Stateless and Verifiable Execution Layer for Meta-Protocols on Bitcoin

Hongbo Wen[1,5], Hanzhi Liu[1,5], Shuyang Tang[5], Tianyue Li[2,5]
Shuhan Cao[3,5], Domo[4], Yanju Chen[1], Yu Feng[1,5]
{hongbowen,hanzhi,yanju,yufeng}@ucsb.edu, htftsy@riema.xyz
til062@ucsd.edu, shuhanca@usc.edu, domodata@proton.me
[1]University of California, Santa Barbara
[2]University of California, San Diego, [3]University of Southern California
[4]Layer 1 Foundation, [5]The Nubit Team

ABSTRACT

The Bitcoin ecosystem has continued to evolve beyond its initial promises of decentralization, transparency, and security. Recent advancements have notably been made with the integration of Layer-2 solutions, which address scalability issues by offloading transactions from the main blockchain. This facilitates faster and more cost-effective transactions while maintaining integrity. The advent of inscriptions and ordinal protocols has further broadened the spectrum of capabilities, enabling the creation of unique, indivisible assets on the blockchain. Despite these technological strides, the inherent limitations of Bitcoin's script being Turing-incomplete restrict complex executions directly on the blockchain, necessitating the use of Bitcoin indexers. These indexers act as off-chain execution layers, allowing for the incorporation of Turing-complete programming languages to manage and update state transitions based on blockchain data. However, this off-chain solution introduces challenges to data integrity and availability, compounded by the decentralized nature of blockchain which complicates data maintenance and accuracy.

To address these challenges, we propose a new modular indexer architecture that enables a fully decentralized and user-verified network, mitigating the risks associated with traditional decentralized indexer networks susceptible to Sybil attacks. Our solution, Indecure, leverages polynomial commitments as checkpoints to streamline the verification process, significantly reducing the overhead associated with integrity checks of state transitions. By implementing a robust data attestation procedure, Indecure ensures the reliability of state information against malicious alterations, facilitating trustless verifications by users. Our preliminary evaluations of Indecure across various indexer protocols—BRC20, Bitmap, and satsnames—demonstrate its superiority in reducing computation time and data block size while maintaining high integrity in state transitions. This modular approach not only enhances the security and efficiency of Bitcoin's off-chain executions but also sets a foundational layer for scalable, secure blockchain applications.

1 INTRODUCTION

Blockchain technology continues to captivate the financial and technological sectors with its promise of decentralization, transparency, and security. The Bitcoin [40] (BTC) ecosystem, a pioneer in this realm, has recently made impressive strides, particularly with the advent of Layer-2 solutions like Stacks [9] and CKB [8], which significantly alleviates scalability concerns by processing transactions off the main blockchain, thus enabling faster and more cost-efficient transactions. Furthermore, the introduction of inscriptions [52] and ordinal protocols [3] has opened new horizons for embedding data and creating unique, indivisible assets directly on the Bitcoin blockchain, fostering a new wave of innovation and utility.

Due to the Turing-incompleteness of the Bitcoin script language, complex program logic cannot be executed directly on Bitcoin, which restricts the functionalities of Bitcoin applications. To overcome this shortage, developers leverage *Bitcoin indexers* to establish off-chain execution layers that can execute programs written in mainstream programming languages. Specifically, an indexer first allows users to upload general data and contract code to Bitcoin, where the contract code is written in some Turing-complete programming languages. Based on these codes and data, the indexers maintain a set of states. For each generation of the Bitcoin block according to the Bitcoin transaction order, the indexer executes the user code on the data and updates its state accordingly. Indexers dramatically enhance the accessibility and usability of blockchain data. There are many popular protocols that specify mechanisms of different indexers, such as BRC20 [2], Bitmap [38], runes [45], satsnames [46], and many others. For instance, the BRC20 protocol, which enables users to create and transfer tokens, has kicked off a $1B+ market sector and a new level of growth on Bitcoin.

Despite their utility, existing Bitcoin indexers face data integrity and availability challenges. Specifically, As an off-chain execution layer, the indexer could tamper with the data leading to bogus states for the user [23, 50]. For instance, a bug in the Ordinals protocol has prevented over 1,200 inscriptions from being validated [23]. To mitigate the data integrity problem, the user could download and maintain up-to-date data from the Bitcoin blockchain and verify the validity of the output by executing her indexer. However, doing so will require substantial storage capacity and computing power to manage the ever-expanding blockchain. Additionally, maintaining up-to-date and accurate indexing amidst the blockchain's decentralized and immutable nature poses ongoing challenges.

Due to the integrity and cost of maintenance concerns mentioned above, one native solution is to leverage a *decentralized indexer network* to conduct the computation [31]. However, since the network is completely permissionless, the consensus mechanism of the existing decentralized indexer network is vulnerable to Sybil attacks, which enable malicious indexer operators to provide users with false states, such as asset ownership and spendable balance.

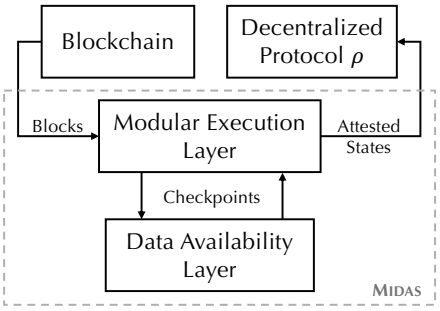

Figure 1: Framework overview.

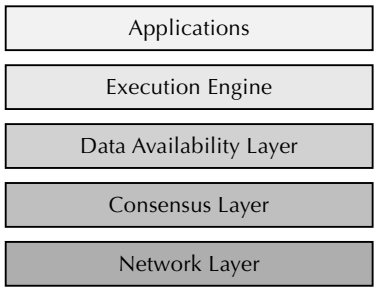

Figure 2: Bitcoin blockchain layers.

To solve this issue, we propose a *modular indexer* architecture that enables a truly decentralized, fully user-verified indexer network. The key challenge here is to design a mechanism that allows users to verify the validity of the states provided by the indexers efficiently and cost-effectively. Our *insight* is based on the following crucial observation: the expensive integrity checking of execution over whole-state transitions can be reduced to checking the validity of a tiny amount of *checkpoints* through the design of proper cryptography protocols.

Our solution. As shown in Figure 1, our modular indexer, INDECURE, works as an off-chain execution layer for the Bitcoin blockchain. Given a decentralized protocol ρ (e.g., BRC20) that interacts with the Bitcoin blockchain by generating and reading transactions, the modular indexer is responsible for the following tasks: ❶ reading each block from Bitcoin and calculating protocol states based on the logic specified by a decentralized protocol ρ, ❷ summarizing these states as a polynomial commitment, namely, *checkpoint*, ❸ For every new Bitcoin block, generate a new checkpoint and publish it to the data availability (DA) layer [41], ❹ retrieving the checkpoint from the DA layer and verifying the correctness of the states. With the proposed attestation procedure, INDECURE can ensure state integrity against malicious attacks with negligible cost for different applications. Here, an off-the-shelf DA layer ensures that transactions on a blockchain are readily accessible to all nodes in the network, which can prevent problems such as data withholding attacks. Ultimately, our system aims to provide fully user-verified modular execution generalizable for different blockchain meta-protocols ρ. As we show later in our technical section, under the 1-of-N trust assumption, the light clients in INDECURE will never accept invalid states from malicious participants, thus dramatically improving the robustness and security of existing indexers and decentralized protocols on the Bitcoin ecosystem.

Evaluation. We evaluate INDECURE on three popular implementations of indexer protocols, namely, BRC20, Bitmap, and satsnames. Through a production testnet with 211,503 transactions and over 5,000 light clients, we confirm the effectiveness of our architecture, which dramatically outperforms a stateful baseline in terms of computation time as well as the size of data blocks. Additionally, we simulated a recent incident due to invalid states from one indexer of the BRC20 protocol, and INDECURE can effectively compute the correct states and report the violation caused by incorrect states.

Contributions. In summary, we make the following contributions:

- We propose INDECURE, the first modular indexer architecture that enables trustless verification in light clients on Bitcoin.
- We propose a decentralized procedure for data attestation with 1-of-N trustless assumption. The attestation procedure guarantees the identification or recovery of states.
- We implement the proposed idea and systematically evaluate its effectiveness through real-world applications.

