

# BitVM3: Efficient Computation on Bitcoin

Robin Linus

## Abstract

BitVM3 is a protocol for verifying SNARK proofs on Bitcoin that dramatically reduces the on-chain footprint of its predecessor, BitVM2. By leveraging optimistic computation with a garbled circuit, BitVM3 shifts the burden of verification off-chain. This design enables an evaluator to generate a compact fraud proof in the event of a dispute. The resulting on-chain transactions are highly efficient: the assertion transaction is approximately 56 kB, while the disproval transaction is just 200 bytes, reducing the on-chain cost of a dispute by over 1,000 times compared to the previous design.

## 1 Introduction

BitVM3 significantly enhances the on-chain efficiency of SNARK proof verification on Bitcoin. It addresses the primary drawback of BitVM2, where the ‘assertTx’ and ‘disproveTx’ were large (2-4 MB). In contrast, BitVM3 reduces the ‘assertTx’ to about 56 kB and the ‘disproveTx’ to a mere 200 bytes.

The core principle remains optimistic computation and the overall transaction graph remains unchanged. However, instead of using Bitcoin Script for on-chain computation, BitVM3 employs a garbled circuit to shift the computation off-chain. This circuit is designed to conditionally reveal a secret, which acts as a fraud proof, only if the garbler provides an invalid SNARK proof. This approach builds upon ideas from Jeremy Rubin and Liam Eagen.

## 2 Computing Gate Labels

The garbling scheme is founded on an RSA-based system.

- **Public Parameters:** The garbler selects and publishes an RSA modulus  $N = P \cdot Q = (2p+1)(2q+1)$  and five public exponents:  $e, e_1, e_2, e_3, e_4$ . These exponents must be invertible modulo  $\frac{\phi(N)}{4} = p \cdot q$ , and for performance, small primes (e.g., 3, 5, 7, 11, 13) are suitable. Let their inverses be  $d, d_1, d_2, d_3, d_4$ . Additionally, the derived exponent  $h \equiv (e_1 e_4 d_2 - e_3) \pmod{pq}$  must also be invertible.
- **Label Generation:** Using the secret trapdoor  $\phi(N)$ , the garbler computes the secret input wire labels  $a_0, a_1, b_0, b_1 \in C_{pq} \subset (\mathbb{Z}/N\mathbb{Z})^*$  by solving the following system for output labels  $c_0, c_1 \in C_{pq} \subset (\mathbb{Z}/N\mathbb{Z})^*$ :

$$a_0^e \cdot b_0^{e_1} \equiv c_0 \pmod{N}$$

$$a_0^e \cdot b_1^{e_2} \equiv c_0 \pmod{N}$$

$$a_1^e \cdot b_0^{e_3} \equiv c_0 \pmod{N}$$

$$a_1^e \cdot b_1^{e_4} \equiv c_1 \pmod{N}$$

The knowledge of  $pq$  allows the garbler to efficiently find a unique solution. The explicit solutions for the secret input labels are:

$$\begin{aligned} b_0 &\equiv (c_1 c_0^{-1})^{h^{-1}} \pmod{N} \\ b_1 &\equiv b_0^{e_1 d_2} \pmod{N} \\ a_0 &\equiv c_0^d \cdot b_0^{-e_1 d} \pmod{N} \\ a_1 &\equiv c_0^d \cdot b_0^{-e_3 d} \pmod{N} \end{aligned}$$

## 2.1 Setup for Tree Circuits (Backward Pass)

For a circuit with a tree structure (fan-out of 1), the garbler generates labels by working backward from the final output gate.

1. For the final gate  $G'$ , choose output labels  $c'_0, c'_1$  and solve for its input labels  $(a'_0, a'_1, b'_0, b'_1)$ .
2. For a preceding gate  $G$  whose output feeds into the first input wire of  $G'$ , its output labels are determined by  $G'$ 's requirements:  $c_0 = a'_0$  and  $c_1 = a'_1$ .
3. Solve for gate  $G$ 's input labels  $(a_0, a_1, b_0, b_1)$  using these newly defined  $c_0, c_1$ .
4. This process is repeated backward through the circuit. Since each gate feeds into exactly one subsequent gate, the output labels for every gate are uniquely determined.

