GOAT BitVM2 White Paper

GOAT Network Research Group

Email: contact@goat.network

May. 2025 - Draft Version 0.3

Abstract. The BitVM2 protocol has been adopted by many Bitcoin Layer 2 solutions, but it still faces several practical challenges in building zkRollups. The GOAT Network team has proposed the GOAT BitVM2, which, with the 1-of-n honest assumption, introduces a *multi-round randomized challenge mechanism* and a *sequencer set commitment scheme* to address the open problems in BitVM2, namely, the operator's Double-spending attack, the inability to support withdrawals of arbitrary amounts, inefficient challenge selection, and the absence of a systematic incentive model.
By introducing zkMIPS, we generate the proof for each L2 block, and aggregate all the block proofs on demand for the operator when it kicks off a reimbursement. This can reduce the time for an operator's proof generation to about 40 seconds.
By introducing *Universal Operator*, an abstract role of BitVM2 operator, BitVM2 challenger and decentralized sequencer, this can balance the risks and benefits of different specific roles, and attract more entities to join the optimistic challenge process. This achieves a robust systematic incentive model and enhances the soundness of BitVM2 protocol.
These enhancements are designed to strengthen the reliability and scalability of the protocol, thus promoting the broader adoption of BitVM2 in the Bitcoin Layer 2 ecosystem.

Keywords: GOAT Network, BitVM2, zkMIPS, zkRollup, Cross-chain Bridge

1 Introduction

BitVM[1] and BitVM2[2], proposed by Robin Linus et al., establish a trust-minimized bridge protocol for Bitcoin [3], enabling its sidesystems through a fraud-proof arbitration mechanism without a hard fork under 1-of-n honesty assumption[1], using presigned transactions, one-time signatures, and SNARK proofs.

Serving as the cryptographic foundation for cross-chain bridges and zkRollup, the original BitVM2 protocol achieves two key breakthroughs: 1) Maintaining Bitcoin's base layer integrity without protocol forks under 1-of-n honesty assumption; 2) Compared to BitVM, BitVM2 allows anyone to challenge and slash a faulty operator with 3 on-chain transactions, with a delay of no more than 2-3 weeks, thus enabling a relatively capital-efficient improvement.

Despite its adoption by many Bitcoin L2 projects, BitVM2 faces some critical challenges in practical zkRollup implementations. We illustrate these challengers from the key roles perspective below.

Operator. The operator's double-spending attack, an operator can fork the L2 blockchain, generate zk-proofs for the fork, and manipulate computation proofs for double-spend attacks. For any honest operator, it needs to generate a proof for each reimbursement, and the peg-in to kick-off duration

[1] 1-of-n honesty assuption: for m operators, only one operator is required to keep liveness; for n challengers, only one challenger is required to behave honestly, namely a honest challenger should be obliged to challege the a dishonest operator's reimbursement

may last for serveral months, and makes the operator cannot generate a validity proof in short time.

Challenger. An inefficient challenge process requires about 2-week dispute periods, the operator and the challenger need to lock their asset on Bitcoin during the period; a worse case is that the initial challenger and disprove executor are probably not the same entity, and this may crash the challenger's spirit to dispute the malicious reimbursement.

Our work aims to address these challenges through three architectural innovations on BitVM2 and for BitVM2, especially for Bitcoin zkRollups, such as GOAT Netowrk[2].

Decentralized Sequencer Commitment Scheme in Bitcoin. BitVM2 assumes that the sidesystem is trusted. Observe two facts, 1) a sidesystem should have its own consensus mechanism to maintain its liveness and security; 2) the sidesystem should be able to release its verifiable logic to users, and users can decide if choosing the verified logic to create a covenant for the follow-up presigning or peg-out. However, there is an obvious gap for Bitcoin and sidesystems using BitVM2 bridge: we cannot trust the operator's public inputs[3], and we cannot commit all public inputs when users deposit assets to sidesystem. We introduce the Decentralized Sequencer Commitment Scheme to commit the public inputs between user's *Peg-in* and operator's *Kick-off* and enforce that public inputs are used in Assert transaction. Instead of computing f(w) = y, we need to compute f(x, w) = y, where x is the public input.

Universal Operator Abstraction. There are three main roles in BitVM2, Operator, Challenger, and Committee. For sidesystems, especially zkRollups, there is at least one sequencer, who is employed to pack the transactions into a block, execute the block, and seal the block. For all above four roles, we introduce the Universal Operator to play all four critical roles. This can balance the risk and benefits of all the entities and make the entire system more stable and robust.

Multi-round Randomized Challenge Mechanism. Based on Universal Operator Abstraction with the constraints of 1-of-n honestly assumption, the traditional BFT consensus algorithm does not work. We introduce a cryptographic sortition to allow all the challengers to generate a random and publish the random, and we take some polices to choose a unique challenger to verify the operator's reimbursement and decide if any challenge is necessary. If this challenger does not do anything, we play another round, and make sure enough rounds should be finished to ensure at least one honest challenger is elected. We do the same for the Disprove transaction. This randomized selection can reduce the challenge-response game from 1-2 weeks to 1 day with high confidence.

2 Background and Building Blocks

2.1 zkRollup

A zkRollup (Zero-Knowledge Rollup) is a layer-2 scaling solution that enhances blockchain scalability by moving computation and state off-chain while storing transaction data on-chain.

[2] GOAT Network: empowering a decentralized economy with sustainable yield, robust security, and a thriving ecosystem, https://goat.network

[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

An essential requirement of a zkRollup is that the L1 acts as a settlement layer and is able to verify the L2's transaction. A zkRollup should allow any user to withdraw their asset from L2 to L1 anytime.

In the context of Bitcoin and BitVM2, a zkRollup is still possible. BitVM2 enables the optimistic verification of arbitrary computations, and this computation is the verification of a SNARK Proof, such as Groth16, or Fflonk specifically. This stack is beyond the traditional zkRollup, however, it is still a zkRollup architecture from our perspective, 1) All the L2's states should be published to Bitcoin; 2) all the computation can be committed by zk-proofs, and Bitcoin can verify the proofs by BitVM2.

We illustrate a basic zkRollup workflow in Fig.1.

