Bitlayer Network: The Computational Layer for Bitcoin

2.0 Preview

Bitlayer Research Team

June 24, 2025

Abstract

Bitcoin's limited transaction throughput and programmability hinder its potential in Decentralized Finance (DeFi). Existing Layer 2 solutions often introduce new trust assumptions, failing to anchor their security directly to Bitcoin's consensus. This paper introduces Bitlayer, a Layer 2 network that solves this challenge using the BitVM paradigm. Our core contribution is a novel, recursive verification protocol that, for the first time, enables a continuous chain of Layer 2 state transitions to be verifiably settled on Bitcoin. This moves beyond mere data inscription to achieve security rooted in Bitcoin's proof-of-work. Furthermore, we deeply integrate BitVM bridge with our rollup protocol to enable secure transfers of Bitcoin assets. Finally, we designed a high-performance execution engine and a fast consensus mechanism to provide users with "sub-second soft finality. Bitlayer unlocks Bitcoin's vast, untapped capital for a new generation of decentralized applications, laying a foundational infrastructure for the Bitcoin DeFi ecosystem.

1 Introduction

Bitcoin holds immense potential for Decentralized Finance (DeFi), but its core design limits transaction throughput and programmability. Activating Bitcoin's vast, untapped capital thus depends on secure and scalable Layer 2 solutions [1].

However, existing approaches to scaling Bitcoin fall short. Sidechains that rely on federated multisignatures introduce centralized trust, fundamentally undermining Bitcoin's security model. Meanwhile, early rollup designs for Bitcoin can post transaction data to the L1 but lack a mechanism to enforce the validity of state transitions on-chain. This leaves them vulnerable, as their security is not fully guaranteed by Bitcoin's consensus [7].

This raises a critical question: is it possible to build a Bitcoin L2 that achieves scalable computation while ensuring state validity is enforced by the Bitcoin mainnet itself, without new trust assumptions?

This paper introduces Bitlayer, a Layer 2 network that provides an affirmative answer through rollup architecture and the BitVM paradigm [2]. overcome the limitations of both the Bitcoin and existing Layer 2 solutions by enabling scalable computation while anchoring its security to the underlying Bitcoin blockchain. Our primary contributions are as follows:

- **A Recursive Bitcoin Settlement Protocol for Rollups**: We design and formalize the first rollup protocol that uses a recursive BitVM-based framework to settle a continuous claim chain of Layer 2 state transitions on Bitcoin. This provides security by anchoring the L2's validity directly to the L1.

- **A Synergistic Integration of Bridge and Rollup**: We design and implement a secure asset bridge inspired by the BitVM bridge architecture. The core innovation is its deep integration with our rollup protocol, which ensures that asset security and rollup validity are governed by a unified trust model, enabling seamless and secure asset transfers.
- **A High-Performance Execution Layer**: We design and implement a high-performance execution layer, powered by a fast consensus mechanism, to achieve Sub-Second Soft Finality. This provides a highly responsive experience ideal for DeFi, gaming, and other demanding applications.

2 Network Architecture

Bitlayer operates on a dual-level architecture that combines a Proof-of-Stake (PoS) consensus for fast block production with a rollup framework that anchors its security to the Bitcoin network. The PoS layer allows validators to sequence transactions and produce blocks rapidly, providing a high-throughput, EVM-compatible environment. The rollup layer then periodically commits and settles the state of this L2 chain onto the Bitcoin blockchain. This design leverages Bitcoin as the ultimate layer for security and data availability, while Bitlayer network serves as a scalable and efficient computational layer.

2.1 Network Participants and Roles

The network is maintained by two key participants: Validators, and Full Nodes.

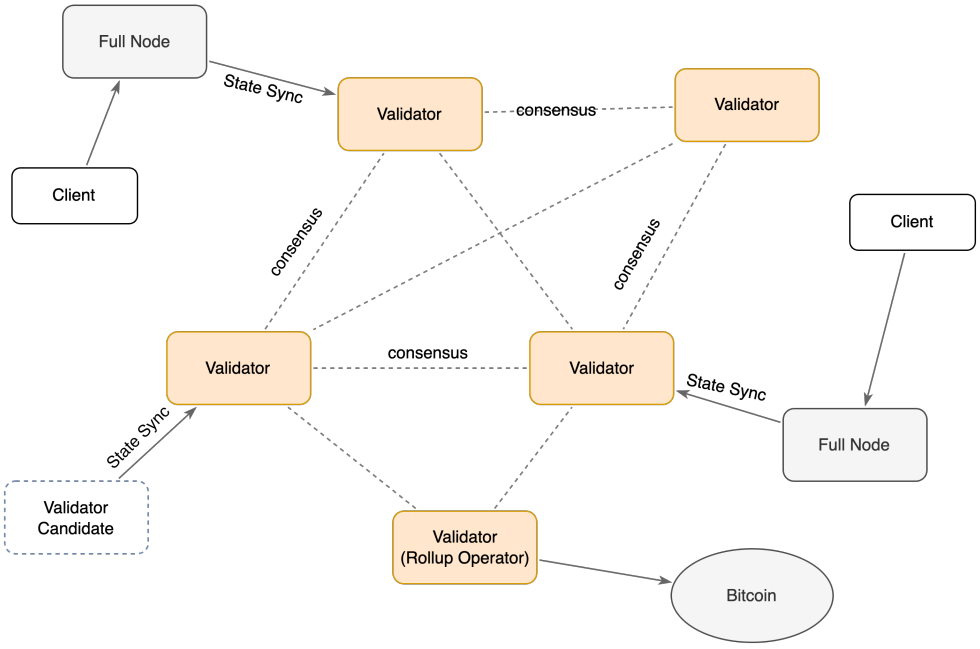

Figure 1: Network Architecture

- **Validators**: Validators form the backbone of the PoS consensus. They are responsible for producing and validating L2 blocks, ensuring the network's safety and liveness. To join the

validator set, a candidate must stake BTR tokens, and their influence in the consensus is proportional to their total stake, which can include tokens delegated by other BTR holders.

  - **Rollup Operator**: The Rollup Operator is a specialized, rotating role assigned to a single validator from the set. This operator is responsible for bundling L2 state transitions into batches, generating cryptographic proofs, and submitting them for settlement on the Bitcoin L1. To ensure accountability and disincentivize fraud, the operator must lock a significant amount of BTC as collateral on L1. The operator role rotates periodically to prevent censorship and centralization.

- **Full Nodes**: Full nodes maintain a complete copy of the Bitlayer network blockchain, independently verifying all transactions and state transitions without trusting validators. They play a crucial role in enforcing the protocol rules and ensuring network transparency.

2.2 Dual-Level Transaction Finality

Bitlayer offers a dual-level finality model, giving users and applications a choice between speed and Bitcoin-level security.

- **Soft Finality**: A transaction achieves soft finality in seconds once its containing block is confirmed by Bitlayer's PoS consensus. This provides a fast and responsive user experience, with security backed by the economic stake of the validator set.

- **Hard Finality**: Hard finality is the highest security guarantee, achieved when the L2 state containing the transaction is settled and finalized on the Bitcoin blockchain. Due to the optimistic rollup's challenge period, this takes approximately seven days. The security for hard finality relies on only a single honest party to challenge fraud, making it nearly equivalent to Bitcoin's own security.

In the rare event of a successful L1 challenge that creates a discrepancy between the L2 state and the settled L1 state, the protocol is designed to halt. The network's recovery would then be guided by social consensus among stakeholders to ensure the integrity of user assets.

3 Settling L2 State on Bitcoin

As a Layer 2 rollup, Bitlayer derives its security from Bitcoin. This chapter details the core mechanism that underpins this relationship: settlement. Settlement is the process by which L2 state transitions, executed in Bitlayer's high-throughput environment, are committed to and finalized on the Bitcoin L1. This allows Bitlayer to inherit Bitcoin's security guarantees. The challenge, however, is achieving this on Bitcoin's constrained, non-Turing-complete script environment.

Our solution is a novel settlement protocol inspired by the BitVM paradigm. This chapter systematically deconstructs this protocol. We first define the concepts of a state claim and explain our hybrid verification approach. After introducing the necessary cryptographic primitives, we detail the protocol for settling a single state claim. Finally, we show how this is extended into a recursive protocol that settles a continuous chain of L2 claims, forming the backbone of the entire rollup.

3.1 Defining the L2 State Claim

At its core, a blockchain is defined by a **State Transition Function (STF)**, denoted as Υ. This deterministic function dictates how the network's **State (s)** evolves. A state, which includes all account balances and contract data, is represented by a 32-byte Merkle root. The STF takes the current state s_t and a batch of L2 **Transaction Batch (T)** to produce the next state s_{t+1}:

$$s_{t+1} = \Upsilon(s_t, T)$$

where t is the index of transaction batch. The entire history of the blockchain unfolds from an initial **genesis state (s_0)**.