## 2.2 Limitation of the Base Scheme: Fan-out $> 1$

The backward-pass setup fails for general circuits where a wire's fan-out is greater than one. Consider a gate  $G$ 's output wire feeding into both gate  $G'$  (requiring input label  $a'_k$ ) and gate  $G''$  (requiring input label  $b''_k$ ). The labels  $a'_k$  and  $b''_k$  are determined independently by the structures of  $G'$  and  $G''$  respectively, meaning in general  $a'_k \neq b''_k$ . Gate  $G$ , however, can only produce a single output label  $c_k$ . This creates an impossible constraint where  $c_k$  must equal both  $a'_k$  and  $b''_k$ .

## 3 Static Fan-out Handling with Adaptor Elements

To handle fan-out in general circuits, we introduce static multiplicative "Adaptor Elements." If an output wire  $W_y$  (with labels  $\ell_{y,0}, \ell_{y,1}$ ) feeds an input wire  $W_{xi}$  that requires different labels, the garbler pre-computes and publishes a static factor  $T_{i,k}$ :

$$\ell_{xi,k} \equiv \ell_{y,k} \cdot T_{i,k} \pmod{N}$$

The garbler, knowing all base labels during setup, computes this factor as  $T_{i,k} \equiv \ell_{xi,k} \cdot (\ell_{y,k})^{-1} \pmod{N}$ . These adaptors become part of the public circuit parameters.

## 4 Reblinding

To reblind the circuit, one can raise the input labels to a secret exponent. The adaptor elements must also be reblinded. For  $k$  rounds of reblinding, the garbler publishes pairwise coprime public exponents  $u_1, \dots, u_k$  and a secret-derived value  $s = \prod_i u_i^{-1} \pmod{\phi(N)}$ . The garbler then publishes the reblinded adaptor points  $T_{i,k}^s$ .

This allows the evaluator to non-interactively compute any singly rebled adaptor point:

$$T_{i,k}^{\frac{1}{u_i}} = (T_{i,k}^s)^{\prod_{j \neq i} u_j}$$

The evaluator can also recover the plaintext  $T_{i,k}$  by raising  $T_{i,k}^s$  to the power of  $\prod_i u_i$ .

## 5 Verifiability and Circuit Correctness

The evaluator can verify the correctness of the garbled circuit’s structure by checking each gate in plaintext. Consequently, the garbler only needs to prove in zero-knowledge that the circuit’s inputs and outputs (which are committed to) were rebled correctly. Thus, the proving complexity amounts to proving in zero-knowledge about 2400 exponentiations with small exponents.

## 6 Communication Complexity and Onchain Footprint

The primary communication cost is the off-chain transfer of the garbled circuit. A SNARK verifier circuit (e.g., Groth16) may have  $\sim 5$  billion gates. With an average fan-out of 2-4 and a 256-byte RSA modulus, the adaptor elements dominate the circuit size. For each fan-out connection, two adaptors are needed (for logic 0 and 1).

- **Off-chain size:**

$$5 \cdot 10^9 \text{ gates} \cdot 2 \frac{\text{fan-out}}{\text{gate}} \cdot 2 \frac{\text{elements}}{\text{fan-out}} \cdot 256 \frac{\text{bytes}}{\text{element}} \approx 5 \text{ TB}$$

Although sharing the circuit takes about 1.8 days with a 250 Mbps upload speed, this is a one-time setup cost.

- **On-chain ‘assertTx’ size:** For a proof of 128 bytes and a 20-byte public input, the garbler must commit to the input labels. This is optimized by publishing encrypted labels during setup and revealing 16-byte decryption keys on-chain.

$$128 \text{ bytes} \cdot 8 \frac{\text{wires}}{\text{byte}} \cdot \left( 2 \frac{\text{labels}}{\text{wire}} \cdot 16 \frac{\text{bytes}}{\text{label}} + 1 \frac{\text{dec.key}}{\text{wire}} \cdot 16 \frac{\text{bytes}}{\text{dec.key}} \right) \approx 56 \text{ kB}$$

- **On-chain ‘disproveTx’ size:** This transaction is minimal. It simply reveals the hash of the output label for ‘0’, signifying that the SNARK proof was invalid.

## 7 Conclusion

BitVM3 significantly advances Bitcoin’s contracting capabilities by using an RSA-based garbled circuit to move SNARK verification off-chain. This approach reduces the on-chain footprint to a 56 kB assertTx and a 200-byte disproveTx, but requires a multi-terabyte off-chain data setup. While this trade-off enables much more capital efficient trust-minimized bridges for second layers like rollups and sidechains, future work must focus on reducing the off-chain data burden. Ultimately, BitVM3 demonstrates a viable path toward using Bitcoin as a secure settlement layer for arbitrarily complex computations.