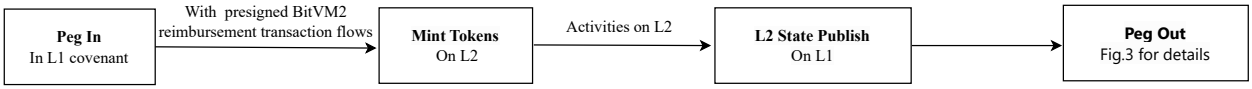

Fig. 1: zkRollup with optimistic fraud proof

Peg In. A user deposits Bitcoin by generating a BitVM2 transaction flow, namely the Bitcoin covenant, and L2 nodes monitor this deposit and generate a Mint event on the L2 network. Finally, the user gets the WrappedBTC on L2.

Transaction ordering and block building. A sequencer orders L2 transactions and constructs the corresponding L2 block, broadcasting it to other nodes to achieve consensus on the next block.

State difference publishing. The sequencer periodically publishes the state differences from L2 transaction execution on L1.

Peg Out. When the user withdraws WrappedBTC on L2 to Bitcoin, it starts a reimbursement process, burns the equivalent WrappedBTC on L2, and triggers the BitVM2 covenant on L1, which is described in Fig. 3.

In summary, any zkRollup that uses the BitVM2 protocol must address the following three baselines: *Liveness*, *Security*, and *Effectiveness*.

Liveness. Use a consensus protocol to ensure the liveness of the BitVM2 operator, challenger, committee, and sidesystem sequencer. The protocol should follow up the BitVM2's honesty assumption.

Security. Achieve BTC-level finality by generating zero-knowledge proofs of off-chain computations and verifying them on Bitcoin. 1) *L2 Computation Verifiability:* Ensure that any off-chain computation can be verified, that is, both the execution of all L2 blocks and their consensus system should be validated on L1. 2) *Available Escape Hatch:* In extreme situations, all users must be able to safely withdraw their assets under any condition.

Effectiveness. All participants' behaviors must be efficient and sustainable. Through effective incentive and penalty mechanisms, the system must ensure that at least one honest participant can participate in every critical step, allowing the system to operate continuously and sustainably over the long term.

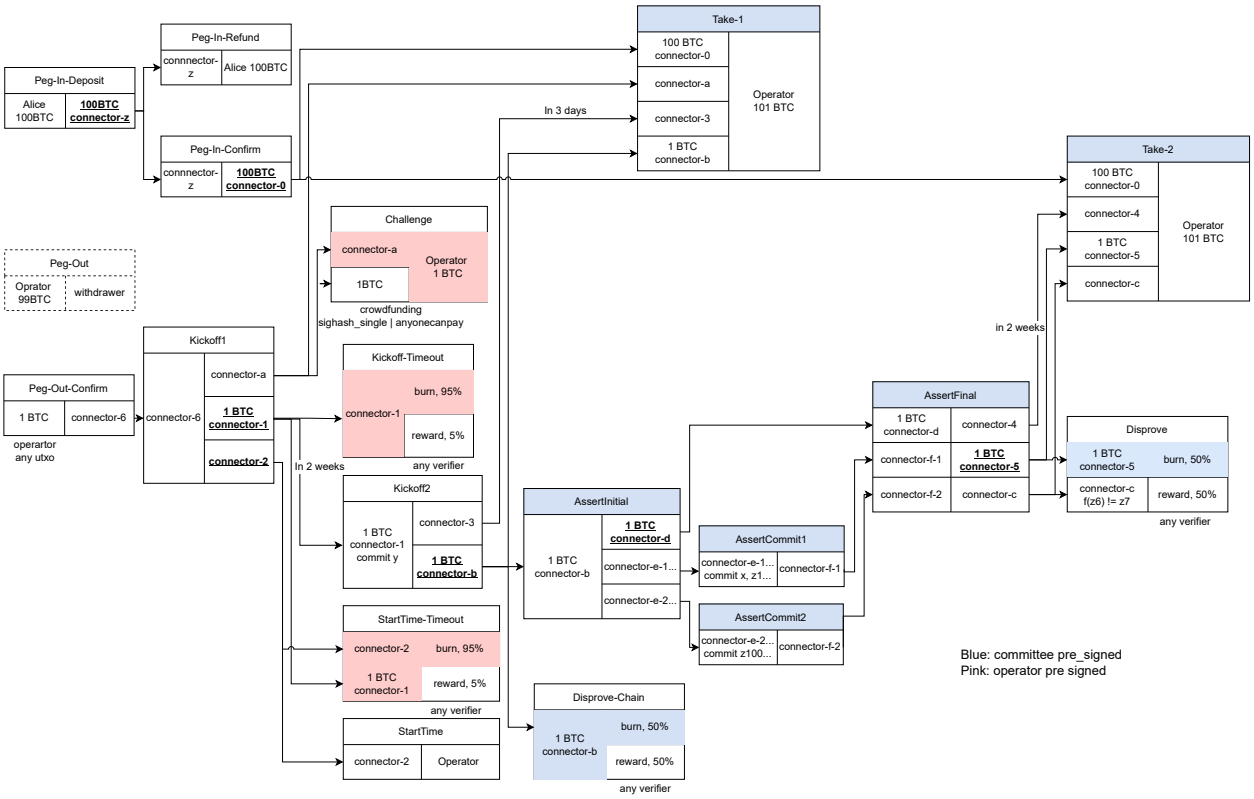

Fig. 2: Original BitVM2 Bridge Protocol

2.2 BitVM2 Protocol Overview

As shown in Fig.2, BitVM2 uses the presigned transaction flow to achieve logic persistence, and uses the one-time signature [4] to implement storage persistence. In combination with timelock and Taproot, BitVM2 can be used to implement trust-minimized bridge protocol between Bitcoin and other L2.

The overview of BitVM2 protocol is as follows:

1. Users initiate a *Peg-In-Deposit* transaction. The operator must promptly initiate a *Peg-In-Confirm* transaction to respond to the user's action, and the L2 network will mint the corresponding assets. If the operator does not respond in time, after a certain period, users can initiate a *Peg-In-Refund* transaction to redeem their locked L1 assets.
2. When users burn L2 assets, the Operator initiates a *Peg-Out* transaction on L1 to advance payment to the user. The Operator then initiates a *Peg-Out-Confirm* transaction to ensure successful reimbursement and follows with a *Kickoff1* transaction, immediately publishing a *StartTime* transaction. If the *StartTime* transaction isn't published promptly, any sequencer can issue a *StartTime-Timeout* transaction to terminate *Kickoff1*.
3. After two weeks, the operator can initiate a *Kickoff2* transaction. If the Challenger and Operator observe different *SuperBlocks*, the Challenger can initiate a *Disprove-Chain* transaction to contest the behavior.

