IEEE ICBC'24 1570978634

Protecting Non Fungible Mutable Tokens: an Application in the Metaverse

Anonymous Authors

***Abstract*—Non-Fungible Tokens (NFTs) are currently used in a large number of scenarios, from digital art to the metaverse, to trace the ownership of assets exchanged between users. However, most NFT defining standards, such as the widely adopted ERC 721 for the Ethereum protocol, have been designed with immutable assets only in mind. As such, they are not suitable for representing assets with features that may need to be updated during their lifetime. To overcome this issue, in the literature have been proposed new models that properly represent and protect mutable assets through NFTs, such as Non Fungible Mutable Tokens, NMTs. In this paper, we expanded the NMT model with a security support meant to protect assets' features updates through access control policies that are defined by the asset creator and the current asset owner and enforced during the assets' lifetime. Policing updates is of paramount importance, because it protects the asset from unintended updates that could greatly alter the asset itself and its value. The main contributions of this paper are a detailed description of the NMT smart contracts architecture and internal dependencies, as well as an experimental validation of NMTs by providing the implementation of a NMT representing a wearable (a jacket) in Decentraland, a popular metaverse environment.**

***Index Terms*—Blockchain, NFT, Access Control, XACML, Metaverse, Decentraland**

I. Introduction

Non-Fungible Tokens (NFTs) have appreciated a steady increase in popularity during the last years. NFTs are mainly used to uniquely represent digital assets and trace their ownership, following well defined standards, such as the ERC 721 one for the Ethereum protocol. They are currently used in a large number of distinct scenarios involving different kind of assets, such as digital art, gaming, and metaverses. As an example, the Bored Ape Yacht Club (BAYC)[1] is a collection of 10,000 (at the time of writing) images algorithmically generated, managed through an ERC 721 NFT smart contract on the Ethereum blockchain and hosted on IPFS [1]. Traditional NFTs are meant to represent immutable assets, i.e., assets that do not change any of their features over their lifetime (such as the BAYC images or, in general, collectable cards). However, this limitation makes NFTs not usable by the many scenarios involving assets that are mutable over time [2]. For instance, we can consider a wearable object in the metaverse, such as a jacket: its characteristics could change over time, as buttons could be replaced, sleeves could be shortened, or some decorative patches could be added. In gaming applications, mutable assets are used to enhance the player experience by dynamically representing in-game characteristics, such as avatars evolving in appearance [3], stats increasing as players accumulate experience [4], or items in play-to-earn RPG card games [5]. Another example is real estate tokenisation, where mutable assets can be used for representing properties (e.g., lands or buildings) while capturing their changing factors, such as building expansions that are added to the properties, maintenance works that have been carried on, and so on. For this reason, the definition of a new more flexible standard is required for representing and managing mutable assets, while retaining all useful NFTs properties.

In [6], the authors propose the novel concept of Non-fungible Mutable Token (*NMT*) to support full NFT features mutability. However, updating the features of an asset may deeply impact its nature, potentially changing it beyond the original creator intent, and possibly considerably altering its value. This is why it is paramount to provide clear rules on the conditions behind NMT updates. Rather than requiring each NMT creator to write and embed their own custom set of authorisation rules in each NMT, that would result in an heterogeneous and prone to error NMTs ecosystem, we propose to employ already well established Access Control systems. To this aim, in this paper we leverage the proposal of [7] to employ smart policies, i.e., smart contract based representations of traditional Access Control policies expressed in the XACML standard. Such Access Control systems already operating on chain well integrate with the decentralised nature of NMTs.

Following these considerations, the main contribution of this paper concerns NMT protection, and consists in the extension of NMTs with a security support which regulates the right of changing their features, thus avoiding unauthorised and dangerous alterations of the represented asset. In particular, mutable assets' features are stored on-chain, and the methods which implement such changes are defined by the NMT creator when they define the NMT contract itself. This methods execution is allowed only if the access control policies linked to the NMT, one defined by the NMT creator and another defined by the current holder of the NMT, are satisfied. This new kind of NFTs on the one hand provides their users the possibility of updating the features describing the asset itself, thus supporting the aforementioned scenarios impossible with traditional NFTs. On the other hand, NMTs allow their creators to retain some degree of control on the changes that will be made by users to the NMTs they created by means of proper access control policies. The current holder of the asset can define their own access control policies as well, in order to allow third parties to operate updates on the asset within their

Acknowledgements removed for double blind review

[1] https://boredapeyachtclub.com/

desired specifications. Another relevant contribution of this paper is a detailed description of the architecture of the smart contracts implementing NMTs, focusing on the mechanism for guaranteeing policy enforcement. Moreover, to validate the proposed approach, a reference implementation in a metaverse scenario is described, and a set of experiments have been carried out to measure the costs of using NMTs. In this work we will use Ethereum as reference underlying blockchain protocol, as most of the existing NFTs are based on it.

This paper is organised as follows. Section II provides some background notions concerning NFTs, Access Control Systems, and the Decentraland metaverse. Section III presents the proposed approach to represent and manage Non-Fungible Mutable Tokens, Section IV describes our implementation of such approach on the Decentraland Metaverse, while Section V evaluates the costs of deploying and using such NMTs. Finally, Section VI describes some related work, and Section VII concludes the paper.

