Goldinals: A Trust-Minimized Fungible Token Standard on Bitcoin

The Nubit Team

December 2024

Special Thanks to Concept Contributor Domo for invaluable feedback and insightful discussions.

1 Introduction

The current landscape of Bitcoin-based assets is highly fragmented. Multiple standards—such as BRC-20 [1], Ordinals [2], Runes, and BRC-420—coexist without interoperability, forcing end-users and wallets to navigate a confusing array of formats and execution models. This fragmentation stems from Bitcoin's inherent constraints: it lacks a global state and a Turing-complete programming environment. As a result, protocols like BRC-20 rely on off-chain indexers [5] to maintain the execution state while embedding application-specific metadata on-chain, producing a patchwork of incompatible asset formats and undermining trust, since users must rely on indexers' integrity.

To address these issues at their core, two essential components are needed:

1. A Turing-complete programming model natively integrated with Bitcoin's security guarantees, eliminating the need for ad-hoc indexers and custom metadata formats.

2. A unified asset standard built on this computation model, compatible with existing asset protocols—such as Runes and Ordinals—and capable of guiding the ecosystem toward a more coherent, reliable, and extensible future.

The first component is a Turing-complete programming model integrated directly with Bitcoin's security. BitVM [4] provides such functionality: it leverages an off-chain, interactive proof system and Bitcoin's native scripting constraints to enable arbitrary computations without modifying Bitcoin's consensus. This essentially brings Turing-completeness to Bitcoin at a layer above its base protocol, eliminating the need for ad-hoc off-chain indexers to maintain state.

Building on top of this Turing-complete foundation, the second component, Goldinals, introduces a unified asset protocol that leverages BitVM's capabilities to encode complex logic directly on-chain. Its design separates operations

into a "Prepare" and "Kickoff" phase, secured by zero-knowledge proofs and a challenge mechanism, ensuring trust-minimized state verification. Through this approach, GOLDINALS offers a single, coherent framework that can integrate and standardize diverse asset formats—such as BRC-20, Runes, and Ordinals—under a fully programmable model. In doing so, it promises to reconcile today's scattered ecosystem into a seamless, Bitcoin-secured environment where all assets can be easily managed and evolved.

2 Background

We begin by presenting the core concepts that underpin the protocol's design and operation.

2.1 Bitcoin UTXO Model

Bitcoin operates under the Unspent Transaction Output (UTXO) model, in which every fraction of Bitcoin exists as a discrete output of a previous transaction or mining reward. Each output includes a lock script defining the conditions under which it can be spent, and the spender must provide an unlock script satisfying these conditions. This design ensures that Bitcoin cannot be created or destroyed arbitrarily: tracing any unit of Bitcoin always leads back to a valid mining reward output.

However, while this model upholds Bitcoin's security guarantees, it introduces significant challenges for implementing custom tokens or complex asset protocols. In Ethereum's account-based system, each node maintains a global state that records all balances. Simple operations, such as transferring ERC-20 tokens, update this global ledger by decreasing the sender's balance and increasing the recipient's balance. The entire network can easily verify that the sender's account has sufficient funds and that the recipient's balance is correctly adjusted.

In contrast, Bitcoin nodes do not maintain a single global state of user balances. Instead, each user's holdings must be inferred from scanning relevant UTXOs across the blockchain. This stateless nature makes it difficult to implement token-like logic similar to ERC-20. Without a native global state, verifying that a sender has enough tokens to transfer, or recording the resulting balance changes after a transfer, cannot be done as straightforwardly within Bitcoin's base layer.

Moreover, embedding all necessary token information directly into Bitcoin's standard transaction scripts is non-trivial. Simply placing recipient and amount data in these scripts would violate Bitcoin's consensus rules, causing transactions to fail. The UTXO model's strict constraints on script execution and data storage thus impede the straightforward adoption of Ethereum-style token protocols, necessitating additional layers or mechanisms, such as those introduced by GOLDINALS, to achieve similar functionality in a secure and trust-minimized manner.

