Technical Documentation for Minstordinals

Galois Field, Noé de Larminat

January 11, 2024

Contents

1 Introduction

1.1 Purpose of the Technical Documentation

The purpose of this technical documentation is to provide comprehensive information about the Minstordinals project. It is intended for developers, users, and anyone interested in understanding and implementing Minstordinals. This documentation covers the technical aspects, installation, configuration, usage, and more, enabling readers to navigate and utilize Minstordinals effectively.

1.2 Prerequisites

Before diving into the technical details and implementation of Minstordinals, it's essential to be aware of the prerequisites necessary for working with this project. Here are the prerequisites:

1.2.1 Hardware Requirements

As Minstordinals is still under development we are purposing here the maximal requirement for us to generate fully this service.

- Adequate storage space for the application and data. Bitcoin full node and `ord` indexed (around 650Gb in total). Moreover, storage for `minstordinals` indexing ;
- A reliable internet connection for interacting with blockchain networks. Calls to archival nodes, bitcoin rpc APIs and ordinals indexers ; (not free see Mempool Entreprise yearly BTC subscription from 0.5 BTC to 2 BTC per year)
- Enough computer power for the Verifier (BOS workflow verification process).

1.2.2 Software Requirements

To install and run Minstordinals, you'll need the following software :

- Operating System: Minstordinals is platform-agnostic and should be used on every major operating systems, including Windows, macOS, and Linux.
- Dependencies: Minstordinals relies on specific near and taproot/ordinals software libraries and tools, which would be covered in the "Installation and Setup" section.

1.2.3 Skills and Knowledge

While Minstordinals aims to be accessible to a wide audience, having the following skills and knowledge will be advantageous:

- Basic understanding of blockchain technology, NFTs (Non-Fungible Tokens), and Bitcoin Ordinals. Understanding of `minsta` framework ;

- Familiarity with programming languages, mainly `javascript, typescript` and `Rust` to modify the code ;
- Knowledge of how to use command-line tools for software installation and management. Mainly `ord` and `near-cli-rs`.

1.3 Terminology

To facilitate understanding, let's clarify some of the key technical terms and jargon used throughout this documentation:

- **Bitcoin Ordinals/inscription**: Encoding data directly into the Bitcoin's transaction outputs through Ordinal enveloppe ;
- **JSON-based protocol**: A standardized format for representing protocol deployed on Bitcoin ordinals transactions.
- **Cross-Minting process**: The process of creating or issuing NFTs, with a unique token on near and corresponding inscription on Near Protocol ;
- **Verification**: Confirming the authenticity of a `minstordinals` Bitcoin inscription.

Throughout this documentation, these terms will be used consistently to explain and describe the various aspects of Minstordinals.

2 Installation and Setup

Still under consideration. Each tech are not decided.

2.1 System Requirements

2.2 Installation Process

We should use a client server model. Where the client should be built with `npm` and server in Rust compiled through `cargo`.

2.3 Configuration

It will depend about services used to rely each data. We can built our own nodes to rely on ourselves or use tiers-API/RPC to access required data.