II. Background

A. Non Fungible Tokens (NFTs)

Non Fungible Tokens are defined in Ethereum by the ERC 721 standard [8], and they are meant to track the ownership of digitally represented assets, thus including also real-world assets having a digital representation. For the sake of simplicity, in the rest of the paper we will focus on digital assets only.

Each NTF is defined as a couple of values: the identifier of the owner, and the reference to the digital asset the NFT links to, named *token ID*. Moreover, the NFT is paired with an asset descriptor, which embeds information related to the asset representation (which we will refer to as asset attributes in the rest of this paper). Generally, the descriptor, or item's metadata, plays a critical role in the world of NFTs and is referred to as an off-chain resource, usually pointed by an *URI*. Typically, the owner's address on the blockchain, which is obviously unique, is used as owner's identifier, while the asset reference is obtained by computing a cryptographic hash of the asset digital representation (often stored by IPFS), thus being unique as well, as long as assets are all unique. This solution ensures that any tampering of the digital asset representation is immediately detected since the reference to the digital asset representation embedded in the NFT would not be valid anymore.

B. Access Control System (ACS)

An Access Control System (ACS) is used to regulate and manage access to digital resources. Access rights are expressed through Access Control policies, which typically consist of a set of conditions that are evaluated each time an access request is received, considering the current access context. Several models have been introduced in the literature for defining access rights; among these, the Attribute-based Access Control (ABAC) model carries out the decision process taking into account as decision factors the attributes associated with the user requesting the access, the resource being accessed, and the environment in which the access request is made. Examples of attributes related to a subject (S) might encompass the following: S's ID, the ID of the company that employs S, S's role within this company, and so forth. Attributes paired with a resource R could include: the ID of the creator, the ID of the owner, the creation/deployment date, and so on. Several languages are currently available for expressing ABAC policies, with the eXtensible Access Control Markup Language (XACML) [9], defined by the OASIS consortium, being one of the most widely adopted one.

C. Decentraland

Decentraland[2] is a metaverse environment that allows its users to participate in a unique virtual community for a distinctive virtual experience. Decentraland is a project built on the Ethereum (ETH) blockchain, with the goal of establishing an open virtual world where users can operate much like they do in the physical world by using one or more virtual user characters called "avatars". Virtual worlds and metaverse platforms such as Decentraland (or The Sandbox[3]) use NFTs to represent a wide variety of assets such as parcels of virtual land or resource assets displayed within its scenes. Users can buy, sell, and develop these virtual properties, creating virtual economies within these digital spaces [10].

Wearables[4] in Decentraland is a category of virtual items that can be worn by avatars within the virtual world. Wearables refer to a range of garments, body features, and accessories and many other customised elements that allow to personalise the appearance of a Decentraland avatar. Each wearable has a specific category, characteristic, and rarity attributes that determines its identity and which body part in the avatar system (e.g., head, upper body, etc.) the wearable will be applied to. A variety of default wearables are accessible to all avatars (free of charge) in Decentraland, but the platform also embraces the development and use of custom wearables represented by NFTs. This unique feature permits the generation or minting of a limited quantity of distinct wearables on the blockchain, similar to land parcels. All wearables are NFTs and have their own marketplace[5] which means they can be bought and sold on the open market. There are mainly two categories of wearables in Decentraland: *Regular Wearables* and *Linked Wearables*[6]. Regular Wearables can be bought, sold, and traded on the Decentraland marketplace and are not unique. Linked wearables, instead, refers to three-dimensional representation of NFTs generated outside the Decentraland ecosystem and introduced by third party entities.

It's important to note that Linked Wearables differ significantly from standard wearables as Linked Wearables are not part of the conventional wearable collection, they exclusively serve as "in-world representations mapped to external NFTs by a Third Party". Before any Linked Wearables can be introduced into the Decentraland ecosystem and subsequently

[2]https://decentraland.org/
[3]https://www.sandbox.game/
[4]https://docs.decentraland.org/creator/wearables/wearables-overview/
[5]for instance: https://opensea.io/collection/decentraland-wearables
[6]https://docs.decentraland.org/creator/wearables/linked-wearables/

minted, they must pass a preliminary review by a Curation Committee for approval. This step is necessary to be admitted by the DAO[7] (Decentralised Autonomous Organisation, a key component of Decentraland governance and decision-making process, in charge of managing the most relevant aspects of the virtual world) as an enabled Third Party (for the original creator of the external NFT) by submitting a proposal using the template in the new category Linked Wearables Registry.

III. Protecting Non-Fungible Mutable Tokens

Despite their flexibility, NFTs were originally designed with monolithic descriptors in mind, i.e., representations opaque about their internal attributes. For this reason, NFTs may be appropriate for static digital assets (e.g., for representing images), but not for digital assets whose attributes are mutable over time and that need to be accessed on chain. This can be shown through an example in the metaverse, focusing on Decentraland. Let us consider a wearable object (see Section II-C), such as a jacket that might change over time because some of its internal attributes may be updated, e.g., the colour is changed, buttons are replaced, sleeves are removed, or some decorative patch is added. These changes should be reflected in the corresponding digital representation of the asset but, at the same time, they should leave the jacket asset identifier (i.e., the token ID) unchanged. Currently, assets of this type cannot be represented with standard NFTs exploiting hash-based external URIs, because each bit changed within the asset descriptor (which contains what in ERC721 is called the item's metadata) will compromise its hash, i.e., its token ID. Therefore, the new token ID will no longer match the one recorded in the NFT smart contract. We argue that the existing Dynamic NFTs proposal, see Section VI, is inadequate as well, as it decouples the dynamic metadata from the asset descriptor and performs updates at class level, i.e., the NFT smart contract, making it impossible to adapt changes based on individual assets.

