Scalable Non-Fungible Tokens on Bitcoin

Jordi Herrera-Joancomartí[1], Cristina Pérez-Solà[1], and Toni Mateos[2]

[1] Dept. d'Enginyeria de la Informació i les Comunicacions,
Universitat Autònoma de Barcelona,
CYBERCAT - Center for Cybersecurity Research of Catalonia,
08193 Bellaterra, Catalonia, Spain
{jordi.herrera, cristina.perez}@uab.cat
[2] Freeverse.io & LAOS Network Research
toni@laosfoundation.io

Abstract. This paper presents a protocol for scaling the creation, management, and trading of non-fungible tokens (NFTs) on Bitcoin by extending bridgeless minting patterns previously used on other blockchains. The protocol leverages on-chain Bitcoin data to handle all aspects of token ownership, including trading, while integrating a secondary consensus system for minting and optionally modifying token metadata. To minimize its on-chain footprint, the protocol utilizes the OP_RETURN mechanism for ownership records, while complementary NFT-related actions are stored on the LAOS blockchain. All data remains permanently on-chain, with no reliance on bridges or third-party operators.

Keywords: Bitcoin · NFT · Bridgeless Minting · LAOS.

1 Introduction

Bitcoin's blockchain [12] is the most secure and resilient decentralized ledger deployed to date, operating continuously since its inception in 2009. While primarily serving as a peer-to-peer digital currency, Bitcoin's robust infrastructure can also function as a decentralized record-keeping system, extending its utility beyond simple payments.

Efforts to enhance Bitcoin's programmability have led to the development of off-chain solutions ([15], [4], [11], [2], [9], [10]), which typically anchor partial or compressed snapshots of transactional data directly to Bitcoin's blockchain while keeping the bulk of data off-chain. However, the overall security of these solutions is severely compromised by the specifics of how this on-off-chain interplay is designed.

Meanwhile, alternative approaches, like Inscriptions [17], Runes [18], and BRC-20 tokens [21], store all relevant data directly on-chain, ensuring full transparency and decentralization. However, these methods face inherent limitations in scalability and cost due to Bitcoin's block size constraints and the increased demand for block space.

The protocol presented in this paper follows an always-on-chain approach, providing a scalable solution for the creation, management, and trading of non-fungible tokens (NFTs) on Bitcoin, significantly improving upon Inscriptions. Instead of inscribing NFTs on individual satoshis, it utilizes an OP_RETURN mechanism similar to that of Runes. The protocol optimizes data storage on Bitcoin by keeping on it only the information that adds more value to the network, namely, the data required to fully determine and transfer ownership of each token on-chain. Leveraging the Bridgeless Minting pattern ([1], [14]) previously used on other blockchains, the protocol stores the bulk of the data, particularly asset metadata, on-chain within a separate consensus system, the LAOS Network, an Ethereum Virtual Machine (EVM)-compatible blockchain that operates as a parachain on Polkadot [24]. As a result, its security is directly ensured by one of the most robust consensus systems, providing strong guarantees for Data Availability (DA) and the prevention of invalid transactions, as all transactions are explicitly verified by Polkadot's relay chain.

LAOS's EVM compatibility adds programmability to NFT metadata management without interfering with Bitcoin's role in handling ownership. This allows NFT creators and users to define the logic that best suits their application. For example, they can choose to keep all metadata on-chain, set up collections with a fixed supply, or establish rules governing who can mint and when. Additionally, they can enforce metadata immutability or implement smart contracts to enable controlled updates. The latter is particularly relevant, as it allows a single NFT registered on Bitcoin to undergo metadata modifications (e.g., reflecting its owner's in-game usage) in a structured and traceable manner, with all updates recorded as part of DA on LAOS.

Regarding Bitcoin's role in ownership management, collection creation is fully permissionless, and trading requires explicit Bitcoin signatures from rightful owners, with P2PKH, P2WPKH and P2TR being supported schemes. Additionally, bulk transfers of a potentially large number of NFTs, even across multiple collections, are supported. This includes the atomic inclusion of both NFT transfers and Bitcoin payments, ensuring trustless execution of sales and purchases. Throughout the paper, we shall refer to the presented protocol as BRC721 (**B**ridgeless or **B**itcoin ERC721), due to its similarities to the well established ERC721 standard for NFTs across EVM chains.

The main contribution of this work is thus the design of the BRC721 protocol, that enables secure and scalable NFT management on Bitcoin while minimizing on-chain footprint. Unlike existing solutions, our approach maintains full on-chain ownership while offloading metadata management to a separate but verifiable consensus system. Additionally, we provide a description of the protocol's architecture and detailed specification, along with a thorough analysis of its security properties.

The rest of the paper is organized as follows. Section 2 presents the state of the art of Bitcoin protocols beyond payments. Section 3 describes the overall protocol, with the concrete architecture detailed in Section 4. The protocol specification is presented in Section 5, followed by its security analysis in Section 6. Finally, Section 7 summarizes its potential impact and novelty.

2 State of the art

Bitcoin [12], originally designed as a decentralized monetary system, has evolved into a foundational layer for various protocols that enable asset deployment and management. Protocols such as Colored Coins [3], Omni Layer [22], Counterparty [5], Open Assets [19], RGB [16], Ordinals [17], BRC-20 [21], Taro [10] and Runes [18], among others, highlight the versatility of Bitcoin's infrastructure. Each of these protocols extends the functionality of Bitcoin to support use cases such as tokenization, asset management, and digital artifacts, leveraging the blockchain's robust security and immutability. Despite their differences in implementation, they all share a reliance on Bitcoin's existing key infrastructure, particularly the keys associated with UTXOs (Unspent Transaction Outputs), which are integral to asset control and wallet compatibility.

By associating assets with the keys that control UTXOs, these systems integrate seamlessly with Bitcoin wallets, simplifying user adoption. Wallets that manage standard Bitcoin transactions can be extended to support these protocols with minimal modifications, allowing users to handle assets like tokens or NFTs alongside their regular Bitcoin holdings. This reliance on Bitcoin's cryptographic architecture ensures that these protocols remain decentralized, secure, and interoperable, making them a natural extension of Bitcoin's capabilities.

The protocols listed above vary significantly in how they embed and manage information on the Bitcoin blockchain. Early systems such as Colored Coins, Omni Layer, Open Assets and Counterparty, and even some new proposals such as Runes, rely on the OP_RETURN field to embed metadata directly into Bitcoin transactions. In contrast, newer protocols like RGB, Taro Protocol, and BRC-20 utilize Bitcoin's advanced features, such as Taproot, to store cryptographic commitments within Merkle tree leaves. A different approach is taken by Ordinals since the data is stored in the witness field of the inputs.

Each of those described methods to store information on the Bitcoin blockchain have distinct benefits and drawbacks. OP_RETURN is simple and widely supported, allowing protocols to store small amounts of data (up to 80 bytes) directly in Bitcoin transactions. This approach allows for easy implementation and clearly separates data from spendable outputs. However, OP_RETURN has limitations in scalability, as it consumes block space and sacrifices privacy, with all data fully exposed on the blockchain. In contrast, Taproot leaves, used in modern protocols, perform better in terms of scalability and privacy. By storing cryptographic commitments within Merkle tree leaves, Taproot minimizes the on-chain data footprint and supports advanced features like selective data disclosure and complex scripting. However, implementing Taproot-based solutions requires greater technical expertise, since it adds new levels of complexity by relying on off-chain data mechanisms. Table 1 summarizes the benefits and drawbacks of each approach.

