# GOAT BitVM2 White Paper

GOAT Network Research Group

Email: contact@goat.network

Jan. 2026 - Version 1.0

**Abstract.** The BitVM2 protocol is often regarded as one of the most secure bridge frameworks for Bitcoin and its side-systems. However, it still faces challenges in a real-world zkRollup setting and deployment, such as the inability to support arbitrarily sized withdrawals, inconsistent challenge incentives, potential double-spending attacks by operators, and large on-chain data for dispute resolution. To address these issues (without compromising the 1-of-n honesty assumption), we introduce a trust-minimized settlement protocol—enabling native zkRollups on Bitcoin—called GOAT BitVM2. 1) This includes a method for committing ZK proof public inputs to Bitcoin as part of BitVM2's optimistic computation, e.g., a commitment to the L2's sequencer-set configuration. With this, both state transitions and the latest L2 state can be verified by the sequencer set on Bitcoin, resisting potential double-spending attacks by operators. 2) The operator and challenger collateral in the BitVM2 protocol are moved onto L2, with the operator's reimbursement constrained by a chain based synchronization primitive, in combination with CPFP[1], improving the operator's capital efficiency. 3) By integrating with Atomic Swap, this protocol allows end-users to withdraw from L2 to L1 with arbitrary amounts, and the operator reimburses themselves using L2 state transition proofs.
The subsequent innovation of the BitVM2 architecture, integrating garbled circuits (GC) with designated-verifier (DV) SNARKs, further optimizes performance and reduces on-chain costs. By shifting computations (e.g., the SNARK verifier) off-chain with GC, committing the GC's inputs and output labels on-chain and revealing the input labels when challenged to compute the output cleartext, the system minimizes its on-chain footprint. The correctness of the GC is guaranteed by zero-knowledge proofs, while the validity of the DV-SNARK's Structured Reference String (SRS) is ensured via pairing verification for the *BN254* curve, and through a combination of the Cut & Choose technique and a random challenge mechanism for the *Sect233k1* curve, thereby achieving a balance between on-chain and off-chain complexity.
All implementations are publicly available on GitHub at BitVM, BitVM2-node, and bitvm2-gc, facilitating reproducibility and further research.

**Keywords:** GOAT Network, BitVM2, Garbled Circuits, zkVM, zkRollup, Cross-chain Bridge

## 1 Introduction

BitVM [12] and its evolutions BitVM2 [13], proposed by Robin Linus et al., establish a trust-minimized bridge protocol for Bitcoin [14]. They enable Turing-complete smart contracts and side systems (like zkRollups) through a fraud-proof arbitration mechanism without requiring a hard fork, operating under the 1-of-n honesty assumption[2]. The protocol leverages pre-signed transactions, one-time signatures, and SNARK proofs to verify off-chain computations on-chain, conceptually similar to Optimistic Rollups in Ethereum [16][2].

Serving as the cryptographic foundation for cross-chain bridges and zkRollups, the original BitVM2 protocol achieves two key breakthroughs: 1) Maintaining Bitcoin's base layer integrity

[1] CPFP: Child Pays For Parent (CPFP) is a fee bumping technique where a user spends an output from a low-feerate unconfirmed transaction in a child transaction with a high feerate in order to encourage miners to include both transactions in a block, https://bitcoinops.org/en/topics/cpfp/

[2] The 1-of-n honesty assumption requires that liveness is maintained by at least one Operator, and that validity is enforced by at least one honest Challenger who is obligated to challenge a dishonest Operator.

without protocol forks under the 1-of-n honesty assumption; 2) Compared to BitVM, BitVM2 allows anyone to challenge and slash a faulty Operator with only three on-chain transactions, with a delay of no more than 2-3 weeks, thus enabling a relatively capital-efficient improvement.

However, the BitVM2 protocol continues to grapple with significant on-chain costs and operational inefficiencies. Delbrag [17] (and subsequently BitVM3 [11]) dramatically reduces on-chain costs using garbled circuits (GC) [19]. During setup, a Garbler commits to a SNARK verifier circuit in a Taproot tree and shares it with an Evaluator. If the Garbler submits an invalid SNARK proof, the Evaluator generates a fraud proof using the pre-shared circuit. This approach retains BitVM2's trust model while minimizing its on-chain footprint. This is achieved by directly verifying the output label for publicly verifiable garbled circuits.

Representing the verification circuit as a Boolean circuit results in an extremely large circuit size, with the number of gates reaching billions. This massive circuit imposes a significant burden on both off-chain data storage requirements and the computational complexity of proving the correctness of the GC construction. Although algebraic GC circuits offer a way to reduce the circuit scale, current schemes [8] can only support computations over integers and cannot efficiently simulate modulo operation over a large prime. Alpen Labs' [3] proposed using SNARK schemes with designated verifiers (DV-SNARK) on the binary curve to reduce the size of the GC circuit in the ZKP verifier, as a sequencer converts Elliptic Curve Pairing to Multiple Scalar Multiplication and greatly reduces the circuit size. For applications requiring different trade-offs, we provide two elliptic curve options: *BN254* and *Sect233k1*. The *BN254* curve is pairing-friendly, allowing efficient pairing operations, which enables straightforward verification of the SRS validity, and it is crucial to support more concise and efficient protocol sequences. Alternatively, the *Sect233k1* curve is defined over a binary extension field. This characteristic supports constructing significantly smaller GC implementations and reduces the GC size by two orders of magnitude.

In addition to the significant on-chain inefficiencies, use of BitVM2 to build a practical Bitcoin zkRollup faces some critical challenges, listed below.

**Operator's Double-Spending Attack**: When the operator gets challenged and reveals the execution trace of the ZK proof's verifier, and the proof is the commitment of the L2's state transition, then an Operator could potentially create a valid proof for an incorrect or forked state (e.g., from an L2 fork) and use it to fraudulently withdraw funds, executing a double-spend. How to determine the correct latest state root becomes an issue since we can't trust the operator to publish the proof's public input, especially when the L2 has its own decentralized sequencer network.

**Inability to Withdraw Arbitrary Amounts:** The operator in BitVM2 is able to withdraw from Tapscript in the `Peg-in` transaction, however it is not possible for every end-user of an L2 to run an operator. Furthermore, the peg-in transaction amount is fixed, hence it is necessary to allow end-users to withdraw arbitrary amounts from L2 to L1.

**Misaligned Incentives:** BitVM2 lacks a practical incentive mechanism suited for real-world deployment. This could result in serious security vulnerabilities if there are not enough challengers or operators participating in the network. For instance, if no fraud occurs over an extended period, challengers might not receive sufficient rewards to stay motivated and could decide to exit the protocol. An additional concern is that challengers may not receive their rewards. During the crowdfunding process for the Challenge transaction, the original challenger may not be the one who ultimately submits the final Disprove transaction - a Bitcoin miner could front-run the transaction to claim the rewards instead. As a result, the staked funds may not be properly distributed to the honest challenger.

GOAT BitVM2 combines three architectural innovations to overcome these limitations.

**Decentralized Sequencer Set Commitment Scheme:** This innovation addresses the double-spending attack by securely anchoring the L2 state on Bitcoin. Instead of directly trusting the Operator's public input, the system commits to the state of a decentralized sequencer set (e.g., their public keys) via a Bitcoin transaction. This committed state $x$ then becomes a verifiable public input[3] for the computation $f(x, w) = y$ in the `Kickoff` process, ensuring the Operator's proof is based on the canonical L2 state.

**Universal Operator Abstraction:** To balance risks and incentives, the distinct roles in the system (Operator, Challenger, Committee, Sequencer, and Watchtowers, etc.) are unified into a single Universal Operator role. Participants stake funds and are assigned different roles over time through a rotation mechanism. This ensures that no single entity is permanently burdened with high-cost roles or only profitable ones, leading to a more stable and sustainable economic model.

**NIZK based Verifiable Garbled Circuit:** Using GCs to improve the optimistic challenge efficiency - instead of verifying Groth16 on Bitcoin, DV-SNARKs can significantly reduce on-chain computation complexity. In such a setting, the L2's state transition is proven with Ziren[4] and the STARK proof is wrapped to the DV-SNARK proof, before the DV-SNARK verifier is proven by Ziren for BitVM2's dispute resolution.

Combining these innovations, GOAT BitVM2 improves on the theoretical framework of BitVM2, resulting in a pragmatic, efficient, and secure foundation for Bitcoin zkRollups.

## 2 Background and Building Blocks

### 2.1 zkRollups

A zkRollup is a Layer-2 (L2) scaling solution that executes transactions off-chain while submitting succinct validity proofs to the underlying Layer-1 (L1) blockchain. By leveraging zero-knowledge (ZK) proof systems (e.g., Groth16), zkRollups cryptographically guarantee the correctness of off-chain state transitions without requiring transaction re-execution on L1.

The core architecture of zkRollups is guided by the following two principles:

1. **Succinct Verification:** All off-chain computation is verified on L1 via succinct ZK proofs, preserving integrity while avoiding redundant execution.
2. **Data Availability:** Only essential state data (e.g., state differences or compressed transaction data) is published on L1 to ensure data availability.

Figure 1 illustrates the generic workflow of a zkRollup system. Users deposit assets on L1, which are then represented and transacted on L2. After off-chain execution, the updated L2 state is committed to L1 together with a validity proof, enabling trust-minimized withdrawals back to L1.

**Deposit (also referred to as `Peg-in` within the BitVM2 context):** A user deposits assets by locking them on L1 non-custodially. The zkRollup sequencer detects this event, validates the finality of the deposit transaction, and updates the L2 state accordingly—typically by minting equivalent wrapped tokens on L2.

