NASH: A secure asset hash with self-contained 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, and yet the technology is advertised as fulfilling all these criterias. 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 3D blocks built from an asset hash.

1 Introduction

In a world where blockchain technologies shaked the entire economic system, a part of the community sensed 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 ownership of a wide range of assets ranging from tweets to digital arts and allow token holders to speculate on their value.

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 [5]. 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 storage (e.g: IPFS) which is more resilient by design but not ideal to retrieve assets.

Uniqueness of an NFT may also be hard to enforce without the help of some third-party tracker [6]. We've stated before that the token belongs to a digital ledger (e.g: Ethereum) meaning its corresponding asset is considered unique as long as we only take the ledger it belongs to and the token itself as a reference, in other words a token is very much unique but the asset it represents is not. 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 change things and make it harder to track potential duplicates residing on different blockchains.

Finally, authenticity is most of the time not ensured in any way, meaning that anyone can effortlessly mint any asset that do not belong to them. They run legal risks by doing this but it still happens and tracking down people behind a cryptographic address is also harder.

An interesting take on solving the issues discussed above is to use generative art algorithms based on data unique to a token such as a transaction hash [7]. Generative art most often refers to art has been created with the use an algorithm, it puts the programming part as the artist's medium and let the machine express the algorithm's artistic vision. While this technique satisfies the criterias, it lacks of consistency between asset types and remains too opaque for most users which makes it hard to know what's going on. [2] On top of that, if there is no enforcing rules it might be easy to bruteforce outputs by inserting a nonce in the transaction data field which inherently lowers the value of the asset. It may also be worth noting that minting such assets on Ethereum blockchain (and most likely on any smart-contract blockchains) comes at a pecuniary cost which is not suitable for everyone.

We propose an out-of-the-box solution we name **N**FT **A**sset **S**ecure **H**ash (NASH) in the form of a human-readable self-contained NFT asset model with immutable plaintext contract ruling its lifecycle in a trustless environment. In this model, an asset is essentially created from two human-readable values *plaintext* and *proof* both written in a natural language such as English or French making accessible for anyone to create and verify with or without any technical knowledge.

2 Single-hash asset

A single-hash asset self-contains its lifecycle and some data supposed to make it valuable (e.g: a poem [3]) in its plaintext. One issue we face with this model is how to determine if 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 by an AI, however another deadliest threat awaits around the corner by exploiting ostensibly honest values which can act as a nonce to bruteforce assets in a very efficient way. For instance, an attacker could use an Ethereum address as a nonce in the plaintext which could be easily created any number of time without betraying his honesty.

Overall the model is too predictable to be used leading us to opt for a double-hash proof method introducing an uncertainty to the miner in the form of an *intermediate proof asset*.

3 Double-hash proof asset

A double-hash proof asset works the same as a single-hash asset but adds a unpredictable Human-PoW (H-PoW) layer where the hash of the plaintext is taken to map 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 a hashing function. Let n_x a chosen plaintext. Let $S = \{a_0, \dots a_n\}$, a finite set of assets, 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

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

If a miner is able to find a plaintext mapping to the hash of any *intermediate asset proof* having a known proof, he would then be able to use that proof without having to do any additional work. In its simplest form, an attacker could create one valid proof then use an ostensibly honest value in the plaintext as a nonce 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. The method could be improved by using a database of known proofs instead of a single one but the collisions probabilities would still be too low to be worth trying.

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 asset type, 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 its proof, hoping for it to be valid with his *intermediate proof asset.* While the attack is feasible in theory, 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 its corresponding 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 taking this into account would then need to preprocess every proofs before using them to ensure that none of them will betray him which can quickly become very hard given their intrinsic semantical freedom.

As the old adage says: *"A chain is only as strong as its weakest link"*. Our model weakest link is the *proof* value as it is the last component before getting the final asset hash, meaning that if anyone finds 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 many Unicode zero-width padding characters as needed until finding a valuable asset. Everyone 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 in various scamming attacks. Note that this threat can easily be mitigated by using tools verifying if an input has hidden unicode characters in it.

A variant of the invisible Unicode character padding attack would be to add ostensibly honest artificial linguistical 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 creating a story in order 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 without diverging too much from the *intermediate proof asset* meaning that miners must try to be as clever as possible in their proof writing to 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 a whole as both its material aspect and its philosophy then nobody would be able to buy our alternative Mona Lisa because the very act of buying it would ignore its philosophy component. This paradox is the very basis of how contracts work.

We define a *Contract* as the portion of the plaintext data ruling 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 its semantic remains the same.

6 Uniqueness

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

"I may only be minted on Ethereum by 0x... 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 $(address, tokenId)$ tuple then everyone can know for sure that the asset is burnt and holds no value.

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 simple ownership clause as follow:

"I belong to Ethereum address 0x... and may only be sold by it."