A **State Claim (Φ)** is a formal assertion submitted by a Rollup Operator to a smart contract on the Bitcoin L1. Its purpose is to commit to a new L2 state that has resulted from processing a specific transaction batch. This claim acts as the anchor, linking L2 activity to the L1 and enabling Bitlayer network to inherit Bitcoin's security.

$$\Phi = \{s_{t-1}, s_t, T\}$$

3.2 Cryptographic Primitives

The settlement protocol relies heavily on two advanced cryptographic primitives: Succinct Non-interactive Arguments (SNARGs) and Hash-based One-Time Signature scheme.

3.2.1 Groth16 SNARG

Following the Groth16 paper [4], a SNARG for a relation R consists of three probabilistic polynomial-time algorithms ($\texttt{Setup}, \texttt{Prove}, \texttt{Vfy}$):

- $\delta \leftarrow \texttt{SNARG.Setup}(R)$: A setup algorithm that produces a common reference string δ for a given relation.
- $\pi \leftarrow \texttt{SNARG.Prove}(R, \delta, \Phi, \omega)$: A prover algorithm that, given the common reference string δ, a claim Φ, and a witness ω, generates a proof argument π.
- $0/1 \leftarrow \texttt{SNARG.Vfy}(R, \delta, \Phi, \pi)$: A verification algorithm that accepts or rejects the proof.

The SNARG satisfies perfect completeness, computational soundness, and what we define as full succinctness.

Definition 1 (Full Succinctness)**.** *A protocol ($\texttt{Setup}, \texttt{Prove}, \texttt{Vfy}$) is fully succinct if the verifier $\texttt{Vfy}$ runs in time polynomial in the security parameter λ, and the size of the proof π is also polynomial in λ.*

3.2.2 Hash-based One-Time Signature (HOTS)

The Bitcoin script language, with its `OP_CHECKSIG` opcode [6], is designed to verify signatures for transactions, not for arbitrary off-chain messages. While proposals like BIP348 exist to extend this functionality, they require a network consensus change. To overcome this limitation, We utilize a Hash-based One-Time Signature scheme (`HOTS`) [5, 8]. This approach is particularly advantageous as hash functions are native and computationally inexpensive operations within Bitcoin script.

Our variant of `HOTS` consists of four algorithms:

- $(sk, pk) \leftarrow$ `HOTS.setup`(λ): Generates a secret key and public key pair from a security parameter.
- `s` $\leftarrow$ `HOTS.publish`(pk, b): Publishes a commitment to the Bitcoin script, preparing it to verify a signature for a message of length b.
- `w` $\leftarrow$ `HOTS.sign`(sk, m): Signs a message m with the secret key to produce a witness `w`.
- $(0/1, m) \leftarrow$ `HOTS.verify`$(pk,$ `w`$)$: Verifies the witness `w`. If valid, it returns '1' and reveals the original message m on the stack for further on-chain processing.

This final property—the on-chain revelation of the signed message—is a critical component for linking consecutive state claims, as will be detailed in Section 3.5.

3.3 Protocol Overview

The entire settlement protocol is embodied in a BitVM-style smart contract, which is not a single, monolithic contract but rather a complex graph of pre-signed Bitcoin transactions. Participants must jointly pre-sign this transaction graph and are bound to interact strictly according to its predefined pathways. Whereas the original BitVM protocol focused on settling claims about events on both external chain and the Bitcoin for bridging purposes [3], Bitlayer's protocol is more intricate. It must settle a continuous *sequence* of claims, each representing a discrete change in the L2 state, and guarantee that this sequence is consecutive and unbroken.

The protocol can be conceptualized as a recursive structure. In Section 3.4, we will first elaborate on the sub-protocol for settling a single state claim. Then, in Section 3.5, we will detail how this single-claim verification mechanism is recursively embedded within a broader protocol that settles a continuous chain of claims. By combining these two components, we construct the complete rollup protocol for settling the Bitlayer network state on Bitcoin.

3.4 Settling a Single Claim

3.4.1 The BitVM2 Paradigm

The on-chain verification of a claim is conducted optimistically. The verifier program in our case is expressed in Bitcoin script. However, as demonstrated by the groundbreaking work of the BitVM Alliance on a Groth16 verifier, a monolithic implementation of such a verifier is far too large to execute directly within a single Bitcoin transaction. Therefore, the BitVM2 paradigm [2] splits the large verifier program into a chain of smaller sub-programs, or "chunks." The protocol then proceeds as a fraud-proof game, where it is assumed the operator's claim is correct unless a challenger can pinpoint an incorrect computation step between two specific chunks.

3.4.2 Protocol Roles

The BitVM smart contract for claim settlement involves a well-defined set of participants:

1. **Attesting Committee:** Rather than forming a new entity, the existing validator set of the Bitlayer network serves as the attesting committee. This committee is collectively responsible for pre-signing the transaction graph that defines the protocol.
2. **Protocol Participants:** The active participants in the settlement game include a single, designated **Operator** responsible for submitting claims and any number of **Watchers**. Watchers can be anyone, including other validators, and their role is to monitor the operator and challenge fraudulent claims.

3.4.3 Single Claim Verification Protocol

The protocol for verifying a single claim unfolds as a timed challenge-response game governed by Bitcoin time locks. It ensures that both the Operator and any Watcher must act within specified time bounds or face penalties. The protocol can be broken down into three primary stages, which correspond to a series of interconnected Bitcoin transactions pre-signed by the Attesting Committee.

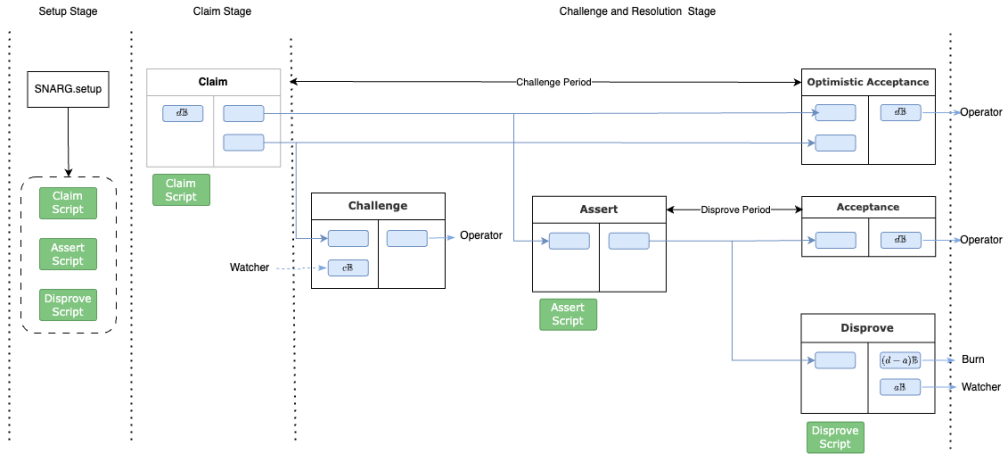

Figure 2: Single Claim Verification Protocol

1. Setup Stage Before the game begins, a trusted public setup generates the necessary scripts. The STF (Υ) is used to create a common reference string δ via `SNARG.Setup`. The verifier program $f(\pi) = \texttt{SNARG.Vfy}(R, \delta, \Phi, \pi)$ is split into a set of smaller sub-programs $\{f_i(I_i)\}$. Three types of locking scripts are created: a `Claim Script` to initiate the process, an `Assert Script` for the Operator to post results of all f_i, and a set of `Disprove Scripts`, one for each sub-program, which Watchers can use to prove fraud. The Claim Script and Assert Script are composed of multiple HOTS, which force operator to post signed claim and results of all sub-programs. The Disprove Scripts include all sub-programs and are organized within a Taproot address. These scripts are then embedded into the outputs of the pre-signed transaction graph.

2. Claim Stage The Operator initiates the process by publishing a *Claim* transaction, which spends from a previous UTXO and locks their collateral (e.g., d BTC). To unlock this input, the Operator must use their HOTS key to sign and reveal the state claim Φ_t. This action starts a timer, $\Delta_{\text{challenge}}$. Watchers observe this claim and can run the STF locally. If they find the claim to be invalid, they can issue a *Challenge* transaction before the timeout. If no challenge occurs, the Operator reclaims their collateral via an *Optimistic Acceptance* transaction, and the claim is considered settled.