4. If the zk-proof is valid and no challenge occurs within three days, the operator can submit a *Take1* transaction to get reimbursed, following the optimistic path, namely the happy path.
5. If the challenger finds the zk-proof invalid, they can initiate a *Challenge* transaction, and the operator must respond with an *AssertInitial* transaction. Subsequently, *AssertCommit1* and *AssertCommit2* transactions are published, followed by an *AssertFinal* transaction, namely the unhappy path.
6. If the operator behaves maliciously, the challenger can submit a *Disprove* transaction based on the ambiguity commitment to prevent the operator's malicious act. If no *Disprove* transaction is submitted within two weeks, the operator can issue a *Take2* transaction to be reimbursed successfully.

2.3 zkVM and zkMIPS

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 without revealing sensitive data. Key components include *zkCompiler*, *Prover* and *Verifier*.

zkCompiler: Compile high-level code into ISA-specific binaries (e.g. MIPS, RISC-V), generate execution traces, and convert the traces into polynomials for constraint satisfaction check.

Prover: Commit the polynomials, and generates zk-proofs. Compared to native execution, the Zero-knowledge proving is slower by 100-1000x nowadays. Hardware acceleration, Continuation[4], and pipelined proving are exploited to reduce computing overhead and latency.

Verifier: A program to verify the zk-proof, written by a smart contract or Bitcoin covenant, ensuring trustless verification.

Compared to specialized zkEVMs (which are EVM-equivalent), zkVMs are architecture-agnostic and support a broader range of applications, including zkRollups, cross-chain interoperability, and verifiable AI.

zkMIPS [5] 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 ZKM[5], it optimizes zero-knowledge proof generation through efficient zkCompiler, pipelined proof architecture, and cutting-edge proof system, ensuring high instruction proof efficiency and reduced audit requirements.

For GOAT BitVM2, we generate the proof for each block using zkMIPS' Real-time proving initiative on Ethproofs[6], and aggregate all block proofs on demand for the operator when it kicks off a reimbursement.

Formally, for any block i, p_i is the time to generate a proof for this block, C is the constant time to aggregate any two block proofs. Let $a(i, j)$ be the time to generate an aggregated proof

[4] Continuation: https://docs.zkm.io/design/continuation.html
[5] ZKM: https://zkm.io
[6] Ethproofs: Progressing towards fully SNARKing the L1, https://ethproofs.org/

from block i to j, $i \leq j$. We aim to minimize the total proving time $a(0, m)$, where m is the block that includes the operator's burning transaction, by formula 2.3:

$$a(i,j) = \begin{cases} p_i, & if\ i = j \\ max\{a(i, j-1), p_j\} + C, & if\ i < j <= m \end{cases}$$

where we consider the fact that most L2s have a very short blocktime, e.g. 3s, when a new block produced, the previous block may not be proven, but must be executed by the ISA emulator(the execution is extramly fast, recall that the proving is about 100-1000x slower than the native execution), and once a block is executed, all the chunks can be dispatched to a cluster-level in-parallel proving. Finally, we wrap the aggregated proof into a SNARK proof, which requires another constant time D.

From our experiment with GPU acceleration enabled, C is about 10s, D is about 5s, and $max\{p_i\}$ is about 25s for a block with 100 transactions, the total time to generate a proof for the operator would be about 40s. This makes GOAT BitVM2 practical to serve as a Bitcoin zkRollup.

3 Model Design and Assumptions

3.1 Open Problems in BitVM2 and Bitcoin zkRollup

From our observations, there are some open problems when we use BitVM2 to build a zkRollup on Bitcoin.

Operator Double-spending Attack. Currently, in the BitVM2 protocol, the user commits the zk-proof verifier (Groth16 verifier specifically) circuit when they peg in, and the operator publishes the proof and output y to spend the pegged Bitcoin. However, y may represent a computation result that is verifiable but incorrect. For example, the sidesystem might experience a fork, and the forked chain could still be used to produce valid proofs, thus the operator could complete a withdrawal and carry out a double-spending attack. The root cause of this attack comes from the truth that public inputs can not be determined in the *Peg-in* transaction, and the operator's commitment y can be manufactured by the above fork attack.

Inefficient Reimbursement. A single stake corresponds to a single *Kickoff* and a separate challenge, causing large presigning computation burdens for Operators and Committees during multiple *Kickoff* events.

Unable to support arbitrary-amount *Peg-Out.* if a user's asset on the sidesystem cannot match a *Peg-In* transaction, they cannot exit BTC from L2 to L1.

Moreover, the optimistic challenge periods are long, leading to low capital efficiency for the Operator. The reasons for this include: 1) a lack of an efficient and secure consensus mechanism among challengers; 2) inefficient proof generation, most of the zkVMs nowadays are not able to produce real-time proofs when an operator kicks off a reimbursement.

Lack of Incentive Mechanism. BitVM2 lacks a reasonable incentive mechanism for real-world systems. This may lead to some critical security problems if there are not enough challengers or operators to join the network. For example, if no fraud happens for a long time, the challenger can not get enough incentive and may choose to quit the protocol.

A special issue is that challengers may not receive rewards. When crowdfunding for the *Challenge* transaction, the initial challenger may not end up being the one who submits the final *Disprove* (usually a Bitcoin miner will apply a front-running attack to get the rewards), which means that the staked funds might not be correctly allocated to the honest challenger.

4 GOAT BitVM2 Design

GOAT BitVM2 aims to build a native Bitcoin zkRollup on BitVM2, GOAT Network's decentralized sequencer, and ZKM's zkMIPS proof network.

In this section, we define the problem and describe the complete protocol.

4.1 Design Overview

Define the problem. Instead of computing f(w) = y, we extend this model to calculate f(x, w) = y, where x is the public input. The offchain computation f is presigned/committed at *Peg-in* phase, w and y are committed in the operator's *Kick-off* transaction, and x should be committed by the sidesystem between *Peg-in* and *Kick-off* transaction. In order to switch the context from general BitVM2 to GOAT BitVM2, we use Layer 2 (L2) to represent sidesystem in the following sections.