To properly represent this novel type of mutable digital assets, we employ fully fledged Non Fungible Mutable Tokens (NMTs) (originally introduced in a preliminary form in [6]), enriched with an access control system based on the ABAC model. The integration of an access control system within the NMT design is of paramount importance because it allows to protect mutable assets from malicious and undesired changes made possible by assets mutability.

A. Protecting NMTs through access control policies

To support the most general possible digital assets, it is desirable to define and manage the mutability operations inside the asset itself, alongside its attributes. Providing operations for updating asset attributes also requires the introduction of a support for authorising their execution, i.e., an Access Control System (see Section II-B), based on policies established by the asset creator and by the current holder. As a matter of fact, unauthorised or improper changes could dramatically decrease the value of the asset itself or change asset's features in a way such that the current holder or the creator would not desire. Therefore, a key aspect of our NMT proposal is the enforcement of access control policies to provide a dynamic regulation of the rights to update the internal attributes of assets. The creator would define the general constraints on attributes updates to not alter the asset nature, while the current holder may express additional rules depending on their preferences. In practice this is achieved by having two Access Control policies managing the same asset at the same time, one defined by the asset creator, and the other by the current holder. Consequently, a request to change an internal attribute of an asset will be actually executed only when it is authorised by both policies.

Referring to the previous example, we can imagine that the jacket mutable asset in the metaverse is produced by a famous clothes designer, who would like to decide which functional and artistic changes such jacket can undergo and who is authorised to perform them. For instance, the designer might decide to allow to change the colour of the jacket, but only within a given set of hues. Moreover, the designer could express in their policy the addresses of the subjects that are allowed to operate such changes (e.g., a set of trusted tailors), because these subjects are in charge of producing the new 3D representation of the mutable asset (reflecting the new attributes) which is then shown in the metaverse. Thus, to regulate changes to the jacket, the designer writes an Access Control policy that is associated with the Jacket Mutable asset. On the other hand, the policy written by the current holder h of the asset might specify the address of a subject (who should be chosen among the ones in the set of the creator trusted tailors) that is trusted by h to perform the change colour action, and could state that the colour of the jacket can be only black or green. To enforce such restrictions, the holder writes an Access Control policy that is associated with the Jacket Mutable asset as well.

B. NMT Architecture

In our proposal, digital assets are represented and managed by specific smart contracts named *Mutable Asset smart contracts*. This allows for mutable attributes to be updated on chain through a set of proper update functions. Since such update functions are executed through smart contract method calls, the evaluation of the previously mentioned policies protecting mutable assets needs to be executed on chain as well. In this paper, instead of relying on users to write the custom code implementing the desired policies in a smart contract, we outsource the whole Access Control process to dedicated smart contracts by following the *smart policy* approach described in [7]. This approach allows users to write policies exploiting the traditional XACML standard, that are then automatically translated as deployable smart contracts, reducing human error, encouraging reuse, and increasing flexibility.

Figure 1 shows the architecture implementing our proposal and customised for the metaverse Jacket reference example. Our architecture consists of three classes of smart contracts, rather than a single one as for standard NFTs:

[7]https://dao.decentraland.org/

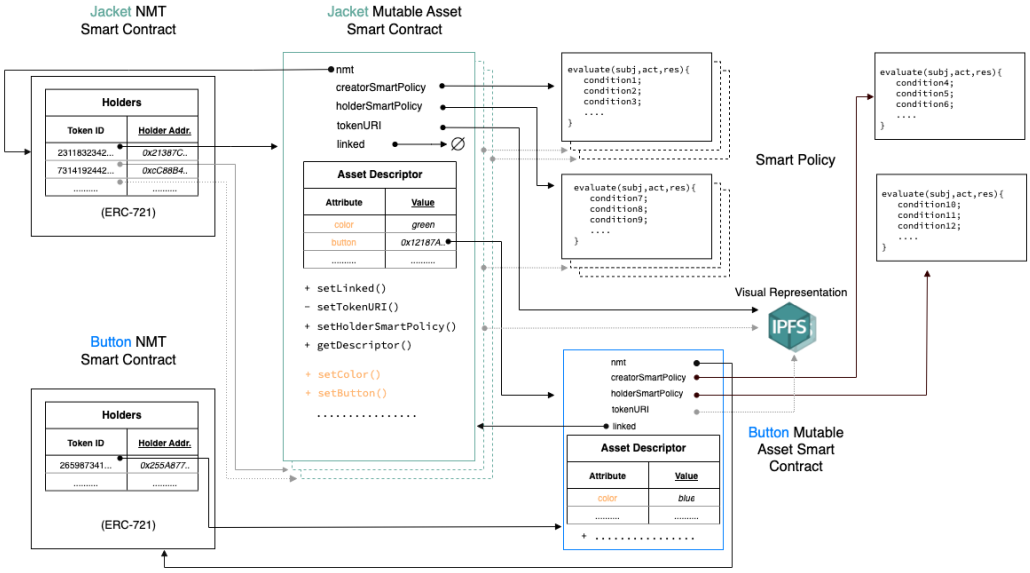


Fig. 1: NMT architecture

- NMT smart contracts to track ownership of unique assets of homogeneous type;
- Mutable Asset smart contracts to represent assets as first class on chain entities;
- smart policy smart contracts to implement mutable assets Creators and Holders Access Control policies.