**Transaction Execution and State Commitment:** A sequencer batches L2 transactions, executes them off-chain, and generates a new L2 state root (e.g., a Merkle root representing the latest

[3] Public inputs: a virtual machine reads the inputs, runs the logic, and generates the outputs. For a zkVM, to commit the inputs, public inputs are introduced, which usually consist of a part of the inputs and all outputs.

[4] https://github.com/ProjectZKM/Ziren.

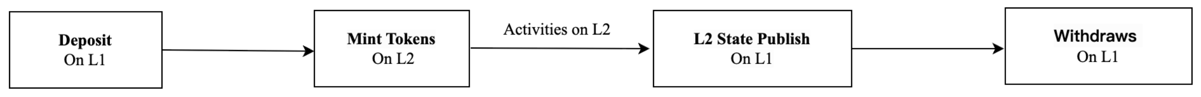


Fig. 1: zkRollup Workflow

state). The sequencer then publishes the raw transaction data (for data availability) and a validity proof (e.g., a zk-SNARK) to the L1 verifier contract. The proof attests to the correctness of the state transition.

**Withdrawal:** To withdraw assets from L2 to L1, a user submits a burn transaction on L2. The sequencer includes this in the next batch, generating a validity proof for the updated state (which reflects the burned L2 tokens). The user then submits a Merkle proof to the L1 contract, which verifies the validity proof and releases the locked L1 assets.

## 2.2 BitVM2 Protocol Overview

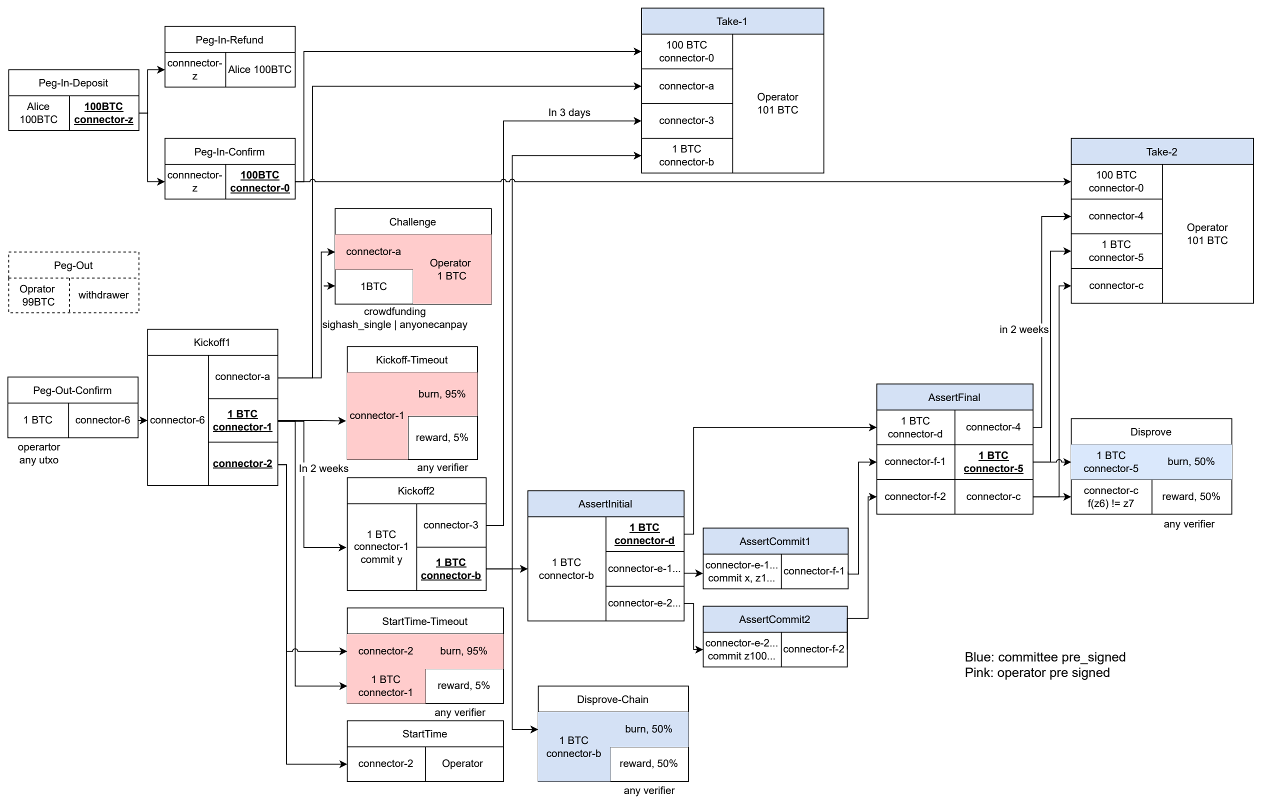


Fig. 2: Original BitVM2 Bridge Protocol

As shown in Fig.2, BitVM2 uses the presigned Transaction Graph[5] to achieve logic persistence, and uses a one-time signature [10] to implement storage persistence. In combination with timelock and Taproot, BitVM2 can be used to implement a trust-minimized bridge protocol between Bitcoin and an L2.

The protocol operation can be condensed into three key phases:

– Asset Deposit (Peg-in): A user locks BTC on L1 via a `Peg-In-Deposit` transaction. The Operator must respond with a `Peg-In-Confirm` transaction for the user to receive equivalent assets on L2. If the Operator fails to respond, the user can retrieve their BTC using a `Peg-In-Refund` transaction after a timeout.
– Withdrawal Initiation and Challenge Window (Kick-off): When a user burns L2 assets to withdraw, the Operator advances the BTC payment on L1 and initiates a series of `Kickoff` transactions. This starts a dispute period (e.g., two weeks). A Challenger can contest the withdrawal during this time if they detect fraud (e.g., the Operator using an invalid L2 state).
– Settlement:
  - Optimistic Path (Happy Path): If no challenge is raised after the dispute period, the Operator submits a `Take1` transaction to get reimbursed seamlessly.
  - Challenge Path (Unhappy Path): If a challenge is issued, the Operator must publicly prove the withdrawal's validity on-chain. If the proof is invalid, the Challenger can block the reimbursement and claim a reward. If the challenge fails or times out, the Operator eventually gets reimbursed via a `Take2` transaction.

## 2.3 BitVM Bridge

BitVM is one of the promising protocols to build bridges between Bitcoin and side-systems, with its core innovation being a trust-minimized solution achieved through the BitVM smart contract. This solution is based on a key design principle: the security of funds is guaranteed as long as there is at least one honest participant in the system who can detect and challenge malicious behavior. This approach fundamentally eliminates the dependence on centralized authorities or multisig set-ups.

## 2.4 Ziren

A zkVM (Zero-Knowledge Virtual Machine) is a cryptographic system that enables verifiable computation by generating a ZK proof for arbitrary program executions. It allows developers to write code in high-level programming languages (e.g., Rust, Go) and compile it into instructions compatible with specific Instruction Set Architectures (ISAs). The zkVM executes these instructions, generates a trace of the execution recording register states and memory accesses at each clock cycle, and produces a succinct proof to validate the correctness of the computation. Key components include *zkCompiler*, *Prover* and *Verifier*.

**zkCompiler:** Compiles high-level code into ISA-specific binaries (e.g. MIPS, RISC-V), generates execution traces, and converts the traces into polynomials for constraint satisfaction checks.

[5] In BitVM2, a Transaction Graph is a pre-defined, interconnected set of Bitcoin transactions that form a directed acyclic graph. This structure is used to enforce a complex challenge-response protocol on Bitcoin without requiring changes to Bitcoin's core protocol.

**Prover:** Commits the polynomials, and generates ZK proofs. Compared to native execution, Zero-knowledge proving is currently slower by 100-1000x. Hardware acceleration, continuation[6], and pipelined proving are exploited to reduce computing overhead and latency.

**Verifier:** A program to verify ZK proofs, run in an L1 smart contract or covenant to share the security of an L1's consensus and achieve the finality and settlement of an L2's state transitions.

Ziren [18] is a production-grade zkVM based on the MIPS32r2 instruction set, a stable RISC architecture known for deterministic execution and minimal circuit overhead. Developed by the ZKM[7] team, it optimizes zero-knowledge proof generation through efficient zkCompiler, pipelined proof architecture, and a cutting-edge proof system, ensuring high instruction proof efficiency and reduced audit requirements. Ziren demonstrates remarkable performance when accelerated by a single NVIDIA RTX 4090 GPU, achieving a computational throughput of more than 4 million Hz. This high-performance capability is particularly evident in its handling of the Poseidon2 hash function, for which it can generate over 300,000 proofs per second.

### 2.5 Garbled Circuits and On-Chain Verification

A Garbled Circuit (GC) is a cryptographic protocol, introduced by Yao [19] in 1986, that enables secure two-party computation. It allows two distrustful parties to jointly compute a function on their private inputs without revealing those inputs to each other. The core idea is to encode the function as a Boolean circuit (composed of gates like AND, OR, XOR etc.). One party, the Garbler, encrypts or "garbles" the truth table of each gate, turning meaningful binary values (0/1) into random-looking labels. The other party, the Evaluator, then obliviously evaluates this encrypted circuit using these labels - through a process like Oblivious Transfer (OT) to obtain the labels corresponding to their own inputs - and learns only the final output without gaining any information about intermediate values or the Garbler's inputs. This approach supports arbitrary functions with a constant round of interaction. While early GCs had high communication costs, key optimizations like free-XOR [9] (making XOR gates virtually free in terms of cost) and point-and-permute (which allows the Evaluator to decrypt only one entry per garbled table) have significantly improved efficiency.

GC is compatible with optimistic verification computation on Bitcoin as introduced in Delbrag [17]. In a GC protocol, the protocol proceeds in three rounds: garble, authenticate, and evaluate - and is carried out entirely off-chain.