3. Challenge and Resolution Stage If a Watcher issues a *Challenge* transaction (which may require them to post a smaller collateral, c BTC, to deter frivolous challenges), the pessimistic path is taken. The Operator is now compelled to execute an *Assert* transaction. In doing so, they must run the chunked verifier program locally and use their HOTS key to sign and publish the results of *all* sub-programs $\{f_i\}$. This action starts another timer, Δ_{disprove}.

The Watcher now examines the sequence of results published by the Operator. If the Watcher can find a single sub-program f_i where the Operator's published output is inconsistent with the

execution of f_i on the Operator's published inputs, they have found fraud. The Watcher can then construct a *Disprove* transaction. This transaction spends the output corresponding to the fraudulent sub-program, executing that small chunk of logic on-chain to prove the inconsistency. A successful *Disprove* transaction forfeits the Operator's collateral, a portion of which (a BTC, where $a > c$) is awarded to the Watcher. If the Operator correctly publishes results of all f_i, and the Watcher fails to submit a valid *Disprove* transaction in time, the Operator can finalize the claim via an *Acceptance* transaction and recover their collateral.

3.4.4 Security Properties

This protocol is designed to be secure under the assumption of at least one honest Watcher. Its security rests on three key properties, which are analyzed in detail in Chapter 6.2.

- **Completeness:** An honest Operator who correctly follows the protocol and submits valid state claims will never be unjustly penalized.
- **Soundness:** A dishonest Operator who submits a fraudulent claim cannot avoid being penalized, as an honest Watcher will always be able to construct a valid *Disprove* transaction.
- **Efficiency:** The entire claim verification process, whether it results in acceptance or rejection, is guaranteed to terminate within a bounded timeframe defined by the protocol's time locks.

3.5 Settling a Chain of Claims

The protocol described above is sufficient for settling a single, isolated claim. However, a rollup requires the continuous settlement of a sequence of claims that represents the ongoing evolution of the L2 state. This is achieved by extending the protocol to recursively chain claims together.

3.5.1 Linking Claims with HOTS

The key to chaining claims lies in the transaction graph's structure. Each *Claim* transaction, in addition to its other outputs, creates a special UTXO called a **claim connector**. To submit the next claim (Claim $N + 1$), the Operator must spend the claim connector UTXO created by the transaction for Claim N. The locking script for this connector requires the Operator to use their HOTS key to sign and reveal the data package for Claim $N+1$. This design naturally links adjacent claims into a chronological and unforgeable chain, as each claim transaction can only be created by consuming an output from its direct predecessor. Bitcoin time locks are used to enforce a regular cadence, preventing the Operator from submitting claims either too quickly or too slowly.

3.5.2 The Trunk Transaction Graph and Parallel Verification

This recursive structure results in a transaction graph with a primary **trunk** that links the sequence of claims. At each claim on the trunk, a complete sub-graph for single-claim verification (as described in Section 3.4) branches off.

A critical feature of this design is that the submission of the next claim does not need to wait for the final resolution of the previous claim's verification sub-protocol. The Operator can submit Claim $N+1$ while the challenge window for Claim N is still open. This parallelism is efficient but requires a mechanism to handle cascading failures. If Claim N is successfully challenged, the protocol ensures that its state is invalid, which automatically invalidates the premise of all subsequent claims $(N + 1, N + 2, \ldots)$. A rational Operator, upon having a claim successfully challenged,

is economically incentivized to cease submitting further claims, as each would require posting collateral that is doomed to be forfeited. The trunk would then terminate via a `ClaimTimeout` transaction.

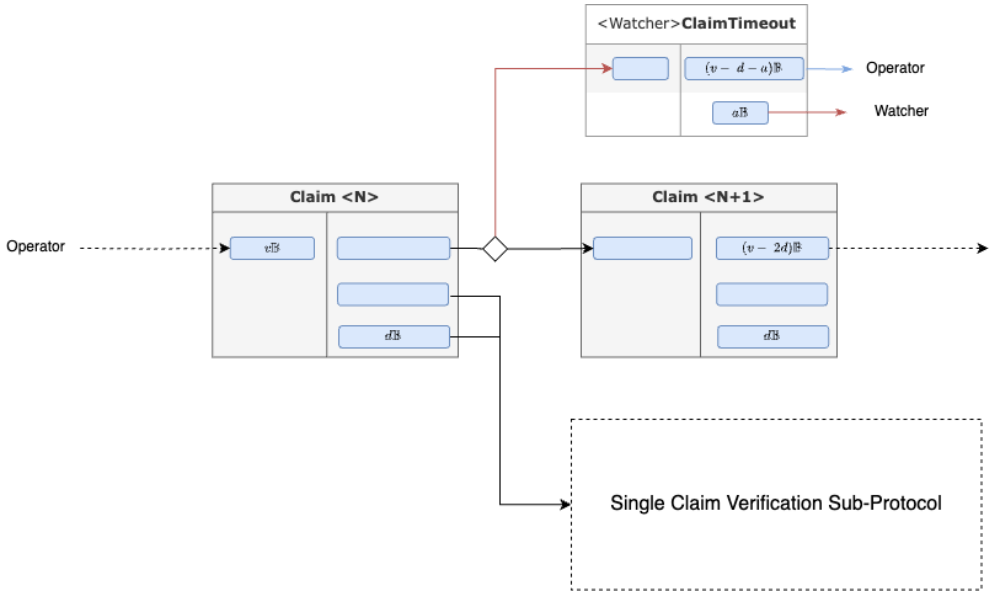

Figure 3: The Trunk Transaction Graph

3.5.3 Transaction Graph Reconfiguration and Epochs

Constructing, pre-signing, and storing a transaction graph intended to last for the entire lifecycle of the rollup (e.g., 100 years) is computationally and logistically infeasible for validators. It would also require an impossibly large amount of BTC to be locked as collateral upfront and would preclude any future protocol upgrades.

To solve these problems, we introduce **Reconfiguration**. The protocol's timeline is divided into discrete **epochs**, with each epoch consisting of a fixed number of claims (e.g., lasting for two weeks). At the transition between epochs, a reconfiguration event occurs. For each attesting ceremony, the validator set only needs to pre-sign the trunk transaction graph for the upcoming epoch. This makes the burden on validators manageable.

The Exit Window Reconfiguration is also the point at which protocol upgrades or changes to the validator set can occur. These changes may alter the security assumptions or trust parameters of the system. To protect user sovereignty, Bitlayer provides a mandatory **Exit Window**. The configuration for Epoch $N + 2$ is proposed and finalized during Epoch N. This gives users the entirety of Epoch $N + 1$ to review the new validator set and transaction graph for Epoch $N + 2$. If a user does not approve of the upcoming changes, they have a full epoch to exit the system by withdrawing their assets (e.g., pegging-out BTC via the BitVM Bridge) before the new configuration takes effect.

Validator Incentives All validators are required to stake BTR tokens to participate. The pre-signing ceremony for each epoch's transaction graph is coordinated through a system contract on the L2. Failure to participate in the ceremony in a timely manner results in the forfeiture

of a portion of the validator's staked BTR, strongly disincentivizing attacks designed to stall the protocol.

3.5.4 The Reconfiguration Process

The reconfiguration process is orchestrated by the L2 system contract. The designated operator prepares all necessary information for the next epoch's transaction graph, and each validator independently generates it, signs it, and submits their signature to the L2 contract. Once a supermajority $(N - f)$ of valid signatures are collected, they are aggregated, and the attestation is complete.

This process culminates in a **Reconfiguration transaction** on Bitcoin. This transaction locks the aggregate collateral required for all claims in the new epoch and records the updated configuration parameters, such as the verifier program commitment δ, the operator's identity, and time lock values. **Reconfiguration transactions** must be issued immediately after pre-signing is completed to promptly announce configurations. The very first such transaction, the **Epoch 0 Reconfiguration transaction**, bootstraps the entire rollup protocol and records the genesis state s_0 of the Bitlayer network.