The advantage of having dedicated smart contracts to represent different concepts, such as asset descriptor and Access Control policies, allows to dynamically make changes to the asset and to the policies without invalidating their relative contracts linking.

The first smart contract, called the NMT smart contract, handles the creation of assets and the tracking of their holders. It adheres to the standard for NFTs by implementing the ERC-721 standard. The uniqueness of each Mutable Asset is guaranteed by defining its token ID as the unique address of the corresponding Mutable Asset smart contract. Such token ID is unique since the address of each smart contract is unique. The holder tracking is guaranteed in the NMT smart contract by a table storing the association of the holder's account address with the token ID. This smart contract is extended every time we want to create a new kind of mutable assets. In Figure 1 there are two distinct smart contracts extending the NMT smart contract, one for jackets and the other for buttons, namely: Jacket NMT Smart Contract and Button NMT Smart Contract.

The second smart contract, called Mutable Asset Smart Contract, is meant to represent mutable assets. This smart contract is extended every time we want to create a new kind of mutable assets. In Figure 1 we defined two extensions of the Mutable Asset Smart Contract for our running example, one for jackets and the other for buttons, namely: Jacket Mutable Asset Smart Contract and Button Mutable Asset Smart Contract.

The Mutable Asset Smart Contract contains:

- a variable, *nmt*, holding the address of the specific NMT Smart Contract to which this specific Mutable Asset Smart Contract belongs. This variable is immutable because each Mutable Asset can only belong to a specific NMT Smart Contract. Through this variable (and the *ownerOf* method of the NFT standard) we get the address of the current holder of the Mutable Asset represented by the smart contract. In our reference example, the *nmt* variable of the Jacket Mutable Asset Smart Contract contains the address of the Jacket NMT Smart Contract;
- a variable, *creatorSmartPolicy*, holding the address of the creator's Smart Policy smart contract, which is meant to regulate the execution of the update methods of this smart contract according to the creator's rules;
- a variable, *holderSmartPolicy*, holding the address of the Smart Policy defined by the current holder of the asset, that is meant to regulate the execution of the update methods of this smart contract according to the rules set by the current holder. The value of the *holderSmartPolicy* variable can be changed only through the *setHolderSmartPolicy* method, that can be executed only by the current holder of the Mutable Asset smart contract. In Figure 1 we can see several instances of the Smart Policy smart contract, used as Creator Smart Policy or as Holder Smart Policy;
- a variable, *tokenURI*, used to link the off-chain metadata, such as the 3D visual representation of the asset. In our reference example, tokenURI is the reference to the Jacket Mutable Asset 3D model stored in IPFS. However, other off-chain storage solutions could be chosen, provided that a proper mechanism to protect the off-chain data is put in place if required by the specific scenario.
- a variable, *linked*, that is used when this mutable asset

is linked to another mutable asset, which is considered as the main asset. In our example, the Jacket Mutable Asset is not linked to any other mutable asset, while the Button Mutable Asset can be linked to a Jacket Mutable Asset. In such a case, the variable *linked* of the Button Mutable Asset smart contract would contain the address of the smart contract representing such Jacket Mutable Asset.

- the asset descriptor containing the mutable item's metadata, which stores the names and the values of all on-chain attributes defining the asset. In our running example, the considered attributes related to the Jacket Mutable Asset are the colour and the buttons;
- the public methods designed by the creator of the asset to allow the update of the asset by changing the value of fields in the descriptor. These methods are meant to be the only way to change the state of the asset. Thus, the asset creator can protect the asset itself by non-authorised updates. In this case, before actually executing such methods, both the Creator and the Holder Smart Policies are evaluated, and only if they are both satisfied the attribute update is executed. In our reference example, we defined the methods *setColor* and *setButton* for changing the value of the attributes *color* and *button* of the Jacket Mutable Asset, respectively.

The third type of smart contracts part of our architecture are smart policies. These smart contracts are needed because one relevant feature of our proposal is that both the asset creator and the asset holder can regulate the changes that will be performed to Mutable Assets by defining their own smart policies, and associating them to the Mutable Asset smart contract through the *creatorSmartPolicy* and *holderSmartPolicy* variables. As previously explained, the values of asset attributes (the ones listed in the Asset Descriptor of the Mutable Asset Smart Contract) can only be updated using the methods established by the creator and embedded in the Mutable Asset Smart Contract. The invocation of such methods requires the evaluation of the above mentioned Smart Policies to decide whether the update can be executed or not.

