On the way to self-contained generative NFT assets immutable lifecycle

Erwan J.
erwan.jpro@gmail.com

2021 November

Abstract

NFTs recently blown-up bringing with it a lot of skepticism from a part of the community and blockchain outsiders. In most cases NFTs lacks of the ability to prove ownership, authenticity and uniqueness. We propose a human-readable self-contained NFT asset model with immutable contract ruling its lifecycle in a trustless environment, then we demonstrate a proof of concept which uses generative 3D blocks built from the asset's hash.

1 Introduction

In a world where blockchain technologies shaked the entire economic system, the community started to sense a larger purpose for it with the democratization of smart-contracts: NFTs were born. NFTs, standing for Non-Fungible Tokens, are unique and non-interchangeable units of data stored on a digital ledger. This token is mostly used to sell supposedly ownership of a wide range of assets ranging from tweets to digital arts.

However the technical details of NFTs implies some flaws that are widely discussed across the community. [1]

On-chain storage is hard, and most of the time impossible because blockchains are not designed to store large asset data so most NFTs are just pointing to an url storing the asset [3]. This entierly breaks the decentralized model and put right back a third-party which is subject to data loss and depreciation of services, meaning that an asset could at some point in time be no longer accessible. This forced the community to come up with a decentralized version of assets storage (e.g: IPFS) which is more resilient by design.

Uniqueness of an NFT may also be hard to enforce without the help of some third-party tracker [4]. We've stated before that the token belongs to a digital ledger (e.g: Ethereum) meaning it is considered unique as long as we only takes the ledger it belongs to and the token itself as a reference. At the time of writing, Bitcoin is on the verge of releasing Taproot, a major update allowing for the creation of smart-contracts which will allow the use of NFTs on the Bitcoin

network. Until now, Ethereum was dominating the market of NFTs, but the Taproot update will probably make it harder to track potential duplicates.

Finally, authenticity is most of the time not ensured in any way, meaning that anyone can mint any asset that do not belong to them without effort. They run legal risks by doing this but still, it happens. This is hard for an artist to prove ownership of an asset in the first place without relying on third-parties.

An interesting implementation solving on-chain storage, uniqueness, authenticity and ownership NFTs is done by using generative art algorithms based on a data unique to a token such as a transaction hash [5]. While this satisfies the criterias discussed before, this model lacks of consistency as each smart-contract may contains different code which is hard to verify.

We propose a human-readable self-contained NFT asset model with immutable contract ruling its lifecycle in a trustless environment.

2 Single-hash asset

A single-hash asset self-contains its lifecycle and some data supposed to make it valuable (e.g: a poem) in its plaintext. One issue we face with this model is how to determine that it was honestly created. Nowaday NLP AIs, with the most recent example being GPT-3, are able to easily create entire convincing plaintexts. We can see how easy it would be for an attacker to create ostensibly honest inputs without having to do any work at all. We could minimize the impact of this threat by writing more intelligent plaintexts which would be harder to generate for an AI, however it exists another threat based on exploiting ostensibly honest values which can be used to bruteforce assets. For instance, an attacker could put an Ethereum address in the plaintext which can be easily created multiple times without having to have any semantical structure.

Overall the model is too predictable to be safe which is why we opt for a double-hash proof asset method.

3 Double-hash proof asset

A double-hash proof asset works the same as a single-hash asset but adds a unpredictable PoW layer where the hash of the plaintext is taken and maps to an **intermediate proof asset** of the same type for which the miners are given the task to provide a valid proof. Both plaintext and proof hashes are then hashed together to get the final asset hash.

Let H be any hashing function. Let n_x any chosen plaintext. Let $S = \{a_1, \dots a_n\}$, a finite set of assets where $|S| \leq |H|$, we find the corresponding asset hash a as follow:

$$a = H(H(n_x) \| H(proof(H(n_x)))$$

The H-PoW system is formally defined as $\forall H(n_x) \in S \quad \exists H(n_x)_{proofs} = \{p_0, \dots p_n\}$ where p is a valid proof.

4 Cheating

The model must be resilient to cheaters as otherwise any asset will lose its value because of the impossibility to determine whether it was created honestly or not. We explore potential attacks and define the core mechanisms the model provides to avoid them.