2.2 Ordinals and BRC-20

The Ordinals protocol recently introduced a way to embed arbitrary data directly into Bitcoin's witness fields, leveraging SegWit and Taproot to permanently store information within UTXOs without exceeding size limits. Building on this capability, BRC-20 aims to issue native assets on Bitcoin by treating Bitcoin as a sequencing layer and storage medium for metadata, while relying entirely on off-chain indexers to maintain the global state of token balances.

In practice, however, BRC-20's reliance on off-chain indexers introduces significant trade-offs. Since Bitcoin provides no built-in global state, BRC-20 delegates state management—account balances, transaction validity checks, and other token logic—to external indexers that must track every relevant transaction. This design forces users to trust either their own or a third-party indexer to correctly compute balances and validate transactions. Running an indexer is resource-intensive, and most users depend on centralized services that are not guaranteed by Bitcoin's consensus rules. Additionally, because Ordinals records data with fine-grained UTXO management, indexers must process the entire Bitcoin history to confirm a token's provenance, making a single BRC-20 transfer both complex and costly, often requiring multiple Bitcoin transactions.

Beyond these operational challenges, BRC-20 lacks native programmability. Without a global state or a built-in mechanism to enforce conditional logic, it cannot support advanced features like stablecoin issuance, custom unlock conditions, or automated policy enforcement. As a result, BRC-20 remains limited to basic, static token functionality, and its long-term adaptability and extensibility are constrained by its reliance on off-chain computation and trust assumptions.

2.3 CAT Protocol

The Covenant Attested Token (CAT) Protocol [3] is an alternative approach to tokenization on Bitcoin, utilizing a UTXO-based token protocol that leverages smart contracts, specifically covenants, to manage token mints and transfers.

One of the key characteristics of the CAT Protocol is the absence of a need for an indexer, as the token's ruleset is guaranteed by the Bitcoin consensus, with both token data and logic residing on-chain through UTXO encoding. This design allows for a more decentralized and trustless approach to token management, as users do not need to rely on external indexers to track token balances and validate transactions.

However, it is worth noting that the CAT Protocol relies on the re-activation of OP_CAT, a Bitcoin opcode that was disabled in 2010 due to security concerns related to its original implementation, specifically potential denial-of-service (DoS) attacks caused by unbounded memory usage. This means that the CAT Protocol cannot currently be implemented on the Bitcoin network.

3 The Goldinals Way

3.1 Overview of Goldinals

Goldinals is a Bitcoin-native asset protocol designed to operate under Bitcoin's constraints while providing functionality similar to familiar token standards on other blockchains. In Ethereum's ERC20 standard, tokens can be deployed, transferred, minted, and burned through a straightforward set of APIs. BRC20, built on top of Bitcoin Ordinals, also aims to support token-like functionality, but it relies heavily on off-chain indexing and other trust assumptions. Goldinals provides an alternative approach that implements a similar set of APIs directly on Bitcoin, with minimal trust assumptions and without sacrificing Bitcoin's security model.

The core APIs of Goldinals are:

Deploy: Initializes the protocol's global parameters and creates a new token instance.

Mint: Increases the token supply by creating new units of the token and assigning them to a specified address.

Transfer: Moves tokens from one address to another, subject to validity checks such as balance sufficiency.

Burn: Reduces the token supply by permanently removing tokens from circulation.

These APIs are logically similar to those in ERC20 and BRC20, but their implementation on Bitcoin is more complex due to Bitcoin's stateless design and Turing-incomplete scripting language. Unlike Ethereum's seamless, atomic calls, Goldinals must carefully manage state updates and validity checks over multiple steps, ensuring that the final confirmed state fully inherits Bitcoin's security.

3.2 Design Principles

Implementing these token APIs directly on Bitcoin requires careful consideration of both trust assumptions and compatibility with Bitcoin's security and design constraints. Two principles guide Goldinals:

First, the system should be **trust-minimized**. It must avoid relying on any single centralized off-chain indexer. Protocols like BRC20 or Ordinals often depend on off-chain indexers that maintain and process the entire transaction history. If these indexers become unavailable or dishonest, no one can validate past transactions. Thanks to BitVM, Goldinals aims to remove this point of trust, allowing users to validate token states independently through Bitcoin's blockchain and cryptographic proofs.