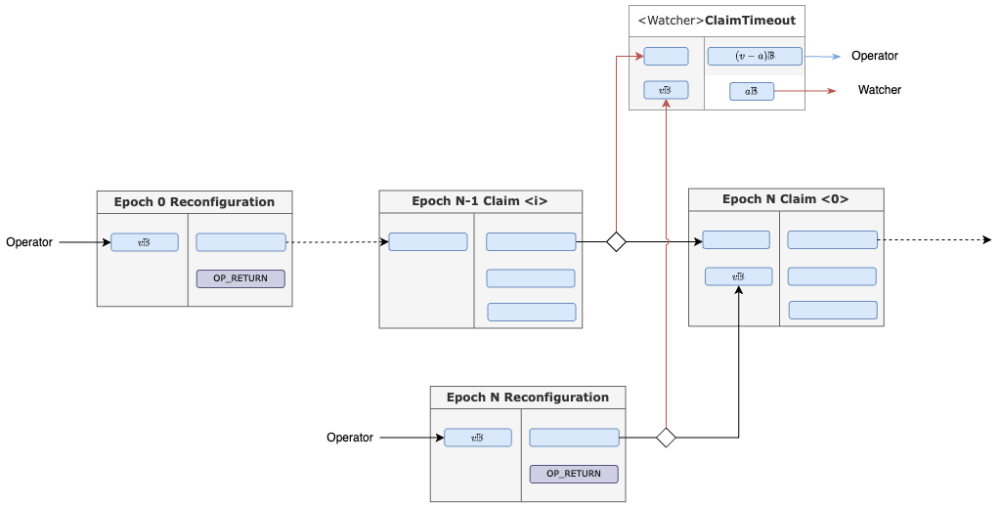

Figure 4: Transaction Graph Reconfiguration

3.6 Summary

In summary, the Bitlayer settlement protocol materializes as a perpetual, yet manageable, BitVM-style transaction graph on Bitcoin. This graph is cyclic, composed of per-epoch sub-graphs that are linked together through reconfiguration transactions. Each epoch's sub-graph contains a trunk of chronologically linked state claims, and each claim is accompanied by its own verification sub-graph—a sophisticated challenge-response game that allows any single honest participant to enforce the correctness of the L2 state. This architecture enables Bitlayer to achieve a high degree of scalability and programmability while being securely anchored to Bitcoin's unparalleled proof-of-work consensus.

4 State Transition Function and Batch Proof

Chapter 3 established how a state claim is settled on Bitcoin. This chapter details what is being settled: the execution of the Bitlayer Network STF over a batch of L2 blocks. We first define the components of our STF, including its unique system contracts. We then present the core technical contribution of this section: a multi-stage, recursive proving pipeline that uses a zero-knowledge virtual machine (zkVM) to generate a single, succinct proof for an entire batch of L2 activity, making verification on Bitcoin feasible.

4.1 The Bitlayer Network STF

Bitlayer Network's STF aligns with the fundamental principles of Ethereum [9]. At the same time, as a Bitcoin rollup, it extends EVM with additional features and processes to address its unique requirements.

4.1.1 Gas and Fee

Transaction fees on Bitlayer Network are paid in BTC. Unlike native tokens that can be minted arbitrarily, BTC is safely injected into the Bitlayer network via the BitVM bridge. Bitlayer Network introduces a distinct gas model where transaction costs are separated into execution, storage, and Data Availability (DA) components. This allows for a dynamic fee mechanism that reflects both the transaction's resource usage on L2 and the current data posting costs on the Bitcoin network.

4.1.2 System Smart Contracts

System Contracts in Bitlayer Network are a set of special-purpose contracts, deployed at genesis, that contain core protocol logic enforced during state transition verification. Changes to these contracts are managed by a governance mechanism, which oversees protocol evolution through a transparent voting process conducted by elected representatives.

System Config The System Config contract defines core operational parameters for the STF, such as block gas limits and dynamic gas pricing. Its primary purpose is to allow for risk-managed adjustments, such as congestion control, in response to network conditions. Unlike the more rigid Reconfiguration process on L1, changes to the System Config are flexible, requiring agreement from the validators to take effect in the subsequent block.

Validator Management The Validator Management contract is crucial for the network's decentralized operation, overseeing the entire validator set on an epoch-by-epoch basis.

- **Participation and Selection**: Participation is permissionless; anyone who meets a minimum staking threshold can stake BTR tokens to join the validator set. New stakers enter a pending queue, from which the protocol promotes new validators to the active set each epoch, up to a configured limit. Similarly, exiting validators enter an inactive queue for a defined period before their stake is released. The active validator set for each epoch is responsible for two primary functions: L2 block production and serving as the Attesting Committee for the Bitcoin Settlement Protocol and the BitVM Bridge.
- **Incentives and Penalties**: The contract enforces a clear economic model. Validators are rewarded in BTR for their work in securing the network, with rewards proportional to their

stake and performance. This model includes role-specific incentives: the Rollup Operator receives additional BTR rewards to offset the costs and collateral risks of L1 settlement, while Attesters are rewarded for their participation in the pre-signing process. Conversely, any validator failing to adhere to the protocol faces penalties, including the slashing of their staked BTR.

Bitcoin Light Client Contract The Bitcoin Light Client ensures that significant Bitcoin L1 events are accurately reflected within the Bitlayer ecosystem.

- **Permissionless Monitoring**: The off-chain "Listeners" monitors Bitcoin for relevant activities (e.g., pegin requests) and submits corresponding block data to the L2 light client contract. The listener is intentionally designed to be permissionless, Anyone can act as a Listener by submitting Bitcoin block headers to the contract. To process these potentially conflicting submissions, the contract enforces Bitcoin's heaviest-chain fork choice rule to maintain the canonical chain. A submitted block is then considered finalized after accruing six or more subsequent confirmations.
- **Liveness Guarantee**: To ensure the L2 state remains synchronized with L1, the Rollup Operator is obligated to submit Bitcoin block updates if no one else does. This is enforced within the rollup's proof system, which verifies that each new state claim corresponds to a minimum number of new Bitcoin blocks having been processed.
- **Censorship Resistance**: To guarantee a level of censorship resistance equivalent to Bitcoin, this contract enables a **Force-Inclusion Transaction** mechanism. A user can broadcast a transaction directly to Bitcoin using a specific format. The Bitlayer protocol obligates the Rollup Operator to process all such transactions "seen" by the light client on L1 within a defined time limit. This is enforced by the proving system: if a proposed batch fails to include a pending forced transaction, the proof generation will fail, leading to the Operator's bond being slashed.

Bridge This contract manages the bidirectional flow of assets between Bitcoin and Bitlayer Network. When the Bitcoin Listener reports a pegin transaction, the Bridge contract is invoked to mint an equivalent amount of the asset on L2. It also handles the initiation of withdrawals (peg-out) from Bitlayer Network back to Bitcoin. The detailed logic of the bridge mechanism is discussed further in Chapter 5.

4.2 Transaction lifecycle

1. **Sequencing:** Users initiate transactions on the L2 network (Step 1a). These transactions are collected by the Sequencer component within the validator, while on-chain interactions (Step 1b), such as deposits and force inclusion transactions, are submitted to Bitcoin. The Bitcoin Listener component continuously monitors the Bitcoin and synchronizes L1 transactions to the Sequencer (Steps 2–3).
2. **Block Consensus:** After collecting the transactions, one of the Sequencers is selected to propose a set of transactions. All the validators could reach an agreement on the ordering and content of the proposal through the consensus (Step 4). Once consensus is achieved, a sequenced L2 block is produced (Step 5).

3. **Execution:** Once a block is produced, it is immediately executed by the validator (Step 6). This includes applying all transactions in the block to update the L2 state and storing the resulting state in the database. This ensures that every validator maintains a consistent and up-to-date view of the state.

4. **State Claim:** After executing blocks, the Operator component fetches block data from the database (Step 7), assembles rollup batch information, and starts the Bitcoin settlement process as described in Chapter 3.

5. **Proof Generation:** When a Watcher challenges the state claim posted by the Operator on Bitcoin, the Operator responds by generating a zero-knowledge proof (ZKP) to defend the correctness of the claimed state. To this end, the Operator submits proving tasks to the ZK Prover (Step 9). The Prover generates ZKPs based on the execution trace and state transition. These proofs are then returned to the Operator (Step 10) and submitted to Bitcoin (Step 11) as evidence supporting the validity of the rollup state.

