September 2022

EXTENSION TO NFT SMART CONTRACTS FOR FINE-GRAIN MANAGEMENT OF COPY RIGHTS FOR DIGITAL MEDIA CONTENTS

HP INC

Extension to NFT smart contracts for fine-grain management of copy rights for digital media contents

Abstract

Non-Fungible Tokens (NFTs) enable tracking ownership of digital media contents, such as images, photographs, videos, music, media streams, … in blockchain based distributed ledgers.

However, in the current common implementation of NFTs, using de-facto standards such as ERC-721, there is no way to protect the digital media data associated to the NFT. So, although the intention of the content creator may be protecting the copy rights of the digital media, this cannot be ensured with current common implementations.

The proposed solution addresses this issue by allowing the content creator to specify in the NFT contract the rights for using the digital content in different reproduction environments and technologies and protecting the data to be used for those reproductions for any other purpose.

Current NFT implementations

NFTs can be used to manage, track, and monetize digital media assets such as: digital images, digital videos, digital music, … This is achieved by creating a smart contract in the underlying blockchain that associates the digital asset file(s) to the NFT. Since the digital assets, are large, and it would be very expensive to store them in the blockchain (decentralized ledger), they are usually stored in an external storage services such as IPFS, and 'linked' in the NFT contract, using an URI.

As an example, the ERC721 specification (the most used standard to implement NFT smart contracts in Ethereum and other blockchains) associates an Universal resource Identifier (URI) to each NFT in the contract, and there is a method:

```
function tokenURI (uint256 _tokenId) external view returns (string)
```

ERC-1155 specification, also used to implement NFT smart contracts has an equivalent method.

Due to the nature of blockchain technology, this URI can be read by anybody, and since in most of the cases the data pointed by the URI is not protected, anybody can access the digital media and reproduce it. This is what is called the 'right-click-save' problem (https://mashable.com/article/non-fungible-tokens-nfts-right-click-save).

Thus, with current standard implementation of NFTs it is not possible to associate copy rights to the NFT so the content creator can specify in which circumstances, and under which conditions the digital content can be reproduced.

Proposed solution

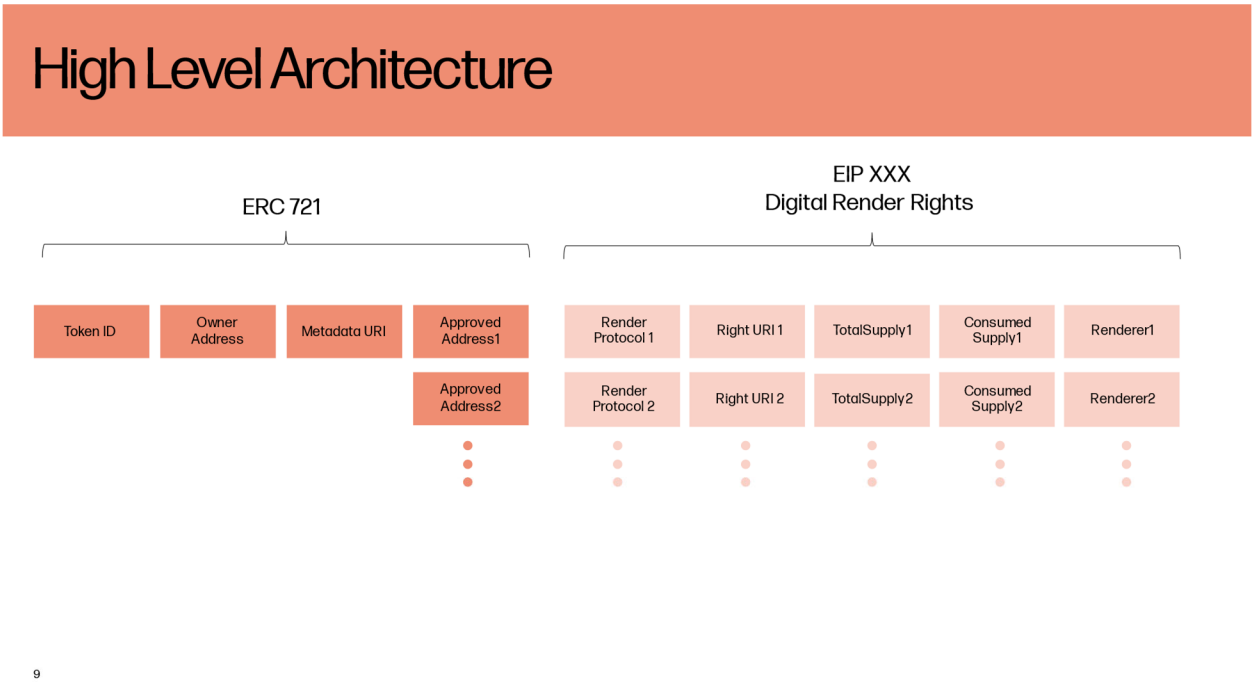
The proposed solution is based on

- adding new functionality to the NFT Smart Contract to manage the rights to render (reproduce) the digital media using different 'protocols' or rendering mechanisms
- creating an ecosystem of 'Rendering Smart Contracts' (or 'consuming smart contracts'_ that manage the reproduction of the digital media using different protocols or mechanisms

