BitVM 3s – Garbled Circuits for Efficient Computation on Bitcoin

Robin Linus
robin@zerosync.org

July 22, 2025

Abstract

BitVM 3s enables efficient SNARK proof verification on Bitcoin by leveraging garbled circuits to shift computation off-chain. BitVM2 requires large on-chain transactions of 4 MB. In contrast, BitVM 3s reduces the assertion transaction to ≈ 66 kB and the disproval transaction to ≈ 20 kB, cutting dispute costs by about 1,000x in comparison to its predecessor, BitVM2. The protocol builds upon the Delbrag scheme but introduces *BitHash*, a novel hash function optimized for Bitcoin Script's constraints, reducing the collateral requirements from over $10,000 to under $100, making it economically feasible for Bitcoin bridges.

1 Introduction

BitVM2 [1] demonstrated SNARK proof verification on Bitcoin through optimistic computation, but suffered from large on-chain transactions that made bridge operation economically challenging. The core challenge was performing complex computations directly in Bitcoin Script, leading to multi-megabyte transactions.

BitVM 3s[1] solves this by moving computation off-chain using garbled circuits. During bridge setup, the garbler commits to a circuit in a Taproot tree and shares it with the evaluator. If the garbler later asserts an incorrect SNARK proof, the evaluator can use the shared circuit to generate a fraud proof that falsifies the assertion. This approach maintains the same trust model and transaction graph as BitVM2 bridges while dramatically reducing on-chain costs.

2 Related Work

Early explorations of garbled-circuit techniques on Bitcoin, such as through expressing Boolean functions in Discrete Log Contracts, date back to at least 2022 [2]. Although several teams investigated similar ideas privately, the first public proposal to use garbled circuits for BitVM-style constructions was Jeremy Rubin's Delbrag scheme, announced in April 2025 [3]. Delbrag's introduction of garbled circuits to the BitVM community inspired multiple subsequent designs, though significant challenges remained for practical implementation:

- **Delbrag**: Conceptually elegant but its reliance on the Blake3 implementation [5] required 300 kB-sized disprove Scripts and thus, over $10,000 in collateral, negating the efficiency benefits that garbled circuits were meant to provide.
- **BitVM3-RSA** [4]: Used RSA-based garbled circuits but was cryptographically broken

[1] BitVM 3s is the secure, simple, and Script-based successor to the earlier RSA-based prototype, now referred to as BitVM3-RSA.

- Citrea employed classical Yao-style circuits [8] with cut-and-choose, requiring 400 GB storage per circuit, which is practical, but not optimal.
- Alpenlabs announced the "Glock" scheme [9], aiming to reduce circuit size to 12 million gates. However, it relies on permissioned validators, exotic cryptography, and lacks compatibility with Groth16 – the industry standard SNARK scheme with extensive tooling and widespread adoption.

BitVM 3s builds on Delbrag's foundation but introduces an efficient hash function, *BitHash*, solving its fundamental limitations while maintaining simple, battle-tested cryptography, and permissionless challenging.

3 Protocol Overview

3.1 High-Level Design

The protocol operates in three phases:

1. **Setup**: Garbler creates a garbled circuit for SNARK verification, commits to it in a Taproot tree, and shares it with the evaluator. The evaluator verifies that the Taproot tree is valid.
2. **Assertion**: Garbler publishes a SNARK proof, claiming that it is valid.
3. **Challenge**: If the SNARK proof or the circuit is incorrect, the evaluator uses the Taproot tree to generate a fraud proof that falsifies the incorrect assertion on-chain. Otherwise, a timeout occurs and the proof is considered valid.