Define the public input. The public input is used to commit the proof and inputs. For a L2, the chain's states can be committed by block hash, and the block hash is signed by the at 2/3 of L2 sequencers, given that most L2s are using BFT-based consensus algorithms.

Since the public input is committed before *Kick-off* transaction, it should be immutable before the *Kick-off* transaction confirmed.

With the above two constraints, we can define the L2 sequencer's public keys as the public input and select the sequencers round-by-round periodically. For each epoch, the committee publishes the sequencers in the next one or two weeks.

Commit the public input. Consider the fact that the public input comes from L2, L2 should be able to ensure the correctness and validity of all public inputs. In combination with BitVM2's committee setting, we use the committee to submit the public input by Threshold Signature Scheme (TSS)[7]. A trusted setup ceremony should be created to commit the TSS' public key, which is committed in a P2WSH owned UTXO via a transaction, and we name it a genesis transaction.

After the genesis transaction, the committee should periodically publish public input by spending the previous commitment UTXO.

Open the public input. The operator needs to open the public input and disclose it in the Assert transaction. The transaction introspection[8] is introduced. Currently, a very simple approach is the Bitcoin Light Client[9] by the Citrea team. For a better understanding of the overall solution, more details will be described in Section 4.3.

[7] A threshold signature is a digital signature that may have been created by an authorized subset of the private keys which were previously used to create the corresponding public key. Threshold signatures can be verified using only a single public key and a single signature, source: https://bitcoinops.org/en/topics/threshold-signature/

[8] Introspection is the ability to inspect different parts of the transaction that is being evaluated while trying to spend a specific coin, source: https://bitcoinmagazine.com/featured/bitcoin-covenants-what-are-they-and-what-do-they-do.

[9] Bitcoin Light Client: https://gist.github.com/ekrembal/2b26e224873a5a0a179d25a4b5f6e58a

Define the Universal Operator. GOAT BitVM2 introduces the concept of a *Universal Operator*, which can switch to different specific roles to perform the corresponding functions.

We describe all the roles involved in GOAT BitVM2 in Table 1. Notably, the sequencer is excluded from the table, as it is a purely L2 component from an architecture perspective, although it is highly related from an economic perspective, hence we incldue all roles in our GOAT Rollup Economic Paper.

Role	Functions	Honesty Assumption
Committee	– n-of-n signers for the pre-signed BitVM2 graph – Commit the sequencer set	1/n
Operator	Anyone can be an operator. – Exchange PeggleBTC to native BTC with users – Generate the validity proof, Kickoff the reimbursement, and respond the challenge – Generate the preimage of the hashed time-lock to each watchtower [6]	1/∞
Challenger	Anyone can be a challenger – Verify the validity of the reimbursement from operators offchain; – Submit the challenge transaction on Bitcoin to force the kick off to unhappy path; – Once the kickoff is on the unhappy path, and the operator unveils all the execution trace(Circuit F below), verify finds the fraud in the execution trace, and can spend the UTXO from Assert transaction, and stop the operator to continue the reimbursement.	1/∞
Watchtower	A special kind of challenger, selected from the Sequencer candidates, maintains the longest chain headers and spends the Watchtower output of the Kickoff transaction.	1/m
Relayer	Transmitting correlation information between BTC and GOAT.	1/∞

Table 1: GOAT BitVM2 Role Definition

Considering that the system introduces multiple roles, each with different responsibilities and costs, designing a fair and efficient roll-up economic model (and incentive mechanism) becomes extremely challenging. The core idea of our economic design is to unify all these roles under a single identity — the *Universal Operator* — and have operators rotate through the different roles over time.

All roles must stake tokens on L2 and are assigned the responsibilities of a specific role in each term.

This approach offers three major advantages: 1) Balancing Income and Costs. No operator will always be just a sequencer (who earns transaction fees by producing new blocks) or just an operator (who bears high costs such as staking and generating zero-knowledge proofs). In the long term, every operator will rotate through both profit-generating and cost-incurring roles, achieving a balance between earnings and expenses; 2) Equilibrium in Incentives. Operators know that if they are assigned high-cost roles (such as the operator or publisher) this round, they are likely to be assigned a profitable role like sequencer in the next round. Through cross-subsidization between roles, the operators' overall profit and loss are smoothed out; 3) Redundancy and Reliability. If an operator fails or goes offline, the system can reassign their duties to others in the next round. Thus, the system does not permanently depend on any particular Sequencer or Operator.

In summary, our system merges different behavioral roles (sequencer, operator, challenger, etc.) into a single pool of staked operators who share all responsibilities.

Multi-round Randomized Challenge Mechanism. Initially, a challenger is selected to verify a reimbursement request. If this challenger does not raise a challenge, another is randomly selected from the challenger pool. If this new challenger detects fraud in the reimbursement process and issues a challenge, all prior challengers who failed to act will have their stakes slashed. A cryptographic sortition is employed to select the challenger randomly. More details will be shown in Section 4.4.

4.2 Protocol Description

The entire transaction flow is presented in Fig. 3. Compared to the original BitVM2 protocol, GOAT BitVM2's enhancements are shown as follows. First, the *Peg-out* transaction is removed, each operator becomes an Over-the-counter trader via Atomic Swap. Second, the slashing transaction's reciptant has been changed from any verifier to the committee, and committee will allocate the rewards to valid verifier in L2. Last but the most important, the watchtower transactions(Bitcoin Light Client mechanism) are integrated into the flow to commit the public input.

In the following, we present the *Bridge in*, *Bridge Out*, and *Sequencer Set Commitment*.

Bridge In. It consists of the following 7 steps:

1. The user constructs a Taproot *Peg-in* transaction (including *Peg-In-Prepare*, *Peg-In-Confirm*, and *Peg-In-Cancel* branches), and broadcasts the *Peg-In-Prepare* transaction.
2. A Relayer monitors the execution of the *Peg-in* transaction and then initiates a deposit transaction on the GOAT side contract, submitting the Txid of the *Peg-In-Prepare* transaction and the unsigned *Peg-In-Confirm* transaction. Alternatively, the user can directly submit an outpoint.
3. After verifying the validity of the deposit transaction, the Committee constructs the BitVM2 transaction flow, pre-signs it, and broadcasts it to all Operators.
4. Upon receipt, operators presign certain transactions (such as Challenge and Kickoff timeout transactions) and store the fully signed transaction flow on IPFS[10].