Extended NFT Smart Contract

The Extended NFT Smart Contract implements the traditional NFT interfaces (ERC721 or ERC1155 in the case of Ethereum, or in general EVM based blockchains)

The additional data needed in a NFT smart contract is depicted in the following picture



9

This image shows the data associated to the standard implementation of an NFT Smart Contract (using ERC721 specification) and the additional data associated to the copy right management. For each copy right, the following information is stored

- Render Protocol: identifier of the rendering mechanism to be used for that right. Examples may be 'Print', '3DPrint', 'DisplayOnHighResScreen', 'PlayOnVRDevice', ... This enables having managing rights for different types of media and different mechanism of copying, playing, or in general rendering it.

- TotalSupply and ConsumedSupply: these fields contain the total number of times that the content can be reproduced using the protocol and the number of reproductions that have already been consumed. For example, in the case of 'Print' or '3DPrint' render protocols, this would be the total number of copies allowed and the total number of copies already printed
- RightURI: this fields points to 'off-chain' data associated to the corresponding digital right. For example, in the case of 'Print' this URI would point to the high-resolution image for printing, in the case of '3DPrint' it would point to the 3MF design content, and in the case of 'DisplayOnHighResScreen', it would point to the high-resolution media. The content pointed by the URI should be protected through encryption. The mechanism to get the content decrypted will depend on the implementation of the corresponding rendering protocol. Technologies such as proxy reencryption may be used
- Renderer: when a rendering (reproduction, copy) of the digital right is in process, this field contains the address of the smart contract managing the rendering

The NFT smart contract MUST have two types of interfaces related to the digital copy rights:

- The *creation and management interface* is used by the NFT creator to manage the digital rights embedded in the NFT. There are two options to manage the copy rights in the NFT:
    - Immutable: the rights associated to the NFT are defined when the NFT is created (minted) and cannot be changed. In this case, the rights associated to the NFT may be 'hardwired' in the smart contract itself, or there may be a 'management API' that configures them before minting
    - Dynamic: the NFT creator may have an interface to add new rights depending on business needs. This interface should NOT allow the removal of existing rights. The rights management API should only be accessible to the owner of the smart contract
- The *consumption interface* is used by the NFT owner to consume the rights and trigger rendering. This interface contains the following functions:
    - Read only functions:

      function getRenderProtocol (uint256 tokenId, uint256 rightId) external view returns (string)

      returns the Render protocol associated to the corresponding right

      function getTotalSupply (uint256 tokenId, uint256 rightId) external view returns (uint256)

      returns the total supply (number of allowed Renders) for the corresponding right

      function getConsumedSupply (uint256 tokenId, uint256 rightId) external view returns (uint256)

      returns the number of consumed supplies for the corresponding right

      function getRenderer (uint256 tokenId, uint256 rightId) external view returns (address)

returns the address of the renderer currently reproducing the content for the corresponding right, 0x0 if there is no Render in progress

- State modifying functions
  function startRender (uint256 tokenId, uint256 rightId, address renderer) external payable
  starts a Render of the content for the corresponding right. This method fails if:
  - the caller is not the owner or approved for the token
  - the content is already being rendered
  - consumedSupply == totalSupply

  The parameter renderer is the address of the Rendering Smart Contract that will render (reproduce, print, ...) the copy (see below for the description of the Rendering Smart Contract behaviour).

  When this method is successfully executed, it calls the rendering contract to start the rendering process.

  function renderDone (uint256 tokenId, uint256 rightId, boolean success) external
  finishes the render in place for the corresponding right. If success == true, consumedSupply is incremented
  This method fails if the caller is not the reproducer

Rendering Smart Contracts

Each rendering protocol must have, at least, a Rendering Smart Contract that manages the reproduction of the digital media when a copy rendering is requested.

These contracts manage the access to the encrypted digital media content to be reproduced and interact with the off-chain components (through oracles, bridges, and/or events) that perform the rendering.

Rendering Smart Contracts must implement a method that must be called by the Extended NFT Smart Contract to kick start the rendering of a copy. When the copy is finished, the Rendering Smart Contract uses the renderDone method in the Extended NFT Smart Contract to communicate the success or error in the copy rendering.

Extension: Rendering Smart Contracts Registry

A registry of the Rendering Smart Contracts available in the ecosystem and which rendering protocols do they support can be built in an additional Smart Contract. If this registry exists, the NFT Extended Smart Contract can consult it to verify that the requested renderer in the startRender call is allowed to process the rendering protocol associated to the requested right.

Disclosed by Josep Abad Peiro, Nigel John Williams and Alberto Such Vicente, HP Inc.