Second, Goldinals should inherit **Bitcoin's security model**. A straightforward approach might record all token-related transactions on a separate layer (L2) and only settle the final state on Bitcoin. However, if that separate layer ceases operation, there would be no definitive way to reconstruct the history or

validate past events. GOLDINALS therefore aims to confirm every token operation directly on Bitcoin, preserving a trust-minimized and secure audit trail that depends only on Bitcoin's finality and data availability.

These constraints make the design non-trivial. Bitcoin's latency, slow block times, and Turing-incomplete scripting language limit the complexity of operations that can be verified on-chain. To meet these challenges, GOLDINALS carefully splits each operation (such as a transfer or mint) into multiple steps and leverages zero-knowledge proofs to ensure correctness without relying on powerful off-chain indexers or large, persistent state machines.

3.3 State Machine in Goldinals

Under these principles, GOLDINALS introduces key concepts and entities that together form a secure, stateful, and verifiable system.

First, GOLDINALS introduces a new entity, the **ZKOracle**, which acts as a **state machine** that simulates and records the entire execution history of GOLDINALS. The ZKOracle maintains global state by scanning Bitcoin blocks for protocol-defined operations. It does not simply store all historical data; instead, it relies on zero-knowledge proofs to compress and verify large portions of history into small, easily verifiable proofs. By doing so, it avoids the pitfalls of naive off-chain indexers and ensures that users can validate the full transaction history with minimal trust and cost.

Second, GOLDINALS introduces the **Challenger**. During the confirmation of token operations, the Challenger proactively verifies the legality of submitted operations. Under the assumption that at least one Challenger is honest (1-of-N assumption), any illegal attempts to finalize an operation can be discovered, challenged, and invalidated before the operation becomes finalized.

To handle the complexity inherent in confirming operations on Bitcoin, GOLDINALS represents each API call as a three-stage state machine: **Prepare**, **Kickoff**, and **Challenge**. In a straightforward environment like Ethereum's ERC20, a single atomic transaction can confirm a transfer or mint operation. On Bitcoin, GOLDINALS splits the workflow:

1. **Prepare:** The Sender submits a Bitcoin transaction that encodes the desired operation (such as transfer or mint) and its relevant parameters. This Prepare transaction is stored on-chain, ensuring permanent data availability. At this stage, the operation is known to the network but not yet finalized.

2. **Kickoff:** After a predetermined number of confirmations and upon assembling necessary ZK proofs, the Sender submits a Kickoff transaction that proves the prepared operation's correctness.

3. **Challenge:** After the Sender submits the required proofs in the Kickoff phase, the Challenger enters a defined challenge period that is an integral part of the BitVM process. Within this timeframe, the Challenger can scrutinize the proofs and, if it detects any irregularities or attempted fraud,

present evidence on-chain. By leveraging BitVM's challenge mechanism, the entire dispute resolution process is enforced by the Bitcoin network itself. If no valid challenge arises by the end of this period, the operation is finalized and the global state—balances, supply, and other protocol-defined variables—is permanently confirmed under Bitcoin's own security guarantees.

By adopting this three-stage design, GOLDINALS navigates Bitcoin's latency and constrained scripting environment. **Prepare** ensures transparency and data availability. **Kickoff** ensures that the correct cryptographic proofs and verification conditions are met before final confirmation. **Challenge** introduces a trust-minimized mechanism for detecting and mitigating any fraudulent or invalid operations.

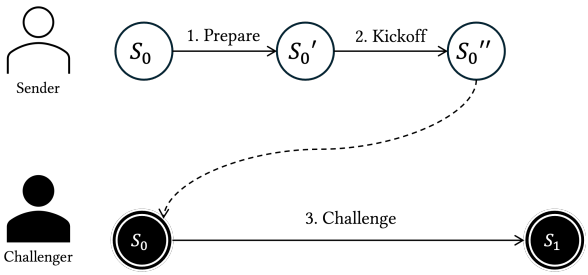

Figure 1: The three-stage state machine for GOLDINALS: Prepare, Kickoff, and Challenge.

3.4 Implementation Details

In this section, we elaborate on the above computation model using the **transfer** and **mint** functions from GOLDINALS.