[10] InterPlanetary File System, https://ipfs.tech/

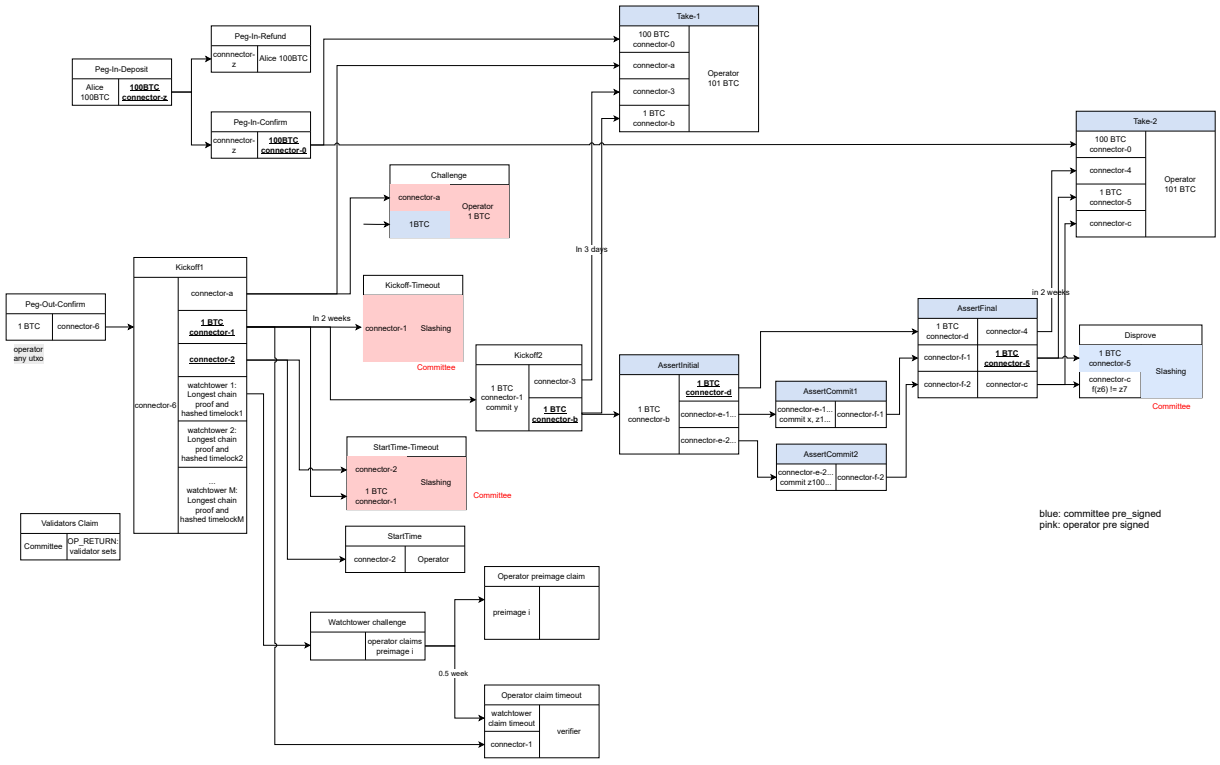

Fig. 3: GOAT BitVM2 Transaction Structure

5. The user checks the transaction flow and verifies the pre-signatures. If everything is correct, the user signs and broadcasts the *Peg-In-Confirm* transaction. The user may also choose to cancel the *Peg-in* process.
6. Once the Relayers confirm the *Peg-In-Confirm* transaction on-chain, they submit the proof to GOAT and issue PegBTC to the user.
7. Relayers form a decentralized message-passing network, but they are not required to ensure the authenticity of the messages themselves.

Bridge Out. The user performs an Atomic Swap with the operator to exchange PegBTC for BTC. After collecting a sufficient amount of PegBTC, the operator proceeds with the BitVM2 reimbursement process. Since the withdrawal is initiated by the operator, there is no need to verify *Peg-Out* completion on Bitcoin. Instead, the operator burns the PegBTC on L2 and, along with a corresponding ZKP, initiates the Kickoff transaction. As shown in Figure 4, a user and operator complete the asset exchange via Atomic Swap. As shown in Figure 5, this is done through the GOAT BitVM2 protocol.

Sequencer Set Commitment. The sequencer set is essential for verifying the L2 consensus. We adopt a periodic commitment approach, where the Committee periodically uploads the sequencer set to Bitcoin. The Committee periodically commits the sequencer set for the upcoming two weeks.

