Instantiating BitVM3 from Label Forward Propagation

Alva Fu[1], Stephen Duan[1,2], Ethan Zhu[1]

1. GOAT Network
2. ZKM

Abstract. BitVM3 optimizes off-chain verification of SNARK proofs on Bitcoin by replacing Delbrag's symmetric encryption in garbled circuits (GC) with RSA-based homomorphic encryption. This eliminates the need for costly zero-knowledge proofs to verify entire GC correctness and removes Delbrag's single-use constraint. In this work, we further enhance BitVM3 by introducing a reusable circuit garbling scheme, which addresses the bottleneck of excessive off-chain data—slashing requirements by 3–4 orders of magnitude (from terabytes to megabytes)—making the solution economically viable. Additionally, we implement several other optimizations: a forward-pass GC generation flow to halve per-gate data, enabling unlimited reuse of GC off-chain data, and formalizing the protocol. These collectively resolve prior scalability issues while maintaining compact on-chain transactions.

Keywords: BitVM3, garbled circuits, verification on Bitcoin, layer 2

1 Background

BitVM3[1] extends BitVM2[2]'s optimistic computation model—which assumes operators are honest unless challengers submit fraud proofs on Bitcoin—by leveraging Garbled Circuits (GC)[3] for encrypted-state computations.

Delbrag pioneered the use of Yao's GC for SNARK(Groth16) verifier while enabling efficient on-chain verification of circuit correctness. The Garbler, acting as an operator in BitVM, constructs a GC and performs the following Protocol:

1. The Garbler commits all input/output labels on Bitcoin and publicly discloses GC ciphertexts with a ZKP[4] proving consistency between committed labels and published ciphertexts.
2. To verify a valid proof on-chain: **a)** The Garbler reveals the proof in plaintext on-chain. **b)** If challenged (validity disputed), the Garbler then reveals the corresponding input labels. **c)** Compute the GC circuit with revealed labels to derive the output label. **d)** If the output label corresponds to 0 value, the challanger successfully disproves the operator's reimbursement.

Delbrag significantly reduces on-chain overhead: assertion transactions are $\tilde{1}00$kB, while disproval transactions are merely 32 bytes—a 1,000× cost reduction compared to BitVM2. However, Delbrag's classical symmetric encryption for GC construction incurs substantial off-chain burdens:

- The ZK verifier Boolean circuit contains $\tilde{5}$ billion gates, requiring hundreds of GB of published GC ciphertext.
- Proving GC correctness off-chain via ZKP is economically infeasible.
- Single-Use Constraint: Only one valid input label set (yielding output = 1) can be revealed. Revealing labels for invalid proofs (output label corresponding to 0) would expose cheating, limiting practical utility.

BitVM3 improves Delbrag[5] by replacing symmetric encryption with RSA. Leveraging RSA's multiplicative homomorphism, it constructs homomorphic GC circuits where intermediate/output labels retain exponentiation properties when input labels are raised to a power. This enables a“reblinding” technique: one set of labels validates circuit correctness, while a ZKP proves committed on-chain input/output labels maintain homomorphic relationships—indirectly ensuring valid GC construction without ZKP verification. Key advantages include:

- Eliminating the cumbersome ZKP for GC correctness.
- Enabling multiple GC reuses (though usage counts must be predetermined, limiting flexibility).

Notably, RSA’s larger bit-width increases off-chain data to terabytes (TB), leaving scalability unresolved. This paper proposes *sub-circuit reuse* to dramatically reduce off-chain data. Since Groth16 verification predominantly repeats modular multiplication ($\tilde{3}$0,000 times), reusing a single sub-circuit’s GC data while publishing only input labels for subsequent invocations reduces off-chain data by 3–4 orders of magnitude. Our scheme renders BitVM3 fully practical, with additional optimizations including:

- **GC generation reversal**: Switching from output$\rightarrow$input to input$\rightarrow$output label generation, halving per-gate data ($4 \rightarrow 2$ values).
- **Unconstrained reuse**: Eliminating preset usage limits for flexible GC recycling.
- **Complete protocol specification**: Providing a comprehensive framework where BitVM3 currently lacks formalization.