Algorithm 1 Pseudocode snippet structure for smart policy enforcement

```
Require: subj: subject, act: action, res; resource
 1: subj ← msg.sender
 2: act ← abi.encodeWithSignature("act_signature",
 3:                                [act_params])
 4: o ← ownerSmartPolicy
 5: c ← creatorSmartPolicy
 6: if (o.eval(subj, act, res) AND c.eval(subj, act, res)) then
 7:     execute(act);                ▷ Permitted action e.g., setColor
 8: else
 9:     throw error;                 ▷ Denied action "access denied"
10: end if
```

Algorithm 1 shows the pseudo-code that outlines a basic structure for the smart policy enforcement, by checking conditions and applying policy actions accordingly. Each Smart Policy smart contract will implement an "evaluate" method. For example in line 6 there are two different evaluate methods (*o.eval* and *c.eval*) that belongs to different policies. An evaluate method performs a switch-case construct for selecting the action to be checked, passed as a parameter (line 2). For each action, a set of conditions to be satisfied is defined inside the Smart Policy. For each MutableAsset method that determines mutability there are associated two (Solidity) modifiers connected to the two Smart Policies (creator, owner). The modifiers role, arranged in series (as shown in line 6), is to safeguard the change of the asset's characteristic. They act by passing the three parameters required by the *evaluate* method of the linked smart policy. If the required conditions and prerequisites meet the criteria, then the modifier allows advancing to the next step (eventually the method body) otherwise it will throw an error (and revert the current transaction).

For Smart Policy definition we adopted the approach defined in [7], which has the following two main advantages. First of all, the access control policy is written exploiting XACML, a very popular language for writing attribute based access control policies. This relieves the Mutable Asset creator from the burden of using an ad-hoc language for defining the policy to protect the Mutable Assets they produce, and allows them to also use already existing user-friendly tools and editors for easily producing such a policy. The second advantage is that the framework in [7] transforms the XACML access control policy directly in a smart contract, the smart policy, which can be deployed on the blockchain and referred in the Mutable Asset Smart Contract. Simply put, the smart policy consists of a list of rules associated with the asset's methods which express the conditions that must be valid to allow the execution of such methods. Listing 1 shows the example of holder smart policy described in Section III-A expressed in XACML language.

IV. NMTs in Decentraland

To validate our proposal and to conduct some experiments involving NMTs representing fully digital assets, we have set up a testbed in the metaverse by leveraging the capabilities of Decentraland and Ethereum technologies. To perform our experiments we adhered to Decentraland's internal development procedures for integrating our Mutable Assets in the metaverse. In particular, we leveraged the Sepolia Ethereum test network to operate within a local testing environment to create a potential wearable item before its formal submission to the DAO. We didn't actually submit our wearable to the Decentraland DAO only to avoid Decentraland approval time and costs. However, for what concerns the aspects of the NMT approach we want to validate, the resulting testbed is very similar to the one we would have obtained by using a DAO approved wearable.

For conducting our tests we considered the following use case scenario. A renowned fashion designer creates a rare jacket design and offers it for sale through their brand's flagship store, PUB (Prestigious University Brand). Since unauthorised changes may decrease the value of the jacket or

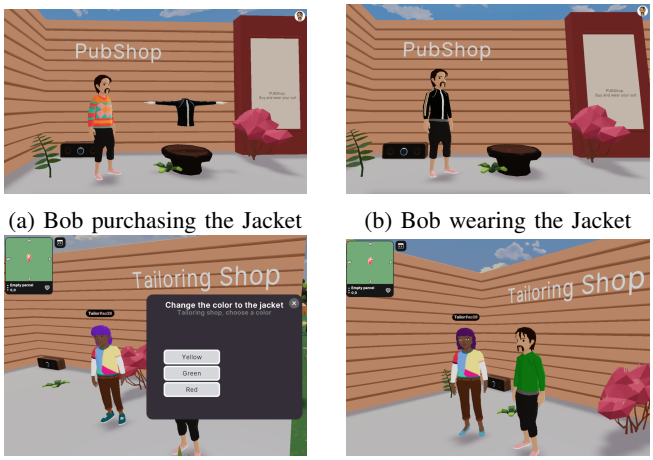
Listing 1 Example of XACML policy

```
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" PolicyId="a123"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:
first-applicable" Version="1.0">
  <Description>creator policy</Description>
  <Target></Target>
  <Rule Effect="Permit" RuleId="allowed colors and tailors">
    <Target>
      <AnyOf>
        <AllOf>
          <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            changecolor</AttributeValue>
            <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:
            action:action-id" Category="urn:oasis:names:tc:xacml:3.0:attribute-
            category:action" DataType="http://www.w3.org/2001/XMLSchema#string"/>
          </Match>
        </AllOf>
      </AnyOf>
    </Target>
    <Condition>
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:
        function:string-at-least-one-member-of">
          <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:
          action-parameter" Category="urn:oasis:names:tc:xacml:3.0:
          attribute-category:action"
          DataType="http://www.w3.org/2001/XMLSchema#string"
          MustBePresent="true"></AttributeDesignator>
          <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-bag">
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            black</AttributeValue>
            <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            green</AttributeValue>
          </Apply>
        </Apply>
        <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:
          function:string-one-and-only">
            <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:
            subject:subject-id" Category="urn:oasis:names:tc:xacml:1.0:subject-
            category:access-subject"
            DataType="http://www.w3.org/2001/XMLSchema#string"
            MustBePresent="true"></AttributeDesignator>
          </Apply>
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
          0x7DE5260b6964bAE3678f3C7a8c82654af2CeAc28</AttributeValue>
        </Apply>
      </Apply>
    </Condition>
  </Rule>
  <Rule Effect="Deny" RuleId="deny-rule"></Rule>
</Policy>
```

alter the original designer vision, in our reference example, the jacket designer wants to restrict who can make these updates, i.e., only tailors that are trusted by the PUB brand, and which modifications can be done, i.e., the jacket colour can be updated only to a predefined set of hues. Moreover, in our use case, the holder of the jacket wants to further restrict the operations that can be performed on their jacket depending on their own taste and style. In particular, the holder wants to have the colour of their jacket changed only to green or black. Moreover, the holder wants that the update of the 3D representation of their jacket can be performed only by a specific tailor they trust, e.g., the tailor having address *0x7DE5260b6964bAE3678f3C7a8c82654af2CeAc28*, who is the only one that meets the holder's own taste.