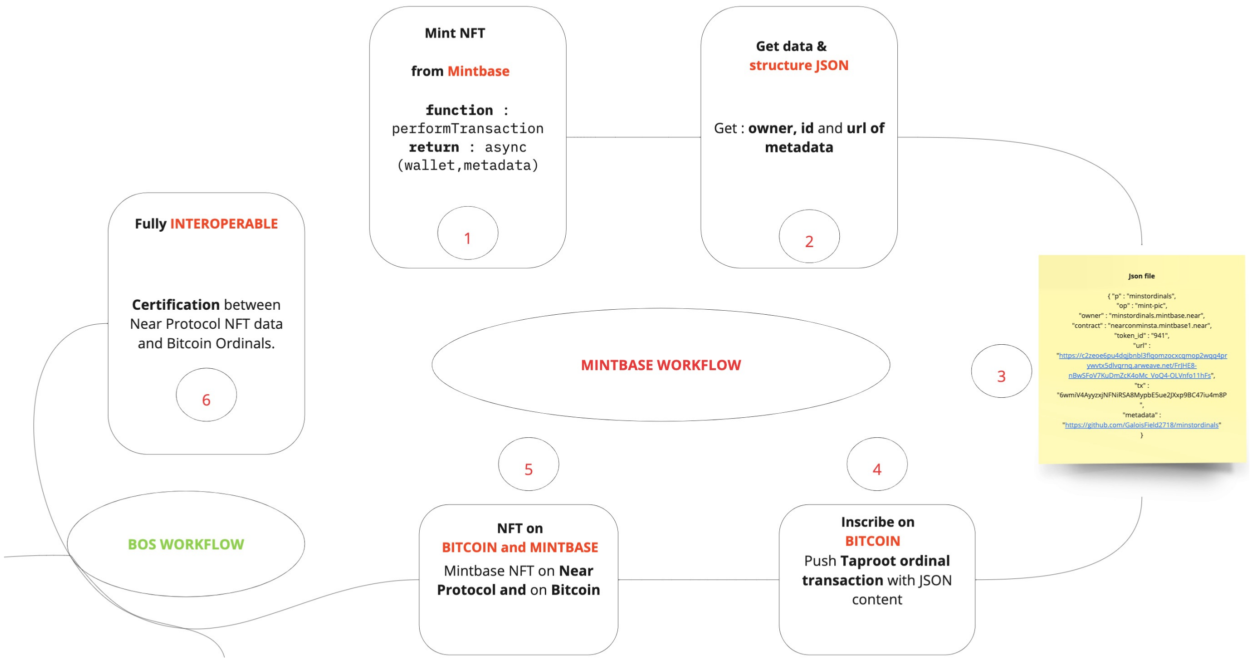
3 Workflow

Workflow is mainly divided into 2 parts : a minting process and a verifying process. Last workflow should only be the user front-end workflow not designed for now.

Those workflows are now illustrating the purpose without implementing them. The implementation process will be discussed in Protocol Details 4.

3.1 Mintbase workflow

The Mintbase workflow covers mainly the minting process in backend from user taking a picture on Minstordinals to Bitcoin inscription.


Mint NFT
from Mintbase
function : performTransaction
return : async (wallet,metadata)
1
Get data & structure JSON
Get : owner, id and url of metadata
2
Json file
3
MINTBASE WORKFLOW
Fully INTEROPERABLE
Certification between Near Protocol NFT data and Bitcoin Ordinals.
6
5
4
Inscribe on BITCOIN
Push Taproot ordinal transaction with JSON content
NFT on BITCOIN and MINTBASE
Mintbase NFT on Near Protocol and on Bitcoin
BOS WORKFLOW

```
{ "p" : "minstordinals",
"op" : "mint-pic",
"owner" : "minstordinals.mintbase.near",
"contract" : "nearconminsta.mintbase1.near",
"token_id" : "941",
"url" :
"https://c2zeoe6pu4dqjbnbl3flqomzocxcqmop2wqq4prywvtx5dlvqrnq.arweave.net/FrJHE8-nBwSFoV7KuDmZcK4oMc_VoQ4-OLVnfo11hFs",
"tx" :
"6wmiV4AyyzxjNFNiRSA8MypbE5ue2JXxp9BC47iu4m8P
",
"metadata" :
"https://github.com/GaloisField2718/minstordinals"
}
```

Figure 1: Mintbase workflow

3.2 BOS Verifier workflow

This workflow aims to describe the effective way to verify veracity of data pushed on Bitcoin and produce a proof available on Bitcoin. For a Bitcoin indexer it can consider only verified inscription.

Provide this two steps allow a more open free mint character rather than compact this both transaction into only one.

Making use of BOS tools allows the verifier to access Near on-chain data, use Near notifications tools and can allow smooth integration into others Near app like near.social.