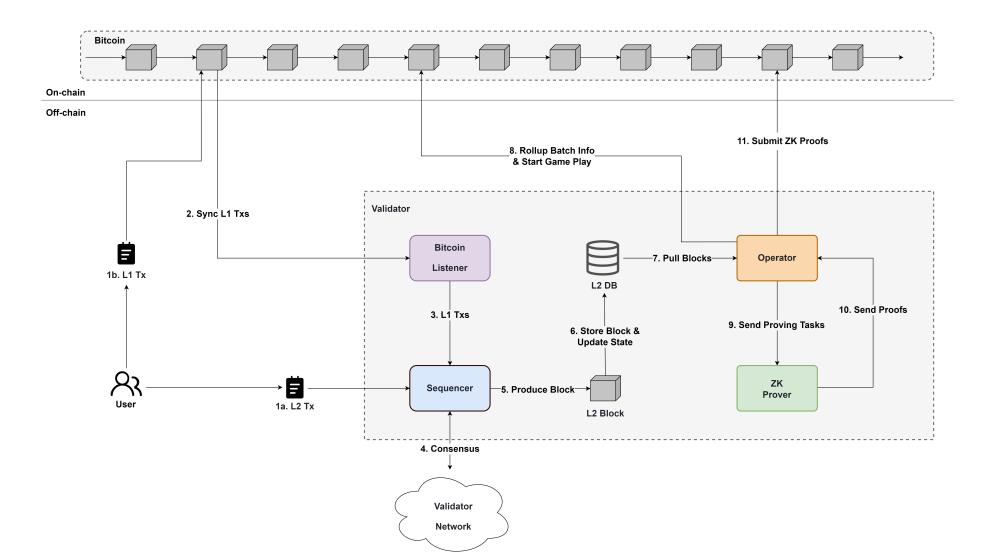

Figure 5: Transaction Lifecycle

In the following section, we provide a detailed explanation of the Prover's architecture and the internal structure of the generated proof.

4.3 Verifying Batch Proof

To verify the state transition function of our rollup protocol, we use a zkVM as the foundation of our constraint system and proof generation protocol. This section describes the architecture and operational flow of the zkVM-based proving system. The design leverages the flexibility of zkVMs to generate proofs directly from the existing execution codebase, utilizing a multi-stage, recursive workflow to ensure both the integrity and validity of computations. Ultimately, this process produces a final proof that can be efficiently verified on-chain.

4.3.1 Recursive Proving Pipeline

The core of the zkVM proving workflow is a sequential pipeline consisting of four distinct stages: STF Execution, Batch Aggregation, Batch Recursion, and Finalization. Each stage receives specific inputs, performs computations within the zkVM, and generates outputs that either feed into the next stage or contribute to the final verifiable proof. This recursive architecture enables efficient aggregation and compression of proofs, enhancing the scalability and efficiency of the system. The proving pipeline is illustrated in the following figure, and each stage will be described in detail.

The integrity of the STF is fundamentally secured by the `CodeCommitment`, the hash of the STF program binary. To ensure the integrity of the entire process, the `CodeControlGroup` is employed. This structure is an append-only Merkle Patricia Trie (MPT) that represents the outcome of social consensus: each leaf node of the MPT corresponds to a valid `CodeCommitment`. Operators usually publish the `CodeControlGroup` prior to a system upgrade, and it is accepted by all participants. This approach enables index-based lookup of valid code commitments, ensuring that all proofs are generated only from authorized and correctly specified program versions relevant to their operational context.

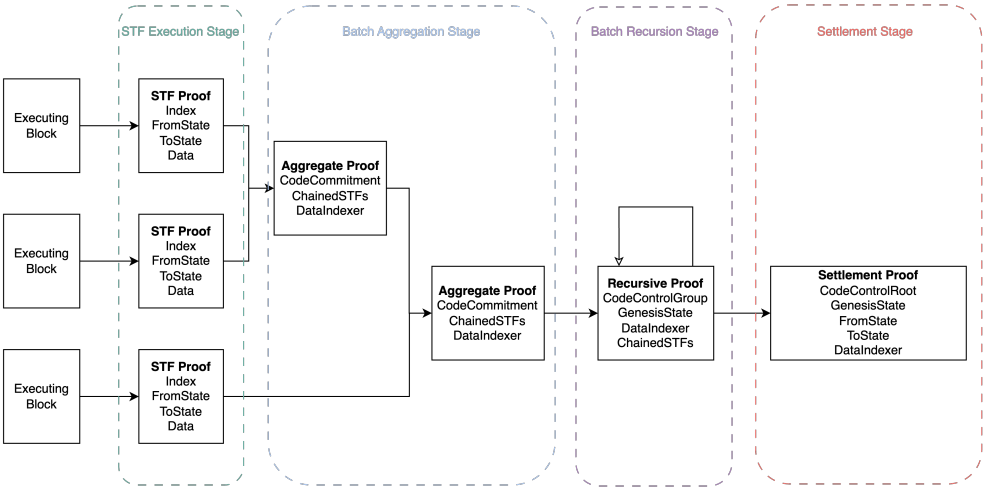

Figure 6: Recursive Proving Pipeline

Stage 1: State Transition Function Execution After the state transition of a single block is completed, the proof of correct execution is initiated within the zkVM for that block. The elements `Index`, `FromState`, `ToState`, and `Data` collectively serve as a claim of the correct outcome for the block's execution. Here, `FromState` and `ToState` are cryptographic commitments (e.g., state roots) representing the state before and after the block's execution, while `Index` provides a unique identifier for the processed block—typically the block number. `Data` represents the commitment to the operations processed within that block. In the current design, this is either the Merkle root of the transactions within the block or another representation of the state changes applied.

Stage 2: Batch Aggregation Batch aggregation is used to combine multiple STF proofs from consecutive blocks and to record their associated STF program commitments. This stage takes several STF proofs as input and produces a batch proof as output. Batch aggregation not only demonstrates the correctness of multiple STF proofs, but also verifies the consistency between

adjacent blocks. Specifically, it ensures that the `ToState` of the STF with `Index` is consistent with the `FromState` of the STF with `Index + 1`. Additionally, the `Data` from all blocks in the batch are combined into a single commitment, `DataIndexer`, indicating that the batch's data has been successfully validated against the data availability layer.

Furthermore, the code commitments of the STF programs used to generate each input STF proof are extracted and recorded as `CodeCommitment`. This stage does not verify these STF code commitments against the `CodeControlGroup`; that verification is deferred to the Batch Recursion stage. The set of all valid STFs from previous stages is denoted as `ChainedSTFs`.

Stage 3: Batch Recursion The primary purpose of the batch recursion stage is to recursively combine batch proofs, enabling the aggregation of proofs over progressively larger sequences of blocks while maintaining a constant-size proof.

In this stage, two zkVM proofs, including the new batch proof and the ongoing recursive proof, are combined into a new, single recursive proof. For the previous batch proof, each claimed STF program commitment (`CodeCommitment`) is validated against the `CodeControlGroup`, and the Batch Aggregation program is also validated against the `CodeControlGroup` based on the relevant execution index.

Likewise, both the previous recursive proof and the current Batch Recursion program must comply with the `CodeControlGroup` constraints for their respective historical index values. This comprehensive validation guarantees that the newly generated recursive proof correctly extends the chain of trust from the genesis block.

Batch Recursive will incrementally scan all claim transactions from the blocks in Bitcoin longest chain, making sure `ChainedSTFs` is consistent with the stated posted by claim transactions. At this stage, `CodeControlGroup`, `GenesisState`, `DataIndexer`, and `ChainedSTFs` collectively serve as the claims for the proof, with `ChainedSTFs` encapsulating the cumulative chained output, including the latest state.

Stage 4: Settlement For settling the proof on the Bitcoin network, we utilize a final SNARG proof, which features polynomial size and polynomial verification time relative to the security parameter λ. This compaction circuit significantly reduces the computational load required for on-chain validation and remains fully compatible with the Bitcoin settlement protocol.

During this stage, a final check is performed to ensure that the recursive proving program version used to generate the input Recursive Proof adheres to the constraints of the `CodeControlGroup`. The `CodeControlRoot`, derived as the state root of the `CodeControlGroup` MPT, serves as a commitment to the entire authorized code history and all current valid code versions.

The claim produced at this stage contains all information necessary for the on-chain verifier, including `CodeControlRoot`, `GenesisState`, `FromState`, `ToState`, and `DataIndexer`. Watchers first observe this claim and may decide to challenge it by submitting a Challenge transaction to the Bitcoin network. If a watcher initiates a challenge, the proof is generated using the protocol described in this section. Otherwise, the proof is produced in the background for use in future claims.

4.3.2 On-Chain Verification via Bitcoin Script

The SNARG claim, generated by the zkVM proving pipeline, is specifically designed for efficient and secure verification on the Bitcoin blockchain using its native scripting capabilities. This verification process relies on a combination of static commitments embedded within the script template

(established during pre-signing transactions) and dynamic inputs provided at runtime with the transaction containing the proof.

`CodeControlRoot` and `GenesisState` are hard-coded in the script template. These two values are established prior to individual proof verification and serve as trust anchors for the system. The settlement proof (i.e., the SNARG proof), along with `FromState`, `ToState`, and `DataIndexer`, are specific to the particular state transition being verified.

The script combines the dynamic fields with the static `GenesisState` and `CodeControlRoot` (which also serve as public inputs to the ZK proof itself). These elements are hashed together on-chain to form a single, comprehensive `ClaimHash`. This `ClaimHash` represents the public statement that the ZK proof attests to, with its integrity ensured by on-chain computation.