2 BACKGROUND

In this section, we survey some basic concepts necessary to understand our system's technical details, followed by a realistic threat model to quantify adversarial behavior in our environment.

2.1 Bitcoin

The idea of Bitcoin was originally introduced by Satoshi Nakamoto [40]. Specifically, the Bitcoin blockchain operates through a network architecture comprised of several layers shown in Figure 2. At the base, the *network layer* manages peer-to-peer communications among nodes, facilitating the seamless transaction propagation and block data across the global network. This layer ensures that all nodes can participate in the network equally and are updated with the latest blockchain state, maintaining Bitcoin's foundational principle of decentralization. The *consensus layer*, primarily powered by Proof of Work (PoW), allows nodes to agree on a single history of transactions, thus securing the network against fraud and double-spending. This layer coordinates consensus across the diverse network of nodes and aligns economic incentives through mining rewards, integrating robust security with a self-sustaining economic model that underpins the entire Bitcoin system. On top of that, the *data availability* layer ensures that all transaction data necessary to validate the blockchain's current state is accessible to any nodes. This transparency is vital for the security and reliability of the network, allowing independent verification of the blockchain without reliance on a central authority. Moving upward, the *execution layer* processes transactions according to the rules set out in Bitcoin's protocol, ensuring that all transaction conditions are met and that the ledger's integrity is preserved. In spite of the Turing-incompleteness of Bitcoin script language, the *application layer* can still simulate complex computations through the assistance of *indexers*.

2.2 Data Availability

Data availability [12] refers to the assurance that data related to transactions on a blockchain is readily accessible to all nodes in the network, enabling them to validate transactions and maintain the network's integrity. This aspect is crucial in preventing problems such as data withholding attacks, where some participants might hide data to manipulate the network's state or to disrupt the consensus process. The importance of robust data availability solutions has grown with the scaling of blockchains, as larger blocks and more transactions intensify the demand on nodes to process and store data efficiently. For instance, Layer-2 rollups execute transactions outside the main blockchain (Layer-1) but post transaction data back to it. Rollups could benefit greatly from robust data availability solutions because they rely on L1 for data validation and security.

2.3 Polynomial Commitment and Verkle Tree

Polynomial commitments. Polynomial commitments are cryptographic primitives that enable a party to commit to a polynomial in a way that allows the committer to later prove properties about the polynomial (such as its value at certain points) without revealing the entire polynomial. These commitments leverage algebraic properties to compress the polynomial's representation and provide proofs of evaluation that are both succinct and easily verifiable. This makes them particularly valuable in blockchain environments and other cryptographic systems where data integrity, privacy, and scalability are crucial, such as in zero-knowledge proof protocols where they help verify complex computations without compromising the privacy of the underlying data.

Verkle Tree. INDECURE utilizes the *Verkle tree* [4, 18] as the data structure for storing checkpoints. The Verkle Tree is a variant of the Merkle Tree, featuring more branching and shallower depth. Its nodes use polynomial commitments instead of the results of cryptographic hash functions as summaries of the child nodes. The Verkle Tree treats the root node as the commitment for the entire tree (also referred to as the checkpoint mentioned earlier), meaning the prover, unlike with a Merkle Tree, does not need to provide all the "sibling nodes" hash values from the root to the leaf node. It only needs to provide all the nodes on the path from the root to the leaf node. Thus, the proof size of the Verkle Tree is smaller than that of the Merkle Tree, especially when users query multiple leaf nodes.

2.4 Indexer

Due to the Turing-incompleteness of the Bitcoin script language, complex program logic cannot be executed directly on Bitcoin, which restricts the functionalities of Bitcoin applications. To overcome this shortage, developers leverage *Bitcoin indexers* to establish an off-chain execution layer that is Turing-complete.

As shown in Figure 3, an indexer is a specialized component that reads and processes data from the Bitcoin blockchain and returns the computation results to clients. The fundamental purpose of an indexer is to enhance the efficiency and usability of blockchain data through off-chain turing-complete computations. Specifically, Bitcoin indexers work by scanning the blockchain, extracting code and data, and maintaining a set of states. For each generation of

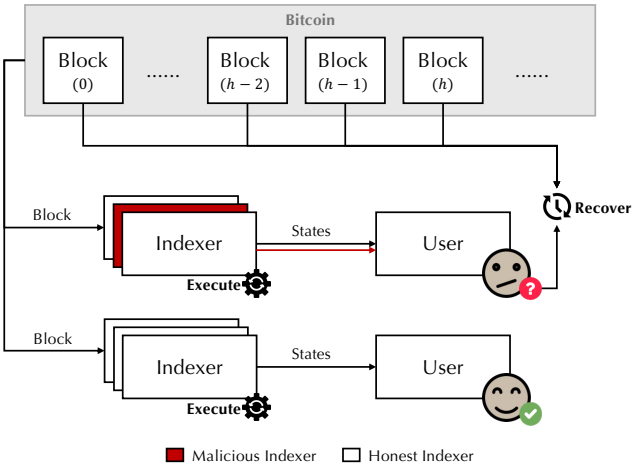

Figure 3: A strawman system for verifiable execution.

Bitcoin blocks according to the Bitcoin transaction order, indexers execute the code on the data and update the state accordingly. Indexers dramatically enhance the accessibility and usability of blockchain data. Because as Bitcoin evolves to include more complex features like smart contracts and layer 2 solutions, the role of indexers becomes even more pivotal. They enable users to verify the validity of outputs in a trustless setup, reinforcing the security and decentralization that are hallmarks of the Bitcoin network. This kind of innovation is crucial as it supports the growing array of applications on Bitcoin, transforming it into a more robust platform for decentralized finance.

In spite of the crucial roles of Bitcoin indexers in querying Bitcoin blockchain data efficiently, face several security concerns. ❶ *Data Integrity*: Indexers must ensure the data they retrieve and store from the blockchain is accurate and unaltered. Manipulated data can mislead users about transaction histories, balances, or smart contract outcomes. ❷ *Privacy*: Indexers often accumulate detailed transaction histories that could be exploited to track users' financial behaviors. Ensuring data is handled in ways that protect user anonymity and financial privacy is crucial. ❸ *Centralization Risks*: Many indexing solutions rely on centralized servers, creating potential single points of failure. These centralized components can be targeted by attackers aiming to disrupt service or steal data. ❹ *Sybil Attacks*: as shown in Figure 3, even with a decentralized network of indexers, due to lack of slashing mechanisms, attackers could still control the majority of the nodes and create misleading states of the computation, potentially affecting trading decisions and market movements.

2.5 A Strawman System for Verifiable Execution

As we mentioned in the previous section, a decentralized network composed of Bitcoin indexers still cannot prevent Sybil attacks because its consensus layer lacks a penalty mechanism and cannot punish malicious actors, which could lead to serious security concerns that compromise data integrity. To address this problem, Figure 3 shows a strawman system that allows users (i.e., light

clients) to verify the correctness of the computation. Specifically, in a completely trustless setup where a malicious indexer could return bogus states, the user could still rely on the economic security of Bitcoin mainnet and mitigate this problem through two steps: ❶ download code and data from all relevant blocks (i.e., from $Block_0$ to $Block_h$), and ❷ Execute each block data locally until recovering the final result of the current state. In this system, transaction order is guaranteed by the Bitcoin blockchain, and for a given decentralized protocol, the data and code on the block are interpreted and executed at the light client side.

Caveat. Even though downstream applications could directly obtain states from indexers, when it comes to discrepancies between different states for the same block, applications are forced to re-index the entire history of blocks. In other words, despite adding an execution layer to the blockchain, the strawman system relies on light clients to resolve conflicts, which leads to a huge burden to end users in terms of computation, storage, and response time.

Threat model. We outline a realistic threat model that forms the basis of our work. The primary threats include adversarial capabilities such as network-level attackers capable of intercepting or altering data, internal threats from users with system access, and external hackers targeting software vulnerabilities. These adversaries aim to disrupt service operations, manipulate data for fraudulent purposes, or illicitly access sensitive information. The system faces risks from various attack vectors including denial-of-service attacks, Sybil attacks to undermine the indexer network, spoofing to falsify identities, and exploitation of potential vulnerabilities both within the Bitcoin protocol and the indexer's architecture.

