A Self-Governing Digital Token

Erwan Jolivet
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 some cases NFTs fails to ensure true authorship, ownership and uniqueness. On-chain generative art is a technique which solves these issues and is what motivated this paper. We propose a Self-Governing Digital Token model providing a self-contained immutable contract and explore how it could be used to enhance generative art NFTs.

1 Introduction

In a world where blockchains shaked the entire economic system, a part of the community sensed a larger purpose for it with the democratization of smart-contracts: the NFTs were born. NFTs, standing for Non-Fungible Tokens, are unique and non-interchangeable units of data stored on a digital ledger. This type of token is mostly used to sell ownership of a wide range of assets ranging from tweets to digital arts, allowing most of the time token holders to speculate on their value. However, the case of NFT art selling includes some technical details which implies some flaws that are widely discussed across the community. [1]

On-chain storage is hard if not impossible as blockchains are not designed to store large data blobs so most NFTs are just pointing to an url storing the asset [5]. As storing datas on centralized servers would entierly break the decentralization by putting right back a third-party in the model, the community had to use decentralized storage solutions such as IPFS which are more resilient by design.

A token belongs to a digital ledger (e.g: Ethereum) meaning its corresponding asset's ownership is as unique as the ledger it belongs to or even some times as the token itself since multiple tokens could point to a same effective asset. In theory, there always exists a true first minted asset by tracing back every versions of it across every blockchains and find the one with the oldest timestamp but it is a bit of a burden. 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 potentially allow the use of NFTs on the Bitcoin network. Until now, Ethereum was dominating the market of NFTs, but the new blockchains supporting smart-contracts will probably makes the tracking of duplicates more struggling than it already was. On top of that, initial ownership inherently linked to authorship is most of the time not ensured in any way, meaning that anyone can mint an unprotected asset that do not belong to them. This led to initiatives trying to track unintended selling of art pieces allowing for the authors to be notified and take actions if necessary. [6]

An interesting take on solving the issues discussed above is the use of on-chain generative art relying on unique token's data such as its transaction hash [7]. Generative art most often refers to art has been created with the use of an algorithm setting the programming task as the artist's medium while letting the machine expresses the artistic vision. While this technique ensures true authorship, uniqueness and ownership, it lacks of transparency as most of the time remains too opaque by relying on smart-contracts to make them work which makes it hard to know what's going on. [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 which is far from ideal while also being less inclusive. Newer promising blockchains tries to tackle this cost issues (e.g: Solana) by providing very efficient and low-cost transactions, however some NFTs projects smart-contracts still come with a mint price putting right back a potentially toxic betting environment in place.

We propose a free-to-mine Self-Governing Digital Token model ensuring true authorship, uniqueness and ownership among other things drawing a path to enhanced generative art with multiple applications one of them being its usage with the NFTs ecosystem.

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 an artist includes the hash of a contract in its piece of art in a way such that it cannot be tampered with without damaging the quality of the piece. As the hash could technically "hold" an entire legal contract disclosable at any time to be verified, we might find ourselves considering that the piece of art inherently dictates its own rules, hence would be self-governing.

In order to understand why this intuitively 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 accept applications of reasonable statements made about itself as its apparent self-awareness intuitively leads us to build a mental representation of it as a distinct entity having some authority over itself. For instance, when a token is self-referencing it might be deemed reasonable to grant it authoritative power over its selling conditions since it initially is something we can attach value to, similar to how we intuitively .

As a more concrete example, we could 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 should be natural to see why this clause stays within the scope of reasonable applications as there is no obvious attempt of any kind of cheating nor one of an "impossible ask".

While defining what is meant by "reasonable clause" in a formal way is a complex task, it should most of the time be relatively trivial to judge using common sense. We consider that the lower approbation there is about a clause, the less applicable value it has. For instance, while our previous example of setting a token selling starting date would most likely get widely approved, a clause such as "This token has been created by Elon Musk" would probably not as it would allow anyone to create artificial value or harmly attributing authorship to someone without any proof which makes no sense, hence it would logically be rejected to avoid such cases.

We note that a token is not enforcing the application of its contract in any technical way. In fact, nothing is preventing someone to try using a token in a way that is not specified it its contract. We demonstrate that such unintended application would most likely fail as it would exponentially increase the complexity of the consensus between "participants". 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 built-in clauses, we find that $|S| < |S^*|$ since application of S is more restrictive by design. In practice, it means that for a token's built-in clause, it exists an infinite number of other non built-in clauses that could virtually be applied. However, such unintended application would become exponentially hard to coordinate between the participants as the number of them increases since there would no longer be any immutable record to rely on to know what application applies to the token. This situation would then most likely inevitabely leads to chaos between the participants incentivizing them to rely back on the token's built-in contract. This observation allows us to assume that $V(S) > V(S^*)$ where V is an abstract function evaluating the "value" of a given set of rules, hence it should always exist one unique true reference about a token governance which is its self-contained contract.

We demonstrate that there is no reliable way for an attacker to destabilize the model just by voluntarily rejecting the model. Let $P_{approve}$ and P_{reject} respectfully be both sets of participants where $|P_{approve}| > 0 \vee |P_{reject}| > 0$ is true, we assume that $P_{approve} \cap P_{reject} = \emptyset$ and find the model's value at a time t as follow: $V_t = \frac{|P_{approve}|}{|P_{approve}|+|P_{reject}|}$. We assume that each participant have an evenly weighted opinion about the model, either approving or rejecting it as doing both is nonsensical. In practice, this means that a someone trying to destabilize the model will have to resort on social initiatives in order to minimize as much as possible the value of V since there is only one possible "vote" per human, which makes it extremely hard for an attacker to convince every participants to reject the model as the number of them increases.