1. The Garbler will choose secret input and output labels and garble the circuit. The Garbler and Evaluator also carry out a protocol to ensure the GC is correctly constructed.
2. The Garbler chooses some inputs and authenticates them by revealing their input labels.
3. The Evaluator will evaluate the GC using the input labels to compute the output labels.

For a SNARK verifier circuit, if the protocol is secure, then the Evaluator should learn the output 1-label if and only if the Garbler authenticated a valid proof input. So, the output 0-label constitutes a fraud proof. Given an authentication mechanism that is compatible with GC and verifiable on chain, we can use this fraud proof to slash on-chain.

GOAT BitVM2 adopts the Poseidon2 hash function $H(\cdot)$ as the cryptographic primitive for garbled circuit (GC) construction, and follows the *free-XOR* optimization together with the *secret-free* garbling technique introduced in [5]. The garbler constructs the GC according to the following procedure.

[6] Continuation: https://docs.zkm.io/design/continuation.html

[7] ZKM: https://zkm.io

*Global Setup.* The garbler samples a global offset $r \in \{0,1\}^\lambda$. For every wire $w$, the associated wire labels $w^0$ (encoding bit 0) and $w^1$ (encoding bit 1) satisfy

$$w^1 = w^0 \oplus r.$$

*Input Wire Label Generation.* For each input wire $w$, the garbler samples $w^0\{0,1\}^\lambda$ uniformly at random and derives $w^1 = w^0 \oplus r$.

*Gate Garbling.* Let $a$ and $b$ denote input wires, $o$ the output wire, and $\mathsf{gid}$ the unique identifier of the gate.

– **XOR Gate (Free-XOR).**
$$w_o^0 = w_a^0 \oplus w_b^0.$$
– **NOT Gate (Free-NOT).**
$$w_o^0 = w_a^1.$$
– **AND Gate.** The garbler computes
$$w_o^0 = H(w_a^0, \mathsf{gid}),$$
and generates a single ciphertext
$$c = H(w_a^1, \mathsf{gid}) \oplus w_o^1 \oplus w_b^1.$$
– **OR Gate.** The garbler computes
$$w_o^0 = H(w_a^1, \mathsf{gid}) \oplus r,$$
and generates a single ciphertext
$$c = H(w_a^1, \mathsf{gid}) \oplus H(w_a^0, \mathsf{gid}) \oplus w_b^1.$$
For every output wire $o$, the 1-label is derived as
$$w_o^1 = w_o^0 \oplus r.$$

*Gate Evaluation.* Given garbled input labels $w_a^x$ and $w_b^y$, where $x, y \in \{0,1\}$ are the encoded bits (note that the NOT gate requires only $w_a^x$), the evaluator computes the output label $w_o$ as follows.

– **XOR Gate:**
$$w_o = w_a^x \oplus w_b^y.$$
– **NOT Gate:**
$$w_o = w_a^x.$$
– **AND Gate (with ciphertext $c$):**
  - If $x = 0$, then $w_o = H(w_a^x, \mathsf{gid})$.
  - Otherwise, $w_o = H(w_a^x, \mathsf{gid}) \oplus c \oplus w_b^y$.

– **OR Gate (with ciphertext $c$):**
  - If $x = 1$, then $w_o = H(w_a^x, \mathsf{gid})$.
  - Otherwise, $w_o = H(w_a^x, \mathsf{gid}) \oplus c \oplus w_b^y$.

### 2.6 Designated Verifier SNARK

DV-SNARK is an optimized variant of SNARK designed to address the high on-chain costs and computational overhead associated with traditional SNARK verification circuits. It achieves this by replacing expensive cryptographic operations with more efficient alternatives, making it particularly suitable for blockchain applications like Bitcoin L2 solutions, where minimizing the on-chain script is critical.

Traditional SNARKs (e.g., Groth16 [7]) rely on elliptic curve pairings for verification, which require complex operations in large prime fields. These operations must be encoded as binary circuits when integrated with GC, leading to massive circuit sizes (billions of gates) and prohibitive off-chain communication cost.

DV-SNARK uses a designated verifier model, where the verifier holds a secret key, allowing the prover to generate proofs tailored to that specific verifier without revealing critical information. It adapts techniques from schemes like DV-KZG [15] to substitute pairing operations with elliptic curve scalar multiplications and largely reduces the circuit size.

## 3 GOAT BitVM2's Design Overview

GOAT BitVM2 aims to enable a trust-minimized Bitcoin zkRollup based on the BitVM2 protocol and Garbled Circuit. Its core innovations include: 1) introducing a chain based synchronization primitive over BitVM2 for operators to share collateral in multiples reimbursements; 2) combining Atomic Swap with BitVM2 to implement a user-friendly Bridge-in and Bridge-out UX (see Section 3.2 for more details); 3) introduction of universal operator abstraction and shifting the collateral inside BitVM2 from L1 to L2 to establish a concise and efficient penalty and incentive mechanism; and 4) integration of GCs with DV-SNARK to optimize verification efficiency.

In this section, we introduce the main roles and components of GOAT BitVM2 in relation to the overall Rollup protocol.

### 3.1 Roles and Universal Operators

The key participants in the GOAT BitVM2 protocol consist of the Committee, Operator, Challenger, Watchtower, Designated Verifier, and Relayer. Their primary duties are defined as follows in Table 1 and the protocol consolidates all system roles (Prover, Challenger, Sequencer, and Designated Verifier) into a single Operator identity, which we call *Universal Operator*. Universal Operators stake tokens on L2 and are assigned different roles in rotating epochs. It achieves certain key advantages:

– Balanced Economics: Operators rotate through both profit-generating (e.g., Sequencer earning fees) and cost-incurring roles (e.g., Prover generating proofs). This ensures a sustainable balance between individual income and costs. See GOAT Network Economic Paper for more details.

– Aligned Incentives: Since Operators know they will perform various roles over time, short-term costs in one epoch can be offset by profits in a subsequent epoch, creating a cross-subsidization effect.

– Enhanced Reliability: The system avoids permanent dependency on any single entity. If an Operator fails, its responsibilities can be reassigned to others in the next epoch, improving resilience.

In summary, this design unifies different functions into a single pool of staked Operators who share and rotate all responsibilities, mitigating centralization risks, and supporting a sustainable incentive mechanism.

Table 1: GOAT BitVM2 Role Definitions

| **Role** | **Responsibilities** | **Honesty Assumption** |
|---|---|---|
| **Committee** | – Acts as the $n$-of-$n$ signers for the pre-signed BitVM2 transaction graph<br>– Commits the active sequencer set to Bitcoin | $1/n$ |
| **Operator** | Anyone can act as an Operator.<br>– Exchanges PegBTC for native BTC with users<br>– Generates validity proofs, initiates reimbursements, and responds to challenges<br>– Reveals pre-images of hashed timelocks to Watchtowers [1] | $1/\infty$ |
| **Challenger** | Anyone can act as a Challenger.<br>– Verifies reimbursement correctness off-chain<br>– Submits challenge transactions to force execution onto the unhappy path<br>– Upon full execution trace disclosure, detects fraud and spends the Assert UTXO to halt reimbursement | $1/\infty$ |
| **Watchtower** | – Acts as monitors for Bitcoin's longest chain<br>– Submits block headers from the longest Bitcoin chain<br>– Commits sequencer updates on Bitcoin to determine public input correctness during challenges | $1/n$ |
| **Designated Verifier** | – Participates in SRS generation<br>– Maintains and reveals secrets for the DV-SNARK verifier circuit | $1/n$ |
| **Relayer** | – Relays correlation information between Bitcoin and GOAT | $1/\infty$ |

### 3.2 Core Workflow

As illustrated in Figure 3, we introduce the lifecycle of the bridge protocol between Bitcoin mainnet (L1) and GOAT Network (L2), and demonstrate the deposit and withdrawal with user Alice.

#### Peg-in

– `Peg-In-Prepare`: User Alice locks 100 BTC in a specific output to initiate the deposit process.

