A Self-Governing Digital Token

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 lack of the ability to ensure true uniqueness as well as initial authenticity. We propose a Self-Governing Digital Token model providing solutions for those issues and explore how it could be used to enhance generative art NFTs.

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 on-chain generative art relying 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 and puts the programming part as the artist's medium and let the machine express its artistic vision. While this technique effectively ensure true authenticity, uniqueness and ownership, it lacks of transparency as most of the time it remains too opaque as dependant on smart-contracts which makes it hard to know what's going on providing a debatable user-experience. [2] It may also be worth noting that minting such assets may comes at a pecuniary cost which relates to a toxic betting environment experience.

We propose a Self-Governing Digital Token model ensuring true authenticity, uniqueness and ownership among other things providing multiple applications to enhance generative art NFT assets while being entierly free to mine.

2 Self-Governing Digital Token

A Self-Governing Digital Token (SVN) is an immutable non-fungible unique asset built from the hash of a self-referencing human-readable plaintext we call "Contract" and the one of a puzzle proof.

An interesting take on getting an initial intuition of how this token works is to imagine a variation of it where we include the hash of a contract in a piece of art such that it cannot be tampered with without damaging the quality of the piece. As the hash could technically holds an entire legal contract which can easily be disclosed at any time to match it against the piece self-contained hash, we could consider that the piece of art inherently dictate its own rules, hence governs itself.

In order to understand why this works, we need to explain what "self-referencing" is and means. Self-reference occurs in natural or formal languages when a sentence, idea or formula refers to itself and has many fundamental applications in mathematics, philosophy and computer programming to name a few. When reading a self-referential sentence, we find ourselves more inclined to accepts reasonable statements made about itself since we intuitively consider the sentence as a distinct entity having some authority over itself. For instance, in the context of a token's contract it would be reasonable to grant authoritative power over its selling conditions as a token is something we can credit value to.

As a more concrete example, let's try to set a token's selling starting date by including the following clause in its contract: "This token must not be sold

before 2022". It is easy that this clause is reasonable as it does not impact anything else other than itself, hence the clause has no reason not to apply.

Defining the reasonability of a clause in a formal way is a complex task but most of the time is relatively trivial to judge for a reader. The sum of readers judgements of a clause can be generalized as a consensus where the lower approbation there is , the less applicable it is. For instance, while our previous example of setting a token selling starting date would most likely get approbated by a majority of participants, "This token has been created by Elon Musk" would probably not. Approving such clause would allow, for instance, anyone to create artificial value or harmly attributing authorship to someone which makes no sense, hence it would naturally be rejected.

We can demonstrate that rational agents have no incentive to reject reasonable clauses in the attempt to harm the model stability by defining the situation as a common interest game leading to a Nash equilibrium where agents chose not to cheat. If we consider two rational agents A and B with two possible pure strategies $S_{approve}$ and S_{reject} to adopt in regard of a seemingly reasonable clause, we can establish a payoff matrix as follow:

	$S^A_{approve}$	S^A_{reject}
$S^B_{approve}$	$2, 2$	$0, 1$
S^B_{reject}	$1, 0$	$0, 0$

We assume each agent as having a participation value of 1 participating to a "consensus sum" which is attributed to any agent approving the clause and consider that when either of them reject the clause it lowers this sum proportionnaly to their participation. If both agents reject the clause they do not gain anything but lose everything. However we notice that in the case where only one of them reject the clause in an attempt to harm the model stability, they lose more than what they gain. This is true because in this situation the other agent still approves the clause, which ends up being an undecidable tie. Let $P_{approve}$ and P_{reject} respectfully be both sets of participants approving and disapproving the model at given time t where $P_{approve} \cap P_{reject} = \emptyset$, we find the model value at t as follow: $V_t = |P_{approve}|/(|P_{approve}| + |P_{reject}|$.

Another take on understanding why reasonable clauses have no reason not to applies is to take a philosophical approach of the application of a token's contract at any given time. In our previous example, the token ruling over its selling date is not enforcing its contract's application in any way other than asking the readers to model it this way while a smart-contract will programmably restrict it with no way to change it. In fact, technically nothing is preventing someone to try to sell the token on another date that the one specified. This is where a philosophical barrier kicks-in, where modeling the token in our head naturally leads us to prefer the token's contract clauses rather than the other "possibilites".

Let a token t_k have two distinct sets of possible applications $S = \{\alpha \mid \alpha \in \lambda\}$ and $S^* = \{\alpha^* \mid \alpha^* \notin \lambda\}$ where λ is the set of t_k contract's clauses applications,

we find that $|S| < |S^*|$ since applications of S are more restrictive by design. In this situation, the law of scarcity tells us that $\alpha > \alpha^*$ which should incentivize readers to prefer the immutable contract's application.

In practice, it basically means that if a token states a restrictive application in its contract then it consequently exists an infinite number of other possible applications, each of them being less valuable than the self-contained one.

We will explore applications of such token for generative NFT assets and will briefly discuss other applications at the end of this paper.