The clause ensure that any address other than the one defined in it cannot sell the asset in a very simple way. More complex ownership clauses can be expressed as we will see in the *Scarcity* section.

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 of a burden for a miner to add a signature to its plaintext as it would add textual garbage and reduce the human-readability. It's worth noting that it would also be difficult to agree on signature formats and key formats as it exists multiple versions of them which would make the verification process cumbersome.

We propose a **k-timestamp hash interactive proof** technique as a variation of S/KEY [4] for trustless environment to ensure authenticity as needed in a lightweight manner by computing $H^k(secret)$ where k is an expiration counter of unit of times. Let τ be a challenge, a prover P can solve $prove(\tau)$ where $\tau < k$ by providing $H^{k-\tau}(secret)$, then a verifier V 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 proof for $\tau - 1$ where τ is the latest known challenge. However, giving away the proof for τ during its corresponding unit of time would provide a reusable proof to an attacker for the time remaining until $\tau + 1$. We provide a protocol ensuring the one-time property of a proof as long as P and V respect the procedure by using a *time beacon*. In this procedure, given a challenge τ, P will privately compute $proof(\tau)$ and send a *time beacon* to V by computing $data\|(proof(\tau)\|data)$ where $data$ can be any plaintext. At this moment, V knows that the beacon was received during the unit of time corresponding to τ while having learned nothing about the proof itself, then P waits until $\tau + 1$ to finally send $proof$ to V. At this point, V has everything to verify both that $proof$ is valid and that the beacon was also computed with it. With this protocol, an attacker learns the proof for τ at $\tau + 1$ which will not be usable again as any verifier following the procotol procedure will ask him a proof for $\tau + 1$.

After k unit of times are elapsed, P will no longer be able to 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 unit of time for k at the cost of higher or lower computation time, for instance a unit of time of 10 minutes allows to build a chain lasting for 100 years with approximately 5 millions hashes which computes in few seconds on most modern computers. P could avoid unnecessary computations by providing an offset timestamp from which to determine the current τ. Note that the more the time passes the more computation V would have to do while V will have less of it. This could be mitigated by using *pebbles* where for instance V knows for sure the latest proof allowing him to use it as verification endpoint lowering its computational effort. Finally, if for any reason P 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 V will need to compute the reamining hashes accordingly.

While this interactive proof model works well between two parties, it can also be used on blockchains. Consider an asset having a contract stating the following clause:

"To mint this asset, a proof for 0x... must be provided."

In this context, P first creates a *beacon transaction* containing the *time beacon* for the current τ, wait for the transaction to be included in a block, and then mint the actual asset at $\tau + 1$ while providing a reference to the *beacon transaction* hash. An eavesdropper would not be able to outrace the miner on the minting process since he only learns the proof when it becomes unusable.

9 Scarcity

Scarcity of an asset refers to the fact that it may exists only a finite amount of it, the less there is the higher the scarcity. Scarcity could be ensured by contracts as much as uniqueness and ownership. For instance, a conditional ownership clause could be defined as follow:

Whomever mint me on Ethereum with a transaction's corresponding block hash beginning with 0xdeed may own me.

For anyone to mint the asset as a whole, his mint transaction must be included in a block where its hash begins with *0xdeed*, meaning that multiple copies of this asset could exists as long as conditions are met.

10 Future-proofness

Since NFTs are stored on a blockchain, it allows anyone to 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 the 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.

11 Proof of concept

We define our 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*. Consequently, our *intermediate proof asset* will be a 3D block which the miners will use to build a valid proof.

12 Algorithms

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

Algorithm 1 Block features algorithm

$uniform \leftarrow xoshiro256++(hash)$
$size \leftarrow uniform(1, 16)$
$colorCount \leftarrow uniform(1, 16)$
$voxelCount \leftarrow size > 1 ? size^3 - (size-2)^3 : 1$
$colorsBias \leftarrow getBiasesWithUniform(colorCount, 1, 1000)$
$colors \leftarrow getUniqArrayWithUniform(0, 2^{24} - 1)$
$voxelsColor \leftarrow getBiasedVoxelsColorArray(voxelCount, colors, colorsBias)$
Return $size\ \ colors\ \ voxelColors$

Rendering the 3D block is done by iterating through $voxelsColor$ 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 a bitmap 1 bits represent a selected voxel for a given face $a \ldots f$.

13 Conclusion

We have proposed human-readable self-contained NFT asset model with immutable plaintext contract ruling its lifecycle in a trustless environment providing a built-in H-PoW layer ensuring assets value. We started by exploring how single-hash method 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 method where a final asset's hash is built from a *plaintext* which is used to create a *proof* for an *intermediate asset hash*. 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://medium.com/treum_io/on-chain-artwork-nfts-f0556653c9f3`.

[3] In: (). URL: `https://text.bargains/amulet/`.

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

[5] 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/`.

[6] 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`.

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