Feature	**OP_RETURN**	**Taproot Leaves**
Data Capacity	Limited (80 bytes max)	Efficient (cryptographic commitments)
Privacy	Low (data is fully exposed)	High (selective disclosure)
Flexibility	Limited (basic metadata storage)	High (complex contracts, multi-asset)
Ease of Use	Simple, widely supported	Complex, requires advanced tools

Table 1. Comparison of OP_RETURN and Taproot Leaves for Storing Information

Although our goal to develop a lightweight non-fungible token over Bitcoin could be achieved using both approaches, and including information in taproot leaves seems to have more benefits, we have an extra goal of having full data availability on-chain. Such property discards using taproot leaves to store information since only commitments can be stored in taproot leaves and the information itself has to be stored off-chain. For that reason, we use the OP_RETURN approach to store the information in the Bitcoin blockchain, minimizing the information footprint of our proposal in that particular blockchain, for ownership purposes, and complementing the data with the information stored in the LAOS blockchain, for storing other specific actions on the NFTs.

2.1 The LAOS blockchain

The LAOS Network [1,14] is a Layer-1 blockchain built as a Parachain on Polkadot, with its core value proposition being the ability to offload certain types of transactions from other blockchains without bridges.

This paper focuses on a specific application called *Bridgeless Minting*, which enables the minting and evolution of Non-Fungible Tokens (NFTs) to be offloaded from a source consensus system to the LAOS Network. At the same time, all aspects related to the ownership of these NFTs, including trading, lending, and DeFi, remain fully on-chain within the source consensus system.

Since September 2024, the LAOS Network has been utilized to scale the minting and evolution of NFTs from major EVM-compatible chains, including Ethereum and Polygon.

Bridgeless Minting introduces a clear separation of roles for applications like video gaming or Real World Assets (RWA). Developers of these applications can mint at a larger scale without congesting Ethereum or incurring native gas fees by offloading to a purpose-built consensus system that leverages Polkadot's underutilized bandwidth. Meanwhile, end users, such as gamers or RWA traders, can continue trading as usual on Ethereum, avoiding the need for bridges or wrapped assets and remaining where liquidity resides.

Bridgeless Minting utilizes the concept of Universal Location (UL), introduced in the third revision of the Cross-Consensus Message Format (XCMv3) within the Polkadot network [25,26]. Earlier versions of XCM have been used within Polkadot for years, enabling its sovereign Parachains to reference various resources from other Parachains. This has facilitated secure cross-chain transfers and even remote execution. The third revision extends part of this functionality to generic consensus systems, not limited to those within Polkadot.

As detailed in the LAOS Whitepaper [1], Bridgeless Minting establishes a fully permissionless pattern in which all data remains permanently available on-chain, eliminating the need to rely on external parties for Data Availability. The source blockchain manages the on-chain ownership of unique (non-fungible) NFTs, or *slots*, which point via Universal Location (UL) to specific locations on the LAOS consensus system.

On EVM-compatible networks, where this pattern has been implemented so far, the ownership logic on the source EVM chain typically follows standardized patterns, particularly those defined by the ERC721 standard [8] and its corresponding interface.

This paper extends this pattern to Bitcoin. Since the core Bitcoin protocol lacks support for smart contract logic, the extension builds on previous work with off-chain solutions, choosing an approach based on OP_RETURN as the most suitable, as discussed above.

3 Bridgeless Minting on Bitcoin

To extend LAOS token management to Bitcoin, we need to define how Bitcoin addresses may take control of LAOS token ownership. As discussed in Section 2.1, LAOS is an EVM-compatible network, where public addresses are 160-bit identifiers. This applies to both smart contract addresses and Externally Owned Addresses (EOAs), the latter being derived cryptographically from corresponding private keys.

Token registration on Bitcoin requires an initial reference to a smart contract on LAOS, to register a collection. This step is then followed by subsquent token registrations, which require the usage of an address mapping between public key hashes on Bitcoin and EVM addresses. The first step is detailed in Section 3.1, while the address mapping is covered in Section 5.1.

The remainder of this section provides a detailed explanation of the process.

3.1 Bitcoin collection registration

A collection on LAOS is a smart contract, associated with an EVM address `LAOScollectionAddress`, which implements the minimal functionality of minting non-fungible tokens (NFTs). Additionally, it may support token updates, though this is optional. The logic and permissions governing these operations are not constrained by the protocol presented here and can be tailored to fit any business requirements. For instance, a simple implementation could involve a smart contract that mints a predefined set of NFTs, with all its data on-chain, while ensuring that no further minting or modifications can be performed by any address.

Once a collection is created on LAOS, it can be registered in the Bitcoin blockchain. The idea is to map a `LAOScollectionAddress` with a `BitcoinCollectionID`. The register collection action is performed through a Bitcoin transaction (referred to as a **register collection transaction type**). Anyone can create such a transaction, even if he is not the owner of the corresponding collection on LAOS. It is an out-off-band mechanism that determines which is the 'proper' map between a `LAOScollectionAddress` an a `BitcoinCollectionID`. In fact, multiple maps can be posted in the Bitcoin blockchain that assign different `BitcoinCollectionID` to the same `LAOScollectionAddress`. Implicitly in the collection registration, a token pre-mint is performed. BRC721 tokens are designed in such a way that they are already pre-minted to a specific 160 bit address. Every collection has, by default, 2^{96} tokens that are pre-minted to addresses identified by 160 bits. The token identifier encodes such relation and it is defined as the bitwise concatenation of the `h160Address` with an arbitrary 96 bits integer, the `slotNumber`:

$$\texttt{TokenID = uint256(uint96 slotNumber || uint160 h160Address)} \tag{1}$$

The `BitcoinCollectionID` is implicitly defined as `blockHeight:txIndex`. Therefore, the identifier is established when the transaction is confirmed (and thus has a position inside a block).

3.2 Mint tokens on LAOS

BRC721 tokens are minted on smart contracts governing collections on LAOS. As pointed out in Section 3.1, the collection smart contract contains the authorization logic to mint tokens in such a collection. Obviously, minting tokens for a collection can only be performed once the collection is created on LAOS.

In the minting process, BRC721 tokens associated with the same `h160Address` are differenciated by selecting different values for `slotNumber`, and hence, generating unique token identifiers `TokenID` by using the mapping in Equation 1.

Tokens outside the context of their collection smart contract must be referred to by the pair `(LAOScollectionAddress, tokenID)`.

3.3 Token ownership registration in Bitcoin

In a standard flow, BRC721 tokens are first minted on LAOS and then registered on Bitcoin. In the first step, any entity with minting permissions for the relevant LAOS collection, whether an EOA or a smart contract (e.g., a multisig), chooses the `h160Address` of each token as its *initial owner*. When minting multiple tokens for the same initial owner, different `slotNumber` values shall be used.

Once a token is minted on LAOS, it can be registered on Bitcoin by its *initial owner*. Specifically, only the owner of the Bitcoin address (as determined by the mapping detailed in Section 5.1) corresponding to the `h160Address` encoded in the tokenID can register the token on Bitcoin.

Registering on Bitcoin a BRC721 token that has not yet been minted, or may never be, results in UTXOs containing null Bitcoin NFTs, i.e., NFTs without metadata, rendering them likely useless. This is analogous to ERC721 tokens on Ethereum that reference external URIs that cannot be accessed or fetched. However, unlike Ethereum, where unreachable metadata can lead to permanent loss of asset information, LAOS is a public, permissionless consensus system, ensuring that if a token was originally minted on-chain, its data, including the minting event, content, and timestamp, is fully traceable and verifiable.

