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 data blobs 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 use decentralized storages solutions (e.g: IPFS) which are more resilient by design but not ideal from an UX standpoint.

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 might indeed be very much unique but the underlying asset is probably 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 might also be harder due to the opacity of the technology providing more anonymity.

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 its 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] It may also be worth noting that minting such assets on smart-contract based blockchains comes at a pecuniary cost which is not suitable for everyone as it provides a potentially toxic betting environment.

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 NFT asset with self-contained 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 any natural language making it accessible for anyone to create and verify 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 uncertain intermediate step for the miner.

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* for which the miner is 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 $\forall H(n_x) \;\; \exists H(n_x)_{proofs} = \{p_0, \dots p_n\}$ where p is a valid proof.

4 k-Novel Proof (k-NP)

We propose a model we name *k-NP* which transforms plaintext's hash output bytes to a letter set L of size k and asks the miner to use consecutive members of it starting from the beginning to build a proof by including words starting with one or more member of the set from the current position in the set. It opens both learning and creative spaces to miners and is designed to be moderately hard to create and easy to verify while being theorically impossible to efficiently be created by computers.

Given a plaintext hash output bytes, we take 8-bits slices s from highest-order bit to low-order bit and compute $s \mod 26$ to get the letter index.

The probability to find a collision given k bytes for a hash function of size n is expressed as $\frac{2^n}{26^k}/2^n$ which is equivalent to $1/26^k$. To give some perspective, the Bitcoin's network difficulty at the time of writing yields hashes starting with 9 zero bytes (18 nibbles) taking 10 minutes on average to be found for a probability of $1/2^{4*k}$ where k is the number of leading zeroes.

We specify some rules for a proof to be valid:

1. Let c be a one-way cursor in L in $[0, k]$. For a word to move c, it must starts with $L_[c, k]$.
2. It must not contain any invisible Unicode character (e.g: zero-width).
3. Interchangeable values (e.g: number) must be context sensitive such that if the value changes the proof become semantically invalid.
4. Ahead-of-time rules could be defined in the plaintext to rule over the proof content.

We explore some of the attack threats that could be used against our model and how we could mitigate them while having in mind that for any of them to be worth the try it would need to have a higher throughput than an actual human.

Naïvely bruteforcing plaintexts with some honest nonce (e.g: Ethereum address) would be wasteful and worthless as we've seen that finding a hash giving the same sequence as a previously proved one before would be harder than mining a Bitcoin block if we define a proof as being safe if it uses 18 or more consecutive byte letters.

An obvious bruteforce method would be to store a lot of textual data then use it to match against a given set L to find a valid sample. In fact, it might no even be too expensive to do so as long as the attacker have a large enough dataset and a clever way of matching the content to L. However, this attack might yield some weird proof contexts or leak in some way that the content was not meant to be used as a proof of our model which could limit the actual usable dataset.

NLP AIs is a major threat to the model as if one is capable enough, it might be able to create valid proof satisfying every rules. An attacker could build a large database of byte to keyword values, match the set L against it and send keywords to the NLP AI to get a valid proof without doing any honest work. We believe that such AI does not exist at the time of writing while not excluding that they could exist in the future such that new rules may be created accordingly to stay ahead of the NLP breakthroughs.

Rule 4 helps strenghten our model by allowing miners to specify some rules about the proof creation ahead of time in the plaintext. For instance, a miner could state:

The proof will tell the story of a lost man on a lonely planet

This way, we drastically reduce the probability of a succesful bruteforce attack by reducing the usable dataset.

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 the work as a whole as both its physical object 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, nobody can tamper with it without altering the *intermediate proof hash* and consequently the final asset hash itself. As soon as any contract's clause is violated the asset is said to be burnt. A burnt asset has no longer value as long as more than 50% of actors consider that the asset's immutable lifecycle must takes precedence over everything else.

Contracts are Turing-complete by design as they are written in any natural language 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 of ensuring uniqueness for 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 (which is unique on Ethereum) 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..."

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 environments 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 saved a proof where $\tau < k$ allowing him to use it as verification endpoint instead of $H^k(secret)$ lowering the verification computational cost. 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 to control the rate at which an asset can be minted 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* otherwise

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.

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 $[a.base64(bitmap), \dots f.base64(bitmap)]$ where a bitmap 1 bits represent a selected voxel for a given face $a \dots f$.

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 $size\ \ colors\ \ voxelColors$

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.