3.2 Garbled Circuit Construction

A Groth16 verifier circuit consists of approximately 10 billion gates (7.7 billion free gates and 2.7 billion non-free gates) [7]. The garbler generates:

- **Wire labels**: Random values for each wire's 0 and 1 states
- **Encryption table**: For each non-free gate, four ciphertexts encrypting output labels
- **Taproot commitment**: A large Taptree with Scripts for verifying each garbling table entry

3.3 Fraud Proof Mechanism

Each gate's garbling table has four rows. For each row, a Taproot Script asserts to the *ciphertext* (the output label encrypted with the key) and:

$$H_{inA} = \text{Hash}(\text{input\_label\_A})$$
$$H_{inB} = \text{Hash}(\text{input\_label\_B})$$
$$H_{out} = \text{Hash}(\text{output\_label})$$

To falsify an incorrectly garbled gate, the evaluator reveals `input_label_A` and `input_label_B`, allowing the Script to:

1. Verify the preimages match the committed hashes
2. Compute the decryption key: $k = \text{Hash}(\text{input\_label\_A} \oplus \text{input\_label\_B})$
3. Decrypt the output label: `output_label` $= \text{ciphertext} \oplus k$
4. Verify that Hash(output_label) $\neq H_{out}$, proving the garbler's assertion was incorrect

3.4 Disprove Script

The disprove Script for a row of a gate in pseudo-code is the following:

```
//
// Unlocking Script
//
<input_labelA>
<input_labelB>


//
// Locking Script
//

// verify the input labels
OP_2DUP
OP_HASH
OP_EQUALVERIFY
<hash_inputA>

OP_HASH
OP_EQUALVERIFY
<hash_inputB>

// compute the key
OP_XOR
OP_HASH

// decrypt the output label
<ciphertext>
OP_XOR

// falsify the output label
OP_HASH
<hash_output>
OP_EQUAL
OP_NOT
OP_VERIFY
```

4 BitHash: Efficient Hash Function for Bitcoin Script

The critical challenge is implementing efficient hash functions in Bitcoin Script without `OP_CAT` and with only 32-bit arithmetic, while maintaining second-preimage resistance against malicious evaluators. Our approach builds upon earlier work on hash commitments in Bitcoin Script [6] and introduces *BitHash*, a novel hash function optimized for the intricacies of Bitcoin Script.

4.1 Construction

BitHash processes a 160-bit preimage bit-by-bit, using each bit to select between RIPEMD160 and SHA256:

```
//
// Unlocking Script: 160 bits of preimage
//
<bit_159> <bit_158> ... <bit_1> <bit_0>

//
// Locking Script
//
<static_initial_value>  // e.g., 0x00000000
repeat 160 times:
    OP_SWAP
    OP_IF
        OP_RIPEMD160
    OP_ELSE
        OP_SHA256
    OP_ENDIF

OP_RIPEMD160
<expected_hash>
OP_EQUALVERIFY
```

4.2 Properties

BitHash160 provides:

- **80-bit collision resistance**: Sufficient for preventing malicious collision attacks
- **160-bit second-preimage resistance**: Protects against preimage attacks
- **768-byte Script size**: reduced from 75 kB for Blake3

4.3 Stack Management Optimization

To reuse preimages after verification, we employ the altstack:

```
repeat 160 times:
    OP_SWAP
    OP_DUP
    OP_TOALTSTACK  // Save for later use
```

```
OP_IF
    OP_RIPEMD160
OP_ELSE
    OP_SHA256
OP_ENDIF
```

This requires 8 opcodes per iteration, increasing the Script size to 1,024 bytes per invocation to enable preimage reuse.

4.4 BitHash Improvements

Using the first 32 bits of the preimage as the initial hash value instead of a constant reduces the Script size by about 256 bytes per invocation. This reduces the size of the disprove transaction by about 1 kB.
Since we do not rely on collision resistance, but only on second-preimage resistance, we may use 80 to 128-bit preimages, which reduces the size of the Script by almost a factor of 2, down to about 500 bytes per invocation.

4.5 Key Derivation

The fraud proof mechanism described above has a critical flaw: Bitcoin Script's `OP_XOR` only operates on 32-bit values, but hash functions output 160-256 bit values. This makes the decryption step impossible as written:

```
OP_XOR          // Works: XOR two 32-bit input labels
OP_HASH         // Produces 160-bit hash = k
<ciphertext>
OP_XOR          // BROKEN: Cannot XOR 160-bit k with ciphertext
```

To solve this, we introduce a key derivation mechanism that is efficient in Script.