Ownership registration in Bitcoin is performed through a **token ownership registration transaction type**. To ensure that only the owner of the `h160Address` defined in the tokenID can perform such registration, the token ownership registration transaction type includes an input that unlocks the output associated with the `h160Address`. Outputs of such a transaction determine the new owner of the registered token. From that moment on, ownership of the token is encoded in a UTXO (whoever can spend the UTXO will be the new owner, and will thus have the ability to further transfer the tokens to other outputs).

Such registration mechanism allows registering the owner of a single token or multiple tokens at once. Notice that token ownership registration implies transferring the token from the `h160Address` to a new UTXO, that can encode arbitrary conditions for being unlocked. Of course, such new UTXO could also be controlled by the original owner, ensuring that the effective ownership does not change in this initial registration, although this is not required. For example, a new UTXO might not be controlled by the registering account if an NFT is sent to someone while being registered, optionally including a payment in the same transaction. Furthermore, notice that after the token ownership registration, the semantics of tokenID codification continue encoding the original owner (`TokenID = uint256 (uint96 slotNumber || uint160 h160Address)`), but this `h160Address` is no longer the owner of such token.

Once a BRC721 token ownership registration has taken place in Bitcoin, we can refer to a LAOS NFT Bitcoin output. A LAOS NFT Bitcoin unspent output, UTXO_L, is a Bitcoin output that holds a LAOS NFT (or a group of them). We can also refer to the UTXO_L set, the subset of all Bitcoin UTXO that owns LAOS NFTs.

3.4 Token ownership transfer in Bitcoin

Once a BRC721 token has been registered on the Bitcoin blockchain, it can be transferred between Bitcoin UTXOs. Such transfer can be implicit or explicit. On one hand, in an **implicit transfer**, a UTXO_L is spent in a transaction and all the NFTs owned by such UTXO are moved to another one. On the other hand, the **explicit token ownership transfer** is performed using a Bitcoin mix transaction type. In such a transaction, more granularity is provided and multiple tokens of a single UTXO or different ones can be repacked into other UTXOs. The outputs of such a transaction will determine the new owners of the tokens. Token ownership transfer does not necessarily imply economic exchange between the sender and receiver of the token, although it is possible. In fact, the explicit transfer is intended to package multiple tokens in a specific Bitcoin UTXO, so the new package can be traded in full at a later time. The implicit transfer is the preferred option for token sales.

3.5 Token sale in Bitcoin

In a BRC721 token sale, the ownership of one or multiple NFTs is transferred in exchange for Bitcoins. There is no special transaction type to perform a sale, as it can be performed through an implicit token ownership transfer. In fact, the token sale is performed by an off-chain protocol that crafts a proper transaction that executes such an implicit token ownership transfer. The off-chain protocol between the seller and the buyer creates a Bitcoin transaction where both ownership transfer and economic exchange are atomic in the sense that both are executed or none of them takes place.

3.6 Rebase

For tokens belonging to collections that have been marked as rebaseable (during their registration in the Bitcoin network), their owners are allowed to change their `LAOScollectionAddress`. Without this transaction type, the roles of NFT collection creators/maintainers and NFT owners remain entirely separate. Collection creators/maintainers retain the right to mint and, if allowed by their smart contract logic, modify NFTs, whereas owners can trade, lend, and perform other ownership-related operations.

Rebase transactions allow NFT owners on Bitcoin to determine the content of their NFTs and define the logic governing its modification. A simple paradigmatic example is the blockchain equivalent of the *one-million-pixel image*, which gained viral popularity in the early days of the internet. Bridgeless Minting enables the creator of such a canvas to mint one NFT for each of the 1M pixels on Bitcoin. Through rebase transactions, users who acquire these pixels on Bitcoin can modify their appearance permissionlessly by updating the color metadata on the LAOS Network.

This approach extends to various use cases, including implementations of decentralized naming systems, such as the Sats Domain Protocol [13], which is designed to allow domain names to evolve while remaining owned by their holders.

4 Architecture

Our architecture is based on a well-defined syntax that specifies the various actions supported by the protocol. For each specific action, the corresponding syntax is encoded in the OP_RETURN field of a Bitcoin transaction. This encoding allows actions such as collection registration, ownership registration, transfers or other operations to be recorded immutably on the Bitcoin blockchain. By leveraging the OP_RETURN field, we follow a sustainable approach that in the blockchain ecosystem is translated to a minimalistic storage model, in which we reduce as much as possible the information included in the blockchain, in this case the Bitcoin blockchain. Furthermore, this sustainable approach ensures that the information included in the Bitcoin blockchain does not interfere with the standard use of the blockchain. To that end, we only include information in OP_RETURN outputs, which are never stored in the UTXO set. Furthermore, every action involves only a single OP_RETURN, even if multiple tokens are managed.

At the core of this architecture lies **the indexer**, a pivotal component responsible for monitoring the blockchains and ensuring the seamless operation of the protocol (Figure 1). The indexer consists of three main modules. The Bitcoin and LAOS modules are responsible for interacting with and tracking relevant transactions and events from their respective blockchains; each operates independently and is fully decoupled from the other. Sitting atop them, the Indexer Logic module, which can be designed to be stateless if desired, handles queries that require merging information from both tracking modules.

4.1 The Bitcoin Module

The **indexer's Bitcoin module** scans Bitcoin transactions to identify those containing the specific syntax defined by the protocol, embedded in the OP_RETURN field. Once identified, it decodes the syntax to determine the actions that need to be executed, such as collection registration, ownership registration, or token transfers. This process ensures the protocol's rules are followed, and token-related operations are accurately interpreted and tracked. By acting as the operational backbone, the indexer

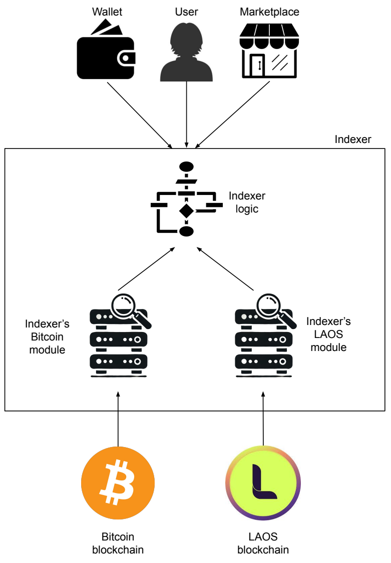

Fig. 1. High-level architecture of an indexer designed to operate within the protocol. The LAOS and Bitcoin modules interact directly with their respective blockchain nodes, parsing relevant data and maintaining an off-chain state. The Logic module communicates with these modules to provide third parties with queries that require merging information from both chains.

transforms raw blockchain transaction data into actionable insights, enabling the efficient management of token activities.

One of the indexer's primary tasks is to maintain an accurate history of token-related transactions. It tracks operations by parsing transaction outputs and decoding protocol-encoded data, such as token identifiers or destination addresses. This ensures that the entire transaction history of a token is reconstructed and token ownership is consistently updated. The indexer aggregates and reconciles all token-related activities, enabling it to maintain up-to-date information on the current ownership of BRC721 tokens linked to Bitcoin addresses. This functionality allows users to view token ownership details, and transaction histories without performing resource-intensive blockchain rescanning. Moreover, the indexer facilitates seamless integration with wallets by presenting this information in an efficient and accessible format, bridging the gap between raw blockchain data and user-friendly interfaces.