If a miner is able to find a plaintext hash of any *intermediate asset proof* which has a known proof, he would be able to use that proof without having to do any additional work. An attacker could build a database of such known hash proofs then use an ostensibly honest value as a nonce (e.g: an Ethereum address) to bruteforce collisions. However, this comes down to executing a traditional preimage attack which is known to be practically impossible if the hashing function is not broken. Another way to attack when $|S| < |H|$ would be to take the hash of the asset type algorithm output instead of $H(n_x)$ as this would give a higher probability of collisions but at the cost of more computation depending of the underlying algorithm. In this variant, the lower $|S|$ is, the higher the probabilities of a worth-it successful attack. Alternatively, depending on the underlying asset type algorith output, an attacker could try to find a plaintext hash which maps to any *intermediate proof asset* which is similar enough to another one with a known proof in order to try to use it. While the attack is theorically feasible, the flexibility provided by our model allows miners to get creative in order to write resilient proofs, for instance a miner could make references to the plaintext in the proof which would likely betray any miner using it with any different plaintext even if its hash maps to the same *intermediate proof asset*. An attacker would then need to preprocess every proofs before using them to sanitize them which might get very hard given their semantical freedom.

As the old adage says: *"A chain is only as strong as its weakest link"*. Our model weakest link is the *proof* as it is the last step before getting the final asset hash, meaning that if anyone find a way to add arbitrary data without invalidating it, then he could effectively bruteforce assets in a very efficient way. Since there is no length limit to a proof an attacker could fill, for instance, as much Unicode zero-width characters as needed until finding a valuable asset. Anyone must be aware of this threat which is not specific to our model as it has been used in the wild for a long time now. Note that this attack can easily be mitigated by having tools verifying if an input has hidden unicode characters.

A variant of the hidden Unicode character attack would be to add ostensibly honest artificial padding content to a proof. For instance, a miner could create a valid proof then be assisted by an AI to generate additional textual content much like a story to try to find new assets. Note that this could be an honest way for a miner to create proofs as long as he is writing the story himself meaning that miners must try to be as clever as possible to create stories related to the *intermediate proof asset* which would be very unlikely to have been created by any AI known to this day. Consequently, any value that appears to have been used as a nonce by the miner (e.g: a numeric value) must be cautiously evaluated to determine if it has honest meaning or not in which case the proof

must be considered invalid.

5 Contracts

If Leonard da Vinci had written the following sentence on the Mona Lisa canvas: "I must not be sold, at any cost.", would anybody be able to buy it?

While anyone would be able to buy the physical painting, could we say the same about its philosophy? If we consider a work as both the material and its philosophy then nobody would be able to buy our alternative Mona Lisa as a whole because the very act of buying it would ignore its philosophy. This paradox is the very basis of how contracts work.

We define a *Contract* as the portion of the plaintext data which rules over the intended asset's lifecycle. Since the *Contract* is part of the asset plaintext hash, nobody can tamper with it without altering the *intermediate proof asset* and consequently the asset itself. As soon as any clause of an asset contract is violated the asset is said to be burnt. As long as more than 50% of actors consider that the asset's immutable lifecycle must takes precedence over anything else, a burnt asset has no longer value.

Contracts are Turing-complete by design meaning that their applications are limited only by the imagination of those who write them, providing a creative framework for miners to explore. Note that a *Contract* syntax is purely a matter of miner's preferences as long as anyone can understand the statements.

6 Uniqueness

We can leverage contracts to provide a way to ensure uniqueness of any asset. We can for instance, add a clause ruling over which blockchain the asset can be sold on and constraining it to a specific tuple $(address, tokenId)$:

"I may only be minted on Ethereum with a tokenId corresponding to my hash"

It is now trivial for anyone to see what the intended lifecycle were from the get-go by looking at the plaintext and if someone tries to sell the asset on the Bitcoin network or use a wrong *tokenId* then everyone can know for sure that's a burnt asset.

7 Ownership

Ownership does not ensure authenticity, the fact that an asset's contract sets ownership to someone doesn't implies that it was created by this person. On paper, this may sound silly as to why anyone would work for free by giving ownership to someone else, however one of many reasons could be to act as a donation or for any creative purpose. For instance, we could define a conditional ownership as follow:

"Whomever mint me with a transaction hash 0xdeed... may own me."

For anyone to mint the asset as a whole, his mint transaction's hash must begin with *0xdeed*. While most of the time ownership will be set to only one entity, this asset allows multiple copies as long as the conditions are met.

8 Authenticity

While authenticity might not be the priority for a miner since ownership is what matters when it comes down to selling an asset, being able to prove that you're the author of it is still necessary.

Most of the time, authenticity is ensured with a public-key cryptographic algorithm such as ECDSA, however this would be too much burden for a miner to add a signature to its plaintext as it would add textual garbage and it might be difficult to agree on signature formats, key formats. Verifying would also be a laborious process.