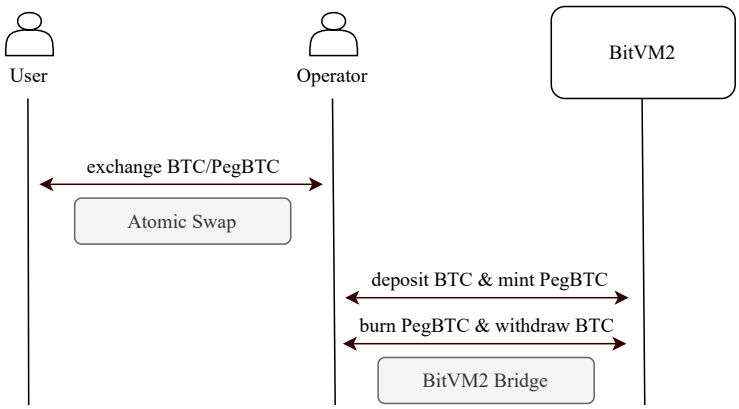

Fig. 4: Bridge Out

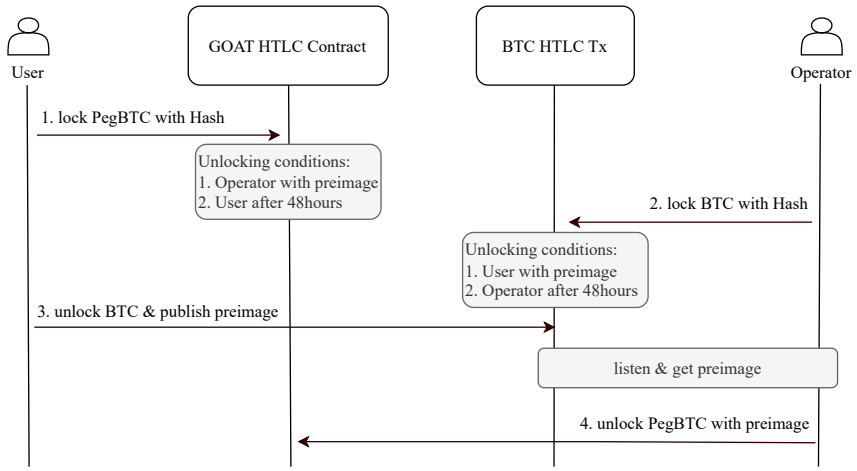

Fig. 5: Cross-chain Atomic Swap

The sequencer set is stored via a Merkle tree, with the Merkle root published in the OP_RETURN field of a BTC transaction (or alternatively via a cost-less Taproot data submission). The Merkle proof of each sequencer is also disclosed.

Since the committed sequencer set cannot be directly used by subsequent BitVM2 Asset or Disprove transactions, we introduce transaction introspection to ensure the existence of historical transactions can be verified.

Initiating the BitVM2 Reimbursement Process. It consists of the following 6 steps:

1. During the Kickoff transaction, the Operator must stake BTC and commit two pieces of information: 1) The GOAT Withdraw Txid; 2) The latest blockhash of what the Operator considers the longest chain.
2. The Challenger initiates a challenge to validate the legality of the Kickoff transaction:

Off-chain verification: 1) The *Peg-Out* transaction is confirmed on BTC, and the amount matches the *Peg-In* amount in this BitVM2 instance; 2) In the GOAT contract, there is a corresponding relationship between the Withdraw ID and the *Peg-Out* transaction.

On-chain verification: The Watchtower completes the longest chain challenge (or skips the challenge, in which case the operator is assumed to have submitted the longest chain after a timeout), as shown in the payout transaction structure diagram below.

3. If no challenge is submitted, the operator can claim the reimbursed BTC through the 'Take 1' transaction after the challenge period.
4. If the Challenger initiates a challenge, the Operator must submit an Assert transaction that commits to the zk.Verifier of the decomposed circuit F (note: the diagram below is simplified. In practice, there is also a time-locked Assert transaction. If the operator fails to submit in time, conditions for a Slash operation are met).
5. Once the Assert transaction is broadcast, the Challenger must submit a Disprove transaction. If the Challenger times out, the Operator completes the reimbursement via the 'Take 2' transaction.
6. A Verifier (not necessarily the Challenger) broadcasts the Disprove transaction and reveals the corresponding faulty traces. If the challenge succeeds and the Disprove transaction is confirmed, the Operator is denied reimbursement. Furthermore, the staked BTC in the Operator's Kickoff transaction is slashed and awarded to the Verifier as a reward.

The circuit in the proof part refers to the Algorithm 2.

4.3 Commit Sequencer Set by Transaction Introspection

Since Bitcoin transactions are stateless and do not share state across different transactions directly. In BitVM2, state sharing across transactions must be achieved using *one-time signatures*. Other solutions, such as ColliderVM [7], ensure consistent state transfer through *hash collision* techniques.

Sharing state across arbitrary transactions is harder. The SuperBlock scheme from the original BitVM2 protocol is used to verify that the Operator-funded *Peg-out* transaction is on the canonical chain —- the SuperBlock scheme is vulnerable to low-hashrate attacks.

The Citrea team proposed a Bitcoin Light Client mechanism. This mechanism uses N watchtowers, the operator allocates each watchtower a hashed timelock, then each watchtower publishes its longest chain headers as an output of the *Kick-off* transaction, the operator comfirms all the watchtower's longest chain data by disclosing the preimage of the hashed timelock. Once the operator discloses all the preimages, it can continue to publish *Kick-off-2*. In high-level summary, the operator builds hashed time-lock channels with the watchtowers to get the different longest chain information. In circuit f, we check that the public inputs, published by the committee of L2, match what has been committed by the Bitcoin Light Client mechanism (by watchtowers or the operator itself). Finally, if one of all the watchtowers is honest and provides a longer chain, the operator cannot finish the reimbursement.

Instead of using Bitcoin Light Coin to watch the *Peg-out* transaction, GOAT BitVM2 uses it to commit the decentralized sequencer's public keys and consider those public keys as the public input of circuit f.

We publish the sequencer's public keys of the next 2 weeks in a transaction periodically, and then prove that the transaction is on the Bitcoin by the light client mechanism. Since circuit f has been committed in the *Peg-in* transaction. The operator must disclose the watchtower's longest chain information and the corresponding preimage. Then we can use the sequencer's public keys to check the signature of the block that includes the burning transction, following with the SPV of the burning tx to the block, and the validity check of the block by the execution light client. We present the sequencer set commitment scheme in the algorithm 1.

4.4 Multi-round Randomized Challenge Mechanism

In traditional Optimistic Rollup designs, any participant can act as a challenger to verify state correctness and submit a fraud proof if an error is found. The security of this model relies on the "1-of-n honest assumption" — as long as one honest participant exists, fraud can be detected. However, this design presents two problems.

Resource Wastage from Redundant Computations. If 100 observers validate a block and it turns out to be valid, the efforts of 99 observers are redundant. If each challenger trusts only their own computation, it results in wasted computational resources, including: 1. The KickOff phase: Checking the validity of the longest chain (including both L1 and L2); 2. The Disprove phase: verifying through public input and the operator's proof.

Insufficient Incentives Due to Lack of Rewards for Challenges. In the absence of fraud, challengers are likely to spend a significant amount of time monitoring the challenge system without receiving any rewards. This lack of incentives could lead to honest nodes lacking motivation to participate in challenges, and, in some cases, a scenario where no challengers step forward may arise.

Additionally, for BitVM2 Challenge transactions, crowdfunding is required to support these challenges. However, there is a high probability that the contributor funding the challenge (e.g., a stakeholder) may not be the actual challenger during the Disprove phase (which is usually carried out by Bitcoin miners). As a result, crowd-funded assets from the challenge phase may not be returned to the correct challenger, leading to misalignment of incentives.