It's important to note that the indexer's Bitcoin module does not interact with the LAOS blockchain nor any other external information sources (different from the Bitcoin blockchain). Therefore, the module parses and stores all transactions identified as belonging to the protocol, regardless of the specific semantics associated to them. For instance, a transaction that registers a collection in Bitcoin before the corresponding LAOS collection has been created or different transactions that register the same `LAOScollectionAddress` in Bitcoin will also be tracked by the module. Then, the behavior of the indexer in each conflicting situation is the responsibility of the Logic module, which can be configured by the user[3] and external sources of verification can be added to the process. This ensures that configuration changes can be made without re-scanning the whole Bitcoin blockchain, a costly procedure.

Beyond its tracking and decoding capabilities, the indexer plays a critical role in optimizing queries and improving the protocol's usability. By maintaining a comprehensive and continuously updated database of token data, it eliminates the need for individual users or nodes to repeatedly scan the blockchain, significantly enhancing efficiency. The presence of multiple independent indexers further ensures decentralization and redundancy, avoiding single points of failure while aligning with Bitcoin's trustless and decentralized principles. Additionally, the indexer connects external systems, such as wallets and token management platforms, ensuring that all token-related activities are synchronized

[3] An example of this situation is given in Section 5.2.

and managed according to the protocol's rules. By centralizing the interpretation and dissemination of blockchain data, the indexer is indispensable for enabling seamless integration, robust token management, and efficient operation across the ecosystem.

4.2 The LAOS Module

Most of the features described for the Bitcoin module also apply to the **indexer's LAOS module**. Like the Bitcoin module, it operates in a fully decoupled manner and optimizes queries to enhance the protocol's usability.

There are, however, two key differences. First, it does not need to track token ownership transfers; instead, it only monitors the minting of tokens and, if permitted by the specific smart contract logic of a collection, modifications to their content. Since all BRC721 token transfers and other ownership-related transactions are fully governed on-chain on Bitcoin, smart contracts on LAOS that adhere to the protocol outlined in this paper must not implement transfer, lending, or similar functionalities. Even if they do, the LAOS module (and the indexer Logic module) would entirely disregard them.

Second, since LAOS is an EVM-programmable blockchain, the indexer does not rely on any off-the-shelf parsing protocols. All necessary data tracking is achieved through on-chain transactions that explicitly adhere to the Ethereum Virtual Machine specification [23].

4.3 The Logic Module

The Logic module is responsible for merging information from both the Bitcoin and LAOS modules and serving it to applications, users, or developers utilizing the protocol presented here.

While a detailed proposal for the Logic module's behavior is outlined in the following sections, we first provide a high-level example to illustrate its functionality.

One example is a marketplace that wants to display all assets minted within a specific Bitcoin collection. Given a query with a `BitcoinCollectionID`, the Logic module first queries the Bitcoin module to verify whether that collection exists on Bitcoin. If the Bitcoin module confirms its existence, it returns the associated `LAOScollectionAddress`. The Logic module then queries the LAOS module to retrieve all NFTs minted under that `LAOScollectionAddress`, which responds with the corresponding `TokenId` values and metadata.

If the marketplace also wants to display the current owners of these NFTs, the Logic module proceeds by querying the Bitcoin module for the ownership status of each `TokenId`. It receives responses for a subset of them: those that have been either registered or traded following the protocol described below. For any `TokenId` that is not part of this subset, the Logic module returns the initial owner, which can be derived directly from the `TokenId` using the address mapping detailed in Section 5.1.

All complexity surrounding blockchain reorgs and data parsing lies beyond the Logic module. In fact, it can be implemented in a stateless manner if desired, simply providing query responses based on the latest canonical chains as defined by the Bitcoin and LAOS consensus protocols.

5 Protocol specification

5.1 Address mapping and encoding

When dealing with EVM compatible addresses, the 160 bits identifier assigned to a BRC721 token, as defined in Equation 1, can be directly identified with an EVM address, so the bijection function between an EVM compatible address and the LAOS asset identifier can be defined by the identity function. However, Bitcoin addresses have a different structure than EVM addresses, so a correspondence between BRC721 token identifiers and Bitcoin addresses needs to be defined. The address mapping is defined in this section (as well as the associated means of proving ownership) and later used in Section 5 to fully specify the protocol.

In the Bitcoin network, ownership of a BRC721 token is assigned to a 160-bit Bitcoin `scriptPubKey` hash (referred to as `h160Address`). Owner authentication is mainly performed using digital signatures, though other verification methods may also be used.

Specifically, to prove ownership of a Bitcoin `h160Address` h, the prover needs to spend a UTXO such that $h = H160(\texttt{scriptPubKey})$. In the Bitcoin context, $H160$ corresponds to applying the RIPEMD-160 hash function to the result of applying the SHA256 function to the input, that is, $H160(x) = \texttt{RIPEMD-160}(\texttt{SHA-256}(x))$.

Ownership can thus be proven by spending any UTXO. Typically, spending a UTXO requires providing a digital signature that validates against the public key encoded in the UTXO. This signature may be either ECDSA or Schnorr, and be included in the transaction's inputs (`scriptSig`) or witness field, depending on the script type.

For instance, in a `P2TR` output, the `scriptPubKey` in the output contains the taproot public key, preceded by the opcode `OP_1`, which signals SegWit version 1. Spending such an output proves ownership of the `h160Address` $h = H160(\texttt{OP\_1 <PK>})$. Using P2TR enables multiple ownership verification alternatives. The standard approach is a key path spend, where ownership is proven by providing a Schnorr signature specified in the witness field:

```
witness:      <S>
scriptSig:    (empty)

scriptPubKey: OP_1 <PK>
              (0x5120{32-byte-public-key})
```

Alternatively, a script path spend allows ownership to be demonstrated using a custom locking script encoded within the tweaked public key. These custom scripts may also involve Schnorr signatures, but can also enforce any arbitrary validation rules expressible in Bitcoin Script.

Bitcoin encodes output scripts in the form of **Bitcoin addresses**. While any valid Bitcoin address can be decoded into an `h160Address`, reconstructing a Bitcoin address from an `h160Address` requires knowledge of the original `scriptPubKey` (since hash functions are one-way). Given the `scriptPubKey`, the corresponding Bitcoin address can be derived based on the specific script type.

5.2 Bitcoin Module specification and syntax

This section details the exact syntax for each defined action of the protocol, as well as the operations performed by the Bitcoin Module on Bitcoin transactions that contain protocol-related information.

Register a collection. Registering a collection on Bitcoin is carried out through a **register collection transaction** type. Since collection registration on Bitcoin is permissionless, there are no restrictions on the inputs of a register collection transaction.

The $\texttt{Output}_0$ is an OP_RETURN that encodes the registration as follows:

– OP_15
– SERIALIZATION OF:
  • `REGISTER_COLLECTION_FLAG`
  • `LAOScollectionAddress`
  • rebaseable (bool)

where `REGISTER_COLLECTION_FLAG = 0` signals that the syntax adheres to that of a register collection transaction.

