their global secret is public, so every future attempt can be cancelled immediately and costs them another bond. The tradeoff is semantic poisoning rather than syntactic operator removal.

5 Practical notes

Non-interactive deposits and signer involvement. After the slot catalog is prepared, deposits require no signer interaction. Depositors only need an unused public slot root and parameters, and can pay directly into the reserve Taptree. Setup participants fix the operator set, check the advertised roots or recursive setup, and, in the deleted-key variant, delete the setup key. They are not online cosigners for each deposit, which removes the per-deposit liveness requirement and reduces their ongoing operational surface.

Single-use assertion artifacts. The covenant recursion and the BitVM3-core assertion artifacts should be kept conceptually separate. The garbled-circuit/adaptor-signature material may be single-use, per deposit slot, or per withdrawal claim. A valid assertion reveals no slashing secret and lets the operator complete the withdrawal after the timeout. An invalid assertion reveals the operator's global secret s_i, after which all future assertions by that operator are immediately cancellable.

On-demand bonds and no kickoff UTXOs. Because TH does not commit to input prevouts, input amounts, or input scripts, the bond is not a separately identified stake UTXO. It is provided on demand by whatever extra input funds the asserted output. This also removes the need for dedicated kickoff or connector UTXOs at deposit time, including awkward zero-satoshi outputs that would be non-standard. The bond becomes slashable only after it is absorbed into the asserted UTXO controlled by the assertion covenant.

Miner-colluding operators. If the entire bond is paid as transaction fees in the fraud path, a miner may rebate those fees to a colluding slashed operator. This does not affect safety: the public fraud witness still prevents withdrawal, and the only effect is temporary censorship or state churn. Under Bitcoin liveness, any non-censoring miner can include the redeposit before the withdrawal timeout expires.

Remaining bottlenecks. BitVM-448 addresses the covenant and deposit-interactivity bottlenecks. It does not by itself remove the off-chain cost of BitVM3-core assertion material, such as adaptor signatures or garbled-encoding extraction, nor does it simplify the underlying light client needed for rollup-style withdrawals. CSFS is essential for the deleted-key recursive covenant, but there is no obvious direct way in this sketch for CSFS to reduce the adaptor-signature cost.

6 Assumptions and summary

The central invariant is simple. Before final withdrawal, every reachable state contains either a reserve UTXO of value D under the bridge covenant, or an asserted UTXO of value $D + B$ whose false path returns D to the bridge covenant. If the assertion is false, BitVM3-core reveals $s_i = L_i^*$ to any challenger who evaluates it off-chain, and s_i enables redeposit before the timeout. If no cancellation appears before Δ, the operator withdraws.

The BitVM-448 construction removes the per-deposit cosigner/covenant-emulation committee from bridge safety and removes the n-of-n liveness requirement for prepared deposits: a user can deposit non-interactively by paying to an unused slot in the public bridge catalog. It still relies on the usual cryptographic assumptions of BitVM3-core, Bitcoin ledger safety and liveness, availability of public assertion artifacts, and a sufficiently live challenger ecosystem during the challenge window. The deleted-key variant additionally assumes the setup key for each prepared template is genuinely destroyed by at least one honest participant. The bond B and/or fee ladder should be chosen so that cancellation transactions confirm within Δ even under adverse fee conditions.

The key modeling move is to replace the old "reserve plus connector" withdrawal with a single asserted UTXO that already contains the operator bond. The TH component then only has to constrain the current UTXO's next transaction, which is exactly what it is built to do. Strict enumeration gives a trustless per-slot graph with 2^n subset states and syntactic operator removal. Bounded enumeration gives syntactic removal for up to T slashes with $\sum_{k=0}^{T} \binom{n}{k}$ reserve states. Semantic-slashing enumeration gives $O(nT)$ attempt-depth unrolling without syntactic removal.

Deleted-key recursion gives the compact infinite-depth version with a per-template deleted-key setup assumption.

References

[1] Robin Linus Woll, Ioannis Alexopoulos, Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, and David Tse. *BitVM3: Efficient Bitcoin Bridges via Garbled Circuits*. IACR ePrint Report 2026/933, 2026. https://eprint.iacr.org/2026/933.pdf.

[2] Gregory Sanders, Antoine Poinsot, and Steven Roose. *BIP–448: Taproot-native (Re)bindable Transactions*. Draft Bitcoin Improvement Proposal. https://github.com/bitcoin/bips/blob/master/bip-0448.md.

[3] Gregory Sanders, Antoine Poinsot, and Steven Roose. *BIP–446: OP_TEMPLATEHASH*. Draft Bitcoin Improvement Proposal. https://github.com/bitcoin/bips/blob/master/bip-0446.md.

[4] Brandon Black and Jeremy Rubin. *BIP–348: CHECKSIGFROMSTACK*. Draft Bitcoin Improvement Proposal. https://github.com/bitcoin/bips/blob/master/bip-0348.md.

[5] Brandon Black and Jeremy Rubin. *BIP–349: OP_INTERNALKEY*. Draft Bitcoin Improvement Proposal. https://github.com/bitcoin/bips/blob/master/bip-0349.md.

[6] Jeremy Rubin. *BIP–119: OP_CHECKTEMPLATEVERIFY*. Draft Bitcoin Improvement Proposal. https://github.com/bitcoin/bips/blob/master/bip-0119.mediawiki.

[7] Anthony Towns. *"Recursive covenant" with CTV and CSFS*. Bitcoin Development Mailing List, March 4, 2025. https://groups.google.com/g/bitcoindev/c/Tu7mr419jWQ/m/4bJ3UpaSAQAJ.