3 MODULAR EXECUTION LAYER

A critical limitation of the strawman system described in Section 2.5 is the lack of a mechanism that ensures data integrity and availability. In particular, to recover the actual under the trustless setup, the strawman system will rely on light clients to recompute the current states by accessing the blockchain's full data history. Such a recovery process could be slow and error-prone, leading to a significant limitation on the efficiency and reliability of blockchain computations. To mitigate this, we introduce *modular execution layer*, a stateless and verifiable execution layer for meta-protocols on Bitcoin.

The section is organized as follows: Section 3.1 presents an overview of the proposed architecture with extended formulation upon the strawman system; Section 3.2 introduces a key component called *modular indexer* for computation and storage; Section 3.3 introduces another key component called *light client* for validation and recovery, as well as its corresponding attestation algorithm for ensuring data integrity.

3.1 Formulation and Architecture Overview

We start by re-iterating the *1-of-N trust assumption* [51] that Indecure applies upon. In a 1-of-N setup, there are many actors, and the system will function as expected as long as at least one of them behaves honestly. Any system based on fraud-proofs falls into this category. We further extend this assumption in the context of Bitcoin whose execution layer leverages indexers.

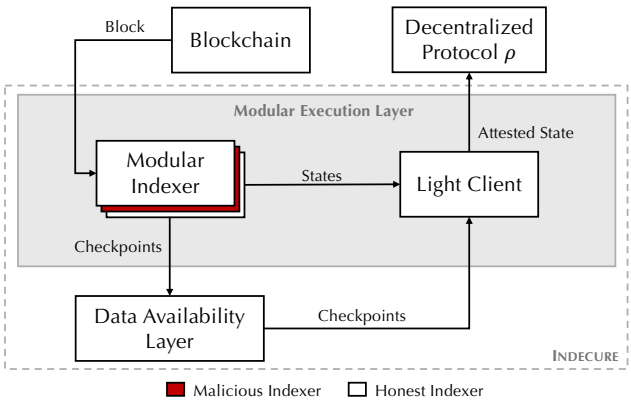

Figure 4: Framework overview.

Assumption 1 (1-of-N Trust Assumption). *For a block at a given height, there is at least one honest indexer that always returns the correct state.*

Definition 1 guarantees that for the execution of each block, the correct state *always* exists among the set of all obtainable (potentially bogus) states from the indexer network. Following the assumption, we can still assure that if all indexers reach the same state for a block, the state is correct; otherwise, divergence needs to be resolved to reach a consensus. Unlike the base system that falls back to re-play of blockchain history, Indecure addresses and resolves the divergence with a checkpoint mechanism that saves, validates and recovers the correct states from within a minimal window of history.

Figure 4 shows the high-level architecture of Indecure and the workflow for generating an attested state. While Indecure can incorporate with various integrity validation techniques, for a concise presentation, we use hash tree techniques[1] for integrity validation by default in our discussion below. In particular, Indecure consists of two layers, namely the modular execution layer and data availability layer. There are two key components in the modular execution layer:

- The *modular indexer* computes and stores states by retrieving and executing blocks from the Bitcoin blockchain. In addition to states, a modular indexer also generates extra data used for state validation called *checkpoints*.
- The *light client* is responsible for state validation and recovery. When a divergence of states happens, the light client recovers the correct one by an *attestation* algorithm with the help of checkpoints.

The data availability provides storage and indexing of checkpoints generated by the modular indexer, making sure of their accessibility to the light clients.

Therefore, for a given block and decentralized protocol, Indecure invokes the following two steps to generated an attested state for this block:

- Step 1 (State computation and storage): Each modular indexer from the network first synchronize with the blockchain

[1]This includes techniques such as Merkle tree, Verkle tree, etc.

by pulling the latest block, and then execute the target decentralized protocol to generate a state. In addition, a checkpoint for the current state is also computed via polynomial commitment algorithm and published to the data availability layer.
- Step 2 (State validation and recovery): A light client obtains states and published checkpoints, and performs an attestation procedure to generated the correct state. Specifically, the attestation procedure resolves potential discrepancy between states proposed by modular indexers by identifying the successfully validated state-checkpoint pair.

In what follows, we discuss these two steps in detail.

3.2 State Computation and Storage

Given a block at height h (denoted by B_h), and a decentralized protocol ρ, the execution (denoted by execute$(\cdot)$) of the data on B_h according to ρ corresponds to the transition from the parent state of height $h-1$ (denoted by S_{h-1}) to the current resulting state S_h:

$$\mathsf{execute}(\rho, B_h, S_{h-1}) \rightarrow S_h.$$

Since such computation happens locally on indexers, a malicious indexer can forge computational results thus creating a bogus state S'_h and submit when queried by other clients from the network.

Modular indexer and checkpoint. As a malicious indexer could cause discrepancy of states and there's no effective way to identify or revoke the bogus states in a base system, Indecure extends the notion of indexer to a new kind called modular indexer that generates an additional "proof" for validating the state proposed by itself.

Such a proof is also referred to as a *checkpoint*. A checkpoint at height h (denoted by C_h) usually takes the form of polynomial commitment, and can be computed by taking into account of different states and checkpoints in the history, e.g.,:

$$\mathsf{checkpoint}(S_{h-1}, S_h, C_{h-1}) \rightarrow C_h,$$

where the current and parent states S_h and S_{h-1}, as well as the parent checkpoint C_{h-1}, are used for checkpoint computation (denoted by checkpoint$(\cdot)$. The resulting checkpoint C_h recursively encodes the history of parent states and checkpoints, which can then be later used by third parties for validation of state integrity. As state validation is closely related to the attestation procedure, we defer the discussion to Section 3.3.

Revisiting the trust assumption. As a modular indexer proposes both states and checkpoints, following Definition 1, when queried for state and checkpoint at certain block height, modular indexers can behave in three different ways:

- *Fully honest.* Both the state and checkpoint proposed are correct.
- *Half-half.* Either only the state or checkpoint is correct.
- *Fully malicious.* Both the state and checkpoint are incorrect.

Of the above types, an incorrect state or checkpoint will mark the indexer as malicious. Therefore, we can extend the notion of honesty to the context of modular indexer.

Corollary 1 (Honest Modular Indexer). *A modular indexer is considered honest at certain block height, if it proposes both correct state and checkpoint; otherwise it's considered malicious.*

Following Corollary 1, only a fully honest modular indexer is considered honest, and the other two types of indexers are considered malicious. We base the follow-up discussion upon this corollary.

Data availability and protocol namespace. To enable data accessibility and robustness, Indecure incorporates a data availability layer where modular indexers can publish the computed checkpoints. The data availability layer assures that data related to transactions on a blockchain is readily accessible to all nodes in the network and can be retrieved efficiently by light clients.

As Indecure is compatible with different decentralized protocols, each protocol establishes its own isolated storage that we call *namespace*. Indexers executing the same protocol share the same namespace.

3.3 State Validation and Recovery

As malicious indexers contribute mainly to state and checkpoint generation, clients that consume the state for down-stream tasks can skip the heavy-lifting work of computation most of the time when the states obtained from various sources are consistent. In the scenario where discrepancy between states appears, the client must resort to a further attestation procedure that resolves a correct state, before handing over to any down-stream tasks.

Light client and state proof. In Indecure, we extend a regular indexer to a special kind called *light client*, which validates states with their corresponding checkpoints, and recovers the correct versions if state discrepancy happens. A light client usually starts by obtaining a set of states (denoted by and checkpoints from the modular indexers and data availability layer respectively.

The validation of the integrity of a state S_h with its corresponding checkpoint C_h can be done by:

$$\mathsf{verify}(C_h, \pi_h),$$

where verify$(\cdot)$ corresponds to the validation operator, and the proof π of the hash tree is given by:

$$\mathsf{proof}(S_h) \rightarrow \pi_h,$$

where proof$(\cdot)$ is the corresponding hash tree proof generator[2]. Depending on the validation result, the light client follows a standard attestation procedure to resolve potential discrepancy among states.

The attestation procedure. Algorithm 1 describes the procedure for a light client to generate attested state and checkpoint at given block height h. In particular, attested states and checkpoints are guaranteed to be correct regarding a given protocol ρ, and can be obtained by the following three ways:

- *Attested by initialization* (line 5). When queried height is 0 (line 4), an empty state $\varnothing$ and checkpoint `0x00` are returned.