2 Encrypting a Gate

2.1 Label generation for two-input Boolean gates

The truth table of a Boolean gate can be represented in Table 1, where z_i denotes a single-bit value 0 or 1.

Table 1: Boolean Gate Truth Table

Input a	Input b	Output c
0	0	z_0
0	1	z_1
1	0	z_2
1	1	z_3

For input wires a and b, their 0/1 labels are denoted $x(a)/y(a)$ and $x(b)/y(b)$ (if they are initial circuit inputs, the labels are randomly generated; if they are intermediate values, they are computed from previous gates).

The Garbler selects and publishes an RSA modulus $N = p \cdot q$ with secret primes p, q and public exponents: e, e_1, e_2, e_3, e_4 (invertible modulo $\phi(N)$). Output labels are computed using the formula from BitVM3 below. This approach differs from the BitVM3 protocol, where output labels are used to compute input labels, whereas our method calculates output labels directly from input labels.

$$\begin{aligned} c_0 &= x(a)^e \cdot x(b)^{e_1} \mod N \\ c_1 &= x(a)^e \cdot y(b)^{e_2} \mod N \\ c_2 &= y(a)^e \cdot x(b)^{e_3} \mod N \\ c_3 &= y(a)^e \cdot y(b)^{e_4} \mod N. \end{aligned} \tag{1}$$

Obtaining the truth table expressed with labels in Table 2.

Table 2: Garbled Truth Table

Input a	Input b	Output c
$x(a)$	$x(b)$	c_0
$x(a)$	$y(b)$	c_1
$y(a)$	$x(b)$	c_2
$y(a)$	$y(y)$	c_3

We then introduce **output adaptors** to unify the output labels of a gate, enforcing that each gate outputs two labels. Let o_0, o_1 be the first output label among c_0, c_1, c_2, c_3 corresponding to plaintext 0 and 1 respectively. For other 0 ciphertexts and 1 ciphertexts, normalize them to o_0 and o_1, respectively:

- If c_i encodes 0, apply output adaptor o_0/c_i;
- If c_i encodes 0, apply output adaptor o_1/c_i.

For all gates that are neither constant 0 nor constant 1, two output adaptors are needed to unify the output labels. Through two output adaptors, we can obtain the final truth table with labels (outputs labels expressed only using o_0 and o_1). Taking gate AND as an example, the final truth table of adapted garbled table is shown in Table 3:

Table 3: Adapted Garbled Table

Input a	Input b	Output $a \wedge b$	Output Adaptor
$x(a)$	$x(b)$	$o_0 = c_0$	/
$x(a)$	$y(b)$	o_0	o_0/c_1
$y(a)$	$x(b)$	o_0	o_0/c_2
$y(a)$	$y(y)$	$o_1 = c_3$	/

2.2 Application Scenario of On-Bitcoin Verification and NOT Gate Handling

When verifying computation correctness via GC on Bitcoin, the Garbler (i.e., Prover) reveals input labels corresponding to valid inputs, enabling the Evaluator (i.e., Verifier) to execute the GC circuit and derive output labels. If the derived labels mismatch the expected output, this indicates prover fraud.

Unlike traditional GC applications of protecting input plaintext, verification on Bitcoin requires public disclosure of the 0/1 semantics for committed input/output labels. During GC evaluation, the Evaluator always knows the plaintext values of each gate's input labels, thus determining adaptor usage to adjust output labels while tracking their plaintext meanings. Concurrently, the Garbler must avoid exposing both labels of any input wire to prevent the Evaluator from computing adversarial output labels that trigger penalties.

For a 2-input GC gate, we denote:

- Input labels: $x(a), y(a), x(b), y(b)$;
- Output adaptors: a_0, a_1;
- Output labels: o_0, o_1.

The output adaptors are public. Given input labels $z(a) \in \{x(a), y(a)\}$ and $z(b) \in \{x(b), y(b)\}$ (revealed by the Garbler or computed from prior gates), the Evaluator:

1. Computes the output label c using Equation (1) without adaptors;
2. Uses the plaintext truth table to select the appropriate adaptor $a \in \{1, a_0, a_1\}$;
3. Compute the output label as $a \cdot c$, and record its plaintext meaning according the plaintext truth table.