3 Token format

A token's backbone is made of two distinct values, the contract establishing governance rules and the puzzle proof. The token in its simplest usgae does not rely on any blockchain, hence we must provide a way to control its creation throughput since anyone could bruteforce an infinite amount of it otherwise. This is what purpose the proof serves, it adds an unpredictable Human-PoW (H-PoW) layer where the hash of the contract is taken to map to an *intermediate proof hash* for which the miner is given the task to provide a valid proof. Both *contract* and *proof* hashes are then hashed together to get the final token.

Let H be a hashing function. Let c_x be a chosen contract. Let ρ be an Oracle Machine capable of solving any puzzle. Let $S = \{t_0, \dots t_n\}$ be the finite set of possible tokens, we find the corresponding token t as follow:

$$t = H(H(c_x) \| H(\rho(H(c_x))))$$

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

4 k-Novel Proof (k-NP)

We propose a model we name *k-NP* which transforms contract'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.

For a given contract 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.

As we expect the hashing function output bits distribution to be uniform, we can consider the probability to find a collision given k bytes for a hash function of size n to be $\frac{2^n}{26^k}/2^n$ 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.

A proof must follow a set of rule in order to be valid:

1. Let c be a one-way cursor in L in $[0, k]$. For a word to move c of n positions, it must starts with $L_{[}c, n]$ where $0 < n < k$.
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 contract 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 contract. 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 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 $(address, tokenId)$:

"This token may only be minted on Ethereum by 0x... with the tokenId being this token's hash."

It is now trivial for anyone to see what the intended lifecycle were from the get-go by looking at the contract 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 holding no value.

6 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:

"This token belongs 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.

7 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 contract 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 [3] 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 token, one must disclose a valid plaintext for 0x..."

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.

8 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 its scarcity. Scarcity can be controlled by contracts, for instance, by defining the following clause:

"This token can only be minted 10 times by 0x... using tuples $(address, H^{[0-9]}(token))$*"*

The token has 10 available mints tuples using the specified address and 10 possible tokenIds ranging from $H^0(token)$ to $H^9(token)$.

9 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 token 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 token. This is an important aspect of the tokens usage with blockchains since we can't predict for sure that humans will always be able to beat AIs at proof writing.

10 Contracts applications

We previously discussed how contracts written in natural language are providing a wide range of possibilites but we should be aware of pros and cons in comparison to smart-contracts to understand that they both are complementary rather than self-excluding. Smart-contracts are decentralized Turing-complete program meant to be executed on a blockchain such as Ethereum which powers a lot of useful protocols without the need centralized third-parties. While we could naïvely argue that our token model contracts can theorically do anything that a smart-contract can as both are "Turing-complete" by design, we should note that in practice this is not always true. Smart-contracts are way better when it comes to manipulate absract datas and do heavy computations since it would be too much of a burden for a human to do it manually. However, as of time of writing smart-contracts have some serious flaws when it comes to requesting data external to the blockchain itself. Blockchain oracles tries to tackle this issue by pushing data on-chain but this introduce a weak point where some of those data could theorically be compromised at some point. Our model can comes handy by implicitly exploiting some kind of the large-scale human Transactive Memory [0] which allowing anyone to establish the current state of a contract clauses relying on external data that can easily be gathered by a reader. A well explaining example is the one of the "Two Degrees" [4] NFT which has built-in destructive code inside its contract listening to NASA's global temperature data via an oracle on Ethereum. If this oracle, by any means is compromised then so is the NFT, however with our model a token could just states the very sentence of the NFT "If global warming reaches 2°C above average this NFT will burn itself." since the information is of public knowledge and should be easily verifiable from multiple trusted sources.

An interesting take on self-governing tokens usage is to provide resiliency, for instance, in the case of blockchain collapse. A classical generative art asset minted on Ethereum along with some resiliency metadata has an obvious issue

which is that the metadata would not prove anything after the collapse. In other hand, a self-governing token having built-in resiliency clauses would still perfectly works as the rules are inherently part of itself without the need of any blockchain in the first place. The token could then, for example, be minted on a designated fallback blockchain without being worried of losing trust as this was provably intended from the get go.

11 Applications

So far, we've discussed theoretical applications to solve specific problems without explaining concrete application of the token itself. A token in essence is just a very large unique value while also being a unique asset. An application where such value shines is generative art. It is trivial to seed a Pseudo-random number generator with such value then algorithmically generate something from there. As long as the cardinality of the algorithm output is large enough, preferably at least as large as the cardinality of the token hashing function then we should be able to assume a bijective mapping between the two sets allowing any token to claim that it owns any such algorithmic arts built from it, hence inherently applying its governance on them.

12 Conclusion

We have proposed Self-Governing Digital Token model with immutable contract establishing its governance rules in a trustless environment while providing a built-in H-PoW layer ensuring its value. We then extended the possibilites of such token in NFT's space by explaining how contracts can ensure ownership, uniqueness and authenticity 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://en.wikipedia.org/wiki/S/KEY.

[4] In: (). URL: https://decrypt.co/72296/this-nft-self-destructs-if-global-temperature-rises-above-2c.

[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.