[2]In real-world practice, to reduce the workload of light clients, proof computation is usually off-loaded to and provided by modular indexers rather than light clients.

Algorithm 1 Attestation Procedure

1: **procedure** ATTEST(ρ, B, h)
2: **Input:** Protocol ρ, Blockchain Data B, Height h
3: **Output:** Attested State $S^\diamond$ and Checkpoint $C^\diamond$
4: **if** $h = 0$ **then** ▷ base case
5: **return** $\emptyset$, 0x00 ▷ initialized state and checkpoint
6: **else** ▷ inductive case
7: $C_h \leftarrow$ QUERYCHECKPOINTS(ρ, h)
8: **if** $\{C\} = C_h$ **then** ▷ checkpoints are consistent
9: $\mathcal{S}_h \leftarrow$ QUERYSTATES(ρ, h)
10: **for each** $S^{(i)} \in$ choose($\mathcal{S}_h$) **do**
11: $\pi \leftarrow$ proof($S^{(i)}$)
12: **if** verify(C, π) **then return** $S^{(i)}, C$
13: **return** $\perp$ ▷ error state
14: **else** ▷ checkpoints are inconsistent
15: $S_{h-1}, C_{h-1} \leftarrow$ ATTEST($\rho, B, h-1$)
16: $S_h \leftarrow$ execute(ρ, B_h, S_{h-1})
17: $C_h \leftarrow$ checkpoint(S_{h-1}, S_h, C_{h-1})
18: **return** S_h, C_h

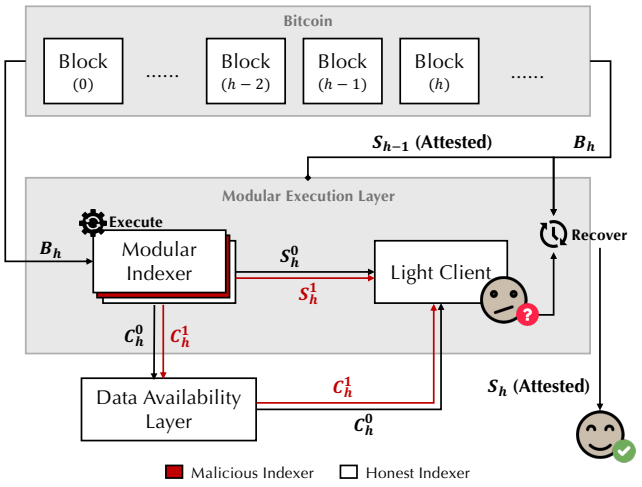

Figure 5: An example of state recovery in INDECURE.

At regular queried block eights (line 6), a set of checkpoints is first obtained from the modular indexers (line 7), and based on the status of these checkpoints, different methods of attestation are invoked:

- *Attested by state selection* (line 8-13). If all the checkpoints obtained are consistent (line 8), according to Definition 1, this consistent checkpoint is guaranteed to be correct. The algorithm then obtains a set of states (line 9) and iterates over them to find out the correct one using the already attested checkpoint (line 10-14). In particular for each state $S^{(i)}$ (line 10), a proof π is first generated (line 11) and paired with the attested checkpoint for verification (line 12). If the verification succeeds (line 12), then the current state $S^{(i)}$ proves correct and is returned with the attested checkpoint; otherwise, the algorithm moves to the next available state until the correct one is found[3].
- *Attested by checkpoint recovery* (line 15-18). If the checkpoints obtained are inconsistent (line 14), a recovery mechanism is then invoked. In particular, the algorithm first delegates the resolution of an attested state and checkpoint of the previous block height $h-1$ to itself in recursion (line 15). Then starting from the attested state of previous block height S_{h-1}, the light client proceeds with execution of block B_h on state S_{h-1} for protocol ρ (line 16), which produces the current state S_h. Since S_{h-1} and C_{h-1} are both attested, S_h is also attested and can be used to derive its corresponding attested checkpoint C_h (line 17) and return (line 18).

Example 3.1. Figure 5 shows an example of state recovery in the presence of inconsistent results. Here, the checkpoints at block height $h-1$ of modular indexers are consistent and attested, but the checkpoint submitted by the committee indexer set selected by users at height h is inconsistent. Without loss of generality, we assume there are only two inconsistent checkpoints. A malicious indexer provided the false checkpoint C_h^1 while the honest indexer provided the true checkpoint C_h^0. Upon discovering the inconsistency between S_h^0 and S_h^1, the light client will undergo the following steps for stateless computation to generate the correct checkpoint, thereby identifying the honest indexer and removing malicious ones from the indexer network.

- **Verify** S_{h-1}: The light client first verifies whether the proof π is correct through Verify(C_{h-1}, π).
- **Generate** S_h: The light client obtains the transactions that need to be executed from the current block h and data V from the parent block's state S_{h-1} required for executing these transactions. Subsequently, the user executes transactions on V, calculating the post-execution state V'. Note that the light client does not need to request the full state (saving the full state requires hundreds of GB of storage space) but calculates from a subset of the full state, containing only those read and written by transactions in the current block h. Also, only one honest indexer needs to provide S_{h-1} as well as the proof π for the transaction execution.
- **Verify Checkpoint** C_h: After verifying the correctness of S_{h-1} and calculating the S_h, the light client can calculate the polynomial commitment of the complete state, as well as the values of the leaves that need to be updated. This means the light client can recover the current block's Verkle Tree T_h from S_{h-1}, S_h and the parent block's checkpoint C_{h-1}, even though it only knows a tiny part of the leaves of the tree, we can still generate the entire tree's checkpoint C_h. Thereby, the light client can determine the correctness of C_h^0 and C_h^1 by comparing these checkpoints with the generated checkpoint C_h, thus identifying the attacker and removing it from the indexer network.

[3]Note that Definition 1 and Corollary 1 ensures at least one of the states is correct. So Algorithm 1 is guaranteed to attest a state and checkpoint without reaching the error state (line 13), which we add for the algorithm's syntactic correctness.

Definition 1 ensures that, as long as there's at least one honest modular indexer, the attested state and checkpoint returned by Algorithm 1 from the light client will always be correct.

Security. Our security goal is to ensure that a user is always able to retrieve the correct state at an arbitrary height. To show this, we proceed with the analysis as follows.

(1) *An honest indexer holds the correct states.* The correctness of local states of an honest indexer is shown by an induction. Assuming secure bootstrapping and the correctness of S_{h-1} and C_{h-1}, $\mathsf{execute}(\rho, B_h, S_{h-1}) \rightarrow S_h$ and

$$\mathsf{checkpoint}(S_{h-1}, S_h, C_{h-1}) \rightarrow C_h$$

deterministically return the correct S_h and C_h.
(2) *At least one honest indexer returns the correct state.* From our 1-of-N trustless assumption, there is always one indexer returning a correct state with a valid proof.
(3) *The user is capable of distinguishing the correct state from forged ones with a proof.* This is guaranteed by the Verkle design, which is based on the binding property of its underlying KZG-based vector commitment. This is shown by induction. For a path of Verkle tree nodes $u_0, u_1, \ldots, u_\ell$ (u_0 is the Verkle root), for any integer i satisfying $0 < i \leq \ell$, u_{i-1} contains the correct vector commitment for a vector including u_i at the desired index. In the other case, it violates the binding property of the vector commitment. If u_{i-1} is the correct Verkle node, then u_i is unforgeable.

Optimization. Additionally as an optimization, since the attested states and checkpoints are guaranteed correct, a light client can then identify and block malicious indexers by checking their proposed states and checkpoints with the attested ones. As an example, the choose operator (line 10) can be programmed to speed up the attestation procedure by selectively skipping malicious indexers (and promoting honest indexers) each time an attestation is required.

3.4 From Stateful to Stateless Computation

Recall that Section 3.2 introduces a *stateful computation* that computes a new checkpoint from three inputs, namely, the current state S_h, the parent state S_{h-1}, and the parent checkpoint C_{h-1}. As we demonstrate later in the evaluation, this approach, while correct, faces significant scalability issues. Because the light client has to retrieve block states that are increasingly growing.