4.5.1 Naive Solution

The naive solution is to simply use the Blake3 implementation to compute the key. This is not optimal, as it requires 18kvB of Script, which dominates the size of the disprove transaction, but is still practical.

4.5.2 Optimised Solution

An optimised solution is to use an AES-128 implementation to derive the key, as it heavily relies on lookup tables and operates on byte-sized words, which is efficient in Script. This reduces the size of the disprove transaction by about 4x, down to about 20kB.
PRINCEv2 [11] or SPECK-128 are 128-bit block ciphers that are very efficient in Script, but they are not as well-studied as AES-128. An optimised implementation [10] is estimated to require about 10kB of Script.

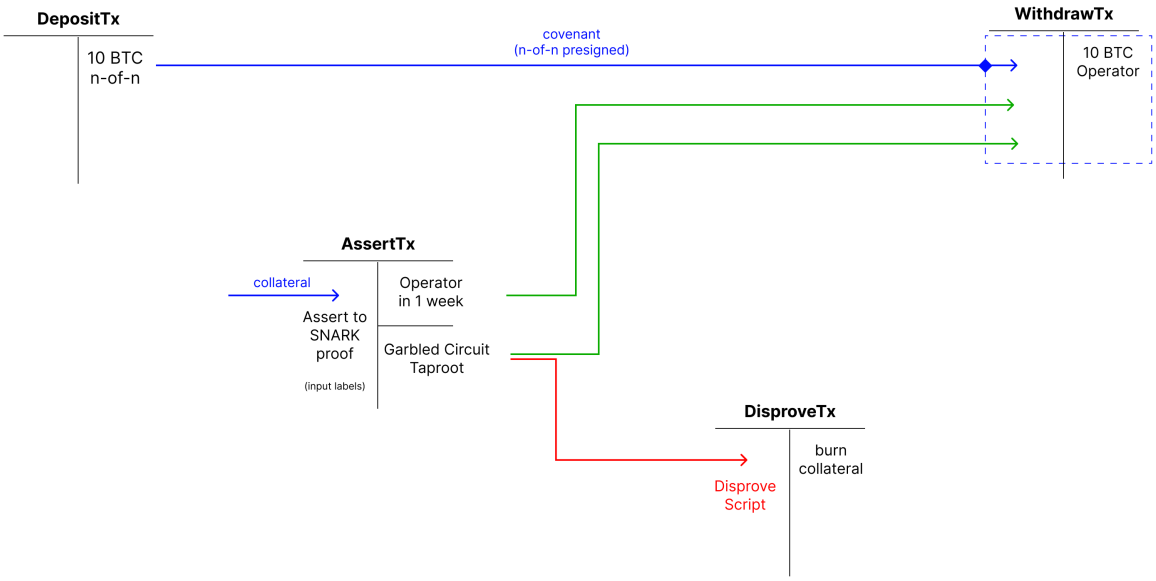

Figure 1: High-level transaction graph of a BitVM3 bridge, using a garbled circuit as drop-in replacement for the BitVM2 verifier. If the circuit or the SNARK proof is incorrect, the evaluator can cancel the withdrawal by publishing a succinct fraud proof.

5 Basic Bridge Design

5.1 Assertion Transaction (≈ 66 kB)

For efficiency, the assertion transaction may use standard hash functions to Lamport sign the input labels, rather than BitHash:

- SNARK proof ($\approx$128 bytes)
- Public input ($\approx$20 bytes)

The overhead of the Lamport commitments for $128 + 20 = 148$ input bytes is ≈ 56 bytes per input bit, totaling $148 \cdot 8 \cdot 56 \approx 66$ kB. The garbler provides a zero-knowledge proof during setup, proving that the Lamport commitments are mapping correctly to the BitHashes in their corresponding disprove Scripts.

Winternitz signatures in combination with ZKPs can reduce the size of the assertion transaction down to $\approx$33 kB.

Since this is all witness data, the 4x witness discount applies, reducing the size of the assertion transaction to about $17 - 33$ kvB.