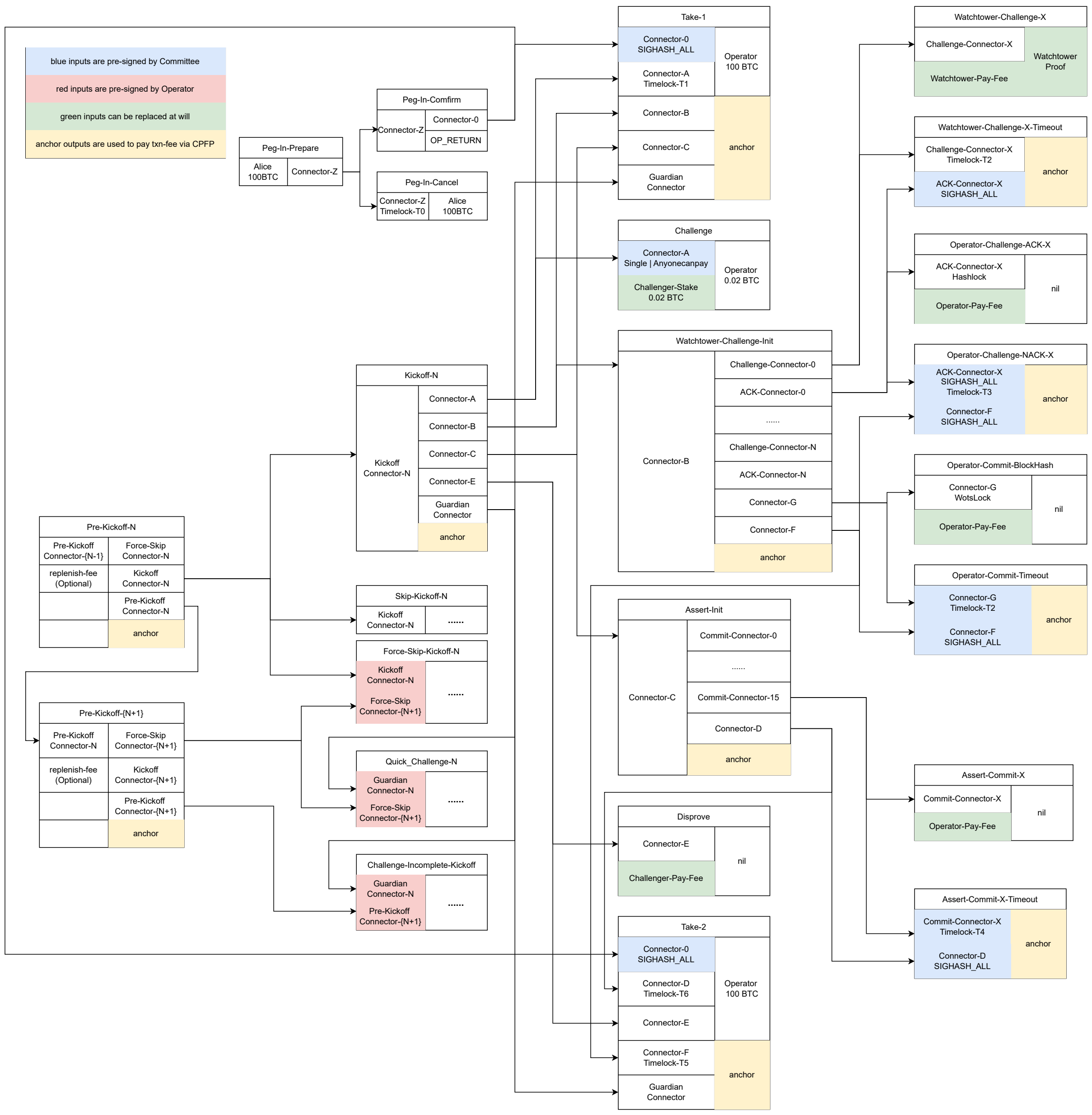


Fig. 3: GOAT BitVM2 Transaction Graph

– `Peg-In-Confirm`: Confirms the deposit operation. Subsequently, Alice receives 100 PegBTC on the L2 chain.
– `Peg-In-Cancel`: A fallback path. Alice can refund her BTC if the 'Peg-In-Conform' transaction is not confirmed in 'T0', when the timelock expires.

**Peg-out** Here, we modify the BitVM2 protocol in two important ways, 1) integrating the Watchtower mechanism, and using Watchtowers to monitor Bitcoin's longest chain; 2) shift the collateral of the operator and challenger to the L2, making it work with Atomic Swap, and incidentally ensuring that the verifier who initializes the 'Challenge' transaction is rewarded the bonus - even if another verifier submits a successful 'Disprove' transaction. Based on this fair incentive mechanism, CPFP is used for operators and verifiers to pay the transaction fee on different exit paths.

1. **Withdrawal Initiation**: When the Operator intends to withdraw BTC to L1 (after burning PegBTC on L2), they initiate the `Pre-Kickoff` and `Kickoff` transactions on Bitcoin. These transactions contain the Operator's commitment of the L2's state root.
2. **Challenge Window**: After the `Kickoff` transaction, a challenge period begins, in which challengers first submit a `Challenge` transaction to end the operator's exit from `Take1`, and then submit the fraud proof to disprove the withdrawal. A special type of challenger, namely Watchtower, should submit their Bitcoin longest chain with block number, including the sequencer set commitment of L2's consensus system, and also the corresponding state root.
3. **Exit Path**:
   – **Optimistic Path (No Challenge)**: If no Challenger disputes the withdrawal within the challenge period, the Operator successfully completes the withdrawal via the `Take1` transaction, receiving 100 BTC.
   – **Pessimistic Path (Challenged)**:
     • The challenger locks 0.02 BTC and broadcasts (`Challenge` transaction).
     • The operator must respond by publishing the `Watchtower-Challenge-Init` transaction and initialize a HTLC with each watchtower. This implies that the operator allocates each watchtower a hash lock - but keeps the pre-image at the beginning - then waits for the watchtowers to submit their witness transaction (`Watchtower-Challenge`). At least one watchtower submits it's own Bitcoin longest header chain in this transaction by a Groth16 proof in our security setting. After a timeout T4, the operator has to acknowledge the watchtower's witness by revealing the pre-image in timeout T5.
     • After the watchtower's challenge phase, the operator needs to reveal the execution trace of the ZK proof's verifier in the `Assert-Init` and `Assert-Commit`) transaction to prove its correctness.
     • After the execution trace has been revealed, any challenger is eligible to post collateral on L2, find a fraud proof in the trace, and submit a `Disprove` transaction with the fraud proof to prevent the operator's exit via `Take2`. If no challenger submits `Disprove`, the operator can proceed with the withdrawal using `Take2`.

**Optimizing the Transaction Graph** As shown in Figure 3, GOAT BitVM2 focuses on three pivotal upgrades designed explicitly to enhance the Transaction Graph's performance and reliability:

– **On-Chain Consensus Layer Verification via Watchtower Mechanism.** To anchor the system's security directly to Bitcoin, the Watchtower mechanism was introduced. This innovation involves the Watchtower mechanism on-chain that verifies the consensus of the L2 network. It ensures that the state used for proofs is based on the canonical L2 chain, effectively mitigating the risk of Operator-led double-spending attacks by leveraging Bitcoin's own security model.

– **Chain based Synchronization Primitive for Shared Collateral.** To address capital inefficiency, a Pre-Kickoff chain was implemented. This allows an Operator to post a single collateral that can be used for multiple `Peg-out` requests sequentially. As outlined in Figure 4, For operator, 1) if the Operator completes reimbursement in Graph-N, or actively skips it, they can directly enter the next Graph (i.e., broadcast Pre-Kickoff-N+1). If, upon entering the (N+1)th Graph, the Nth Kickoff has been broadcast but reimbursement is not completed, the challenger can issue a challenge via Challenger-Incomplete-Kickoff, which will result in the Operator's Graphs after N+1 all becoming unavailable. If, upon entering the (N+1)th Graph, the Nth Kickoff or SkipKickoff has not yet been broadcast, the challenger can issue a challenge via Force-Skip-Kickoff, which will result in the Operator's Nth Graph becoming unavailable.

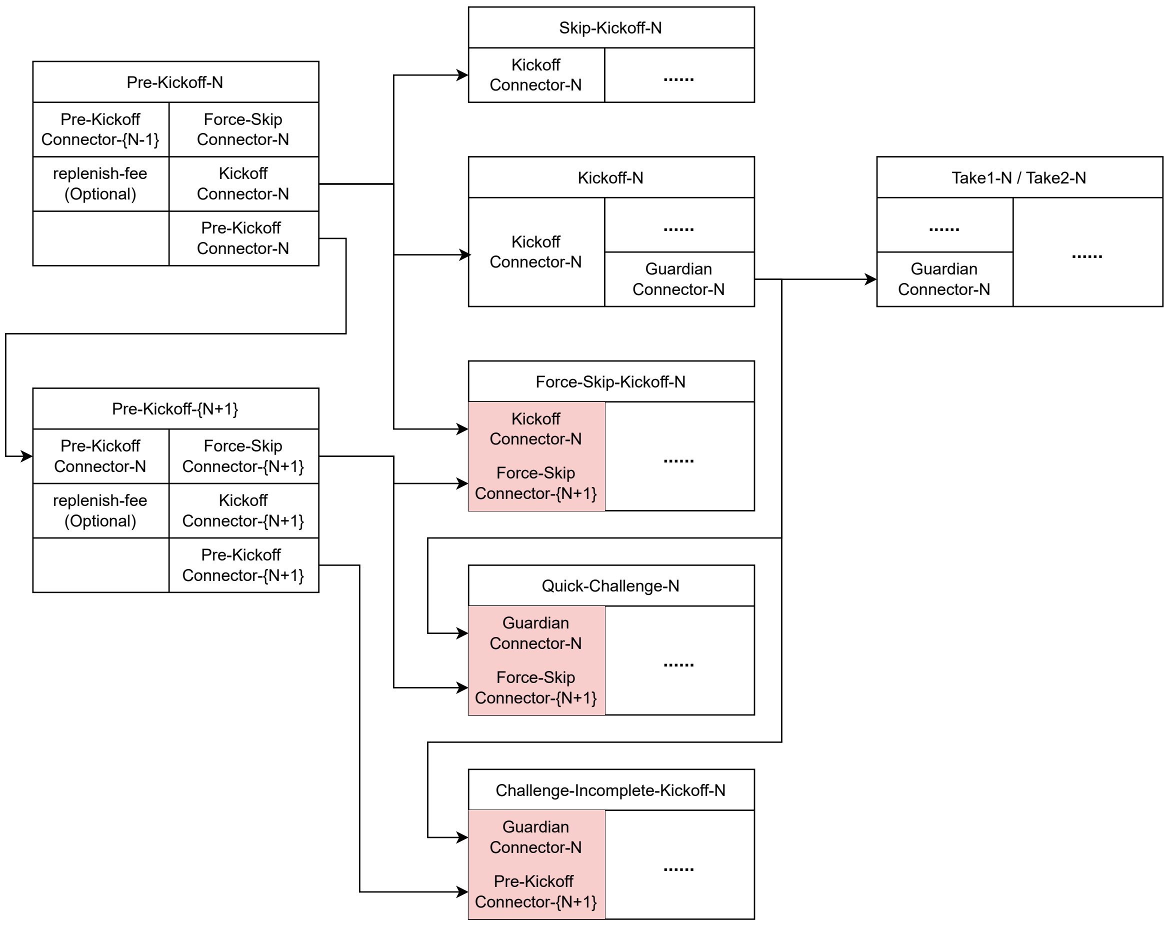


Fig. 4: Chain based Synchronization Primitive for Shared Collateral

– **Adoption of Child-Pays-For-Parent (CPFP) for Fee Payment.** To ensure the reliable confirmation of the pre-signed transactions that comprise the BitVM2 Transaction Graph—

especially during periods of Bitcoin network congestion—the protocol integrates the CPFP fee mechanism. This allows any participant to attach a child transaction with a higher fee to incentivize miners to confirm a stalled parent transaction, thereby safeguarding the liveness and progression of the challenge-response protocol.