For the single-input NOT gates, the Garbler requires no cryptographic computation during GC construction—only semantic inversion of label meanings (0→1, 1→0) for subsequent gates. The Evaluator similarly performs semantic inversion during evaluation. Consequently, NOT gates are free in our GC construction (no cryptographic operations or extra data needed).

3 GC: Encrypting a Boolean Circuit

We demonstrate multi-gate composition in a GC circuit using a 2-bit adder example.

Given inputs $(in_0, in_1), (in_2, in_3)$, representing values:

$$a = 2 \cdot in_1 + in_0, \quad b = 2 \cdot in_3 + in_2.$$

The 3-bit output (out_2, out_1, out_0) represents:

$$4 \cdot out_2 + 2 \cdot out_1 + out_0 = a + b.$$

Example: 2+3=5 encodes as (1,0)+(1,1)=(1,0,1).

3.1 Converting to Boolean Circuit

We will use the following three binary gates: XOR $\oplus$, AND $\wedge$, OR $\vee$ in Table 4.

Table 4: $\oplus, \vee, \wedge$ Truth table

Input a	Input b	Output $a \oplus b$	Output $a \wedge b$	Output $a \vee b$
0	0	0	0	0
0	1	1	0	1
4	0	1	0	1
4	1	0	1	1

A 2-bit adder can be constructed using seven gates g_0 to g_6 with the Boolean circuit diagram shown in Figure 1.

3.2 Constructing the GC Circuit

The Garbler constructs the GC of the above 2-bit addition.

First, the Garbler generates two random values (labels) for each input bit in_0, in_1, in_2, in_3: $x(in_0), y(in_0), x(in_1), y(in_1), x(in_2), y(in_2), x(in_3), y(in_3)$.

Then the Garbler calculates gate-by-gate using the calculating formulas and output adaptor rules. Let $a_0(g), a_1(g)$ represent the two output adaptors corresponding to gate g.