Following the BitVM2 approach (see Chapter 3), the ZK proof verification logic is invoked optimistically. If the BitVM transaction graph accepts the proof, it confirms that:

1. The claimed state transition from `FromState` to `ToState` is computationally valid according to the rules enforced by the zkVM programs;
2. The programs used were authorized as defined by the `CodeControlRoot`;
3. All relevant data was made available as defined by the `DataIndexer`;
4. The entire history traces back to the `GenesisState`.

5 Bridging Bitcoin and Bitlayer Network

A secure rollup requires a correspondingly secure mechanism for asset transfers between the L1 and L2. This chapter details the Bitlayer Asset Bridge, the mechanism for transferring assets between Bitcoin and the Bitlayer Network. The bridge is built upon the same BitVM paradigm as our settlement protocol, ensuring a unified security model for both state validity and asset custody.

5.1 Roles

The bridge protocol involves several key roles:

1. **Users:** Asset holders who initiate transfers between Bitcoin and Bitlayer Network.
2. **Broker:** Assists users in preparing deposits and withdrawals, including constructing initial transaction graphs and obtaining signatures from Attesters. Brokers directly interface with users, abstracting the complexity of the BitVM protocol and enabling seamless interaction.
3. **Attesting Committee:** This is the same validator set from the rollup protocol. The committee elected for a specific Epoch N is responsible for pre-signing the transaction graphs for all bridge requests initiated within that epoch.
4. **Watcher:** Permissionless observers who monitor the protocol and challenge malicious behavior.

5.2 Asset Cross-Chain Flow

Below we use BTC as an example to introduce the complete process of asset deposit and withdrawal.

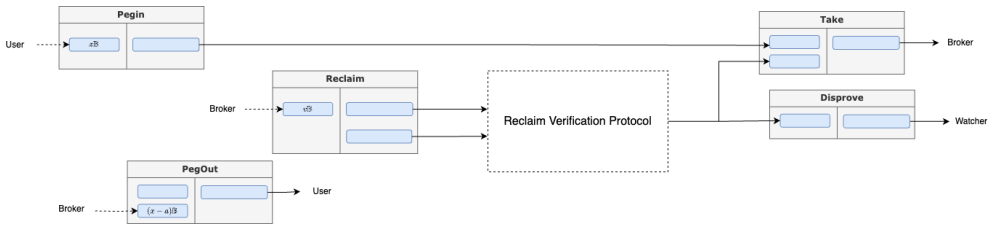

Figure 7: Asset Cross-Chain Flow

5.2.1 Asset Deposit (Peg-in)

The peg-in process moves assets from Bitcoin to Bitlayer and is initiated by the user in several steps:

1. **Initiate Request:** The user submits a `PeginRequest` to all brokers, specifying the UTXOs for deposit, the target Bitlayer Network address, and a Bitcoin address for transaction change.

2. **Preparation:** A broker responds with a complete `Pegin` transaction and the associated transaction graph. The user must carefully verify its correctness, ensuring all parameters meet their expectations.

3. **Broadcast & Mint:** After verification, the user broadcasts the `Pegin` transaction on the Bitcoin network. Once the Pegin transaction is confirmed on L1, the Bitcoin Light Client on L2 (as described in Chapter 4) recognizes it. Anyone can then submit a proof of this transaction to the Bridge contract on L2, which triggers the minting of an equivalent amount of BTC to the user's specified address.

5.2.2 Asset Withdrawal (Peg-out)

The standard peg-out process is designed for efficiency, relying on Brokers to provide upfront liquidity for a fast user experience:

1. **Initiate Burn:** The user initiates a **Burn transaction** on the Bitlayer Network. This transaction burns a specific amount of BTC on the L2 and specifies two key parameters: the amount of BTC the user wishes to receive on the Bitcoin mainnet and the recipient's address. The difference between the amount burned on Bitlayer Network and the amount to be received on Bitcoin constitutes the fee for the Broker.

2. **Broker Fronts Funds:** The Broker monitors for these **Burn transaction**. If a Broker finds the fee acceptable, they will immediately front the requested funds to the user's specified address on the Bitcoin mainnet, efficiently completing the withdrawal.

After fronting the funds, the Broker needs to reclaim their capital from the protocol through the security mechanism detailed below.

5.3 Broker Funds Reclamation

To recover their fronted funds, the Broker initiates a verification process by submitting a **KickOff transaction** to the protocol. This submission serves as an assertion that the Broker has legitimately fulfilled a valid **Burn transaction**. The verification follows the same optimistic, challenge-response game used for state settlement in Chapter 3, where the Broker's assertion is assumed correct unless challenged.

- **Challenge Process:** Watchers verify the legitimacy of this Reclaim Claim. If any invalidity is found (e.g., the corresponding `Burn` transaction does not exist or is invalid), a Watcher will publish a `Challenge` transaction.

- **Assertion and Penalty:** Upon being challenged, the Broker must respond within a specified time with an **Assert transaction**, which must contain a **Groth16 ZKP**. If a Watcher can verify that this proof is invalid, they can publish a **Disprove transaction** to penalize the Broker and receive a portion of their bonded collateral as a reward. This game-theoretic process is mechanically identical to the Single Claim Verification Protocol described in Chapter 3.

This reclaim verification mechanism relies on the **Bitlayer Light Client** and depends on the Bitcoin mainnet for the finality of Bitlayer Network transactions. Therefore, a **Burn transaction** is considered valid only after it has been included in a Batch and achieved Hard Finality on Bitcoin. If challenged, the Groth16 proof provided by the Broker must contain a complete verification chain from the Bitlayer Network's Genesis State to the current state to prove the validity and authenticity of the **Burn transaction**.

5.4 Escape Hatch

The bridge includes an escape hatch to guarantee user sovereignty over their assets, even if the L2 protocol halts. A halt can occur if the Operator repeatedly fails to submit new claims or if a submitted claim is successfully challenged. In this scenario, while the Operator's collateral is slashed and the L2 state is protected from further invalid updates, user funds could become locked. The escape hatch provides a new path for withdrawal, which also relies on Brokers for liquidity.

The process unfolds as follows:

1. **User-Initiated Forced Withdrawal**: A user initiates an emergency withdrawal by broadcasting a force-inclusion withdrawal transaction directly to the Bitcoin L1. This transaction contains a signature proving ownership of the L2 account and specifies the L1 address for receiving the funds. While this L1 transaction cannot be fully processed by the stalled rollup, it serves as an immutable, on-chain withdrawal request.

2. **Broker Fronts Funds**: Brokers monitor the Bitcoin L1 for these forced withdrawal requests. After aggregating a sufficient number of requests to meet a predefined threshold, a Broker can choose to front the liquidity, sending the funds directly to the users' specified L1 addresses.

3. **Broker Funds Reclamation**: To reclaim their fronted capital, the Broker submits a reclaim claim to the bridge protocol, accompanied by a single Groth16 proof. This proof must validate three distinct conditions:

   (a) **Proof of L2 Halt**: Evidence that the rollup protocol is stalled. This is confirmed either by showing a CommitBatchTimeout transaction (indicating the Operator's failure to submit a new batch) or a successful slash transaction (indicating the last submitted batch was fraudulent).

   (b) **Proof of Valid User Request**: Evidence that the user's withdrawal request is legitimate. This requires proving the existence of the force-inclusion transaction on L1 (via the Bitcoin Light Client) and confirming the user had a sufficient balance in the last correctly finalized L2 state.

(c) **Proof of Fulfillment**: Evidence that the Broker has already sent the corresponding funds to the user on L1, also confirmed via the Bitcoin Light Client.

This escape hatch mechanism ensures that users always retain control of their assets, relying only on the security of the Bitcoin L1 and the economic incentives of the Broker network. We will explore using account abstraction to define more intelligent withdrawal logic and extending this mechanism to support the emergency withdrawal of assets held within smart contracts.

6 Security Analysis

This chapter presents a comprehensive analysis of the security that underpins the Bitlayer Rollup. We begin by introducing a general security model for BitVM-style smart contracts, followed by definitions and proofs of their safety and liveness properties. We then conduct a detailed analysis of the Bitcoin settlement security properties discussed in Chapter 3. Finally, we briefly described the inherent censorship resistance provided by decentralized networks.

6.1 BitVM-Style Smart Contract Security

BitVM-style smart contracts follow a universal transaction graph structure. In this section, we provide a general security analysis applicable to all BitVM-style contracts, including the Bitlayer Rollup contract.