## 3.3 Operator Locking and Slashing

In the original BitVM2 protocol, the operator locks it's collateral on L1 directly and leverages an experimental opcode OP_BLOCKHASH to check its confirmation in the longest chain, but this is not operational. Instead, GOAT BitVM2 requires the operators to lock collateral on L2, and use the Atomic Swap to pay the end-user. In this way, we can simplify the original protocol.

The Operator locking contract serves as a prerequisite for network participation and duty fulfillment, such as handling Peg-in/Peg-out operations and generating proofs. Its core objectives include providing security assurances and enabling permission authentication. The locking mechanism consists of two essential steps - facilitating the secure transition of funds from an unrestricted state to one governed by the protocol.

Based on the locking contract, the committee member can verify the operator's qualification to act as a valid operator for the user's Peg-in and have enough funds to kick off the withdrawal.

To unlock the funds from the locking contract, the operator must:

1. Submit an unlock request to the committee and provide proof that all associated `Take1/Take2` transactions have been fully processed or discarded.
   – This can be done by having the operator spend the latest `PreKickoff-connector` output via a `Non-PreKickoff` transaction, demonstrating that no further `Kickoff` transactions can be initiated.
2. The Committee verifies the proof. If valid, a sufficient number of Committee members sign to approve the request.
3. With enough Committee signatures, the unlock transaction can be submitted to the contract, and unlocks the operator's funds from the locked state back to Locking, which allows the operator to exit the locking contract.

The slashing mechanism is the core defense against malicious operator behavior. System security is not based on trusting the Operator's honesty but on a crypto-economic game theory mechanism. By making the economic cost of malicious behavior far exceed any potential gain, the slashing mechanism fundamentally discourages the Operator from acting maliciously. We need to state that as long as one Operator acts honestly, any malicious conduct by the remaining Operators will not result in loss of user assets.

## 3.4 On-chain Sequencer Set Commitment

GOAT BitVM2 runs on a decentralized sequencer network to resist liveness attacks, and publish the sequencers' public key on Bitcoin. The core idea is to use a chain of pre-signed Bitcoin transactions to ensure that any change to the L2 Sequencer set is approved by a sufficient number (e.g., 2/3) of the current members and anchored on the Bitcoin.

The commitment of the sequencer set's public key needs a trusted setup ceremony and then the publishing of the new set once it is updated in the sequencer network by opening the old commitment via Bitcoin threshold signature in Figure 5.

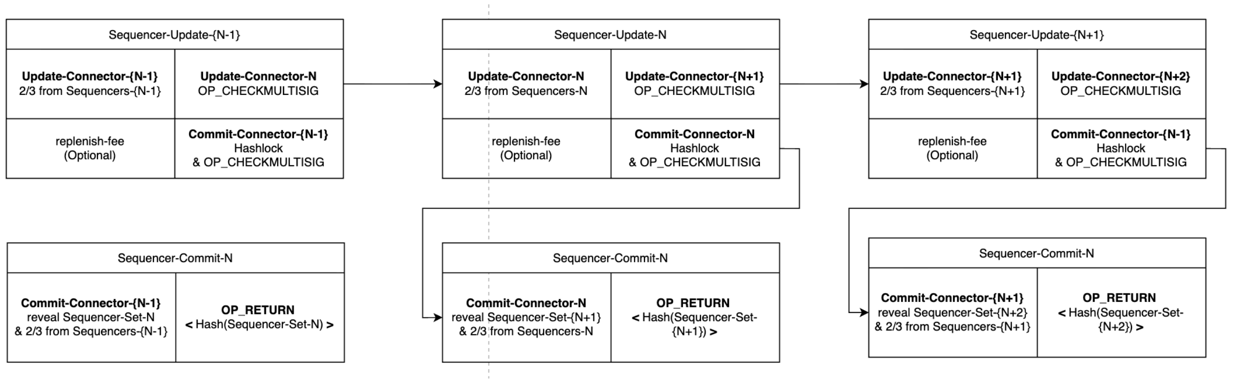


Fig. 5: Sequencer Set Update Flow

The core of the update process revolves around two key transactions: the `Sequencer-Update` transaction, which primarily triggers and organizes the update process, and the `Sequencer-Commit` transaction, which carries the hash commitment of the next cycle's (N+1) Sequencer set, signed by a majority of the sequencers.

When the sequencer set is changed in round N, all sequencers use the `Update-Connector-N` as the first input of the transaction `Sequencer-Update-N` and commit the new sequence set's hash in the transaction `Sequencer-Commit-N`. Specifically:

- Once the new Sequencer set is determined, the structures of the `Sequencer-Update-N` and `Sequencer-Commit-N` transactions are fixed, except for the UTXO used to supplement fees and the fee rate setting, which require consensus among nodes.
- A fund pool is needed to supplement/pay fees, represented by the 'replenish-fee' input.
- The locking script of the `Commit-Connector` includes a hash lock for the next round's sequencer set. In the `Validator-Commit` transaction, the pre-image of the next round's Sequencer set is revealed, and an `OP_RETURN` output is used to commit to the hash of the next round's Sequencer set for easier verification.

After collecting signatures from 2/3 of the Sequencers for both `Sequencer-Update-N` and `Sequencer-Commit-N` transactions, they are broadcast on the Bitcoin network, finalizing the sequencer set update.

### 3.5 L2 Bridge Contracts

The L2 Bridge contracts consist of a PegBTC, a CommitteeManagement, and a Gateway. PegBTC refers to Wrapped BTC, a tokenized version of Bitcoin's native cryptocurrency, BTC. PegBTC is pegged 1:1 to the value of BTC and complies with the ERC-20 token standard.

The CommitteeManagement contract is a multisig contract to manage the committee member's registration, rotation, and exit, leveraging the decentralized sequencer network to achieve the agreement.

The Gateway contract acts as the bridge to work with BitVM2 pre-signed transactions to implement the core rollup.

- PegBTC Mint. The relayers monitor the Peg-In-Confirmation transaction and submit it to Gateway contract, and the contract verifies the Peg-In-Confirmation transaction's confirmation on Bitcoin by SPV.

– Peg-out Initialization.
  - `Init-Withdraw`: An operator initiates the withdrawal process by calling this function. It locks the user's PegBTC tokens and the corresponding Peg-in record on the L2, preventing double-spending.
  - `Proceed-Withdraw`: After the initiation, this function is used to submit the final withdrawal transaction, leading to the burning of the PegBTC tokens on the L2.

– Slash and Incentive.
  - `Disprove-Withdraw`: triggers a successful L1 challenge, confiscates the operator's collateral, and rewards the challenger.
  - `Finish-Withdraw`: this function is called to finalize the process once the operator's reimbursement is successful.

## 4 Rollup Protocol

The BitVM2 protocol is widely used to build bridge protocols with certain advantages:

– 1-of-N Honesty Assumption: As long as at least one participant deletes their private key and supervises malicious actions, then fund theft can be effectively prevented. This security model significantly reduces the trust assumptions compared to models that require a majority of honest participants.

– Verifiable Deployment for Deposits: Peg-in users can verify the correctness of a specific smart contract instance before depositing any funds. This ensures that financial resources are only committed after successful validation, eliminating risk for all parties if no deposit occurs.

– Pre-defined Exit Paths: The transaction graph pre-defines all possible exit paths for the locked funds. This ensures that unauthorized withdrawals are impossible, as the pathways for moving assets are established and immutable from the outset.

These designs ensure that funds are secured by the cryptographic guarantees of the smart contract itself, rather than by a traditional, centralized third party.

BitVM2 enables the creation of covenants on Bitcoin, which involves planning and securing a set of Bitcoin transactions before any funds are moved. The detailed steps are:

1. Define the Transaction Graph: First, designers map out the entire contract's logic as a detailed flowchart of Bitcoin transactions. This "transaction graph" acts as the contract's rulebook, specifying every possible action participants can take.
2. Pre-sign the Graph: Next, this transaction graph is reviewed and approved by a designated group. Using multi-signature authority, the members of this attesting committee jointly sign the graph. This step finalizes the rules, effectively "locking in" the contract and preventing any single party from changing it later.
3. Publish the Graph: Finally, the pre-signed transaction graph is made publicly available. This ensures that all participants have access to the same unchangeable set of rules, which can be distributed by way of a decentralized storage network.

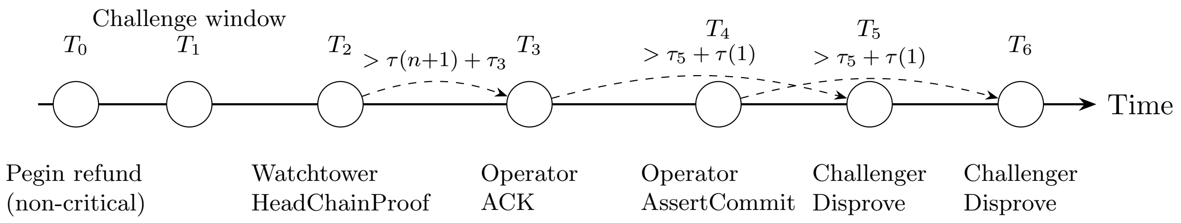
### 4.1 Deposit and Withdrawal in a Bitcoin zkRollup

The protocol combines the following:

- Transaction graphs with pre-signed spending paths.
- Timelock-enforced refund and challenge mechanisms.
- An optimistic Peg-Out design with Watchtower based dispute resolution.