3 Token format

The backbone of a token is made of two distinct values, the contract establishing governance rules and the puzzle proof. Since the token does not rely on any blockchain allowing to control its creation throughtput, the token must provide a built-in throttling method 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 $S = \{t_0, \ldots t_n\}$ 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 $\forall H(n_x)\ \ \exists H(c_x)_{proofs} = \{p_0, \ldots p_n\}$ where p is a valid proof.

4 k-Novel Proof (k-NP)

We propose a model we name *k-NP* which uses the *intermediate proof hash* to create a letter set L of size n and asks the miner to use an arbitrary number k where $k \leq n$ of consecutive members of this set starting from the beginning to write a creative "novel" by including words starting with one or more member of the set from the current set cursor position. It is designed to be moderately hard to create and easy to verify while being theorically impossible to efficiently be created by computers. Such proof could be considered as a Proof-of-Useful-Work providing a common creative space for miners.

Let H be a hashing function. Let ρ be the token's plaintext. We find the n-th letter α_n by computing $\alpha_n = H^n(\rho) \mod 26$.

As we expect the hashing function output bits distribution to be uniform, we can compute the probability of finding a collision for k letters as $1/26^k$. To give some perspective, the Bitcoin's network difficulty at the time of writing yields hashes starting with 9 zero bytes (18 nibbles) and is taking 10 minutes on average to be found. This correspond to probability of approximately $1/2^{4*9}$.

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

1. Ahead-of-time commitment must be defined in the contract.

2. Let c be a one-way cursor in L in $[0, k]$. For a word to cause c to move of n positions, it must starts with $L_[c, n]$ where $0 < n < k$.

3. It must not contain any invisible Unicode character (e.g: zero-width).
4. Interchangeable values (e.g: number) must be context sensitive such that if the value changes the proof become semantically invalid.

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 trying it would need to have a higher throughput than an actual human.

Naïvely bruteforcing plaintexts with some honest contract's 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 easily be harder than mining a Bitcoin blocks.

A better 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 k-NP proof which could prevent the usage of a potentially large subset of the dataset.

However, considering an advanced enough NLP AI model, an attacker could build a large letter-to-keyword database, match a given set L against it and use keywords to instruct the model to write a valid proof without doing any honest work. We believe that such AI does not exist at the time of writing but we are not excluding that they could exist in the future leading to new rules definitions in order to stay ahead of the NLP breakthroughs.

A key component to reducing the above weaknesses is to use what we call an Ahead-of-time commitment in the token's contract. Ahead-of-time commitments 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 following commitment:

The proof will tell the story of a ape on a lonely blockchain searching for his owner.

This drastically reduces the probabilities of a succesful bruteforce attack while providing tools for miners to stay ahead of IAs by allowing them to craft restrictive and creative proofs with little to no sacrifice on their side.

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 authorship, 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.

Authenticity can be ensured with a public-key cryptographic algorithm such as ECDSA. In our case it would mean signing a part of the contract with a known public key and adding the signature to it. However, such process might be too much of a cumbersome as agreeing on key formats, algorithms and what portion of data should be used to create and verify the signature would most likely be hard.

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 $H^k_{n=1\to k}(n > 1 \to (H^{n-1} \| data) \land (n = 1 \to secret))$. 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 is mintable 10 times by 0x... using the n-th hash in the chain as tokenId starting from this token's hash.

The token has 10 available mints tuples using the specified address and possible tokenIds $H^{[0-9]}(token)$.

9 Future-proofness

As NFTs are stored on a blockchain which act as a trusted decentralized timestamp server anyone is able to verify the time a token was minted at, meaning that even if it uses a proof that is no longer considered valid, as long as it was at the time it was minted then no impact on the value of the token should be witnessed. It is an important aspect of the combination of self-governing tokens with blockchains.

10 Contract applications

Contracts are providing a wide range of possibilites but we should be aware of their advantages as well as their caveats in comparison to smart-contracts to understand that they both are complementary rather than self-excluding and that SVN Tokens are only focused on the token itself while smart-contracts have a broader set of applications. 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 of 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 external data. 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 comes handy in this situation by implicitly exploiting some kind of the large-scale human Transactive Memory [0] allowing anyone to establish the current truthness of a contract clauses by relying on external data that can easily be gathered and fact-checked. 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 datas via an oracle on Ethereum. If this oracle, by any means is compromised then so is the NFT, however for it to work with our model, a token could just state the very sentence "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 without the need of anything else.

Another interesting possibility of self-governing tokens 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 comes with 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 Token Applications

All of the theory inevitably comes down to what can be created with such tokens. A token in essence is just a very large unique value being a unique asset by itself. A domain where such values shine is generative art programming as it would be trivial to seed a Pseudo-random number generator with a token's hash value then algorithmically generate something unique from there. Note that an algorithm's cardinality should preferably at least match the cardinality of the token's hash function as well as providing uniformly distributed outputs to avoid duplicates. However, an interesting property of SVN tokens is that even if an algorithm has a low cardinality, each token mapping to the same given output will still have different values as their respective contracts and proof will most likely be differents.

12 Conclusion

We have proposed a 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 the NFTs space by explaining how contracts can ensure ownership, uniqueness, scarcity and authorship 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] "DeviantArt Protect: Helping Safeguard Your Art". In: (). URL: `https://www.deviantart.com/team/journal/DeviantArt-Protect-Helping-Safeguard-Your-Art-884278903`.

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