6.1.1 System Model & Assumptions

In a BitVM-style smart contract, at least three roles are required to collaborate:

- **Transaction Graph Proposer:** The Proposer is responsible for initiating a contract instance, the Proposer must stake a predefined amount of BTC, serving as both a commitment and a deterrent against misbehavior.
- **Attesters:** We assume there are n Attesters, among whom m are honest. The remaining $n - m$ are semi-honest, meaning they follow the protocol and collaborate to construct the presigned signature but may behave unpredictably off-protocol, such as retaining keys after presigning. Each presigning requires the participation of at least $n - m + 1$ Attesters.
- **Watchers:** Watchers monitor the on-chain state submitted by the Proposer to ensure correctness. If misbehavior is detected, they can hold the Proposer accountable by invoking penalties on the staked BTC. The model assumes the existence of at least one rational, honest, and active Watcher.

Additionally, we assume a **synchronized network**, where all communications between participants and the Bitcoin network occur within a known bounded time Δ. All participants are assumed to be rational and polynomial-time bounded, meaning all cryptographic tools used in the BitVM-style smart contract are secure.

6.1.2 Transaction Graph Model

The Transaction Graph serves as the backbone of the BitVM-style smart contract, structured as a directed acyclic graph (DAG). This model provides clarity and enforceability to the contract's execution.

- **Preceding Txs:** The transactions provide the initial outputs necessary for the contract's execution, which include the Proposer's stake reserve and the Watcher's reserve. The Attesters must validate the existence and correctness of these transactions before presigning.
- **Presigned Txs:** The transactions that Attesters need to presign, which determines the logic of the BitVM-style contract.
- **Sink Txs:** The transactions, lacking outgoing edges in the DAG, signify the release of funds.

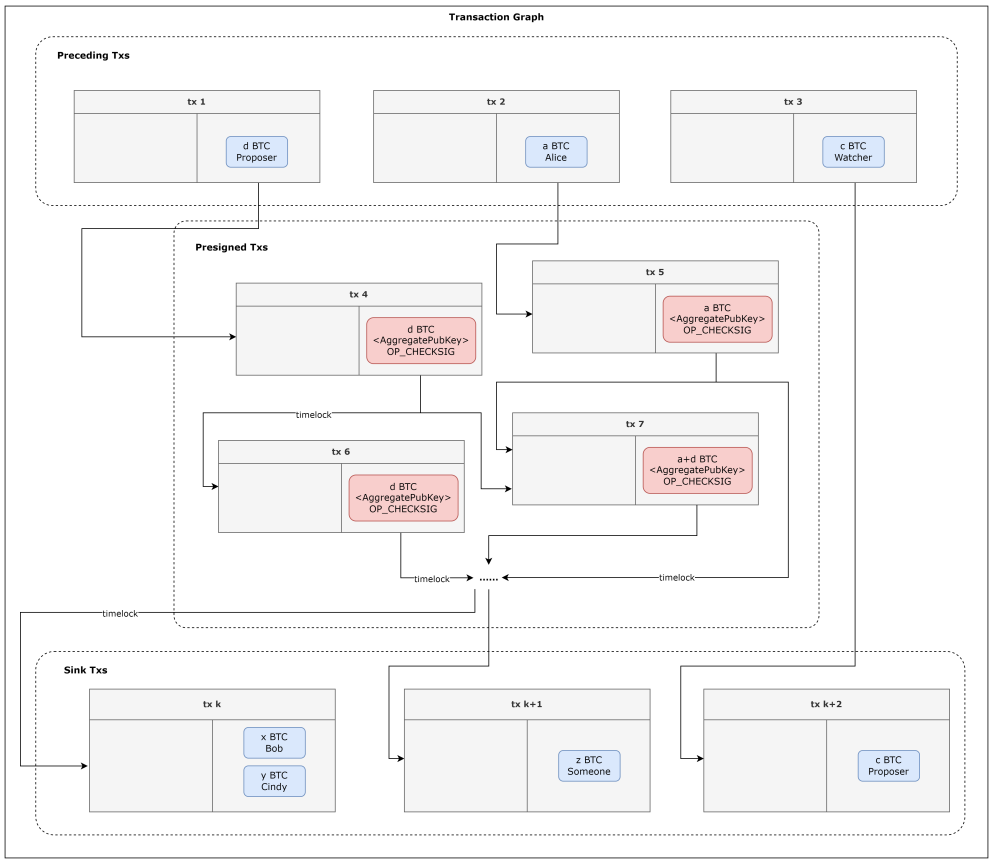

Figure 8: Transaction Graph DAG Model

6.1.3 Design Principles

- **Stake:** The Proposer must stake a specified amount of BTC to initiate the contract. (d BTC in the graph).
- **Slashable:** Incorrect STF submitted by the Proposer can result in the slashing of their staked BTC.
- **Termination:** All outputs containing amounts in the presigned transactions must have a timelock path (which may involve multiple transactions) leading to Sink Txs, ensuring the contract eventually terminates.

6.1.4 Safety

Safety Goals

- **Validity:** Every transaction in the Transaction Graph must be valid post-presigning.
- **Integrity:** No new transactions can be added to the Transaction Graph after presigning.
- **Flexibility:** The BitVM-style smart contract can accommodate different security assumptions, depending on the application scenario.

Lemma 1. *Let* $\{tx_1, ..., tx_n\}$ *be the presigned transactions spending* $utxo_a$*. No transaction* $tx' \notin \{tx_1, ..., tx_n\}$ *can spend* $utxo_a$*.*

Proof. We prove this important lemma by contradiction. Assume a presign committee $\{attester_0, ..., attester_{n-m}\}$ performed the setup presigning. If tx' exists, it indicates that the Attesters have performed additional signing outside of the setup phase, which implies that these $n - m + 1$ Attesters are semi-honest. This contradicts the assumption. □

Lemma 2. *Each presign committee must include at least one honest Attester.*

Theorem 1 (Validity)**.** *If a valid presigned signature* δ *is produced for a transaction* tx*, then* tx *is valid.*

Proof. By Lemma 2, at least one honest Attester s_i participated in the presigning and contributed partial signature σ_i for tx. Hence, tx received by s_i must be valid. Since the validity of δ relies on all Attesters contributing partial signatures to tx, it must be valid. □

Theorem 2 (Integrity)**.**

Proof. Except for Sink Transactions, all outputs must require a multi-signature from the presign committee. By Lemma 1, we can conclude that all participants can only spend the UTXOs in the Transaction Graph along the predefined path, ensuring the integrity of the BitVM-style smart contract. □

Theorem 3 (Flexibility)**.**

Proof. We can dynamically adjust the security assumptions of the Attesters based on the requirements of the application scenario, as long as the presign committee ultimately includes at least one honest node. Based on Lemma 1 and Lemma 2, validity and integrity can then be deduced. □

6.1.5 Liveness

Liveness at the Setup Phase The setup phase is inherently fragile in terms of liveness. If even a single Attester refuses to cooperate, the entire contract setup will fail. This underscores the importance of coordination and trust among participants during the initial setup phase.

Attester Rotation To address the liveness fragility problem without compromising security or fungibility, we propose maintaining a pool of Attesters on the Bitlayer Rollup. Instead of requiring all Attesters to be online, a fixed number of Attesters will be randomly selected from this pool to form the presign committee for each instance. If the presign process fails to complete within a specified time, a new committee will be selected. Additionally, Attesters who fail to respond will be penalized and temporarily excluded from participating in subsequent committee selections.

Liveness at the Execution Phase

Liveness Goal

- **Funds Liquidity:** Funds involved in the contract's Preceding Txs must not remain indefinitely locked.

Theorem 4 (Funds Liquidity)**.**

Proof. Since the timelock duration is known and finite, the Termination principle of the Transaction Graph ensures that all funds will eventually be unlocked and flow to Sink Txs within a finite time. □

6.2 Bitcoin Settlement Security

This section focuses on proving the Bitcoin settlement security properties introduced in Chapter 3.4.4.

Lemma 3 (Completeness)**.** *An honest Operator who correctly follows the protocol and submits valid state claims will never be unjustly penalized.*

Proof. An honest operator publishes valid claims and sub-program results within the required time windows, ensuring no inconsistencies arise. As a result, no watcher can unlock a *Disprove Script*, and the operator is not penalized. To save space, the details are omitted here. □

Lemma 4 (Soundness)**.** *A dishonest Operator who submits a fraudulent claim cannot avoid being penalized, as an honest Watcher will always be able to construct a valid Disprove transaction.*

Proof. If the dishonest operator does not publish Φ within Δ_{claim}, the operator will be penalized. During the Claim Phase, the dishonest operator publishes Φ. `SNARG.Vrfy` will fail locally for watchers, they raise a challenge within $\Delta_{\text{challenge}}$, moving the protocol to the Challenge Phase. If the operator does not publish the result of all sub-programs within time Δ_{assert}, the operator will be penalized.