Given the impracticality of handling the full state due to its massive size, often in the hundreds of GB, the light clients should only request a tiny subset of the full state necessary for the *current* transaction execution. Using the properties of Verkle Tree, our key insight is to compute the full state's checkpoint from a subset of states that are modified, effectively turning the original stateful computation into its stateless version.

Definition 1 (Critical State). *For a block at a given height h, a critical state δ at h is a subset of all Keys $\{K_0, K_1, ..., K_m\}$ that are either read or written by the transactions in block h.*

Therefore, we propose an effective and stateless computation for checkpoints based on critical states:

$$\mathsf{checkpoint}^*(\delta, C_{h-1}) \rightarrow C_h,$$

which dramatically reduces the amount of data on the light clients.

4 IMPLEMENTATION

We implemented the proposed ideas in a tool called Indecure, which contains two components, namely, the **modular-indexer** and the **light-clent**. In particular, the modular indexer's main branch comprises 3,964 lines of code and the light client's main branch contains 3,687 lines of code. Both components are developed in Golang, which is an ideal language for blockchain-related programming involving concurrent operations and low-level memory management.

4.1 Optimization Strategies

In addition to the main ideas in our technical sections, our implementation incorporates several optimizations to enhance performance and reliability in a Bitcoin blockchain environment.

Handling Bitcoin Reorganizations. One of the key challenges in working with Bitcoin blockchain is handling reorganizations, where recent blocks might be replaced, requiring a rollback of state changes. To address this, we've implemented a novel approach using an array structure to store state differences (`statediffs`) at each block height. This method allows for efficient rollbacks up to six blocks, aligning with Bitcoin's maximum reorganization depth.

By preparing for reorganizations, modular indexer system can quickly adjust to changes in the blockchain without extensive recalculations. Meanwhile, instead of storing entire state histories or multiple versions of the Verkle Tree, only the differences are stored, significantly reducing the storage overhead.

Verkle Tree Storage Optimization. Further optimizations were implemented in the storage mechanism of the Verkle Tree, particularly in handling key-value (KV) pairs with large values. Given the constraints of the Verkle Tree structure where large values might span multiple slots, we optimized the storage of such KV pairs by modifying their key encoding.

The key alteration involves appending an increment to the last bit of keys that are associated with large values. This method ensures that these values are split and sequentially stored within the same leaf node.

This optimization offers several advantages:

(1) Computational Efficiency: By storing related large-value KV pairs in the same leaf node, we minimize the computational overhead of generating multiple commitments for disparate nodes.
(2) Space Optimization: Sequential storage reduces the need for additional space that would otherwise be required to store metadata about node locations or additional leaf nodes, thus minimizing space wastage and maximizing the use of allocated space.

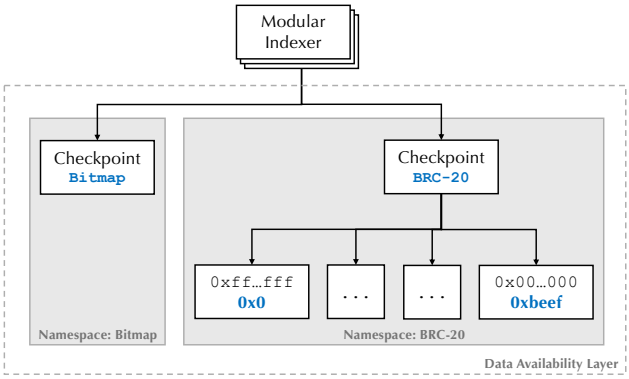

Figure 6: Integration with meta-protocols.

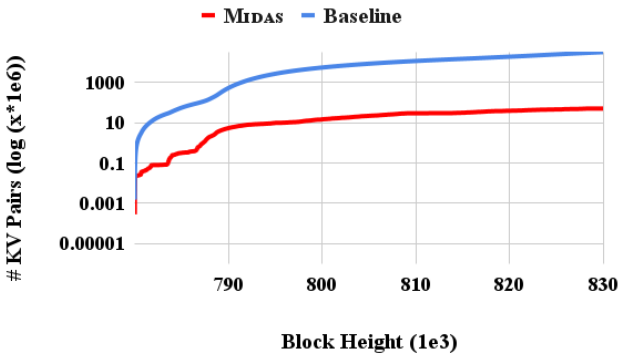

Figure 7: Comparison between Indecure and the base system on the cumulative state size required for proof generation at each block.

Honesty Tracking. To reduce overhead and data transmission size during state attestation procedure, Indecure tracks the honesty status for every modular indexer it has interacted with, which we refer to as *honesty tracking*. As an optimization, honesty tracking happens after a state is successfully attested by a light client. By maintaining a white list of modular indexers, Indecure can effectively choose to only prioritize or block certain indexers, thus reducing communication overhead with those tagged as malicious.

4.2 Integration with Meta-protocols

We further elaborate on how to integrate Indecure with existing meta-protocols:

- **Modular Indexer**: The indexer independently stores and indexes each meta-protocol, thus calculating and publishing checkpoints for each meta-protocol. For example, if an indexer chooses to serve both BRC-20 and Bitmap protocols, it needs to publish the current state commitment tuple:

  $$\langle \text{blockheight } h, \text{blockhash } p, \text{checkpoint } C \rangle$$

  separately to the DA Layer for BRC-20 and Bitmap. Here, checkpoints from different meta-protocols are stored in different *namespace* in the DA layer.
- **Meta-Protocols**: On each ⟨block height h, block hash p, checkpoint⟩, the indexer maintains a Verkle Tree storing all the state variables of the meta-protocol. This Verkle tree uses a 32-byte storage identifier as the key and the current value of the state variable as the value. The meta-protocol is responsible for defining how to generate the storage identifier and ensure its uniqueness. For example, in the BRC-20 meta-protocol, a state variable that needs to be maintained is the user's current available balance under a certain instance. Therefore, the meta-protocol needs to define how each user's available balance under each BRC-20 instance is mapped/hashed to a unique storage identifier.

In terms of state correctness, as shown in Figure 6, any changes in a state variable will lead to a change in the meta-protocol checkpoint. Upon detecting inconsistency in the meta-protocol checkpoints, the light clients can verify and determine the trustworthy indexers through the verification process described in Section 3.2.

5 EVALUATION

In this section, we describe the results of the experimental evaluation of Indecure. Our evaluation is designed to answer the following key research questions:

- **RQ1 (Effectiveness):** Is Indecure's trustless infrastructure with stateless computation and attestation effective?
- **RQ2 (Generality):** Can Indecure be effectively generalized to different protocols?
- **RQ3 (Robustness):** Can Indecure prevent real-world attacks caused from malicious nodes in the network?

Evaluation setup. Indecure is deployed within AWS cloud environment for testing and evaluation purpose, with four machines each running a equal number of modular indexers. These indexers are deployed on compute instance of type "R6i 4xlarge", and operate in Ubuntu 22.04 operating system with 16 vCPUs, 128 GiB of memory and 2TB SSD storage.

We instantiate a state in the system as a set of key-value pairs. Each modular indexer node starts processing at the Bitcoin block height of 779,832, which is the height marks the initial deployment of inscription on the Bitcoin blockchain. Processing has continued up to a block height of 830,000 at the time of reporting. This corresponds to a total of 50,168 blocks processed.

All evaluations are conducted on a private testnet forked from Bitcoin blockchain, with large-scale real-world user participation.

5.1 Comparison on Effectiveness

We evaluate the effectiveness of Indecure from two perspectives, namely transmission size and system recovery time.

Data size in network transmission. As the size of data required for communication between nodes of a system in the network is an important indicator to measure a system's effectiveness, we measure the total size of data transmitted between nodes during system runtime for `BRC20` protocol.

Figure 7 shows the cumulative number of key-value pairs transmitted between nodes as the block height grows. Compared with

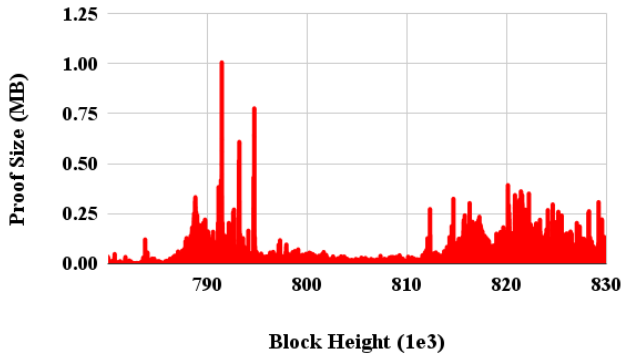

Figure 8: Size of generated proof in INDECURE as block height grows.