When processing such a transaction, the Bitcoin Module must execute the following steps. If any verification step fails, the Bitcoin Module must discard the entire transaction:

1. Discard all transaction inputs.

2. Discard all transaction outputs except $\texttt{Output}_0$.
3. Verify the OP_RETURN syntax of $\texttt{Output}_0$.
4. Assign a `BitcoinCollectionID` to the newly created collection, determined by the transaction ID of the register collection transaction. This assignment occurs once the transaction has been confirmed in the blockchain, using:

$$\texttt{BitcoinCollectionID = BlockID:TxIndex}. \tag{2}$$

5. Store the pair `(BitcoinCollectionID, LAOScollectionAddress)`, establishing the link between the newly created collection and its corresponding LAOS collection.

Let us reiterate that the Bitcoin Module operates independently of whether a smart contract exists on LAOS at `LAOScollectionAddress`. While the standard flow generally involves creating a collection on LAOS before registering it on Bitcoin, the Logic Module is responsible for merging information from both blockchains.

Token Ownership Registration. BRC721 token ownership registration is carried out through a **token ownership registration transaction** type. The purpose of this transaction is to register one or more NFTs within a series of m UTXOs, ensuring that each NFT is owned by the lock script of its respective UTXO and can potentially be traded.

This transaction must include at least one input and at least $m + 1$ outputs:

$$\texttt{TX = \{ Input}_0\texttt{ \} \{Output}_0\texttt{, Output}_1\texttt{, } \cdots \texttt{, Output}_m\texttt{ \}}$$

All inputs beyond $\texttt{Input}_0$ and all outputs beyond the first $m + 1$ are discarded by the protocol.

The first input must spend an output whose `h160Address` corresponds to the last 160 bits of a `TokenID` (see Section 5.1 for details).

$\texttt{Output}_0$ contains an OP_RETURN that encodes the `BitcoinCollectionID` and specifies which of the 2^{96} tokens associated with that address are registered in this transaction. Once a register collection transaction is accepted, every `h160Address` becomes the initial owner of 2^{96} tokens within that collection, with each `TokenId` uniquely determined by the pair `(slot number, h160Address)` as per Equation (1).[4] Thus, the initial owner of a set of tokens can fully specify which tokens to register on Bitcoin by simply providing a set of slot numbers.

We introduce the concept of a *slot range*, S, which represents a list of consecutive slots encoded by the first and last slot indices. It is defined as $S = (\alpha_i, \alpha_j)$, where $0 \leq \alpha_i < \alpha_j < 2^{96}$. If a slot contains only a single element, it can be represented using a single index, $S = \alpha_i$.

The OP_RETURN contains:

- OP_15
- SERIALIZATION OF:
  - `REGISTER_OWNERSHIP_FLAG`
  - `BitcoinCollectionID`
  - 1: $(S_1, \cdots, S_i)$
  - ...
  - m: $(S_j, \cdots, S_k)$

where `REGISTER_OWNERSHIP_FLAG = 1` signals that the syntax adheres to that of a register ownership transaction.

Besides this OP_RETURN, the ownership registration transaction must include at least m additional outputs, at `TxIndex = {1,...,m}`, each receiving ownership of the corresponding slot ranges defined in the OP_RETURN, while preserving the same order.

When processing such a transaction, the Bitcoin Module must execute the following steps. If any verification step fails, the Bitcoin Module must discard the entire transaction:

[4] Determining which subset of these tokens is actually minted on the LAOS chain falls outside the scope of the Bitcoin Module.

1. Verify the OP_RETURN syntax of $\texttt{Output}_0$.
2. Verify that a collection with the specified `BitcoinCollectionID` has been previously created on Bitcoin.
3. Verify that the total number of outputs in the transaction, including the OP_RETURN, is at least $m+1$.
4. Discard all inputs except $\texttt{Input}_0$ and all outputs beyond the first $m+1$. The `h160Address` of the UTXO spent by the first input is used as the last 160 bits of all `TokenID` values registered in this transaction.
5. Verify that transaction is valid (including that the script is successfully validated).
6. Verify that none of the tokens included in the k slot ranges specified in the OP_RETURN have been registered before on the Bitcoin blockchain for the given `h160Address` and `BitcoinCollectionID`.
7. Assign NFT ownership of all tokens with `TokenId` formed via equation (1), using the input `h160Address` and each slot specified in $\texttt{Output}_0$, to the corresponding $\{\texttt{Output}_1, \cdots, \texttt{Output}_m\}$.
8. Assign (implicitly or explicitly) a sorting order to the NFTs within each output, following the natural consecutive ordering inherited from the slot range specification on each UTXO.
9. Add $\{\texttt{Output}_1, \cdots, \texttt{Output}_m\}$ to the UTXO_L set of all unspent LAOS NFT Bitcoin outputs.

Token Ownership Transfer.

Implicit: BRC721 token ownership transfer can be performed implicitly by simply spending a UTXO in the UTXO_L set. If a transaction includes an input that spends a UTXO_L, the Bitcoin Module must transfer NFT ownership to an output of the same transaction as follows:

– Inputs that do not contain NFTs are discarded.
– Outputs that are pure OP_RETURN are discarded.
– NFTs are moved sequentially from the remaining inputs to the remaining outputs.
– If there are more remaining outputs than remaining inputs containing NFTs, the excess outputs are discarded.
– If there are fewer remaining outputs than remaining inputs containing NFTs, the last remaining output will receive all unassigned NFTs from the remaining inputs.

If the only outputs in the transaction are OP_RETURN, the ownership of the NFTs included in the inputs is burned, effectively assigning them to the null `h160Address`. Any future attempt to register or transfer ownership of burned tokens will therefore fail.

Explicit: Mix Transaction BRC721 token ownership transfer can also be explicitly performed through a **mix transaction**, which allows for more granular control over the transfer process compared to an implicit transfer. This type of transaction must contain a set of n inputs (for $n \geq 1$) from the UTXO_L set and at least $m+1$ outputs (for $m \geq 1$).

$$\texttt{TX} = \{\ \texttt{Input}_0,\ \texttt{Input}_1,\ \cdots\ ,\ \texttt{Input}_{n-1}\ \}\ \{\texttt{Output}_0,\ \texttt{Output}_1,\ \cdots,\ \texttt{Output}_m\ \}$$

All inputs and outputs with indices greater than $n-1$ and $m+1$, correspondingly, are discarded, being n and m values specified in the OP_RETURN.

$\texttt{Output}_0$ contains an OP_RETURN that encodes the rules for how the mix will be performed in the transfer. Specifically, the mix must define a mapping between each NFT in the n inputs, $\texttt{Input}_0, \cdots$, $\texttt{Input}_{n-1}$, and each of the m outputs, $\texttt{Output}_1, \cdots, \texttt{Output}_m$.

Notably, different inputs may contain NFTs registered under different `BitcoinCollectionID`s. Likewise, as will become evident when defining the mix transaction, even a single UTXO in the UTXO_L set can hold NFTs belonging to multiple `BitcoinCollectionID`s.

A straightforward method to define the mix mapping, while avoiding unnecessary complexity, is to assign a consecutive index to each NFT in the inputs. Within a single input, all NFTs inherit a natural ordering, either from their initial registration in a collection transaction, which establishes a defined sequence as described earlier, or from a previous mix transaction, which, as will be shown, also preserves a structured ordering.

The sorting across inputs follows a straightforward process:

- The first NFT in $\texttt{Input}_0$ is assigned index $i = 0$.
- The first NFT in $\texttt{Input}_1$ is assigned index $i = N_0$, where N_0 represents the number of NFTs in $\texttt{Input}_0$.
- ...
- The last NFT in $\texttt{Input}_{n-1}$ is assigned index $i = N_n - 1$, where N_n denotes the total number of NFTs across all inputs.