Transfer

In a transfer operation, the Sender first broadcasts a Prepare transaction to specify the transfer of tokens from their address to a recipient's address. The ZKOracle reads this transaction and records it along with other operations in the block. Before finalizing the transfer, the Sender must produce a Kickoff transaction that includes a zero-knowledge proof (ZKP) confirming that the Sender indeed has sufficient balance and that the transfer obeys the rules defined during Deploy. The Challenger can then review this Kickoff transaction and raise a dispute if the proof is incorrect. If no dispute is raised after the challenge period, the transfer is marked as finalized, and the recipient's balance is permanently updated on Bitcoin.

1. Prepare: In the Prepare stage, the Sender broadcasts a Bitcoin transaction containing all the essential data that defines an intended operation, including the referenced deploy instance (`deployRef`), the operation code (`opcode`), related parameters, and a dedicated Bitcoin address (receiver). Unlike protocols that require special-purpose UTXOs, GOLDINALS allows this transaction to consume any available UTXO as its input, eliminating extra preparatory steps. Using `OP_RETURN` fields, the transaction output embeds these details directly on-chain. By including the receiver's address in the output, Bitcoin Core's native RPC `scanblocks` can readily detect and index the Prepare transaction without relying on external indexers.

Since Bitcoin itself does not maintain a global state or inherently verify the legality of operations, GOLDINALS relies on the ZKOracle—a stateful, off-chain verifier—to track and validate the entire protocol state. The ZKOracle continuously scans new Bitcoin blocks to identify all Prepare transactions related to GOLDINALS. Starting from an initial empty state, it updates the global state as each new operation is encountered and validated. Through zero-knowledge proofs, the ZKOracle can succinctly verify both the correctness and inclusion of these operations, producing a state commitment for every block.

Concretely, when the Sender's Prepare transaction appears in Bitcoin Block H, the ZKOracle checks its legality and incorporates it into a Merkle tree containing all GOLDINALS operations from that block. The Merkle root of this tree forms the **Goldinals State Commitment** for Block H—an immutable, cryptographically verifiable representation of the state after processing that block's operations. The Sender then obtains two critical pieces of information from the ZKOracle: the State Commitment itself and an inclusion proof for their Prepare transaction, ensuring that their operation is both recorded and provably part of the recognized global state.

2. Kickoff: Taking a Transfer operation as an example, once the Prepare transaction is on-chain, the Sender must initiate a Kickoff transaction after a specified waiting period (Δ_0 blocks) to finalize the operation. This Kickoff transaction must be signed by the Sender and must embed a zero-knowledge proof (ZKP) in its witness field, along with the corresponding public inputs. Together, these conditions ensure that the Kickoff transaction adheres to the **KickoffScript**, a set of rules verifying that the required ZKP and public inputs are included in the witness field via a correct format.

3. Challenge: During a subsequent challenge period (Δ_1), the Challenger examines the Kickoff transaction's validity. It begins by confirming that the Kickoff transaction references the correct GOLDINALS State Commitment from the ZKOracle. If the included State Commitment does not match the one established for the Bitcoin block containing the original Prepare transaction, the Challenger discards the Kickoff transaction as invalid. If it does match, the Challenger retrieves the Verification Key from the corresponding Deploy transaction and uses it to verify the ZKP and its public inputs. If the ZKP

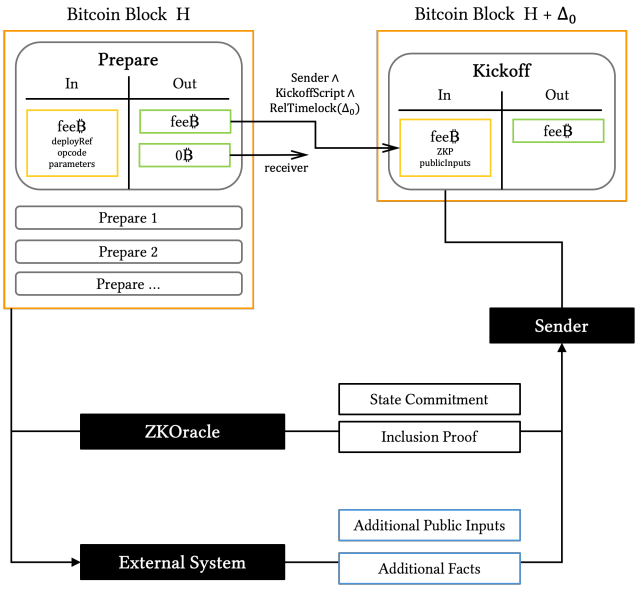

Figure 2: ZKOracle compressing and verifying GOLDINALS State Commitments.