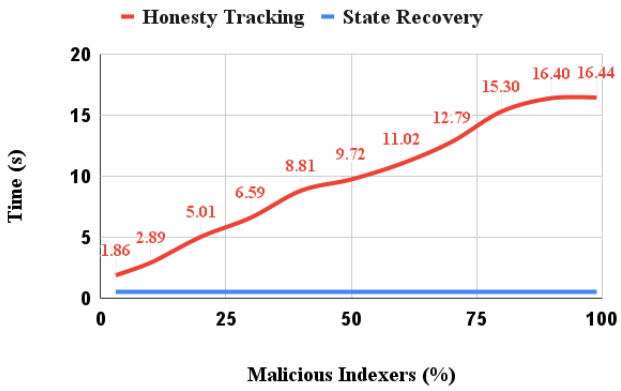

Figure 9: System recovery time in the presence of malicious indexers.

the base system whose data transmission cost grows nearly exponentially with an average of 650k pairs, INDECURE's growth is close to linear with an average of 1k pairs, demonstrating a strong evidence of the effectiveness its stateless computation design as INDECURE saves more than 99% of required data during computation and validation. Even though INDECURE requires an additional proof on top of a checkpoint for state validation, as shown by Figure 8, INDECURE only needs an average of 0.03MB of proof for state validation. In addition, the proof size does not grow as the total size of the entire state grows, due to the design of critical state that only keeps track of changes rather than a state's entire data structure.

System recovery time. Besides data size in network transmission, we further measure the time spent for a light client to attest the current state and recognize honest indexers when inconsistencies between states and checkpoints are detected, which we denote as *system recovery time* T_r:

$$T_r = T_s + T_h$$

where T_s denotes time spent on state recovery and T_h denotes time spent on honesty tracking. System recovery time evaluates a network's effectiveness in resolving state inconsistencies.

We conduct the experiments in a network with in total 100 modular indexers. We then measure the system recovery time of INDECURE under different levels of trustless assumption, ranging from a preset of 1% to 99% malicious indexers in the network. We assume that malicious indexers only appear in the current block height that the client queries. Figure 9 shows the distribution of the system's state recovery time and honesty tracking time as the percentage of the network's malicious indexers grows. As we can see, INDECURE's state recovery time is a constant (0.5s) since the client needs to query the state at the last block height in order to compute and recover the correct state, which is agnostic to number of malicious indexers. That is, INDECURE can terminate immediately regardless of degree of inconsistency. For honesty tracking, since the client needs to compare the correct state with all the states proposed by different indexers, the honest tracking time naturally grows in a linear way.

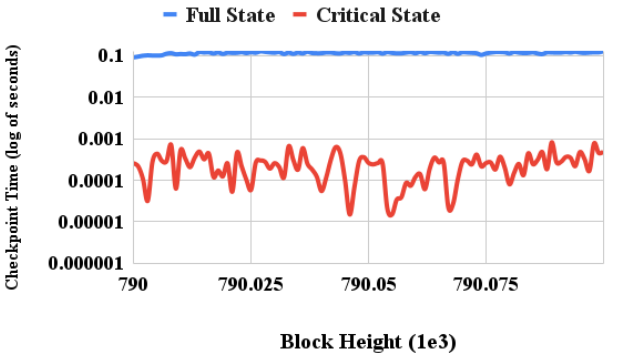

Figure 10: Checkpoint time.

Result for RQ1: INDECURE transmits significantly smaller sizes of data as block height grows, demonstrating a more effective way of stateless computation over the base system.

5.2 Ablation Study

To find out how the stateless computation in INDECURE contribute to its performance, we conduct a ablation study. In particular, we create an ablative version of the system that we denote as INDECURE^o which enables stateful instead of stateless computation. That is, we remove the computation of critical state in INDECURE^o, and it only relies on full states for checkpoint computation. Figure 10 shows a comparison between the full-fledged version INDECURE and the ablative version INDECURE^o on the time spent for checkpoint computation. As we take logarithm of the reported time, we can clearly see that, with critical state incorporated, INDECURE is about 100× faster than INDECURE^o, indicating that stateless computation is effective in saving both data in network transmission and complexity of checkpoint computation. We believe this also provides an evidence towards answering RQ1 in a positive way.

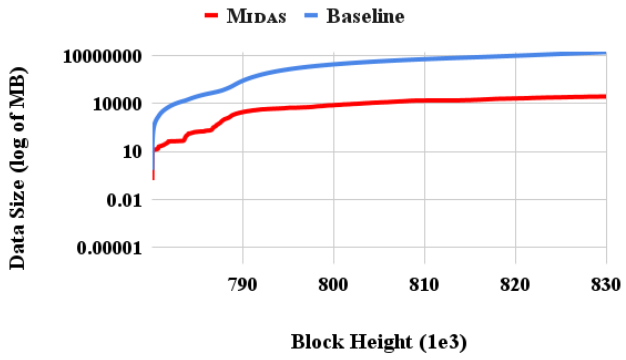

Figure 11: Comparison between Indecure and the base system on the cumulative data size transmitted for proof generation as block height grows when running `BRC20` protocol.

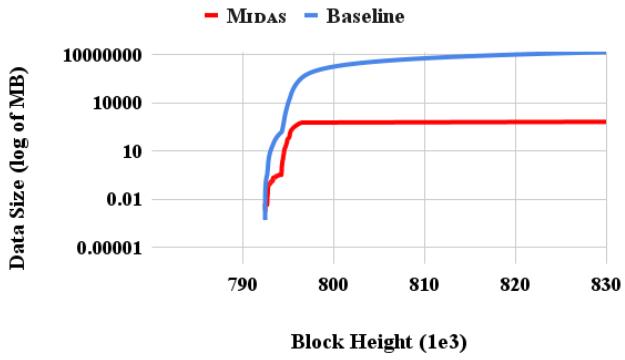

Figure 12: Comparison between Indecure and the base system on the cumulative data size transmitted for proof generation as block height grows when running `bitmap` protocol.

5.3 Generalizing for Different Protocols

As the trustless computation architecture of Indecure does not assume any preconditions of the protocols applied, besides `BRC20`, we also deploy other two representative decentralized protocols on top of Indecure:

- `Bitmap` is a consensus standard that allows anybody to claim the geospatial digital real estate of a Bitcoin Block, which takes advantage of the nature of data's unique ability to be parsed from multiple angels.
- `SatsNames` is a standard for writing human-readable Web3 usernames to Bitcoin using ordinals. The goal is to build a name ecosystem for Bitcoin, that is built by Bitcoiners, and developed entirely on Bitcoin.

Figures 11 to 13 show the comparisons between Indecure and the base system for running the three protocols `BRC20`, `bitmap` and `satsnames` respectively. The three figures all demonstrate a consistent and significant trend of better cost effectiveness of Indecure over the base system.

Result for RQ2: Indecure's computation and validation architecture generalizes to different protocols with consistent cost effectiveness.

5.4 Case Study: Mitigation of Real-World Attacks from Malicious Indexers

In this evaluation, we simulate Indecure in the context of a real-world incident [50] caused by inconsistent states from different indexers.

The problem: On Nov 26th, 2023, the Bitcoin native wallet, UniSat, returned two inconsistent results for one of the user's address [4]. There appeared to be a major discrepancy in the supply of BRC-20 on Binance due to this issue. Binance holds around more ORDI ($100m) than what should exist. In particular, three indexers (i.e., UniSat, OKX, and Ordinalscan) return a balance of 7,250,285.19634297 $ORDI while the fourth indexer (i.e., ordiscan) returns a balance of only 2,304,693 $ORDI.