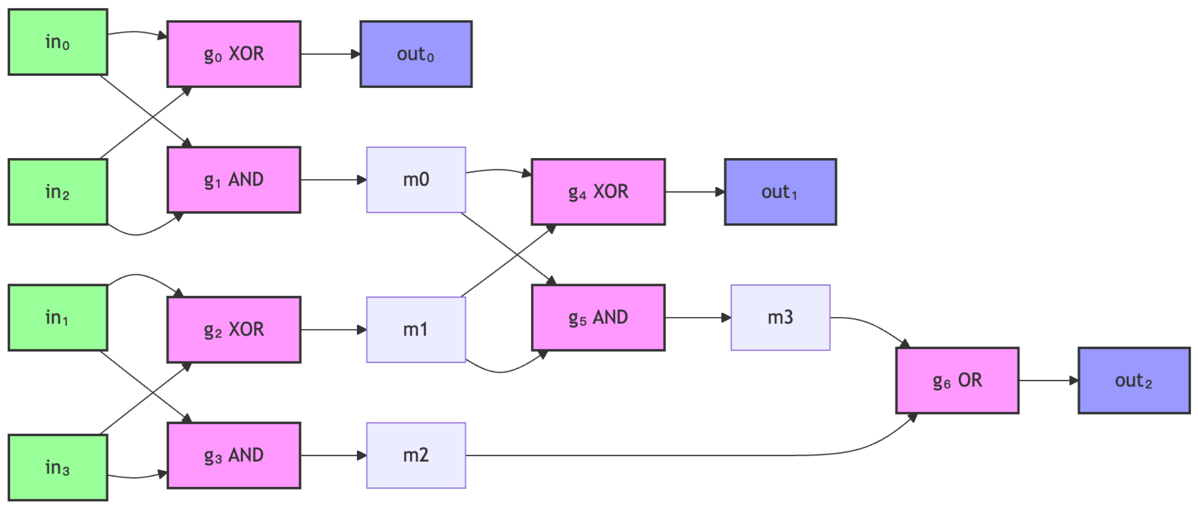

Fig. 1: A 2-bit Adder Circuit

- The Garbler first uses the input labels to calculate g_0, g_1, g_2, g_3, deriving the output labels of $out_0, m_i (i = 0, 1, 2)$: $x(out_0), y(out_0), x(m_i), y(m_i)$ and output adaptors: $a_0(g_i), a_1(g_j), j = 0, 1, 2, 3$ for each gate.
- Then, the Garbler uses the labels of m_0, m_1 to compute g_4, g_5, obtaining the gate output labels $x(out_1), y(out_1)$ and $x(m_3), y(m_3)$, as well as the gates' output adaptors $a_0(g_k), a_1(g_k), k = 4, 5$.
- Finally, the Garbler uses the labels of m_2, m_3 to compute g_6, obtaining the gate output labels $x(out_2), y(out_2)$ and the output adaptors $a_0(g_6), a_1(g_6)$ for g_6.

3.3 Evaluating the GC

A GC is described through input labels, output adaptors for all gates and output labels. Given a set of input labels for 2-bit adder: $z(in_0), z(in_1), z(in_2), z(in_3)$ ($z = x$ or y, 1 label for each variable), the Evaluator can use all the output adaptors of the circuit gates to perform the computation and obtain a set of output labels: $z(out_0), z(out_1), z(out_2)$.

Simulate $2 + 3 = 5$: using **input labels** $x(in_0), y(in_1), y(in_2), y(in_3)$, evaluating the circuit derives **output labels** $y(out_0), x(out_1), y(out_2)$.

4 Reblinding Schemes for GC

4.1 Input Adaptors

Adapters can align input labels between GC or sub-GC circuits. When applied to input labels of circuit/sub-circuit, such adapters are termed input adaptors.

Consider two GC/sub-GC circuits with their respective input labels, output adaptors, and output labels denoted as (L, A, O) and (L', A', O'). Using the input adaptor $B = L/L'$, the input labels L' can be transformed into L, yielding the GC circuit (L, A, O). This conversion leverages the homomorphic properties of GC construction to reuse portions of the circuit description, thereby reducing communication complexity.

4.2 Reblinding the Sub-GC

GC construction with input adaptors. Consider a Boolean circuit $\mathbb{C}$ that repeatedly invokes sub-circuits $\mathbb{C}_i, i = 1, 2, \cdots, n$, where each $\mathbb{C}_i$ is called k_i times.

Now, the Garbler first choose k_i reblinding factors $u_{i,j}$ for sub-circuits $\mathbb{C}_i$. Given input labels L, the Garbler construct the GC with the following two modification:

- The Garbler record the input labels L_i, the output adaptor A_i, and the output labels O_i for the first invocation of sub-circuits $\mathbb{C}_i$ before using reblinding factor $u_{i,1}$ for adjustment.
- For the j-th invocation ($j = 1, 2, \cdots, k_i$) of sub-circuits $\mathbb{C}_i$, using the input adaptor $B_{i,j} = L_i^{u_{i,j}}/L_{i,j}$ (where $L_{i,j}$ is the input labels before adjustment), the Garbler adjusts the input labels from $L_{i,j}$ to $L_i^{u_{i,j}}$. (The first invocation of sub-circuits $\mathbb{C}_i$ also needs to adjust the input labels from L_i to $L_i^{u_{i,1}}$during the GC construction.)

We clarify that only the input labels of reused sub-circuits are adjusted. When these labels are used as inputs to other gates in the circuit, they remain unchanged.

GC Description. Now the new GC description comprises:

- Input labels L for the main circuit,
- Output adaptors A (excluding reused sub-circuits),
- Triples (L_i, A_i, O_i) for the unadjusted GC of first invocation to sub-circuit $\mathbb{C}_i$,
- Reblinding factors $u_{i,j}$ and input adaptors $B_{i,j}$,
- Output labels O.

GC Evaluation. Though we omit detailed description of the GC tuple $(L'_{i,j}, A'_{i,j}, O'_{i,j})$ for the j-th invocation of $\mathbb{C}_i$, the input adaptors yield:

$$L'_{i,j} = L_{i,j} \cdot B_{i,j} = L_i^{u_{i,j}}.$$

Leveraging homomorphic properties in our GC construction – specifically multiplicative/divisional composition of gate labels, output adapters, and input adaptors – we establish:

$$A'_{i,j} = A_i^{u_{i,j}}, O'_{i,j} = O_i^{u_{i,j}}.$$

Thus, given $A, (L_i, A_i, O_i), u_{i,j}, B_{i,j}$, the Evaluator can evaluating the GC with per-bit input labels $L^* \subset L$, and derive the corresponding per-bit output labels $O^* \subset O$.