The specification of the mix transaction follows a pattern similar to that of the ownership registration but uses *index ranges*, $I = (i, j)$, with $0 \leq i < j < N_n$, instead of *slot ranges*. A special symbol, $\overline{I}$, is reserved for the *complementary index range*, defined as all indices i with $0 \leq i < N_n$ that are not explicitly referenced by any index range in the OP_RETURN of this transaction. This *complementary index range* $\overline{I}$ must be present as exactly one of the full set of index ranges.[5] Enforcing the inclusion of this set prevents ambiguities regarding input NFTs that are not explicitly referenced. A common (though not mandatory) convention is to assign the last transaction output, $\texttt{Output}_m$, as the collector for all NFTs that are not explicitly specified.

The OP_RETURN must contain:

- OP_15
- SERIALIZATION OF:
  - `MIX_FLAG`
  - 1: $(I_1, \cdots, I_i)$
  - ...
  - $m - 1$: $(I_j, \cdots, I_l)$
  - m: $\overline{I}$

where `MIX_FLAG = 2` signals that the syntax adheres to that of a mix transaction.

When processing such a transaction, the Bitcoin Module must execute the following steps. If any verification step fails, the Bitcoin Module must discard the entire transaction:

1. Verify the OP_RETURN syntax of $\texttt{Output}_0$.
2. Verify that $m \geq 1$.
3. Verify that all inputs belong to the current UTXO_L set.
4. Verify that $\overline{I}$ is present as exactly one of the index ranges in $\texttt{Output}_0$.
5. Verify that the total number of outputs is at least $m + 1$.
6. Discard all outputs beyond the first $m + 1$.
7. Compute N_n as the total number of NFTs contained across all inputs.
8. Compute N_T as the largest index appearing in the index ranges specified in the OP_RETURN.
9. Verify that $N_n \geq N_T > 0$.
10. Assign each NFT ownership to the corresponding output UTXO according to the index range map specified in $\texttt{Output}_0$.
11. Verify that there is no overlapping between all index ranges specified in the OP_RETURN.
12. Assign any NFTs in the inputs that are not explicitly mapped by the index ranges to the output containing $\overline{I}$. This includes cases where the index ranges do not form a contiguous set, as well as NFTs with index $i \geq N_T$ in cases where N_n exceeds N_T.
13. Establish (implicitly or explicitly) a sorting order for NFTs within each output, following the natural consecutive ordering inherited from the specification of index ranges on each UTXO.
14. Add $\{\texttt{Output}_1, \cdots, \texttt{Output}_{m-1}\}$ to the UTXO_L set of all unspent LAOS NFT Bitcoin outputs.
15. If $\overline{I} \neq \emptyset$, add $\{\texttt{Output}_m\}$ to the UTXO_L set of all unspent LAOS NFT Bitcoin outputs.
16. Assign each NFT ownership to the corresponding UTXO_L.
17. Update the UTXO_L set.

Only the rightful on-chain owner(s) of the NFTs on the Bitcoin network can execute this mix transaction, as the corresponding unlock scripts must be provided for each input.

[5] Note that inclusion is mandatory even if, for a given transaction, $\overline{I}$ is the empty set.

Token Trading. BRC721 token trading is performed through an implicit token ownership transfer, following a protocol in which the buyer and seller collaboratively construct a Bitcoin transaction that implicitly transfers one or multiple UTXOs from the UTXO_L set containing the corresponding tokens. From the perspective of all modules within the Indexer, a token trading transaction is indistinguishable from an implicit token ownership transfer, requiring no additional specification. The involved parties are responsible for correctly structuring the inputs and outputs to facilitate the transfer of NFTs for Bitcoin.

Rebase. The final transaction type supported by the protocol is the *Rebase Transaction*, which enables the creation of unique, non-fungible tokens whose content can be modified by their owners at any time.

For a Bitcoin collection to support rebase transactions, it must explicitly declare this capability in its initial Register Collection Transaction, as outlined in Section 5.2. If the `rebaseable` boolean is set to `false`, all rebase transactions directed at that collection must be disregarded.

The rebase transaction is nearly identical to the mix transaction, with a minor difference in the OP_RETURN syntax at $\texttt{Output}_0$. As in the mix transaction, it must include a set of n inputs (for $n \geq 1$) from the UTXO_L set and at least $m + 1$ outputs (for $m \geq 1$).

$$\texttt{TX = } \{ \texttt{ Input}_0\texttt{, Input}_1\texttt{, } \cdots \texttt{ , Input}_{n-1} \} \{\texttt{Output}_0\texttt{, Output}_1\texttt{, } \cdots \texttt{, Output}_m \}$$

All inputs and outputs with indices greater than $n - 1$ and $m + 1$, respectively, are discarded, being n and m values specified in the OP_RETURN.

The syntax of the OP_RETURN must contain:

- OP_15
- SERIALIZATION OF:
  - `REBASE_FLAG`
  - `NewLAOScollectionAddress`
  - 1: $(I_1, \cdots, I_i)$
  - ...
  - m: $(I_j, \cdots, I_k)$

where `REBASE_FLAG = 3` signals that the syntax adheres to that of a mix transaction.

As in the mix transaction, index ranges specify the mapping between all NFTs in the input transactions and the outputs $\{\text{Output}_1, \text{Output}_1, \cdots, \text{Output}_m \}$. Additionally, it defines the new LAOS collection address associated with the NFTs in all outputs.

When processing such a transaction, the Bitcoin Module must execute the following steps. If any verification step fails, the Bitcoin Module must discard the entire transaction:

1. Verify that all NFTs in the inputs belong to one and only one `BitcoinCollectionID`.
2. The following steps are performed exactly as in the mix transaction:
   (a) Apply the same verifications, incorporating `NewLAOScollectionAddress` in the syntax check.
   (b) Assign NFT ownership to the corresponding output UTXO.
   (c) Establish a sorting order for NFTs within each output.
   (d) Add $\{\texttt{Output}_1\texttt{, } \cdots \texttt{, Output}_m\}$ to the UTXO_L set of all unspent LAOS NFT Bitcoin outputs.
3. Additionally, maintain the relationship between all output NFTs and `NewLAOScollectionAddress`.

The typical usage of this pattern will become clearer in Section 5.4, where the Indexer Logic Module is specified.

5.3 LAOS Module specification

The LAOS Module is relatively straightforward, as it primarily follows the standard behavior of indexers on EVM-compatible chains. Its role is to track a set of smart contracts on the LAOS Network and monitor the events they emit when minting and evolving assets.

Since NFT ownership is fully managed on-chain within Bitcoin, the LAOS Module does not need to track any transactions related to ownership—such as trading, even if the relevant smart contracts on LAOS were to include such functionality.

A simple specification that leverages protocol-level methods within the LAOS Network for greater efficiency is as follows. First, recall that a BRC721 token on LAOS is uniquely defined by the smart contract that minted it and its identifier within that contract, i.e., by the pair {`LAOScollectionAddress, TokenID`}.

The LAOS Module must:

- Track all collections created by the Collection Factory precompiled smart contract on LAOS. Upon the creation of a new collection, add its collection address to the `LAOScollectionAddress` set.
- Track all minting events emitted by smart contracts within the `LAOScollectionAddress` set. Upon each new mint, add the newly minted token to `LAOSTokens`, including its `TokenID` and metadata.
- Track all modifications to the metadata of tokens within the `LAOSTokens` set, as performed within the smart-contract on their respective `LAOScollectionAddress`.