**Timelocks and Global Timing Model** The protocol relies on Bitcoin timelocks to enforce safety and dispute resolution.

| Timelock | Scope | Purpose |
|---|---|---|
| $T_0$ | Peg-In | User refund via Pegin-Cancel |
| $T_1$ | Peg-Out | Delay before Take-1 |
| $T_2$ | Peg-Out | Watchtower response window |
| $T_3$ | Peg-Out | Operator ACK / NACK deadline |
| $T_4$ | Peg-Out | AssertCommit deadline |
| $T_5$ | Peg-Out | Connector-F maturity |
| $T_6$ | Peg-Out | Final Take-2 eligibility |



**Key Properties**

- All locks are relative timelocks using `OP_CSV`.
- Safety is enforced by strict ordering between $T_2$–$T_6$.
- Longer timelocks increase robustness but slow down reimbursement.

**Deposit Protocol Flow**

1. The User initiates a Peg-in request via the Layer 2 contract and broadcasts the request to the P2P network, locking the relevant UTXO.
2. Committee members validate the request and respond on-chain. Once a threshold is reached, the participating Committee set is finalized.
3. The User constructs *Pegin-Prepare*, *Pegin-Cancel*, and *Pegin-Confirm* transactions and broadcasts Pegin-Prepare.
4. The Operator verifies the transactions, constructing a transaction Graph, pre-signs required components, and sends the Graph to the Committee.
5. Committee members verify the Operator's stake on Layer 2 and pre-sign the Graph.
6. The Operator aggregates signatures, finalizes the Graph, and broadcasts it.
7. Committee members sign both the Pegin-Confirm transaction and the Graph digest.
8. A Relayer aggregates Committee signatures and broadcasts Pegin-Confirm.

9. After Bitcoin confirmation, the Pegin-Confirm transaction and SPV proof are submitted to the Layer 2 contract.
10. The contract verifies correctness and mints PegBTC to the User.

**Deposit State Machine**

$$\mathsf{Init} \xrightarrow{\text{Prepare}} \mathsf{Prepared} \xrightarrow{\text{Confirm}} \mathsf{Completed}$$

Refund path:

$$\mathsf{Prepared} \xrightarrow[T_0]{\text{Cancel}} \mathsf{Refunded}$$

Invalid behavior results in the $\mathsf{Discarded}$ state.

**Withdrawal Protocol Flow.** The withdrawal protocol enables a user to redeem *PegBTC* on Layer 2 for native BTC on the Bitcoin blockchain. The protocol is realized via a cross-chain atomic swap, combining a hash time-locked contract (HTLC) on Layer 2 with a corresponding HTLC transaction on Bitcoin.

As illustrated in Figure 6, the user first locks PegBTC on Layer 2 using a hash commitment, while the operator subsequently locks BTC on Bitcoin with the same hash. The user redeems BTC by revealing the preimage on Bitcoin, which is then observed by the operator to unlock PegBTC on Layer 2. Timeout conditions on both chains ensure atomicity and protect both parties against counterparty failure.

A basic atomic swap requires the user to manage the preimage explicitly, which may lead to a suboptimal user experience. This requirement can be eliminated by incorporating Bitcoin SPV verification, allowing the Layer 2 system to learn the preimage trustlessly without introducing additional security assumptions.

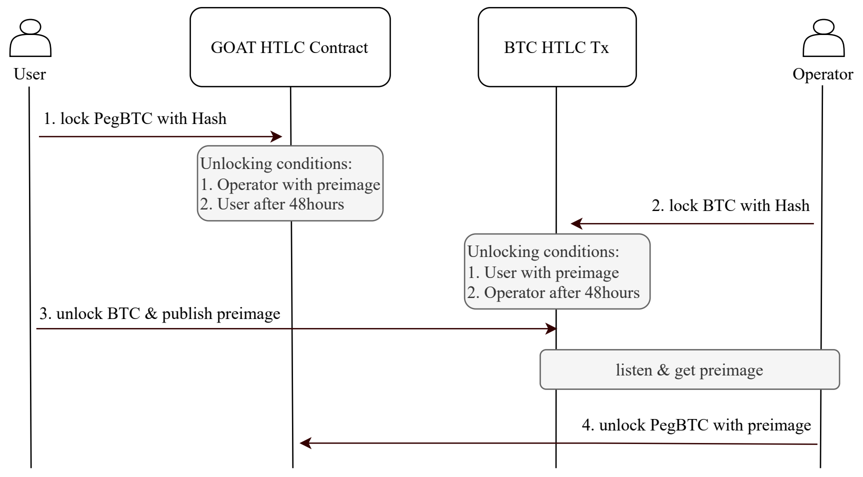


Fig. 6: Cross-chain Atomic Swap

The Peg-Out is the procedure for the operator's funds to exit from L2 to L1, and involves the following steps:

1. An **Operator** locks the corresponding amount of PegBTC in the Layer-2 contract by issuing an **Init-Withdraw** transaction. The contract records the current Bitcoin block height and locks the associated Pegin reference to prevent double-withdrawal.
2. The Operator broadcasts **Kickoff** (possibly preceded by **PreKickoff**), initializing the Bitcoin spending path with minimal fee. CPFP is used to adjust fees dynamically.
3. Relayers or the Operator calls **ProceedWithdraw** on Layer 2, submitting the Kickoff transaction and burning the locked PegBTC. The contract verifies the Bitcoin SPV proof and transaction consistency.
4. Watchtowers monitor the Bitcoin network and submit proofs if they detect misbehavior. A challenge period $T_2$ ensures that Operators cannot bypass external validation.
5. Challengers can dispute Operator behavior by broadcasting **Challenge** transactions. Operators respond via ACK/NACK transactions. Failure to respond or incorrect proofs can result in slashing.
6. After all challenges and Watchtower proofs are resolved, and all timelocks expire ($T_1$ through $T_6$), the Operator may broadcast **Take-1** and **Take-2** transactions to receive reimbursement.
7. If any challenge succeeds, the Operator's stake is confiscated and distributed to the Challenger and relayers, ensuring economic incentive alignment and safety.

**Peg-Out State Machine** We can abstract the Peg-Out protocol into the following simplified state machine:

$$\mathsf{Idle} \xrightarrow{\text{Init-Withdraw}} \mathsf{Kickoff}$$

$$\mathsf{Kickoff} \xrightarrow{\text{No Challenge}} \mathsf{Take\text{-}1/Take\text{-}2} \to \mathsf{Finalized}$$

$$\mathsf{Kickoff} \xrightarrow{\text{Challenge}} \mathsf{Watchtower\text{-}Monitoring}$$

$$\mathsf{Watchtower\text{-}Monitoring} \xrightarrow{\text{Watchtower Proof}} \mathsf{Operator\text{-}Response}$$

$$\mathsf{Operator\text{-}Response} \xrightarrow{\text{ACK / AssertCommit}} \mathsf{Verification} \quad , \quad \mathsf{Operator\text{-}Response} \xrightarrow{\text{NACK / Timeout}} \mathsf{Slashed}$$

$$\mathsf{Verification} \xrightarrow{\text{No Dispute}} \mathsf{Take\text{-}2\ /\ Finalized} \quad , \quad \mathsf{Verification} \xrightarrow{\text{Disprove}} \mathsf{Slashed}$$

This state machine emphasizes the main stages:

– **Kickoff**: Operator initiates a Bitcoin withdrawal.
– **Watchtower-Monitoring**: Watchtowers observe the Bitcoin chain and submit proofs if necessary.
– **Operator-Response**: Operator broadcasts ACK or NACK, and may commit assertions.
– **Verification**: Layer-2 contract validates Operator proofs and Watchtower submissions.
– **Take-1 / Take-2**: Successful reimbursement paths.

- **Slashed**: Penalty state for misbehavior or failed challenge response.

This abstraction hides lower-level transaction details (e.g., CPFP, Anchor outputs, Connector-F) while retaining the essential economic and security guarantees.

This protocol demonstrates that a trust-minimized Bitcoin–Layer2 bridge can be constructed using multisignature Committees, transaction graphs, and optimistic dispute resolution, while preserving strong economic incentives and cryptographic verifiability.

## 4.2 Operator Proof Generation

To ensure that an Operator can legitimately withdraw funds from the Bitcoin network (i.e., initiate a Peg-out), the Operator must prove two critical facts on the L2 chain: 1) that they possess and have burned the corresponding assets, and 2) that this burning operation is included in the correct, canonical L2 state. The validity of this L2 state itself is derived from the most recent Sequencer set commitment, which must be proven to have been legitimately updated and recorded on the Bitcoin blockchain.

The workflow for generating the definitive Operator Proof, which guarantees the legitimacy of a Peg-out, relies on the sequential verification of four core components:

- **Watchtower Proof**: Serves as a monitor for the Bitcoin's longest chain. Each watchtower proposes its own Bitcoin longest chain that includes the sequencer set commitment.
- **Bitcoin Header Chain Proof**: Establishes the canonical Bitcoin chain state. It cryptographically verifies that the transactions containing all Sequencer set updates introduced in Section 3.4 are confirmed on the longest valid Bitcoin PoW chain.
- **Commit Chain Proof**: Commit the Sequencer set updates on the Bitcoin through legal `Sequencer-Update` transaction (i.e., containing enough signatures of the previous Sequencers).
- **L2 State Chain and Asset Burn Proof**: Demonstrates that the canonical L2 state was correctly derived through legitimate transactions and provides cryptographic evidence that the Operator has successfully burned the corresponding assets within this proven state, thereby making them eligible for withdrawal on the Bitcoin network.