5 Instantiating BitVM3 Schemes

5.1 Scheme Description

Suppose the Groth16 verification circuit $\mathbb{C}$ repeatedly invokes sub-circuits $\mathbb{C}_i, i = 1, 2, \cdots, n$, with each $\mathbb{C}_i$ being called k_i times, the Garbler constructs the GC with randomly chosen input labels L: $(L, A, (L_i, A_i, O_i)_{i=1,2,\cdots,n}, u_{i,j}, B_{i,j}, O)$.

Then the Garbler selects two global reblinding factors u, v, and:

- Publishes commitments of L^v, O^v on-chain;
- Publishes off-chain: $A, (L_i, A_i, O_i), B_{i,j}, u, v, L^u, O^u, u_{i,j}$;
- Generates ZKP proving: $(L^v)^u = (L^u)^v, (O^v)^u = (O^u)^v$ and L^v, O^v are consistent with the on-chain commitments.

The GC with input labels L^u is used to verify the consistency of the plaintext, while the GC with input labels L^v constructs an on-chain verification. The validity of committed labels L^v, O^v is guaranteed by the homomorphic properties between the input-output labels L^u, O^u and L^u, O^u.

The Evaluator verify:

- $(L^u, A^u, (L^u_i, A^u_i, O^u_i), B^u_{i,j}, O^u, u_{i,j})$ compose a valid GC;
- ZKP proof is valid.

Without revealing L^v, O^v, we can make sure the committed L^v, O^v are consistent with a valid GC and the corresponding output adaptors and input adaptors are A^v and $B^v_{i,j}$. When the Garbler reveal a set of valid labels (one for each wire) $L^* \subset L^v$, the Evaluator can evaluate the GC using output adaptors A^v, input adaptors$B^v_{i,j}$, and sub-circuit GC (L^v_i, A^v_i, O^v_i), deriving the output labels $O^* \subset O^v$.

Through choosing a different value v, committing to new input/output labels on Bitcoin, and propose corresponding ZKP proof about homomorphism between committed input/output labels and L^u, O^u, the scheme can be used again.

5.2 Complexity Analysis

The Groth16 verification circuit on the BN254 curve comprises approximately 15 billion gates and executes $\sim 30,000$ modular multiplications. Each modular multiplication operation requires $\sim 700,000$ gates, with RSA modulus of 256-byte. Communication complexity is analyzed as follows:

- On-chain complexty follows BitVM3's analysis: 56kB (revealing a set of commited input labels corresponding to plaintext proof) for 'assertTx' and 256 bytes (output label corresponding to 0 output) for 'disproveTx'.
- Off-chain complexity total $\sim 8GB$ dominated by adaptor materials of modular multiplication:
  - Output adaptors for one modular multiplication $700,000 \times 2 \times 256 \approx 350MB$;
  - Input adaptors for 30,000 times modular multiplication $30,000 \times 2 \times 2 \times 256 \times 256 \approx 7.7GB$.

The current 8GB off-chain data requirement remains prohibitively large for large-scale applications. In subsequent versions, we will implement nested reuse of submodule to reduce off-chain data by a factor of 10 to 100.

References

1. R. Linus, "BitVM3: Efficient computation on Bitcoin," https://bitvm.org/bitvm3.pdf, 2025, [Online; accessed 2025].
2. R. Linus *et al.*, "BitVM 2: Permissionless Verification on Bitcoin," https://bitvm.org/bitvm_bridge.pdf, 2024, [Online; accessed 2025].
3. A. C. Yao, "Protocols for Secure Computations," https://research.cs.wisc.edu/areas/sec/yao1982-ocr.pdf, 1982.
4. J. Groth, "On the Size of Pairing-based Non-interactive Arguments," https://eprint.iacr.org/2016/260.pdf, 2016, [Online; accessed 2025].
5. J. Rubin, "Delbrag," https://rubin.io/public/pdfs/delbrag.pdf, 2025, [Online; accessed 2025].