Note that collection creators can choose to deploy smart contracts that enforce metadata immutability, preventing any modifications to token content after minting.

Finally, note that the LAOS Module operates entirely independently from the Bitcoin Module.

5.4 Logic Module specification

The Logic Module is responsible for merging information from the Bitcoin and LAOS modules to serve it to applications adhering to the protocol. This module can be bypassed if desired; users, developers, and applications can directly query information from Bitcoin, for instance, without requiring this layer. However, the Logic Module is designed to integrate information in a structured manner, facilitating the use of the protocol.

The Logic Module is specified only for queries regarding collections created on-chain in Bitcoin, identified by their corresponding `BitcoinCollectionID`, although responses may depend on additional information fetched from the LAOS Module. In all cases, responses will always be based on the leading blockchain branches at query time, as is standard in any blockchain system.

Similarly, the module is only specified for queries about BRC721 tokens associated with Bitcoin collections, which are identified by the pair `BitcoinCollectionID, TokenID`.

The module must:

- Consider that a collection exists (or is fully set up) only if both of the following conditions are met:
  - The collection has been registered on-chain in Bitcoin and, hence, it already has an associated `BitcoinCollectionID` and `LAOScollectionAddress`.
  - The associated `LAOScollectionAddress` has been tracked by the LAOS Module and, hence, is part of the `LAOScollectionAddress` set.
- Consider that a BRC721 token exists, identified by `BitcoinCollectionID, TokenID`, only if both of the following conditions are met:
  - The `BitcoinCollectionID` exists, as per the previous condition.
  - A token is on the `LAOSTokens` set of the LAOS Module with the same `TokenID` minted on the `LAOScollectionAddress` associated to `BitcoinCollectionID`.
- Consider that the owner of a BRC721 token, which is a purely Bitcoin-native concept, is as follows:
  - If the token does not exist according to the previous conditions, ownership is not specified.
  - If the token is included in a UTXO within the UTXO_L set, its owner is the owner of that UTXO.
  - If the token is not included in the UTXO_L set, its owner is the initial owner, that is, the `h160Address`, which can be derived from the `TokenID` using Equation (1).
- The content/metadata of a BRC721 token should only be returned if the token exists according to the previous requirements, as follows:

- If `rebaseable = false`, then the content is as defined by the pair {`LAOScollectionAddress, TokenID`} on LAOS, where `LAOScollectionAddress` is the address associated with the provided `BitcoinCollectionID`.
- The above condition is modified only if all of the following are met:
  - `rebaseable = true`, as specified in the collection registration transaction for the provided `BitcoinCollectionID`.
  - The token is registered in a UTXO with an associated `NewLAOScollectionAddress` via a rebase transaction.
  - A `TokenID` exists in the LAOS smart contract at `NewLAOScollectionAddress`.

  In this case, the content is defined by the pair {`NewLAOScollectionAddress, TokenID`} on LAOS.

6 Security analysis

This section provides a security analysis of the protocol, outlining the adversarial model and explaining how the protocol mitigates various adversarial strategies.

6.1 Adversarial model

This section outlines the threat model, focusing on describing the goals and capabilities of potential adversaries of the bridgeless minting protocol.

Our adversary model considers the following objectives for the attacker of the system:

O1 Claim ownership of non-owned tokens.
O2 Force the transfer of non-owned tokens to a third party (or burn them).
O3 Perform a Denial of Service (DoS) attack to prevent users to register collections or tokens, or transfer tokens.

In our adversary model, we consider an **active attacker**, that is, an adversary not limited to passively observing the involved blockchains (Bitcoin and LAOS), but who can actively interact and interfere with the protocol execution. Specifically, the attacker is assumed to be able to observe any information publicly available (e.g. the content of the blockchains), eavesdrop on open P2P network traffic (such as the Bitcoin P2P network), interact with both networks (e.g. send transactions), and modify or drop other parties' interactions with the networks (e.g. change some bytes of a Bitcoin transaction sent by another user to the P2P network).

We also consider a more powerful adversary, the **active attacker with mining capabilities**, who, in addition to the previously described abilities, controls a significant portion[6] of Bitcoin's hash rate and can mine on the Bitcoin blockchain.

The security analysis presented in this section focuses exclusively on the bridgeless minting protocol as defined in this paper. It evaluates potential attack vectors and adversarial strategies within the protocol's design and execution. However, broader security threats that fall outside the protocol's scope (such as social engineering attacks, side-channel exploits, or the compromise of users' wallets) are not considered. These external threats, while relevant to the overall security of a real-world deployment, require separate mitigation strategies beyond the protocol's design.

6.2 Adversarial strategies and prevention

In this section we explain possible strategies an attacker may follow to achieve the goals described in the adversarial model and how the protocol prevents them.

An adversary may attempt to illegitimately claim ownership of tokens (**Objective O1**) by submitting ownership registration or transfer transactions that assign non-owned tokens to a UTXO under

[6] Details on the amount of hash power needed are discussed afterwards in Section 6.2.

their control on the Bitcoin blockchain. The active attacker may attempt this by either crafting and sending the transaction herself to the Bitcoin P2P network (Strategy S1), or by performing a man-in-the-middle attack to a transaction sent by the legit owner (Strategy S2). An active attacker with mining capabilities also has the alternative to mine the transaction herself, instead of broadcasting it to the P2P network (Strategy S3). Let's analyze each possible strategy an adversary might take to accomplish Objective O1 and examine how the protocol mitigates these threats:

S1 For an attacker to craft a token ownership registration or transfer transactions that assigns tokens to another UTXO under their control, they must first prove ownership of the corresponding `scriptPubKey`. This proof is enforced by Bitcoin's consensus rules and relies on the security of its cryptographic primitives and script evaluation.
For legacy (spendable) standard script types, ownership proof requires providing an ECDSA signature generated with the private key corresponding to the public key specified in the output. Consequently, to successfully execute the attack, the adversary would need to forge ECDSA signatures (or, in the case of P2PKH and P2WPKH, find a hash collision that links a public key under their control to the targeted hash). However, both ECDSA over secp256k1 and SHA-256 (and RIPEMD-160) are currently considered cryptographically secure primitives. Similarly, P2TR outputs mainly require a Schnorr signature, which is also considered cryptographically secure. However, not all `scriptPubKey` require signatures for unlocking. Bitcoin's scripting language allows for arbitrary unlocking conditions, enabling a wide range of spending rules. In such cases, security depends on the correctness of the locking script.
Another approach an attacker might attempt is to broadcast a transaction that spends the UTXO but includes an invalid unlocking script, such as an incorrect signature. However, since Bitcoin nodes strictly enforce transaction validation, any transaction with an invalid unlocking script (e.g. invalid signature) is immediately rejected and fails to propagate through the network.

S2 If an attacker attempts a man-in-the-middle attack by modifying the output of a registration or transfer transaction to redirect ownership to a UTXO they control, the transaction will become invalid. This is because altering the transaction data changes its hash, rendering the original digital signature invalid. Again, this transaction won't propagate through the network.
However, this attack could succeed if the spent `scriptPubKey` does not require a signature for unlocking. In such cases, an attacker could modify the transaction outputs while keeping it valid, effectively achieving their goal. To prevent this, spending conditions must always enforce signature verification.

S3 If the attacker possesses mining capabilities, they might attempt to include a fraudulent transaction which does not include a valid ownership proof in a block they mine. However, this approach is also ineffective because an invalid transaction would invalidate the entire block, causing the network to reject it. Furthermore, mining an invalid block results in economic losses for the attacker, as they would forfeit the associated reward. Given Bitcoin's competitive mining landscape, there is no rational incentive for an attacker to attempt this strategy.