Furthermore, the following attack scenarios are considered: 1) Malicious challenger attacking honest reimbursements: A malicious challenger may attempt to falsely challenge a legitimate reimbursement. 2) Collusion between malicious challenger and Operator: The malicious challenger and the Operator may conspire to avoid initiating a challenge.

For attack 1, since fraud proofs are verified in the Bitcoin script, any malicious challenger will inevitably fail.

For attack 2, we introduce a *multi-round randomized challenge mechanism* with a one-vote veto feature. Initially, a challenger is selected to verify a reimbursement request. If this challenger does not raise a challenge, another is randomly selected from the challenger pool. If this new challenger detects fraud in the reimbursement process and issues a challenge, all prior challengers who failed to act will have their stakes slashed.

When a challenge is required (during the reimbursement process), multiple sequential challengers are randomly selected, and challengers must have staked funds before submitting their challenges on L2.

The random selection algorithm can use Verifiable Random Function (VRF) [8]. The selected challengers are required to initiate a challenge in specific block duration in case of a fraudulent reimbursement request. If a fraudulent operator creates an invalid proof, and a challenger failed to challenge, but a successor managed to disprove it, the previous challenger will be slashed and lose its stake, the successor can get all the rewards. This model can be supported by Lemma 1.

Lemma 1 (Challenger Success Probability). *Assume that each challenger has an increasing probability of success due to accumulated reward incentives, let p_1 be the initialial challenge willingness, Δp be the probability increment after each failure, and n be the number of allowed challengers, modeled as:*

$$p_i = \min\left(p_1 + (i-1)\cdot \Delta p,\ 1\right)$$

Then, the cumulative probability that at least one of n challengers successfully challenges a fraudulent prover is:

$$P_{success}(n) = 1 - \prod_{i=1}^{n} \left(1 - \min\left(p_1 + (i-1)\cdot \Delta p,\ 1\right)\right)$$

Suppose that the reimbursement request appears at block height H, the complete challenging process and time window is shown in Figure 6, which illustrates three step-by-step periods.

- Initial Challenge Period. Whenever a reimbursement request appears on the Bitcoin network, L2 triggers the VRF procedure and selects h challengers randomly, each challenger has 2 hours to finish a challenge, and the total time window is set to 12 hours, about 72 blocks. With Lemma 1, number of challengers is 6, and assume base success probability p_1=0.1, and increase per challenge Δp=0.16, the $P_{success}$ would be 0.9958, which is very confident.
- Assert Period. If no challenge appears before block height $H+72$, the Operator's reimbursement succeeds with transaction *Take 1*. Otherwise, the operator needs to respond with an *Assert* transaction before block height $H + 108$.
- Disprove Period. After the Operator's *Assert* transaction, everyone can submit a *Disprove* transaction before block height $H + 144$.

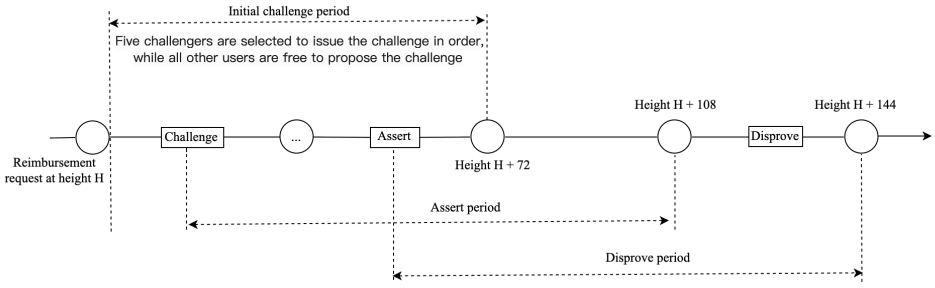

Fig. 6: Full Challenge Period

The Randomized Challenge Mechanism is a form of weak consensus without strong consistency guarantee, where a staked node is randomly selected (using cryptographic sortition) to act as the

challenger. The challenger verifies the current reimbursement process; if fraud behaviors are found, they initiate a challenge to prevent the reimbursement from being completed optimistically. The security assumption here only requires one honest challenger to ensure the system's correctness.

Improved Capital Efficiency: This process is modeled on Lemma 1. Leveraging the decentralized sequencers as an initial set for challenger candidate, helps increase the chanllenger's willingness to conduct a honest challenge, reducing challenge time and reimbursement delays. Using this method, the challenge process can be shortened to under one day.

Security Assurance: The system remains open and fair, allowing any node the opportunity to become a challenger. By introducing economic incentives and slashing mechanisms, it ensures challengers are both motivated and reliable.

The successful challenger will receive a dual-component reward consisting of an initial reward distributed via a Bitcoin-based covenant and penalties collected from selected challengers in L2 who fail to fulfill their duties. Assuming at least 2/3 of sequencers are honest and will act as responsible challengers, the probability of all six selected challengers failing to initiate a challenge becomes smller than 0.004. This reward mechanism creates compounding incentives as successful challengers accumulate escalating rewards when selected challengers remain inactive, ensuring robust protection against malicious reimbursement attempts even during Bitcoin network congestion.

5 Conclusion

GOAT BitVM2 aims to build a practical zkRollup solution with the 1-of-n honesty assumption based on the BitVM2 protocol, in combination with the decentralized sequencer and the proof network.

1) GOAT BitVM2 solves the operator double-spending attack by committing the decentralized sequencer's public keys via the Bitcoin Light Client mechanism.

2) GOAT BitVM2 allows arbitrary-amount withdrawal via Atomic Swap between Bitcoin and decentralized sequencer network, and removing the *Peg-out* transaction from the BitVM2 graph. The operator burns its Pegged Bitcoin on GOAT Network and then starts the reimbursement.

3) GOAT BitVM2 improves the fairness of the challenger, by shifting the challenge incentive from L1 to L2, thereby ensuring that challengers can receive their rewards in a more reliable manner.

4) GOAT BitVM2 introduces a*multi-round randomized challenge mechanism* to accelerate the challenger election, make it possible to finish the entire challenge-response process in 1 day.

4) GOAT BitVM2 introduces Universal Operator Abstract, to balance the risks and benefits of all the roles, enforcing that there are enough challengers in the entire system, this can significantly improve the systematic robustness and security.

5) GOAT BitVM2 introduces an efficient proving method to reduce the operator's proof generation to about 40 seconds.