To implement this example it was necessary to define two distinct contexts and to connect them. The first context focuses on the smart contracts implementing Mutable assets, while the second context is related the definition of Decentraland scenes. The connection between the two contexts is required because some scenes involve interactions with the Mutable Assets and update their attribute values.

The scenes we defined consist of the PUB brand jacket store and a tailoring shop affiliated with the same brand (Figure 2). The user, Bob, immersed in Decentraland through his avatar, visits the jacket store and purchases one of the pre-created jackets (Figure 2a). To produce these jackets, PUB leverages



(a) Bob purchasing the Jacket (b) Bob wearing the Jacket

(c) Bob choosing a new color (d) The Jacket color is changed

Fig. 2: Decentraland scenes containing the PUB and Tailoring shops

the Jacket NMT smart contract for minting new jackets, releasing them into the metaverse market. Once purchased, the user's avatar sees the jacket appear in their inventory (their "backpack"), and it can wear and use it at will (Figure 2b). At some point, the user decides to change the jacket's colour and, to this aim, he visits the tailor shop. Consulting the dyeing tailor, the user explores available colour options (Figure 2c). Let's suppose that the user chooses the green colour. Metaverse functionalities facilitate conveying this information to the *setColor* method of the JacketMutableAsset smart contract. To interact with the Jacket Mutable Asset it is necessary to know the address of the Jacket NMT smart contract and the token ID of the Jacket Mutable Asset. Thanks to the functionalities offered by Jacket NMT it is possible to identify the Jacket Mutable Asset (referring to its token ID) in a similar way to an NFT, as well as being able to interact with the Jacket Mutable Asset methods by invoking them directly. When the *setColor* is invoked, the creator and holder policies are evaluated (the policy evaluation is executed through the *evaluate* smart policy method) and, since the change colour operation and the related parameters satisfy both policies of our example, the jacket's colour attribute is modified, and the tailor releases the modified 3D image into the asset, updating the jacket's appearance (Figure 2d).

V. Cost Evaluation

In order to validate the proposed NMT approach, we performed a number of experiments to evaluate the costs of deployment and execution. In particular, we implemented the smart contracts part of the Jacket Mutable Asset example described in the previous sections, we deployed them on the Sepolia blockchain, and we acquired enough MATIC to invoke such smart contracts' methods.

Table I shows the gas required to deploy the smart contracts implementing our approach. The cost of deploying the JacketNMTsmart contract is the highest, and it is about

Contract	Gas
JacketNMT	1825084
JacketMutableAsset	798140
CreatorSmartPolicy	442469
HolderSmartPolicy	381453
DenyAllSmartPolicy	261679

TABLE I: Deployment Gas

1.8M gas unit. However, this smart contract must be deployed only once, independently on the number of Jackets Mutable Assets that will be created. Deploying a new standalone Jacket Mutable Asset smart contract would cost about 798K gas units, and this cost depends both on the number and type of its internal attributes, and on the methods defined to implement mutability. In the following we will show how this cost changes while varying the number of attributes and structure of the related update methods. Finally, the deployment of the creator and holder smart policies described in the example costs, respectively, 442K and 381K units of gas. This cost heavily depends on the complexity of the policy. For example, the deployment of a *denyall* smart policy, i.e., a very simple policy which forbids any update, would cost about 262K gas units.

Table II shows the costs of execution of the main methods of the smart contracts implementing the mutable jacket example. The cost of executing the mint method of the JacketNMT smart contract to create a new Jacket Mutable Asset is about 798K gas units. This cost is quite high with respect to the others because the mint operation also involves the deployment of the new Jacket Mutable Asset smart contract. The cost of transferring a Jacket Mutable Asset to a new holder invoking the *transferFrom* method of the JacketNMT smart contract is about 55K gas units. The cost of executing the *setColor* method for changing the colour of the asset is about 95K units of gas. This cost is composed of two components: the cost of evaluating the creator and holder smart policies, and the cost of updating the values of the colour attribute in the Asset Descriptor and the tokenURI field, embedded in the *_setColor* method. The execution of the *_setColor* method costs about 70K gas units, while the difference between the costs of the *setColor* and the *_setColor* methods, i.e., about 25K gas units, represents the costs of evaluating the creator and holder smart policies. Changing the holder smart policy with a new one costs about 32K units of gas. Please notice that this cost does not include the deploy of the new smart policy, and it is independent from the complexity of the smart policy itself. As a matter of fact, the holder can change the holder smart policy of a Jacket Mutable Asset exploiting an already existing smart policy.