By chaining these proofs together, the system constructs an irrefutable cryptographic guarantee that the Operator is using the correct and agreed-upon L2 state for the Peg-out. Any attempt to use an invalid or forked state will fail the verification process, allowing a Challenger to detect the fraud, submit a dispute, and trigger a slashing penalty, thereby securing the system against malicious withdrawal attempts.

## 4.3 Data Availability

A data availability (DA) layer for zkRollups must satisfy the following properties: (i) mandatory publication of all state-transition data, (ii) public retrievability by any verifier, and (iii) consensus-level enforcement such that missing data invalidates the corresponding state update. Bitcoin fails to meet these requirements due to the absence of a programmable, stateful execution layer and native mechanisms for enforcing data publication.

Although arbitrary data can be embedded in Bitcoin transactions (e.g., via `OP_RETURN` or witness data), Bitcoin provides no means to reject state transitions when such data is unavailable. Bitcoin consensus validates only transaction correctness and proof-of-work, but does not verify data completeness or retrievability. Consequently, a zk proof may attest to the correctness of a rollup state transition even when the underlying data is unavailable, placing data availability strictly outside Bitcoin's consensus scope. Therefore, Bitcoin-based zkRollups must rely on external DA layers or adopt validity-only (validium-style) trust assumptions.

On GOAT Network, a set of decentralized sequencers forms a permissionless execution layer, where each sequencer maintains a full replica of the Rollup state and collectively guarantees system liveness. All Layer 2 state transitions are recorded in an *L2 State Chain*, whose virtual blocks encode incremental state differences. These state differences are accumulated until reaching a pre-defined size threshold, after which they are published to the Bitcoin blockchain as data commitments.

Using the Watchtower mechanism in conjunction with Bitcoin header chain proofs, the system verifies that the data publication transaction has been confirmed on Bitcoin, thereby providing a verifiable and censorship-resistant record of Layer 2 state evolution.

## 5 Optimizing GOAT BitVM2 with Garbled Circuit and DV-SNARK

### 5.1 Motivation and Core Component

Garbeled Circuits (GC) enables an Evaluator to perform computations on encrypted data (known as *labels*), with visibility restricted solely to the inputs supplied by the Garbler and the resulting labels of intermediate and final values. This cryptographic primitive facilitates verifiable computation on the Bitcoin network. In the proposed design, the Operator (acting as the Garbler) constructs a GC that represents a ZKP verifier circuit. The Operator commits to a set of all possible input and output labels. During the verification phase, if a proof is invalid, revealing the corresponding labels results in a 0-output label. This specific value can then be used on-chain to `Disprove` the malicious Operator's claim and trigger a penalty.

By streamlining the `Disprove` process, this approach significantly reduces the complexity of the dispute procedure and its on-chain cost, making BitVM2 practically feasible.

The integration of a Designated Verifier substantially reduces the complexity of the underlying verification circuit, resulting in a GC that is more efficient to construct and verify.

The integration of GC into the BitVM2 framework primarily involves three phases: `Peg-in`, `Assert`, and `Disprove`. The latter two phases are activated during the `Peg-out` process when a challenge is presented.

The detailed workflow is structured as follows:

– **Phase 1: Peg-in**
    - The Designated Verifier generates the Structured Reference String (SRS) while holding the trapdoor (secret values). The Operator verifies the correctness of the SRS generation using a pairing-based check for the pairing-friendly curve *BN254*. For curves that do not support efficient pairings, such as *Sect233k1*, a combination of the Cut & Choose technique and a random challenge mechanism is employed to attest to the validity of the SRS (see Section 5.2 for details).
    - After verifying the correctness of the SRS, the Operator generates a GC based on the DV-SNARK verifier circuit, publishes this GC off-chain, and uses a ZKP to prove that the GC was constructed correctly.
    - The Operator commits to the input and output labels of the GC on-chain. The Verifier commits the trapdoor on-chain.

– **Phase 2: Assertion after A Challenge**

  - The Operator has already published a SNARK proof, asserting that the off-chain state transition is valid. Once challenged, the Operator reveals the corresponding input labels excluding those related to the Verifier's trapdoor on-chain.
  - The Designated Verifier responds by disclosing the trapdoor on-chain.
  - The Operator then publishes the input labels corresponding to the revealed trapdoor.
- **Phase 3: Disprove**
  - If the SNARK proof or the circuit execution is invalid, any Challenger will:
    - Evaluate the garbled circuit using all the published input labels.
    - Derive the specific output label representing a logical '0' (indicating failure).
    - Submit this '0'-output label on-chain as a fraud proof to invalidate the Operator's `Kick-off`.

The architectural shift necessitates substantial engineering changes in both on-chain and off-chain components to support the GC-based paradigm:

- Restructuring On-Chain Verification Scripts. The on-chain script logic must be rewritten to handle GC-based verification instead of multi-round interactions. The scripts now focus on verifying cryptographic commitments and GC output correctness. For example, scripts will heavily rely on hash functions like `OP_SHA256` to validate that the submitted GC data matches the committed hash.
- Off-Chain Storage Architecture. With the computational burden moved off-chain, managing the large amount of GC circuit data becomes critical. A robust off-chain storage and distribution network, such as IPFS, must be integrated to ensure that verifiers can reliably access the complete GC data when needed.
- Off-Chain Computation and Proving Systems. A high-performance proof generation system (i.e., Ziren) is essential. This involves proving the validity of withdraw operations and the correctness of GC generation.

To ensure security, the protocol requires at least one honest Designated Verifier. During the `Peg-in` setup, $n$ (i.e., 10) verifiers are selected, and the GC's are produced in $n$ independent copies. The system guarantees that if at least one of the verifiers is honest and performs correctly, it can prevent a malicious Operator from successfully executing an illegal BTC withdrawal.

### 5.2 SRS Verification

The DV-SNARK-based BitVM2 scheme significantly reduces the size of GC and decreases off-chain communication overhead. When instantiated with the pairing-friendly *BN254* curve, the GC size can be reduced by approximately five times. In contrast, using the *Sect233k1* curve—which does not natively support efficient pairings—can achieve a reduction of about two orders of magnitude, compressing the GC from hundreds of gigabytes to just a few gigabytes compared to traditional SNARKs that rely on pairing verification.

For the *BN254* curve, the pairing property can be leveraged to directly verify the correctness of the SRS [6]. However, using *Sect233k1* introduces new challenges for SRS verification, as alternative methods—such as interactive proofs or circuit-based checks—must be employed in the absence of efficient pairings.

**SRS Correctness Burden for *Sect233k1*.** Each Designated Verifier must generate an individual SRS (no universal reference string exists), requiring rigorous validation. When using non-pairing-friendly elliptic curves such as *Sect233k1*, verifying SRS correctness becomes computationally intensive - often exceeding the cost of GC verification itself when using methods like non-interactive zero-knowledge (NIZK) proofs.

**Combining Cut & Choose and Randomized Challenge Solution.** Instead of direct SRS verification, the protocol employs a probabilistic game to ensure SRS reliability under *Sect233k1* using parameters $(n, m, k)$:

– Cut-and-Choose Procedure:
  1. The Designated Verifier generates $n$ SRS instances and commits to each (e.g., via short digests of the SRS and associated trapdoor).
  2. The Prover randomly selects $n - m$ instances and requests the Verifier to reveal the corresponding trapdoors.
  3. The Prover recomputes the SRS using the revealed trapdoors and verifies consistency with the commitments. If inconsistencies are detected, the protocol terminates.

– Random Challenge Procedure (applied to the remaining $m$ SRS instances):
  1. The Verifier sends the full SRS data for the $m$ instances to the Prover.
  2. The Prover generates $k$ random proofs (using random public inputs) for each SRS and sends them to the Verifier.
  3. The Verifier uses a NIZK proof to demonstrate that these proofs pass the verification circuit associated with the committed trapdoor.

– On-Chain Deployment: The $m$ GC circuits corresponding to the $m$ trapdoors are deployed on-chain. Verification results are combined using an **OR** logic, allowing the Prover to withdraw funds if **at least one** GC circuit validates successfully. But it's still necessary to avoid the collusion between operators and designated verifiers, like doing a random selection during the peg-in pre-signing phase off-chain, or in the kickoff transaction.

With parameters $n = 20$, $m = 2$, and $k = 10$, the probability of an honest Porver fails to pass the DV-SNARK verification circuit in face of a malicious Verifier is bounded by:

$$\frac{2}{n(n-1)} \cdot (1-\alpha)^n \alpha^2 \leq \frac{1}{190} \cdot \left(\frac{10}{11}\right)^{20} \cdot \left(\frac{1}{11}\right)^2 \approx 6.5 \times 10^{-6}.$$

## 5.3 Parameters Used in GC-optimized BitVM2 Construction

The cryptographic parameters for BitVM2 with DV-SNARK are configured as follows:

– Label Width: 128 bits (Security against brute-force attacks).

– Poseidon2 Hash Configuration (adopted for GC circuit proving efficiency, the 32-byte result is truncated to the higher-order 16 bytes):
  - Base Field: KoalarBear prime field $\mathcal{F}_p$ with $p = 2^{31} - 2^{24} + 1$.
  - Rate: 8.
  - Permutation width: 16.
  - Round Structure: 8 for external rounds and 13 for internal rounds.

– Pairing-friendly Elliptic Curve *BN254* for efficient SRS verification.

– Or, Efficient Elliptic Curve for Boolean circuits:
  - Curve: $E_{k233}/\mathcal{F}_{2^{233}} : y^2 + x \cdot y = x^3 + 1$.