fails validation, the Challenger submits a challenge transaction, marking the operation as illegal. If no challenge arises within the allotted time, the Transfer operation is deemed valid, and its effects become permanent. After the challenge period concludes, the Collector can claim any remaining fees associated with the operation without affecting its final state.

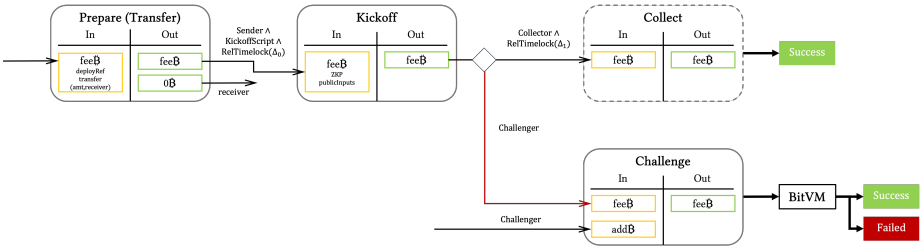

Figure 3: Workflow of a GOLDINALS Transfer operation.

Mint

For a mint operation, the Deployer's initialization sets rules for minting new tokens (for example, supply caps or distribution constraints). The Sender who wishes to mint new tokens first broadcasts a Prepare transaction specifying the mint parameters. The ZKOracle records it, and then, as with transfer,

the Sender must produce a Kickoff transaction with a proof of validity. If no Challenger steps forward to contest this proof, the new tokens are confirmed as added to the specified recipient's balance.

Additional rules can be introduced to manage mint-specific threats, such as malicious users who never finalize their mint operations. For example, after a set time, the Deployer can revert stale Prepare transactions, ensuring that malicious or negligent participants cannot indefinitely lock token supply. Similar designs also address the possibility of refunding users in failed mint scenarios, requiring them to finalize or lose their claim within a specific timeframe.

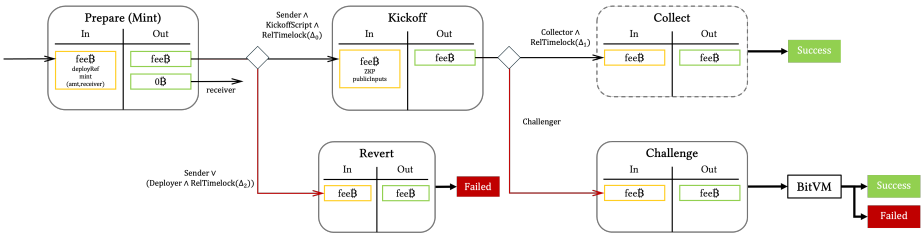

Figure 4: Workflow of a GOLDINALS Mint operation.

Inefficiencies and Exploitation Safeguards: In scenarios where tokens are distributed freely, a malicious participant could repeatedly submit Mint requests without proceeding to the Kickoff stage, thus leaving a large number of unfinalized Prepare transactions. This strategy effectively locks up the token supply and hinders fair distribution, all at a relatively low cost to the attacker. To counteract this, the protocol imposes a time limit (Δ_2) on Prepare transactions associated with Mints. After this period, the Deployer may reclaim and mark these stagnant transactions as **Reverted**, freeing the blocked token supply and enabling other participants to proceed with their own Mint operations.

A similar concern arises when token distribution requires BTC payments. While it may seem logical to allow a user to reclaim their BTC if the Mint fails due to supply constraints, this opens the door to strategic abuse. A user could wait indefinitely, holding their BTC in limbo until favorable market conditions arise. To prevent such manipulation, we introduce a payout mechanism that forces the Sender to complete the Kickoff within Δ_2 blocks. If the Sender fails to meet this deadline, the Deployer can reclaim the UTXO. This ensures that no party can exploit timing to their advantage and that the entire Mint process remains both efficient and fair.