Contract	Method	Gas
JacketNMT	mint	797620
JacketNMT	transferFrom	54791
JacketMutableAsset	_setColor	70554
JacketMutableAsset	setColor	95248
JacketMutableAsset	setHolderSmartPolicy	32426
HolderSmartPolicy	add setColor	≈12K

TABLE II: Execution Gas

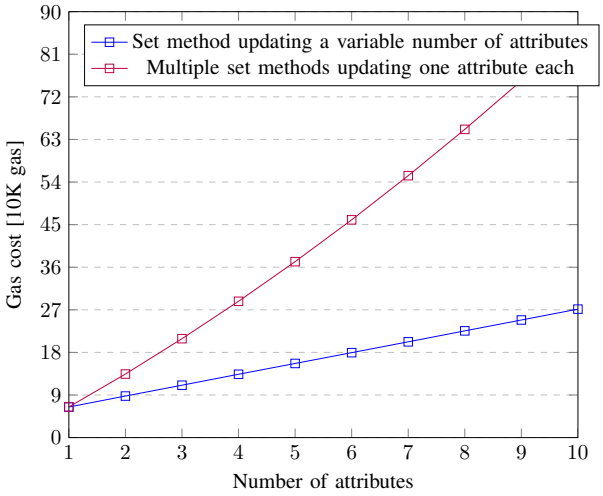


Fig. 3: Cost of executing attribute update methods varying the number of methods or number of attributes updated by a single method

Operation	NFT	NMT
mint	70811	797620
tansferFrom	54563	54791
update (without policy enforcement)	125374 (simulated with burn+mint)	70554
update (with policy enforcement)	X	95248

TABLE III: Costs of the main operations for Non-Fungible Token (NFT) and Non-Fungible Mutable Token (NMT)

In the Jacket Mutable asset smart contract, the *setColor* method changes the value of one attribute only, the colour of the Jacket. However, other kinds of mutable assets could have multiple attributes, and could define methods that update more than one attribute at the same time. Hence, we have considered the update of the simplest possible attribute (a simple integer value). The blue line in Figure 3 shows the execution cost in gas units of such attribute update method varying the number of attributes it updates in the range $\{1..10\}$. The red line, instead, shows the cost of updating i attributes by invoking i times an update method changing one attribute only, with i in the range $\{1..10\}$. From the figure we can easily notice that both plots increase with the number of attributes, but the execution time of the methods changing n attributes at the same time is considerably lower than the cost of executing n times an update method changing one attribute only. In the previous experiments we used a *permit all* policy such in a way that we focused on the set attribute method cost only.

Finally, in Table III we try to compare some of the costs of NMT management related to the Jacket Mutable Asset scenario with the cost of executing the same (or similar)

operations on a standard NFT used to represent a static Jacket. First of all, we notice that the cost of minting a new NFT is considerably lower (one order of magnitude) than the cost of minting an NMT. This is due to the fact that the NMT minting cost includes the deployment of a Jacket Mutable Asset smart contract, and this heavily affects the cost. For what concerns the transfer of a Jacket from an holder to another, the cost is quite similar for NFT and NMT approaches. Instead, if we need to update an attribute of a standard NFT, we could simulate it by destroying the original NFT (i.e., transferring it to the Burn Address) and creating a new NFT from scratch. Obviously, simulating the attribute update on a standard NFT in this way has two major drawbacks (which, indeed, are the main motivation for our proposal): this operation would change the original ID of the NFT (i.e., the resulting NFT is actually a new NFT) and it is not possible for the original NFT creator to regulate the execution of updates on the NFTs they create. Besides this noticeable drawbacks, if we compare the gas cost alone, we notice that our NMT approach outperforms the NFT one, because the cost of updating an NMT without policy enforcement is about 70K gas units, while the cost of simulating an update in case of an NFT is about 125K gas units. If we include the smart policies evaluation, the NMT update cost is about 92K gas, that is still smaller than the cost of simulating an NFT update where the policy enforcement can not be implemented.

VI. Related Work

Although a large number of works concerning NFT applications, even in metaverse, [11], [12], are currently available in the scientific literature, the concept of tokens mutability has been covered by few papers.

For instance, it has been just mentioned in [13], which defines an NFT taxonomy with respect to NFT metadata.

The work that is somehow related to our proposal is described in [14], which introduces the concept of *dynamic NFT (dNFT)*, NFT smart contract embedding some metadata used to manage dynamic characteristics. The main problem of the dNFT approach is that it burdens the NFT contract, whose goal is to regulate NFT minting and track ownership, with additional logic required to manage asset mutability, i.e., the metadata updates, increasing the contract complexity at the expense of modularity. Moreover the dNFT approach is not very flexible, as NFT tracking and updates are intertwined, making it harder to change the update rules and conditions once the contract is deployed. This makes also harder to define different rules for different assets managed by the same NFT contract. The dNFT concept has been introduced using Chainlink to collect information from real world through oracles to update tokens metadata. [15] provides a more detailed description of dNFTs, including a list of use cases. A similar proposal is presented in [16] where the authors discusses a dynamic NFT generation system enabling users to create Dynamic NFTs (DNFTs) or convert existing NFTs, expanding their potential use cases. In contrast to the conventional method of storing NFTs on IPFS, this proposed system stores them as on-chain metadata in Scalable Vector Graphics (SVGs). Another application is the system proposed by [17] where authors modify token details using an ERC-721 token standard that include encrypted and stored associated media files on IPFS through IPNS entries where key distribution, via Shamir's Secret Sharing, ensures security. NFT mutability is considered a risk from the security point of view by the authors of [18]. However, the concept of mutability in that work is related to the off chain descriptor of the NFT, it is not used to indicate the mutable nature of the asset it represents. As such is not more relevant to our proposal than to NFTs in general. The authors of [19] raise concerns about NFT mutability as well. In this case too, the concerns are related to the referred off chain descriptor.