Table 2: Complexity comparison of the DV-SNARK verifier GC.

| **Metric** | **BN254** | **Secp256k1** |
|---|---|---|
| Non-free gates | $5.06 \times 10^8$ | $8.8 \times 10^6$ |
| GC communication | 7.9 GB | 141 MB |
| ZKP hash operations | $1.6 \times 10^9$ | $1.76 \times 10^7$ |
| SRS size | 863 MB | |
| On-chain setup | $\approx$ 84 KB | |
| On-chain assert | $\approx$ 42 KB | |
| On-chain disprove | $\approx$ 16 B | |

- Base Field: $\mathcal{F}_{2^{233}} = \mathcal{F}_2[X]/(X^{233} + X^{74} + 1)$.

**Complexity Analysis** The DV-SNARK verifier circuit based on the `Secp256k1` curve comprises approximately **8.8 million non-free gates**. Our analysis considers a *single* garbled circuit (GC); in practice, $n$ distinct GC circuits are required for $n$ verifiers.

*On-chain Complexity.* The total size of on-chain inputs is **268 bytes**, including the SNARK proof, public inputs, and trapdoor values. We employ *Lamport signatures* with a **160-bit hash output** to commit to all input bits and the output bit. For each bit, two **128-bit labels** are generated as secret keys.

The resulting on-chain communication overhead consists of: (i) a *setup* phase for committing to all input bits, (ii) an *assert* phase revealing one label per input bit, and (iii) a *disprove* phase revealing only the output label. The concrete costs are summarized in Table 2.

*Off-chain Complexity.* Off-chain costs include both communication and computation overheads arising from garbled circuit transmission, structured reference string (SRS) distribution, and zero-knowledge proof generation. A detailed comparison between BN254 and `Secp256k1` is also shown in Table 2.

## 5.4 Performance Testing

The primary computations within the process are concentrated in the Bridge-In phase. The Assertion in the Reimbursement Process only involves a proving for the DV-SNARK circuit, and the Disprove in the Reimbursement Process only requires evaluating the GC. The key computational steps are illustrated in the Fig. 7 .

The process begins with a DV-SNARK circuit (designed for verifying off-chain computations) and a verifier's Boolean circuit. First, the designated verifier generates the SRS and must subsequently prove its correctness or ensure its validity via a specific protocol. Then, the prover constructs the GC and must prove the correctness of the GC.

During the Assertion phase, the prover generates a proof and discloses the input labels corresponding to both the proof and the trapdoor (later revealed by the Designated verifier). If the proof

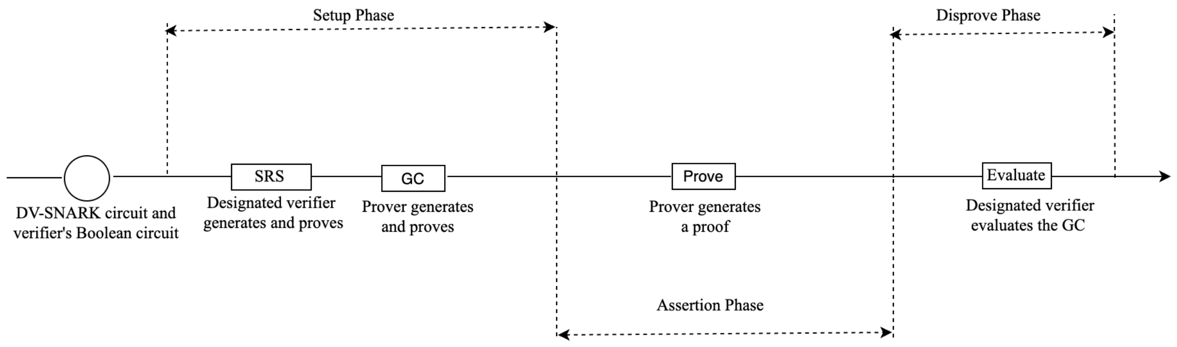

Setup Phase
Disprove Phase
SRS
GC
Prove
Evaluate
DV-SNARK circuit and verifier's Boolean circuit
Designated verifier generates and proves
Prover generates and proves
Prover generates a proof
Designated verifier evaluates the GC
Assertion Phase

Fig. 7: Key Computational Steps in the Overall Workflow

is invalid, the designated verifier (or any other Challenger) can initiate a Disprove by evaluating the GC circuit.

The measured execution times for these major computational steps are summarized in the accompanying hackmd doc. With the exception of the GC proving step, all other processes can be completed quickly (within minutes) on a 16-core AMD EPYC 7R13 processor. The computationally intensive GC proving is accelerated by 8 NVIDIA RTX 4090 GPUs. The total number of Boolean gates is 2.0/2.4 billions, with 506/8.8 millions non-free (AND) gates for *BN254*/*Sect233k1*, and The GC is transformed into about 344/66 billion cycles respectively. Currently, with an 8-GPU configuration, the Ziren system can prove approximately 24 million cycles per second. This performance translates to a total GC proof generation time of about 4 hours for the *BN254* curve and a much faster 0.8 hours for the *Sect233k1* curve, demonstrating significant potential for real-world deployment. Furthermore, since we can partition the total circuit into smaller independent sub-circuits, the entire proving process can be substantially accelerated through parallel sub-circuit execution with additional GPU scaling.

## 6 Conclusion

GOAT BitVM2 significantly advances practical zkRollups on Bitcoin by systematically extending the canonical BitVM2 protocol.

First, GOAT BitVM2 generalizes the original BitVM2 construction for building native Bitcoin zkRollups. This extension integrates an Atomic Swap that supports arbitrary withdrawal amounts, together with a chain-based synchronization primitive that enables Rollup operators to share collateral across multiple reimbursements in a secure and capital-efficient manner.

Second, building upon the extended BitVM2 protocol, GOAT BitVM2 introduces on-chain consensus layer verification using a watchtower mechanism and a decentralized sequencer architecture, materially enhancing the Rollup's liveness and decentralization. Furthermore, the Universal Operator Abstraction formalizes the allocation of revenue and risk among the participating roles, thereby synergistically incentivizing broader participation by challengers and strengthening the overall security of the network.

Third, the GOAT BitVM2 implementation demonstrates significant performance improvements that effectively bridge the gap between theoretical design and practical application. By integrating with the Ziren proof system, Operator reimbursement proofs can now be generated in approximately

40 seconds - a substantial reduction that makes frequent on-chain verification feasible. Furthermore, current GC proof generation times (4 hours for the SRS verification-friendly curve *BN254* and 0.8 hours for curve *Sect233k1* using a composed verification method) demonstrate the practical viability of our approach.

GOAT BitVM2 establishes a solid foundation for the continued evolution of Bitcoin Layer-2 solutions. Future work will focus on strengthening the robustness and formal verification of the optimistic computation model, optimizing SNARK verifier's Garbled Circuit[4] and the corresponding proof generation to achieve sub-minute latency, enabling real-time reimbursement proof generation, and substantially reducing operators' reimbursement costs.

By demonstrating a fully functional implementation that effectively balances cryptographic security with practical performance constraints, this work contributes to the broader ecosystem of trust-minimized Bitcoin scaling solutions.

## References

1. Ekrem Bal, Lukas Aumayr, Atacan İyidoğan, Giulia Scaffino, Hakan Karakuş, Cengiz Eray Aslan, and Orfeas Stefanos Thyfronitis Litos. Clementine: A collateral-efficient, trust-minimized, and scalable bitcoin bridge. *Cryptology ePrint Archive*, 2025.
2. V. Buterin. Ethereum: A next-generation smart contract and decentralized application platform. *URL: https://github.com/ethereum/wiki/wiki/White-Paper*, 2014.
3. L. Eagen. Glock: Garbled locks for bitcoin. *https://cdn.prod.website-files.com/67cfca80708eb505376820af/68a3e174eaff71d197ac4080$_g$lock.pdf*, 2025.
4. Liam Eagen and Ying Tong Lai. Argo MAC: Garbling with elliptic curve MACs. Cryptology ePrint Archive, Paper 2026/049, 2026.
5. S. Zahur et al. Two halves make a whole reducing data transfer in garbled circuits using half gates. *https://eprint.iacr.org/2014/756.pdf*, 2015.
6. Sean Bowe et al. A multi-party protocol for constructing the public parameters of the pinocchio zk-snark. *https://eprint.iacr.org/2017/602.pdf*, 2017.
7. J. Groth. On the size of pairing-based non-interactive arguments. *In Advances in Cryptology - EUROCRYPT 2016*, 2016.
8. D. Heath. Efficient arithmetic in garbled circuits. *https://eprint.iacr.org/2024/139.pdf*, 2024.
9. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free xor gates and applications. *https://www.cs.toronto.edu/vlad/papers/XOR ICALP08.pdf*, 2008.
10. Leslie Lamport. Lamport signature - short private key. 1979.
11. R. Linus. Bitvm 3s – garbled circuits for efficient computation on bitcoin. *https://bitvm.org/bitvm3.pdf*, 2025.
12. Robin Linus. Bitvm: Compute anything on bitcoin. *URL: https://bitvm. org/bitvm. pdf-(12.12. 2023)*, 2023.
13. Robin Linus, Lukas Aumayr, Alexei Zamyatin, Andrea Pelosi, Zeta Avarikioti, and Matteo Maffei. Bitvm2: Bridging bitcoin to second layers. 2024.
14. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
15. Michele Orr'u. Revisiting keyed-verification anonymous credentials. *https://eprint.iacr.org/2024/1552.pdf*, 2024.
16. J. Poon and V. Buterin. Plasma: Scalable autonomous smart contracts. *URL: https://plasma.io/plasma-deprecated. pdf*, 2017.
17. J. Rubin. Delbrag. *https://rubin.io/public/pdfs/delbrag.pdf*, 2025.
18. ZKM Team. zkmips: Universal zero-knowledge virtual machine on mips32r2 isa. 2023.
19. A. C. Yao. Protocols for secure computations. *https://research.cs.wisc.edu/areas/sec/yao1982-ocr.pdf*, 1982.