5.2 Disproval Transaction (≈ 20 kB)

Either falsifying the circuit by revealing:

- Input wire labels for the disputed gate (≈ 32 bytes)

- Merkle proof for the Tapleaf in the Taproot, identifying the gate ($\approx 1kB$)
- Script demonstrating the garbling error ($\approx 20kB$)

Or alternatively, falsifying the SNARK proof by revealing the circuit output commitment for the "false" output label, representing that the verification failed.

6 Complexity Analysis

6.1 Communication Complexity

For the complete Groth16 verifier circuit that must be communicated during setup:

$$\begin{aligned}\text{circuit size} &= 2.7 \text{ billion non-free gates} \times (4 \text{ rows} \times 16 + 2 \cdot 20) \text{ bytes} \\ &= 280 \text{ GB}\end{aligned}$$

This represents the data the garbler must send to the evaluator during the setup phase and the evaluator must store for the challenge phase.

6.2 Computational Complexity

Taproot commitment generation requires hashing ≈40 billion Scripts. It can be optimized by: Moving parameters to Script endings for hash reuse, which reduces the per-script hashing from 10 kB to 100 bytes. Furthermore, we can leverage parallel computation across multiple cores. However, the bottleneck is rather the garbler's upload speed than the evaluators hashing speed, which is more than 1 GB/s on a single core of a consumer laptop.

6.3 Economic Analysis

At 2025 storage costs (≈$30/TB), the 280 GB requirement costs under $10 per challenger. Combined with sub-$100 on-chain collateral, this enables economically viable bridge operations. A BitVM-like bridge with 100 operators would require about 28 TB of storage, which is economically feasible (less than $400 in storage cost as of 2025).

7 Conclusion

BitVM 3s demonstrates that efficient SNARK verification on Bitcoin is achievable through garbled circuits. By introducing BitHash and moving computation off-chain, the protocol reduces on-chain costs by about 1,000× compared to BitVM2 while maintaining the same security guarantees.
The 280 GB off-chain storage requirement per circuit, while substantial, is economically feasible and enables capital-efficient trust-minimized bridges for second layers on Bitcoin. Future work should focus on circuit size reduction and exploring alternative garbling schemes to further improve efficiency.
This represents a significant step toward scaling Bitcoin as a settlement layer for arbitrarily complex computations, fundamentally reducing the cost of bridging Bitcoins to sidechains and rollups.

References

[1] Robin Linus et al. *BitVM2: Bridging Bitcoin to Second Layers*. https://bitvm.org/bitvm_bridge.pdf

[2] Robin Linus. *Boolean Functions in Discrete Log Contracts*. 2022. https://gist.github.com/RobinLinus/4035ced3fa04cc3745a30dcda09e2367#boolean-functions

[3] Rubin, Jeremy. *Delbrag*. April 2025. https://rubin.io/public/pdfs/delbrag.pdf

[4] Robin Linus. *BitVM3-RSA: Efficient Computation on Bitcoin Bridges*. 2024. https://bitvm.org/bitvm3-rsa.pdf

[5] BitVM Alliance. *Blake3 Bitcoin Script Implementation*. https://github.com/BitVM/BitVM/pull/67

[6] *Robin Linus: Preimage Sequence*. https://github.com/coins/bitcoin-Scripts/blob/master/preimage-sequence.md

[7] BitVM Alliance. *Garbled SNARK Verifier Implementation*. https://github.com/BitVM/garbled-snark-verifier

[8] Yao, Andrew Chi-Chih. *How to Generate and Exchange Secrets*. In 27th Annual Symposium on Foundations of Computer Science, pp. 162-167. IEEE, 1986.

[9] Eagen, Liam. *Exotic Cryptography for Every Day Problems – Liam Eagen*. May 2025. https://www.youtube.com/watch?v=X6mtGtumdGY

[10] BitVM Alliance. *PRINCEv2 Bitcoin Script Implementation*. https://github.com/BitVM/bitvm-js/blob/main/scripts/opcodes/PRINCEv2/prince_v2.js

[11] Dušan Božilov et al. *PRINCEv2 - More Security for (Almost) No Overhead*. Cryptology ePrint Archive, Paper 2020/1269, 2020. https://eprint.iacr.org/2020/1269