GOAT BitVM2 becomes the most secure and efficient Bitcoin zkRollup protocol to date, significantly accelerating the mass adoption of BitVM2.

References

1. Linus R.: Bitvm: Compute anything on bitcoin. URL: https://bitvm. org/bitvm. pdf-(12.12. 2023), 2023.

2. Linus R., Aumayr L., Zamyatin A, et al.: BitVM2: Bridging Bitcoin to Second Layers, 2024.
3. Nakamoto S.: Bitcoin: A Peer-to-Peer Electronic Cash System, 2008.
4. Lamport L.: Lamport Signature - Short Private Key, 1979.
5. Team Z.: zkMIPS: Universal Zero-knowledge Virtual Machine on MIPS32r2 ISA, 2023.
6. Bal E., Aumayr L., İyidoğan A, et al.: Clementine: A Collateral-Efficient, Trust-Minimized, and Scalable Bitcoin Bridge. Cryptology ePrint Archive, 2025.
7. Kolobov V I., Levy A M., Naor M.: ColliderVM: Stateful Computation on Bitcoin. Cryptology ePrint Archive, 2025.
8. Goldberg S., Reyzin L., Papadopoulos D, et al.: RFC 9381: Verifiable Random Functions (VRFs), 2023.

Algorithm 1: Check Sequencer Set On Chain

Input:

For all watchtower:

- `[⟨headers, total_work⟩]`: The block headers and total work information representing the longest chain as perceived by the watchtower.
- `OTS`: A one-time signature from the watchtower over the ⟨headers, total_work⟩.

For operator:

- ⟨`all_headers, all_total_work`⟩: The block headers and total work information representing the longest chain as perceived by the operator.
- `sequencer_set`: The set of public keys for the sequencer set, included as the Witness information of the commitment transaction.
- `sequencer_set_commit_utxo`: The UTXO information corresponding to the committed sequencer set, including the txid, vout, and scriptPubKey (which contains the relevant OP_RETURN).
- `sequencer_set_commit_blockhash`: The block hash where the sequencer set commitment transaction is included.

Circuit:

```
correct_watchtower = {}
largest_total_work = 0
// Iterate over all watchtowers
for each watchtower i:
    if verify_ots(watchtower_i):
        insert (watchtower_i, 1) into correct_watchtower
    if not spv_verify_headers_and_total_work(⟨headers, total_work⟩_i):
        continue
    if not spv_verify_utxo_in_blockhash(all_headers, sequencer_set_commit_blockhash,
sequencer_set_commit_utxo):
        continue
    if not verify_sequencer_set_in_tx_note(sequencer_set_commit_utxo, sequencer_set):
        continue
    total_work_i = calculate_total_work(watchtower_i)
    if total_work_i > largest_total_work:
        largest_total_work = total_work_i
if not spv_verify_operator_all_total_work_and_headers(⟨all_headers, all_total_work⟩):
    fail_verification()
if largest_total_work > operator.all_total_work:
    fail_verification()
else:
    pass_verification()
```

Output:

- `sequencer_set`
- `correct_watchtower`
- `all_headers`
- `sequencer_set_commit_utxo`
- `sequencer_set_commit_blockhash`

Algorithm 2: GOAT BitVM2 OffChain Computation: Input

1. Check the Hashed Timelock Preimages are matching with correct_watchtower;
2. Verify that the sequencer_set_commit_utxo contains the correct sequencer set;
3. Verify correct execution of the L2 block;
4. Verify that the signature of the L2 block was made by a member of the sequencer set.

Input:

For each Watchtower:

- ⟨`headers, total_work`⟩: The block headers and total work information representing the longest chain that Watchtower i believes in.
- OTS: A one-time signature by Watchtower i over ⟨`headers, total_work`⟩.

For the Operator:

- `[`⟨`all_headers, all_total_work`⟩`]`: The block headers and total work information representing the longest chain that the Operator believes in (private input).
- `sequencer_set`: The public key set of the committed sequencer set, contained in the Witness data of the commitment transaction.
- `sequencer_set_commit_utxo`: The UTXO information of the sequencer set commitment transaction, including `txid`, `vout`, and `scriptPubKey` (which includes the corresponding OP_RETURN); (private input).
- `sequencer_set_commit_blockhash`: The block hash where the sequencer set commitment transaction is included.
- `Pre MPT State`: The L2 state root corresponding to the last successfully verified reimbursement.
- `Post MPT State`: The L2 state root corresponding to the block that contains the current reimbursement transaction.
- `SignatureOfPostMPTRoot`: The signature made by a sequencer over the `Post MPT State` (private input).
- `Accounts`: The account data used by transactions between `Pre MPT State` and `Post MPT State` (private input).
- `AccessList`: The storage information of the contracts accessed by the L2 chain (private input).
- `Transaction List`: The list of all transactions from `Pre MPT State` to `Post MPT State` (private input).
- `BurnTxProof`: The proof that the Operator has burned the corresponding amount in the L2 transaction for reimbursement (private input).
  It is assumed that the L2 system uses a VM (Virtual Machine) as its smart contract execution engine.

Algorithm 3: GOAT BitVM2 OffChain Computation: Circuits and Outputs

Circuit:

(`sequencer_sets`, `blockhash`, `correct_watchtower`, ...) =
    CheckSequencerSetOnChain(
        [⟨`all_headers, all_total_work`⟩],
        `sequencer_set`,
        `sequencer_set_commit_utxo`,
        `sequencer_set_commit_blockhash`);
// Validate block execution
vmInstance = VM(`Accounts`, `AccessList`);
Post MPT State == vmInstance.execute(`Pre MPT State`, `Transaction List`);
//Validate the block hash
lbh := LargestBlockhash(`all_headers`);
lbh == Hash(`Post MPT State`);
//Validate consensus information: ensure the latest state root is correctly signed by the sequencer set
extractedSequencerSet := parseOpReturn(`sequencer_set_commit_utxo`);
extractedSequencerSet == `sequencer_set`;
verify(`SignatureOfPostMPTRoot`, `extractedSequencerSet`) == true;
//Validate the Burn transaction
SPV(`BurnTxProof`, `Transaction List`) == true;

Output:

- `sequencer_sets`;
- `Pre MPT State`;
- `Post MPT State`;
- `sequencer_set_commit_blockhash`.