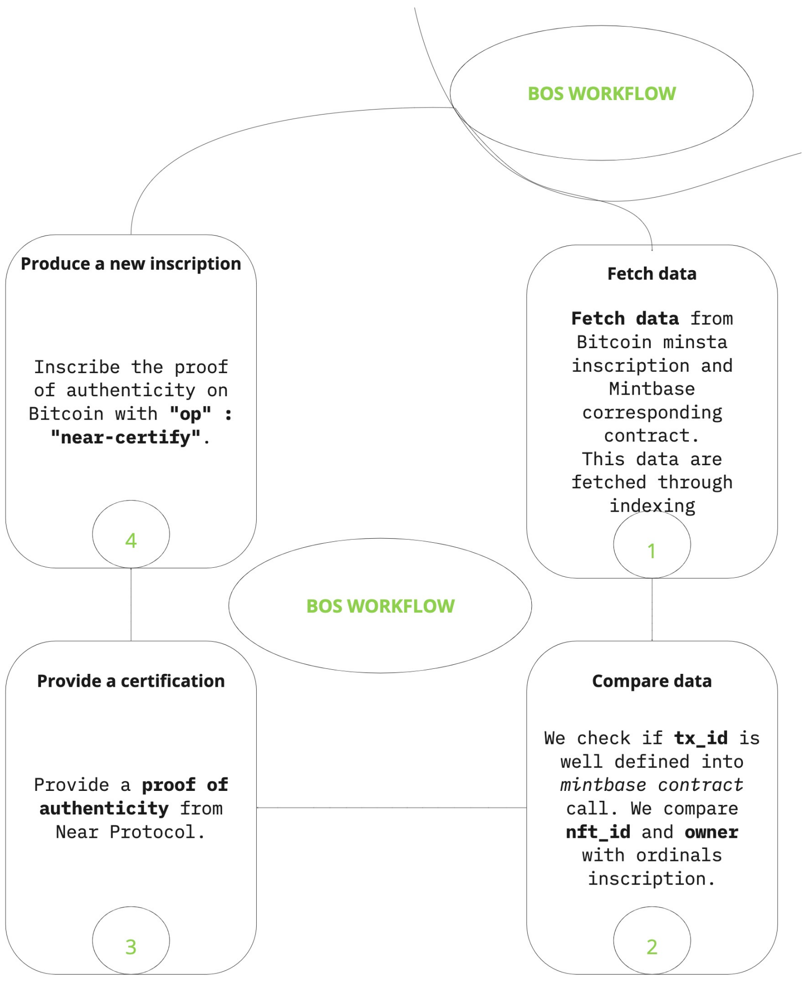

BOS WORKFLOW
Produce a new inscription
Inscribe the proof of authenticity on Bitcoin with "op" : "near-certify".
4
Fetch data
Fetch data from Bitcoin minsta inscription and Mintbase corresponding contract. This data are fetched through indexing
1
BOS WORKFLOW
Provide a certification
Provide a proof of authenticity from Near Protocol.
3
Compare data
We check if tx_id is well defined into mintbase contract call. We compare nft_id and owner with ordinals inscription.
2

Figure 2: BOS Verifier workflow

3.3 Global overview

This schema is the overview of the backend process to mint and verify all of these cross chain operations.

4 Protocol details

This part aims to provide the JSON based files for our Minstordinals protocol.

We describe keywords and important data to catch in the first part. In a second part we look into operations offered by the protocol and structure. Finally we discuss data usage for indexing purpose.