Check Balance

To determine their account balance, users need only employ Bitcoin Core's native `scanblocks` RPC to locate all Prepare transactions associated with their addresses and examine their final statuses. If the user acted as the Sender in a Transfer operation, the confirmed outcome of that Transfer reduces their

balance. Conversely, if the user appeared as a Receiver in either a Transfer or a Mint operation, any confirmed outcome from those Prepare transactions increases their balance. This straightforward approach removes the need for external indexers or complex off-chain services. Users need not sift through the entire historical record; scanning only for the Prepare transactions relevant to their addresses and checking their confirmed results is sufficient to accurately compute their current holdings.

3.5 Threat Model

Goldinals operates under a 1-of-N trust assumption. As long as there is at least one honest Challenger among the set of participants, no illegal operations will remain unchallenged. The ZKOracle is trustless, relying on cryptographic proofs rather than trust in a single operator. Decentralizing the ZKOracle further, such as through a committee of independent ZKOracle operators, reduces the likelihood of collusion or censorship and ensures that users can always find at least one honest source of proofs and state commitments.

3.6 Programmability

Goldinals enhances programmability by allowing the Sender to embed additional external conditions and facts into each operation, enabling logic that goes well beyond Bitcoin's native scripting capabilities. These external inputs—whether cryptographic proofs, off-chain attestations, or metadata—are incorporated into zero-knowledge proofs (ZKPs) submitted alongside the Prepare and Kickoff transactions. This approach ensures that any desired conditions, however complex, can be verified trustlessly on-chain, without relying on centralized parties or indexers.

This flexibility opens the door to diverse use cases. For example, consider a multi-signature wallet setup that requires multiple distinct approvals before transferring tokens. Although Bitcoin natively supports only limited scripting logic, Goldinals allows the Sender to supply non-Bitcoin-compatible signatures as external facts. The ZK proof then confirms that all required keys approved the transfer, while the Challenger verifies the proof's correctness. As a result, even advanced signature schemes or custom key-management policies can be fully enforced within the Goldinals framework.

Beyond multi-signature scenarios, programmability can extend to conditional transfers, token gating, oracles feeding price data, or membership proofs. A Transfer operation might require verification that a certain off-chain event has occurred, that the recipient holds a specific non-fungible credential, or that the Sender meets certain external criteria. By incorporating these checks into ZK proofs, Goldinals seamlessly integrates complex logic into Bitcoin's trust-minimized environment, ensuring that all conditions are cryptographically validated and securely anchored in the blockchain's immutable record.

3.7 Key Advantages Compared to Prior Work

Goldinals enables users to interact directly with protocol-defined operations without depending on centralized indexers. By leveraging zero-knowledge proofs, operations can include additional programmable logic or conditional checks that go beyond simple transfers and mints. Finally, Goldinals requires fewer Bitcoin transactions to complete a single operation compared to protocols like BRC-20, reducing both complexity and cost while maintaining Bitcoin-level security and trust minimization.

4 Conclusion

Goldinals introduces a trust-minimized standard for fungible tokens on Bitcoin. By removing the need for centralized indexers and enforcing on-chain verification of token operations, Goldinals aligns with Bitcoin's principles of security and decentralization. This approach opens new possibilities for securely issuing custom assets on Bitcoin, such as crowdsales, stablecoins, and liquid staking.

References

[1] BRC-20 Documentation. `https://layer1.gitbook.io/`, 2023. Last accessed on Dec 11, 2024.

[2] Ordinal Theory Handbook. `https://docs.ordinals.com/`, 2023. Last accessed on Dec 11, 2024.

[3] CAT Protocol. `https://catprotocol.org/`, 2024. Last accessed on Dec 11, 2024.

[4] Robin Linus, Lukas Aumayr, Alexei Zamyatin, Andrea Pelosi, Zeta Avarikioti, and Matteo Maffei. Bitvm2: Bridging bitcoin to second layers. 2024.

[5] Hongbo Wen, Hanzhi Liu, Shuyang Tang, Tianyue Li, Shuhan Cao, Domo, Yanju Chen, and Yu Feng. Stateless and verifiable execution layer for metaprotocols on bitcoin. 2024.