VII. Conclusions and future work

This paper refines the Non-Fungible Mutable Tokens (NMTs) solution for representing on the metaverse digital assets that change their attributes over time. The main contribution of this paper is the introduction of Access Control policies aimed at regulating the attribute updates that can be performed during the asset lifecycle, in order to avoid changes that undermine or devalue the asset itself. This paper also presents a refined and revised architecture for NMTs where a proper solution for implementing such policies has been chosen, and a mechanism for guaranteeing their enforcement has been introduced. The proposed approach has been validated by implementing a reference example in Decentraland, and by evaluating the costs of deploying and invoking the smart contracts of our solution, varying some design parameters.

Possible future research directions include extending and refining the proposed solution for addressing portability of NMTs across distinct metaverse protocols and underlying blockchains.

References

[1] J. Benet, "IPFS - content addressed, versioned, P2P file system," *CoRR*, vol. abs/1407.3561, 2014. [Online]. Available: http://arxiv.org/abs/1407.3561

[2] Dynamic nft examples - 16 use cases. [Online]. Available: https://freshcredit.com/blockid-dnft/

[3] Lamelo ball. [Online]. Available: https://lameloball.io/#/

[4] Boxing boy. [Online]. Available: https://boxingboyz.com/

[5] Dreams quest. [Online]. Available: https://dreams.quest/

[6] D. Di Francesco Maesa, A. Lisi, P. Mori, L. Ricci, and S. Schiavone, "Non fungible mutable tokens: dynamic assets traceability for the metaverse," in *Proceedings of IEEE International Conference on Metaverse Computing, Networking and Applications (IEEE MetaCom 2023)*, 2023, pp. 393–397.

[7] D. Di Francesco Maesa, P. Mori, and L. Ricci, "A blockchain based approach for the definition of auditable access control systems," *Comput. Secur.*, vol. 84, pp. 93–119, 2019. [Online]. Available: https://doi.org/10.1016/j.cose.2019.03.016

[8] W. Entriken, D. Shirley, J. Evans, and N. Sachs. ERC-721: Non-Fungible Token Standard. [Online]. Available: https://eips.ethereum.org/EIPS/eip-721

[9] OASIS, "eXtensible Access Control Markup Language (XACML) version 3.0," OASIS XACML TC, January 2013.

[10] H. joo Jeon, H. chang Youn, S. mi Ko, and T. heon Kim, "Blockchain and ai meet in the metaverse," in *Advances in the Convergence of Blockchain and Artificial Intelligence*, T. M. Fernández-Caramés and P. Fraga-Lamas, Eds. Rijeka: IntechOpen, 2021, ch. 5. [Online]. Available: https://doi.org/10.5772/intechopen.99114

[11] L. David and L. S. Won, "NFT of NFT: Is our imagination the only limitation of the metaverse?" *The Journal of The British Blockchain Association*, 2022.

[12] S. B. Far, S. M. H. Bamakan, Q. Qu, and Q. Jiang, "A review of non-fungible tokens applications in the real-world and metaverse," *Procedia Computer Science*, vol. 214, pp. 755–762, 2022, 9th International Conference on Information Technology and Quantitative Management. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050922019482

[13] E. Hartwich, P. Ollig, G. Fridgen, and A. Rieger, "Probably something: A multi-layer taxonomy of non-fungible tokens," *arXiv preprint arXiv:2209.05456*, 2022.

[14] Chainlink, "What is a dynamic nft (dnft)?" Online, https://chain.link/education-hub/what-is-dynamic-nft [Accessed 15 January 2023].

[15] M. Solouki and S. M. H. Bamakan, "An in-depth insight at digital ownership through dynamic nfts," *Procedia Computer Science*, vol. 214, pp. 875–882, 2022.

[16] S. P. Shah Kaushal, Uday Khokhariya, "Smart Contract-Based Dynamic Non-Fungible Tokens Generation System," Preprint. In Review, April 13, 2023. https://doi.org/10.21203/rs.3.rs-2796956/v1.

[17] G. C. P. Karapapas Christos, Iakovos Pittaras, "Fully Decentralized Trading Games with Evolvable Characters Using NFTs and IPFS," In 2021 IFIP Networking Conference (IFIP Networking), 1–2. Espoo and Helsinki, Finland: IEEE, 2021. https://doi.org/10.23919/IFIPNetworking52078.2021.9472196.

[18] Y. Gupta and J. Kumar, "Identifying security risks in nft platforms," *arXiv preprint arXiv:2204.01487*, 2022.

[19] L. Balduf, M. Florian, and B. Scheuermann, "Dude, where's my nft: distributed infrastructures for digital art," in *Proceedings of the 3rd International Workshop on Distributed Infrastructure for the Common Good*, 2022, pp. 1–6.