An adversary may attempt to force the transfer of non-owned tokens to a third party or render them unspendable (i.e. burn them) (**Objective O2**) following the same strategies as those employed to achieve Objective O1. However, instead of including outputs under their own control in the fraudulent transactions, the adversary now includes outputs belonging to other parties or inherently non-spendable outputs. The prevention mechanisms in this case are exactly the same than for Objective O1.

An active adversary may try to perform a Denial of Service (DoS) attack (**Objective O3**) by dropping all intercepted protocol messages from the victim to the Bitcoin P2P network (Strategy S4). If the attacker has mining capabilities, she can also censor those transactions in the blocks she mines (Strategy S5). However, due to Bitcoin's decentralized architecture, sustaining such attacks over time is both technically challenging and economically costly for the attacker.

S4 In this scenario, the adversary must monopolize all of the victim's connections to the Bitcoin P2P network. If even a single alternative connection remains open, the victim's transactions can still propagate through the network, making the attack unsuccessful. Achieving complete control over a victim's connections requires significant resources, making this strategy difficult to execute consistently ([7], [20], [27]).

S5 In the second case, the attacker attempts to censor transactions by excluding them from the blocks they mine. To exert continuous control over transaction inclusion in the blockchain, the adversary must control a substantial portion of the network's total hash power. Previous studies suggest that an attacker needs to control at least 33% of the total mining power to significantly and consistently censor transactions [6]. However, given Bitcoin's competitive mining environment and the associated energy costs, sustaining such a level of control is economically prohibitive for most adversaries.

Therefore, the security of the token ownership in the presented protocol is fundamentally anchored to the robustness of the underlying Bitcoin network.

While unlikely, another potential adversarial scenario to consider is the complete disappearance of the LAOS blockchain. This would require an extreme situation in which no nodes are generating new blocks or, even more severely, all LAOS nodes go offline with no archival nodes remaining, rendering the blockchain entirely inaccessible. Although such an event would be detrimental to users of the bridgeless minting protocol, note that it does not affect the attacker's objectives within our adversarial model. Even if LAOS becomes unavailable, BRC721 token ownership on Bitcoin remains unaffected. Token owners may no longer be able to retrieve the content of the tokens, but their ownership will remain intact, and users will still be able to securely transfer ownership of existing tokens.

7 Conclusions

This paper presents a protocol that enables bridgeless minting of NFTs on the LAOS blockchain while managing their ownership natively on the Bitcoin network. By leveraging Bitcoin's OP_RETURN opcode and introducing a minimal and efficient encoding for asset-related data, we achieve on-chain data availability and ownership verification.

The proposed BRC721 standard enables NFTs to be pre-minted on LAOS and registered, transferred, and traded through standard Bitcoin transactions. The ownership model, based on the Bitcoin UTXO structure, ensures compatibility with existing Bitcoin wallets and preserves the trustless nature of the network. A modular architecture with decoupled Bitcoin and LAOS modules, combined with a stateless logic module, ensures the scalability, flexibility, and adaptability of the system. The proposed protocol is designed to be indexer-friendly and maintainable while remaining minimalistic in its blockchain footprint.

Finally, we also provided a detailed security analysis showing the robustness of our approach against active attackers, including those with mining capabilities, demonstrating that ownership integrity relies solely on the well-established security of Bitcoin's cryptographic primitives and consensus mechanism.

Future work will focus on reference implementations, real-world deployment scenarios, and performance optimization of indexers and wallet integrations.

References

1. Alessandro Siniscalchi, T.M., Evans, A.: Laos: Vision for a scalable, bridgelessly connected, truly non-custodial, dynamic nft protocol (2023), https://github.com/freeverseio/laos-whitepaper/blob/main/laos.pdf
2. Ali, M., Nelson, J., Shea, L., Freedman, M.J.: Stacks 2.0: A secure layer-1 blockchain with smart contracts built on bitcoin. Stacks Whitepaper (2019), https://stacks.org/whitepaper.pdf
3. Assia, Y., Buterin, V., et al.: Colored coins whitepaper (2013), https://www.etoro.com/wp-content/uploads/2022/03/Colored-Coins-white-paper-Digital-Assets.pdf
4. Blockstream: The liquid network: Bitcoin sidechain for traders and exchanges (2018), https://blockstream.com/liquid/
5. Developers, C.: Counterparty protocol documentation (2014), https://counterparty.io
6. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. Communications of the ACM **61**(7), 95–102 (2018)
7. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on Bitcoin's peer-to-peer network. In: 24th USENIX security symposium (USENIX security 15). pp. 129–144 (2015)

8. Jacob Evans William Entriken, D.S., Sachs, N.: Erc-721: Non-fungible token standard. ethereum improvement proposals. https://eips.ethereum.org/EIPS/eip-721 (2018), accessed on 2018-01-01
9. Labs, A.: Ark labs: Enabling efficient bitcoin transactions (2024), https://arklabs.com/
10. Labs, L.: Taro: Bitcoin asset protocol using taproot (2022), https://lightning.engineering/posts/2022-04-05-taro
11. Labs, R.: Rsk: Smart contracts for bitcoin (2017), https://www.rsk.co/
12. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf (2008), accessed on 2024-12-09
13. Names, S.: Sats domain protocol documentation (2025), https://docs.satsnames.org/, accessed: 2025-03-24
14. Network, L.: Laos network developer documentation. https://docs.laosnetwork.io (2025), accessed on 2025-01-28
15. Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016), https://lightning.network/lightning-network-paper.pdf
16. Prisco, G., Team, R.: Rgb: Scalable and privacy-preserving smart contracts on bitcoin (2020), https://rgb-org.github.io
17. Rodarmor, C.: Ordinals: Inscribing digital artifacts on bitcoin. https://ordinals.com (2023), accessed on 2023-12-09
18. Rodarmor, C.: Runes protocol: Fungible tokens on bitcoin. https://ordinals.com (2024), accessed on 2024-12-09
19. Team, O.A.: The open assets protocol (2014), https://openassets.org
20. Tran, M., Shenoi, A., Kang, M.S.: On the {Routing-Aware} peering against {Network-Eclipse} attacks in bitcoin. In: 30th USENIX Security Symposium (USENIX Security 21). pp. 1253–1270 (2021)
21. Various: Brc-20: Fungible token standard on bitcoin (2023), accessed from various online resources discussing Ordinals and Taproot.
22. Willett, J.: Mastercoin: A second-generation protocol on the bitcoin blockchain (2012), https://www.omnilayer.org
23. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Yellow Paper (2014), available at https://ethereum.github.io/yellowpaper/paper.pdf
24. Wood, G.: Polkadot: Vision for a heterogeneous multi-chain framework. Whitepaper (2016), https://polkadot.network/PolkaDotPaper.pdf
25. Wood, G.: The cross-consensus message format. https://polkadot.com/blog/xcm-the-cross-consensus-message-format (2021), accessed on 2021-09-06
26. Wood, G., Developers, P.: Xcm v3. https://github.com/paritytech/polkadot/pull/4097 (2023), accessed on 2023-01-17
27. Yves-Christian, A.E., Hammi, B., Serhrouchni, A., Labiod, H.: Total eclipse: How to completely isolate a bitcoin peer. In: 2018 Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC). pp. 1–7. IEEE (2018)