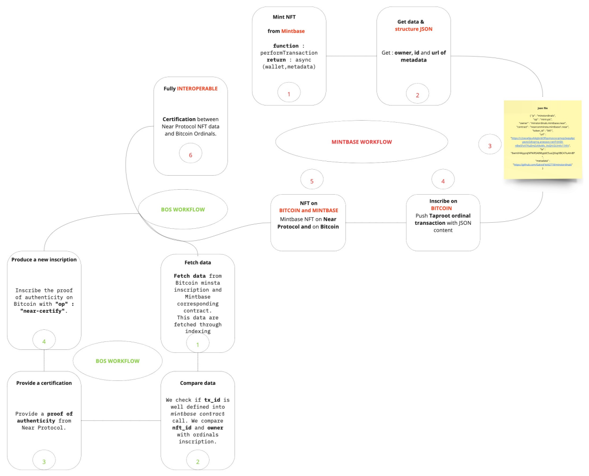

Mint NFT
from Mintbase
function : performTransaction return : async (wallet,metadata)
1
Get data & structure JSON
Get : owner, id and url of metadata
2
Json file
3
MINTBASE WORKFLOW
4
Inscribe on BITCOIN
Push Taproot ordinal transaction with JSON content
5
NFT on BITCOIN and MINTBASE
Mintbase NFT on Near Protocol and on Bitcoin
Fully INTEROPERABLE
Certification between Near Protocol NFT data and Bitcoin Ordinals.
6
BOS WORKFLOW
Fetch data
Fetch data from Bitcoin minsta inscription and Mintbase corresponding contract. This data are fetched through indexing
1
Compare data
We check if tx_id is well defined into mintbase contract call. We compare nft_id and owner with ordinals inscription.
2
Provide a certification
Provide a proof of authenticity from Near Protocol.
3
Produce a new inscription
Inscribe the proof of authenticity on Bitcoin with "op" : "near-certify".
4
BOS WORKFLOW

Figure 3: Global workflow

4.1 File Description

There are basically two operations in the Protocl for now but we think about a `transfer` operation in addition.

An example of `mint` is described :

```

{
        "p" : "minstordinals",
        "op" : "mint",
        "near_tx" : "txid",
        "aarweave_link" : "https://aarweave/hash",
        "contract_name" : "minstordinals.near",
        "function_called" : "mint",
        "data" : "0x....",
        "meta" : "Title of NFT with some additional data if
           required"
}
```

4.2 Operations

Mainly for now `mint` and `verify` operations.

4.3 Discussing data

The metadata field can be a wide range of data. Moreover, we are still thinking about more important data to hand-on with Ordinals Inscription.

4.4 Programming Languages

Basically our main languages should be : `Typescript`, `Javascript` (React), `Rust`.

4.5 Libraries and Tools Used

A brief idea about librairies that could be used :

- An humble Taproot Typescript Bitcoin library
- Rust SDK from Near

5 Further Implementation

Open indexer and Verifier. A lot of developper tools.

6 Usage and API

Make an API such that developper can query data from our App. Should use Flask package in Python or js in first step.

7 Examples

The first experiment was in Inscription 38894303 :

```

{
        "p": "minsta",
        "op" : "mint-nft",
        "owner" : "minstordinals.mintbase.near",
        "contract" : "nearconminsta.mintbase1.near",
        "id" : "941",
        "url" : "https://c2zeoe6pu4dqjbnbl3flqomzocxcqmop2wqq4prywvtx5dlvqrnq.arweave.net/FrJHE8-nBwSFoV7KuDmZcK4oMc_VoQ4-OLVnfo11hFs",
        "tx" : "6wmiV4AyyzxjNFNiRSA8MypbE5ue2JXxp9BC47iu4m8P",
        "metadata" : "https://github.com/GaloisField2718/minstordinals"
}
```

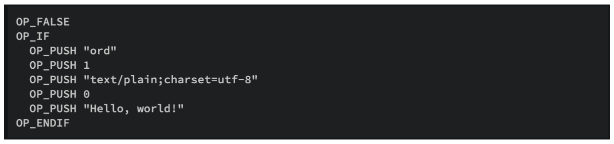
8 Technical Details

The most challenging part is obviously the Verifier and interactions between Near wallet and Near asset with Ordinals assets.

- **Verifier** : Should be built in Rust to offer all the flexibity to interact with `minstordinals` Ordinal protocol and Near NFT.
- **Indexer** : Quite basic but a good typescript indexer could be very valuable for ecosystem.
- **Frontend** : Provide an easy to use interface to add seamlessly new features and gaming options.

9 Troubleshooting

10 Appendix



```
OP_FALSE
OP_IF
  OP_PUSH "ord"
  OP_PUSH 1
  OP_PUSH "text/plain;charset=utf-8"
  OP_PUSH 0
  OP_PUSH "Hello, world!"
OP_ENDIF
```

Figure 4: Ordinal enveloppe from https://docs.ordinals.com/inscriptions.html