If there is a disproved algorithm allowing watchers to unlock a *Disprove Script* with inputs and outputs published by the operator that contradict the sub-program execution, then a dishonest operator cannot escape penalties.

We prove the existence of the disprove algorithm as follows. First, since the inputs and outputs published by the operator contradict the local sub-program execution, there must be at least one inconsistent output produced by the sub-program, saying f'. Then, we check the consistency of the inputs of f'. If all inputs are consistent, we select f' as the challenged sub-program, otherwise recursively run the first step for one of the inconsistent inputs. So, the disprove algorithm must successfully select a sub-program to challenge. □

Lemma 5 (Efficiency)**.** *The entire claim verification process, whether it results in acceptance or rejection, is guaranteed to terminate within a bounded timeframe defined by the protocol's time locks.*

Proof. Each phase of the protocol has a bounded time. If both the Operator and Watcher are honest in following the protocol, the optimistic time bound is $\Delta_{claim} + \Delta_{challenge}$. If either the Operator or any Watcher tries to destroy the protocol, the time-bound will become $\Delta_{claim} + \Delta_{assert} + \Delta_{disprove}$. Thus, the maximum time bound to confirm is $\Delta_{claim} + \max\{\Delta_{challenge}, \Delta_{assert} + \Delta_{disprove}\}$, so that the protocol terminates regardless of whether the claim is accepted or rejected. Thus, the protocol guarantees efficiency by design. □

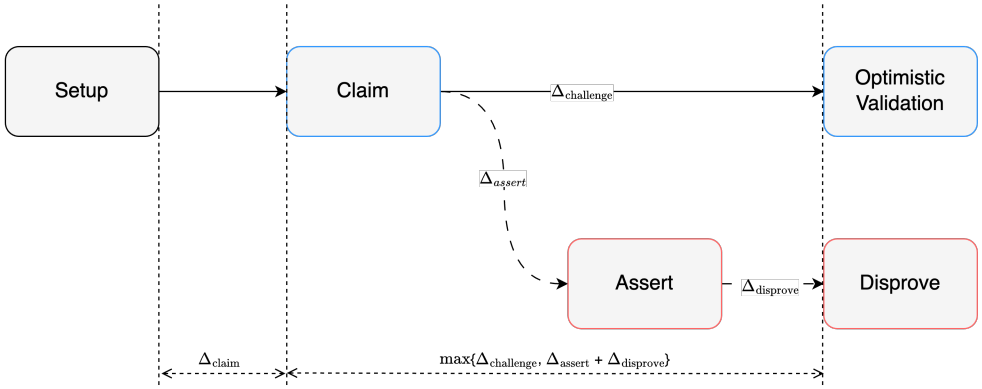

Figure 9: Bitcoin Settlement Security

6.3 Censorship Resistance

Unlike traditional L2 architectures that rely on a single sequencer, our design employs a rotating manner among validators to produce blocks. This decentralized sequencing mechanism ensures that no single party can unilaterally censor transactions. As block production rotates among validators in a permissionless and stake-weighted manner, any attempt to exclude valid transactions can be bypassed in subsequent blocks, providing strong built-in censorship resistance and enhancing the neutrality of the network.

7 Limitations and Future Directions

This chapter reflects on the current design of the Bitlayer Network, discussing its inherent trade-offs and the promising research avenues they inspire. We first outline the primary limitations of our current protocol and then detail future work aimed at addressing these challenges and further advancing the capabilities of Bitcoin L2s.

7.1 Limitations

While the Bitlayer Network provides a robust framework for a Bitcoin computational layer, its current design involves several trade-offs:

1. **Trust Dependency on the Attester Committee:** The security of the bridge and settlement protocol relies on a 1-of-N honest-minority assumption within the Attester Committee. While this provides strong security, eliminating this external committee through future Bitcoin protocol upgrades remains a key goal for achieving a more fully trustless system.

2. **Centralized Operator and Liveness:** The current model uses a single, rotating Rollup Operator for sequencing and settlement. While this is efficient, it presents a potential single point of failure for liveness if the operator goes offline. This motivates the development of a multi-operator mechanism.

3. **Reliance on Broker Liquidity:** The fast withdrawal and emergency escape hatch mechanisms depend on an active network of third-party Brokers to provide upfront liquidity. The system's user experience and capital efficiency could be further improved by protocol-native solutions that reduce this reliance.

7.2 Future Directions

We are actively researching several enhancements to address these limitations and expand the network's capabilities:

1. **Leveraging Future Bitcoin Upgrades (Covenants):** Upcoming potential Bitcoin protocol upgrades, such as those introducing new covenant opcodes (e.g., `OP_CTV` [10], `OP_CAT`, or similar proposals), could pave the way for more trustless smart contract functionalities directly on Bitcoin. We are closely monitoring these developments and plan to integrate such features, if and when they become available and stable. This could allow for:
    - **Elimination of Attester Committees:** Potentially removing the need for an attester committee for certain verification processes, moving towards a more fully trustless model.
    - **Enhanced Permissionlessness:** Reducing reliance on pre-signed transactions or specific roles in the dispute resolution protocol, making the system even more open.
    - **On-Chain Operator Election:** Managing Rollup Operator election and rotation more directly on-chain, further enhancing liveness and decentralization.
    - **Optimized Collateral Management:** Enabling more sophisticated collateral reuse mechanisms within the same epoch without compromising security, thereby reducing the capital costs for operators.
2. **Advanced Proving Systems:** Continuously evaluating and integrating advancements in zero-knowledge proof systems and other cryptographic techniques to improve proof generation efficiency, reduce on-chain verification costs, and enhance overall system performance.

8 Conclusions

In this paper, we have introduced Bitlayer, a scalable and EVM-compatible computational layer for Bitcoin, whose security is based on an honest majority assumption. Bitlayer is built upon the BitVM paradigm to enable complex, general-purpose computation while anchoring its security directly to the Bitcoin network. Our core contribution is a novel recursive settlement protocol, the first of its kind to allow for the continuous, verifiable settlement of Layer 2 state transitions on Bitcoin. This protocol, combined with a synergistic asset bridge sharing the same security model and a fully EVM-compatible execution layer, creates a complete and practical platform for decentralized applications.

We view Bitlayer as a foundational step towards building the premier infrastructure for the BTCFi ecosystem. By demonstrating a clear architecture where Bitcoin acts as the ultimate settlement layer and Bitlayer as an efficient, verifiable computational layer, our work provides a practical blueprint for unlocking Bitcoin's vast potential. We hope that our design, which prioritizes low transaction costs and strong censorship resistance, encourages further research into scalable and secure applications built upon Bitcoin.

References

[1] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System, 2009. http://bitcoin.org/bitcoin.pdf.

[2] Linus, Robin, Lukas Aumayr, Alexei Zamyatin, Andrea Pelosi, Zeta Avarikioti, and Matteo Maffei. BitVM2: Bridging Bitcoin to Second Layers. https://bitvm.org/bitvm_bridge.pdf.

[3] Robin Linus. BitVM: Compute anything on bitcoin, December 2023. https://bitvm.org/bitvm.pdf.

[4] J. Groth. On the size of pairing-based non-interactive arguments, 2016. https://eprint.iacr.org/2016/260.pdf.

[5] Dan Boneh and Victor Shoup. A Graduate Course in Applied Cryptography. https://toc.cryptobook.us/book.pdf.

[6] Bitcoin Wiki. Script, 2025. https://en.bitcoin.it/wiki/Script.

[7] Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S. M., & Felten, E. W. (2018). Arbitrum: Scalable, private smart contracts. In 27th USENIX Security Symposium (USENIX Security 18). https://www.usenix.org/system/files/conference/usenixsecurity18/sec18-kalodner.pdf

[8] J. Buchmann, E. Dahmen, S. Ereth, A. H¨ulsing, and M. R¨uckert. On the security of the winternitz one-time signature scheme, 2011. https://eprint.iacr.org/2011/191.pdf

[9] Wood, G. (2014). "Ethereum: A Secure Decentralised Generalised Transaction Ledger." Ethereum Project Yellow Paper. https://ethereum.github.io/yellowpaper/paper.pdf.

[10] Rubin, J. (2020). "BIP-0119: CHECKTEMPLATEVERIFY." Bitcoin Improvement Proposals. https://github.com/bitcoin/bips/blob/master/bip-0119.mediawiki.