[4] bc1qhuv3dhpnm0wktasd3v0kt6e4aqfqsd0uhfdu7d

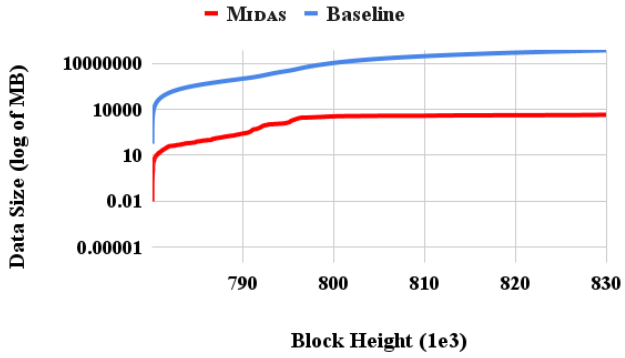

Figure 13: Comparison between Indecure and the base system on the cumulative data size transmitted for proof generation as block height grows when running `satsnames` protocol.

Root cause: It turns out that the balance from the last indexer (i.e., ordiscan) is incorrect due to mishandling an edge case in decimals.

Simulation: Together with regular and benign nodes, we replayed the incident by deploying a "malicious indexer" that deliberately returns the wrong balance to light clients. After that, we monitored the behavior of light clients and see whether any of them will accept the wrong balance from the malicious clients. Through our entire testnet phase that lasted for one week, all light clients were able to accept the correct balance as expected.

Result for RQ3: Indecure is effective in mitigating real-world attacks from malicious indexers.

6 RELATED WORK

Scaling Bitcoin through execution layers and data storage layers is an emerging research topic. In what follows, we discuss prior work that is most closely related to our proposed approach.

Light clients. The most related topic of our design was the light client. Light clients enabled resource-constrained devices to participate in transaction submissions and queries without maintaining the entire blockchain, while still guaranteeing security and liveness. The earliest known light client was Simplified Payment Verification (SPV) [40] for Bitcoin. It was achieved by verifying the hash chain and requesting Merkle proofs for specific transactions. However, it required full synchronization of the light clients. In addition, ecology-specific light clients (or similar schemes) were proposed for other ecologies such as Ethereum [1], Algorand [34], Cosmos IBC [27], ZCash [17], Cardano [21], Mina [16], *etc.*. Furthermore, efforts were devoted to understanding and developing general light clients. Zephyr [29] provided metrics for evaluating blockchain bridge performance of zero-knowledge light clients. SNACKs [10] presented a novel cryptographic primitive that allows light client constructions. Glimpse [47] applied a novel light client with only constant on-chain overhead for PoW-based blockchains. Lu *et al.* [37] proposed a generic light client in the game-theoretic model. In particular, Chatzigiannis *et al.* [22] provided a detailed and comprehensive survey. However, none of the above solutions was applicable to general layer-two Bitcoin applications.

Vector commitments. Vector commitment is the crucial building block of our data security. The earlist vector commitment was the direct application of a Merkle tree. However, it incured long proof size due to its binary nature. Recent advances showed that the proof size of vector commitments could be constant [20, 33, 35, 36]. In particular, Papamanthou *et al.* [42] offered a constant setup size with logarithmic proof size and verification time. However, due to its underlying lattice-based building blocks, its proof size was significantly larger than that of the KZG-based solution. Besides, other schemes above required at least a linear-size setup or linear verification time. Noticeably, some recent works traded efficiency for simpler aggregations or subvector commitments. However, they required either (super)linear verification time [11, 15, 19, 28, 49] or linear setup [48]. Based on the observations above, our modular indexer adopts a solution by leveraging the *Verkle tree* [4, 18] to store states. Its underlying vector commitments are realized by interpolating the vector into a polynomial and committing this polynomial by the KZG commitment [32].

Execution layers of Bitcoin. Before the emergency of generic Bitcoin execution layers, off-chain transaction execution was provided by payment channel networks (PCN) [24, 25, 30], especially the Lightning network [43], and improved by recent advances [13, 26, 44]. In existing bilateral PCN users, data availability was required to prevent sudden withdrawal of other nodes with an early state and loss of deposit. Watchover services [14, 39] were provided to solve the issue, which required the liveness of additional server nodes. Luckily, such an issue of data availability could be solved by modular indexers.

Also, solutions were provided to enable generic layer-two platforms. Stacks [9] and CKB [8] offered scalability improvements through off-chain transactions. Also, inscriptions [52] and ordinal protocols [3] enabled indivisible assets and embedded data for the Bitcoin layer-two ecology. In particular, applications and variants are built on top of the ordinal protocol [5–7, 45]. With our modular indexer, light clients are allowed to participate in the schemes mentioned above in an efficient and secure manner.

7 CONCLUSION

In this paper, we have explored the transformative potential of the Bitcoin ecosystem, where innovations such as Layer-2 solutions and unique protocols like inscriptions and ordinals are paving the way for more scalable, efficient, and robust applications. Our work has particularly focused on addressing the limitations of Bitcoin's Turing-incompleteness by proposing a modular indexer architecture, Indecure, which ensures the integrity and availability of off-chain data processing. This architecture not only mitigates vulnerabilities inherent in decentralized systems, such as Sybil attacks, but also reduces the operational overhead associated with maintaining and verifying large volumes of blockchain data.

The evaluation of Indecure across different indexer protocols has demonstrated significant improvements in efficiency and data integrity, proving the viability of our system in a real-world setting. By implementing a decentralized procedure for data attestation and utilizing a trustless modular execution layer, Indecure stands as a pioneering solution in the realm of the Bitcoin ecosystem, offering a scalable and secure framework that could revolutionize the way digital assets are managed and verified on Bitcoin.

REFERENCES

[1] 2020. Ask about geth: Snapshot acceleration. https://blog.ethereum.org/2020/07/17/ask-about-geth-snapshot-acceleration/. Last accessed on April 16, 2024.

[2] 2023. BRC-20 Documentation. https://layer1.gitbook.io/layer1-foundation/protocols/brc-20/documentation. Last accessed on March 1, 2024.

[3] 2023. Ordinal Theory Handbook. https://docs.ordinals.com/. Last accessed on March 1, 2024.

[4] 2023. Verkle Trees for Statelessness. https://verkle.info. Last accessed on March 1, 2024.

[5] Bitmap 101. https://gitbook.bitmap.land/. Last accessed on April 16, 2024.

[6] Index Names. https://docs.satsnames.org/sats-names/sns-spec/index-names. Last accessed on April 16, 2024.

[7] Indexing. https://layer1.gitbook.io/layer1-foundation/protocols/brc-20/indexing. Last accessed on April 16, 2024.

[8] Nervos Network. https://www.nervos.org. Last accessed on March 1, 2024.

[9] Stacks: Activate the Bitcoin economy with the leading Bitcoin L2. https://www.stacks.co. Last accessed on March 1, 2024.

[10] Hamza Abusalah, Georg Fuchsbauer, Peter Gazi, and Karen Klein. 2022. SNACKs: Leveraging Proofs of Sequential Work for Blockchain Light Clients. In *28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part I.* Springer, 806–836. https://doi.org/10.1007/978-3-031-22963-3_27

[11] Shashank Agrawal and Srinivasan Raghuraman. 2020. KVaC: Key-Value Commitments for Blockchains and Beyond. In *26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part III.* Springer, 839–869. https://doi.org/10.1007/978-3-030-64840-4_28

[12] Mustafa Al-Bassam. 2019. LazyLedger: A Distributed Data Availability Ledger With Client-Side Smart Contracts. *CoRR* abs/1905.09274 (2019).

[13] Lukas Aumayr, Matteo Maffei, Oguzhan Ersoy, Andreas Erwig, Sebastian Faust, Siavash Riahi, Kristina Hostáková, and Pedro Moreno-Sanchez. 2021. Bitcoin-Compatible Virtual Channels. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021.* IEEE, 901–918. https://doi.org/10.1109/SP40001.2021.00097

[14] Georgia Avarikioti, Felix Laufenberg, Jakub Sliwinski, Yuyi Wang, and Roger Wattenhofer. 2018. Towards Secure and Efficient Payment Channels. *CoRR* abs/1811.12740 (2018). arXiv:1811.12740 http://arxiv.org/abs/1811.12740

[15] Dan Boneh, Benedikt Bünz, and Ben Fisch. 2019. Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains. In *39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I.* Springer, 561–586. https://doi.org/10.1007/978-3-030-26948-7_20

[16] Joseph Bonneau, Izaak Meckler, Vanishree Rao, and Evan Shapiro. 2020. Coda: Decentralized Cryptocurrency at Scale. *IACR Cryptol. ePrint Arch.* (2020), 352. https://eprint.iacr.org/2020/352

[17] Benedikt Bünz, Lucianna Kiffer, Loi Luu, and Mahdi Zamani. 2020. FlyClient: Super-Light Clients for Cryptocurrencies. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*. IEEE, 928–946. https://doi.org/10.1109/SP40000.2020.00049

[18] Vitalik Buterin. 2021. Verkle Trees. https://vitalik.eth.limo/general/2021/06/18/verkle.html. Last accessed on March 1, 2024.