We propose a **k-timestamp hash proof** technique as a variation of S/KEY [2] for trustless environment to allow a miner to ensure authenticity as needed by computing $H^k(secret)$ where k is an expiration timestamp in seconds. Let τ be a challenge timestamp, a miner can solve $prove(\tau)$ where $\tau < k$ by providing $H^{k-\tau}(secret)$, then the verifier will be able to verify that $H^{\tau}(H^{k-\tau}(secret)) = H^k(secret)$. Given the one-way property of hashing functions, it is theoretically impossible for an attacker to find a valid input for $\tau - 1$ when τ is the last known proved challenge. A verifier must ask a challenge with τ being as close as possible to the current time to ensure a fair amount of trust in the proof and any challenge with $\tau > currentTime$ must be declined by the miner in order to avoid leaking a valid proof which could then be used in the future by an attacker. After k seconds are elapsed, the miner can no longer provide a valid proof as he would need to compute $H^{-1}(secret)$. Duration between each proof can easily be adjusted by using a different granularity of k at the cost of higher or lower computation time. Note that the more the time passes the more computation a verifier would have to do while the prover will have less of it. If for any reason a miner needs to attach public data in the chain so that it appears in each proof, he can do so by computing $H^k_{n=1\to k}(n > 1 \to (H^{n-1} \| data) \wedge (n = 1 \to secret))$. Note that a verifier will need to compute the reamining hashes accordingly.

9 Future-proofness

Since NFTs are stored on a blockchain, we can verify the time it was minted at. This means that, even if a block uses a proof that is no longer considered valid, as long as it was at time it was minted then we should see no impact on the value of the block. This is an important aspect of the model usage with blockchains since we can't predict for sure that humans will always be able to beat AIs at proof writing.

10 Proof of concept

We define our NFT Double-hash asset as a 3D voxel block built with the PRNG function *xoshiro256++* initialized with the asset's hash as state where the hash function is *sha256*. The *intermediate proof asset* will be 3D block of the same type which the miners will use to build a valid proof.

11 Algorithms

Our algorithm voluntarily include biasing on voxels colors as without it we're likely to get only uniform colors distribution which would make it too hard to find at least one interesting block.

Algorithm 1 Block features algorithm

$uniform \leftarrow xoshiro256++(hash)$
$size \leftarrow uniform(1, 16)$
$colorCount \leftarrow uniform(1, 16)$
$voxelCount \leftarrow size > 1?size^3 - (size - 2)^3 : 1$
$colors \leftarrow []$
$voxelColors \leftarrow []$
$colorsBias \leftarrow x = \sum_{n=0}^{voxelCount-1} uniform(1, 100)$
$i \Leftarrow 0$
while $i < colorCount$ **do**
    $colors \leftarrow uniform(0, 2^{24} - 1)$
$i \Leftarrow 0$
while $i < voxelCount$ **do**
    $voxelColors \leftarrow colors[bias(uniform(0, 2^{24} - 1))]$
Return $size\ \ colors\ \ voxelColors$

Rendering the 3D block is done by iterating through $voxelColors$ in z, y, x axes order. Miners can point to any voxel of the block with the syntax format $[a.base64(bitmap), \ldots f.base64(bitmap)]$ where the bitmap 1 bits represent a selected voxel for a given face $a \ldots f$.

12 Conclusion

We have proposed a model of self-contained hash assets with immutable contracts and a built-in PoW mechanism to ensure asset's value. We started by exploring how single-hash assets could satisfy our needs but demonstrated that the model is too limited to prevent bruteforce attacks. To solve this, we proposed a double-hash proof asset model where a final asset's hash contains a proof for an *intermediate asset hash* based on the hash of a plaintext. We then extended the possibilites of such assets by explaining how contracts can rule

over their lifecycles in an immutable way. This model solves most of the described current NFTs flaws as well as providing a creative framework for miners to explore.

References

[1] In: (). URL: https://news.ycombinator.com/item?id=26444625.

[2] In: (). URL: https://en.wikipedia.org/wiki/S/KEY.

[3] conlan. "Cryptographic Hashing and Why Your Tokenized Art Collection is Worthless Without It". In: (). URL: https://editorial.superrare.com/2020/07/28/cryptographic-hashing-and-why-your-tokenized-art-collection-is-worthless-without-it/.

[4] Lisa Gibbons. "There is a way to protect NFTs from being replicated or lost: This company does just that". In: (). URL: https://cointelegraph.com/news/there-is-a-way-to-protect-nfts-from-being-replicated-or-lost-this-company-does-just-that.

[5] William M. Peaster. "Talking Ringers with Dmitri Cherniak". In: (). URL: https://metaversal.banklesshq.com/p/talking-ringers-with-dmitri-cherniak.