[19] Matteo Campanelli, Dario Fiore, Nicola Greco, Dimitris Kolonelos, and Luca Nizzardo. 2020. Incrementally Aggregatable Vector Commitments and Applications to Verifiable Decentralized Storage. In *26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II*. Springer, 3–35. https://doi.org/10.1007/978-3-030-64834-3_1

[20] Dario Catalano and Dario Fiore. 2013. Vector Commitments and Their Applications. In *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings*. Springer, 55–72. https://doi.org/10.1007/978-3-642-36362-7_5

[21] Pyrros Chaidos and Aggelos Kiayias. 2021. Mithril: Stake-based Threshold Multisignatures. *IACR Cryptol. ePrint Arch.* (2021), 916. https://eprint.iacr.org/2021/916

[22] Panagiotis Chatzigiannis, Foteini Baldimtsi, and Konstantinos Chalkias. 2022. SoK: Blockchain Light Clients. In *Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers*. Springer, 615–641. https://doi.org/10.1007/978-3-031-18283-9_31

[23] cointelegraph. 2023. Bitcoin Ordinals community debates fix after inscription validation bug. https://tinyurl.com/55e8fun9. Last accessed on March 1, 2024.

[24] Christian Decker, Rusty Russell, and Olaoluwa Osuntokun. 2018. eltoo: A Simple Layer2 Protocol for Bitcoin. https://blockstream.com/eltoo.pdf. Last accessed on September 20, 2023.

[25] Christian Decker and Roger Wattenhofer. 2015. A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels. In *Stabilization, Safety, and Security of Distributed Systems - 17th International Symposium, SSS 2015, Edmonton, AB, Canada, August 18-21, 2015, Proceedings*. Springer, 3–18. https://doi.org/10.1007/978-3-319-21741-3_1

[26] Christoph Egger, Pedro Moreno-Sanchez, and Matteo Maffei. 2019. Atomic Multi-Channel Updates with Constant Collateral in Bitcoin-Compatible Payment-Channel Networks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*. ACM, 801–815. https://doi.org/10.1145/3319535.3345666

[27] Christopher Goes. 2020. The Interblockchain Communication Protocol: An Overview. *CoRR* abs/2006.15918 (2020). arXiv:2006.15918 https://arxiv.org/abs/2006.15918

[28] Sergey Gorbunov, Leonid Reyzin, Hoeteck Wee, and Zhenfei Zhang. 2020. Pointproofs: Aggregating Proofs for Multiple Vector Commitments. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*. ACM, 2007–2023. https://doi.org/10.1145/3372297.3417244

[29] Xiangan He. 2024. Zephyr: A Cost-Effective, Zero-Knowledge Light Client for Enhanced Blockchain Interoperability. *SIGMETRICS Perform. Evaluation Rev.* 51, 3 (2024), 16–18. https://doi.org/10.1145/3639830.3639838

[30] Mike Hearn. 2013. Micro-payment channels implementation now in bitcoinj. https://bitcointalk.org/index.php?topic=244656.0. Last accessed on April 16, 2024.

[31] JinseFinance. 2024. BRC20 Indexer War: Will BRC20 be forked at the beginning of the year? what happened. https://www.coinlive.com/news/brc20-indexer-war-will-brc20-be-forked-at-the-beginning. Last accessed on March 1, 2024.

[32] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*. Springer, 177–194. https://doi.org/10.1007/978-3-642-17373-8_11

[33] Russell W. F. Lai and Giulio Malavolta. 2019. Subvector Commitments with Application to Succinct Arguments. In *39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*. Springer, 530–560. https://doi.org/10.1007/978-3-030-26948-7_19

[34] Derek Leung, Adam Suhl, Yossi Gilad, and Nickolai Zeldovich. 2019. Vault: Fast Bootstrapping for the Algorand Cryptocurrency. In *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society.

[35] Benoît Libert, Somindu C. Ramanna, and Moti Yung. 2016. Functional Commitment Schemes: From Polynomial Commitments to Pairing-Based Accumulators from Simple Assumptions. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 30:1–30:14. https://doi.org/10.4230/LIPICS.ICALP.2016.30

[36] Benoît Libert and Moti Yung. 2010. Concise Mercurial Vector Commitments and Independent Zero-Knowledge Sets with Short Proofs. In *7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*. Springer, 499–517. https://doi.org/10.1007/978-3-642-11799-2_30

[37] Yuan Lu, Qiang Tang, and Guiling Wang. 2020. Generic Superlight Client for Permissionless Blockchains. In *25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14-18, 2020, Proceedings, Part II*. Springer, 713–733. https://doi.org/10.1007/978-3-030-59013-0_35

[38] Fadai Mammadov. 2023. Where Bitcoin and Metaverse meet. https://medium.com/coinmonks/what-is-bitmap-9947bca0c6de. Last accessed on March 1, 2024.

[39] Patrick McCorry, Surya Bakshi, Iddo Bentov, Sarah Meiklejohn, and Andrew Miller. 2019. Pisa: Arbitration Outsourcing for State Channels. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21-23, 2019*. ACM, 16–30. https://doi.org/10.1145/3318041.3355461

[40] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[41] Nubit. 2024. The First Bitcoin-Native Data Availability Layer. https://www.nubit.org/. Last accessed on September 20, 2023.

[42] Charalampos Papamanthou, Elaine Shi, Roberto Tamassia, and Ke Yi. 2013. Streaming Authenticated Data Structures. In *32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*. Springer, 353–370. https://doi.org/10.1007/978-3-642-38348-9_22

[43] Joseph Poon and Thaddeus Dryja. 2016. The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments. https://lightning.network/lightning-network-paper.pdf. Last accessed on April 16, 2024.

[44] Xianrui Qin, Shimin Pan, Arash Mirzaei, Zhimei Sui, Oguzhan Ersoy, Amin Sakzad, Muhammed F. Esgin, Joseph K. Liu, Jiangshan Yu, and Tsz Hon Yuen. 2023. BlindHub: Bitcoin-Compatible Privacy-Preserving Payment Channel Hubs Supporting Variable Amounts. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 2462–2480. https://doi.org/10.1109/SP46215.2023.10179427

[45] Runes. 2023. Runes Protocol. https://github.com/luminexord/runes. Last accessed on March 1, 2024.

[46] satsnames. 2024. TOTAL SATS NAME INSCRIPTIONS. https://sats-names.xyz/. Last accessed on March 1, 2024.

[47] Giulia Scaffino, Lukas Aumayr, Zeta Avarikioti, and Matteo Maffei. 2023. Glimpse: On-Demand PoW Light Client with Constant-Size Storage for DeFi. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*. USENIX Association, 733–750.

[48] Shravan Srinivasan, Alexander Chepurnoy, Charalampos Papamanthou, Alin Tomescu, and Yupeng Zhang. 2022. Hyperproofs: Aggregating and Maintaining Proofs in Vector Commitments. In *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*. USENIX Association, 3001–3018.

[49] Alin Tomescu, Ittai Abraham, Vitalik Buterin, Justin Drake, Dankrad Feist, and Dmitry Khovratovich. 2020. Aggregatable Subvector Commitments for Stateless Cryptocurrencies. In *12th International Conference, SCN 2020, Amalfi, Italy, September 14-16, 2020, Proceedings*. Springer, 45–64. https://doi.org/10.1007/978-3-030-57990-6_3

[50] UniSat. 2023. Incorrect cases of the last indexer. https://twitter.com/unisat_wallet/status/1729036612015894630. Last accessed on September 20, 2023.

[51] vitalik buterin. 2020. Trust Models. https://vitalik.eth.limo/general/2020/08/20/trust.html. Last accessed on March 1, 2024.

[52] Pieter Wuille, Jonas Nick, and Anthony Towns. 2020. Taproot: SegWit version 1 spending rules. https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki. Last accessed on March 1, 2024.