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 lacks of the ability to prove ownership, authenticity and uniqueness, and yet the technology is advertised as fulfilling all these criterias. We propose a Self-Governing Digital Token model and then we will discuss applications of such model to solve the previously stated issues. Finally, we will explore further other applications of such tokens.

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 [4]. 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 [5]. 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 [6]. 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 a Self-Governing Digital Token model providing multiple applications one of which is the creation of generative NFT assets with provable ownership, uniqueness and more.

2 Self-Governing Digital Token

A SEGOV Digital Token is a unique and immutable valuable asset built from the hash of a self-referencing plaintext we call "Contract" and a puzzle proof.

An interesting way to picture this model is to consider how including the hash of a contract in a piece of art would impact its lifecycle. Since the hash can technically holds an entire legal contract which can easily be disclosed at any time to match it against the hash value, we could consider that the piece of art effectively dictate its own rules, hence governs itself.

In order to understand why this works, we need to explain what "self-referencing" means. Self-reference occurs in natural or formal languages when a sentence, idea or formula refers to itself [0] 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 build a mental space [0] where the sentence is considered as a distinct entity having reasonable authority over itself. For instance, in the context of a token's contract we could build a space where the token has reasonable authoritative power over its lifecycle or over its selling conditions.

As a more concrete example, we could try to set a token's ownership by including following clause in its contract: "This token belongs to the city of New York". This clause builds a new space where the token (which lives in the base space or reality space) belongs to the city of New York. As this clause should be deemed reasonable, we can assume that it is true both in its own space as well as in the base space.

The reasonability of a contract clause is complex to define in a formal way but is relatively trivial to judge for a human. As we demonstrated with our previous example, everything depends on the context. For instance, it is easy to see how a statement trying to borrow the value of the Mona Lisa is silly and should be rejected by the vast majority of the contract's readers. As contracts are built in natural languages, they provides a creative framework for miners to explore.

Using Game Theory we can demonstrate why reasonable clauses have no reason to be rejected by a rational agent by defining the model as a common interest game leading to a Nash equilibrium where both agents do not 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:

	$S_{approve}$	S_{reject}
$S_{approve}$	$2, 2$	$0, 1$
S_{reject}	$1, 0$	$0, 0$

This matrix considers an optimal payoff value of 2 and consider that when A or B rejects the clause it proportionally impacts the other agent value. If both agents reject the clause they do not gain anything but lose everything as the model become useless. However in the case where only A or B reject the clause in an attempt to harm the model stability, they still lose everything but only remove a value proportionnal to their participation to the other agent. This is true because the other agent still approve the clause, which ends up in an undecidable tie. Given the two previous scenario, it is trivial for the two rational agents to see why approving the clause is more profitable for them since they both gain something in the process, the ability to use a token.

While we will linger on its usage for generative NFT assets, we will also briefly discuss other applications at the end of this paper.

3 Token format

A token 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 a hashing function. Let c_x a chosen contract. Let $S = \{t_0, \dots t_n\}$ a finite set of tokens, we find the corresponding token t as follow:

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

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 burnt and holds 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 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:

"For this token to be correctly minted on Ethereum, its transaction's block hash must begin with 0xdeed, otherwise this is a burnt token"

For anyone to mint the asset as a whole, his mint transaction must be included in a block where its hash begins with *0xdeed* which inherently regulate how many assets can be minted in a given period of time.

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 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.

10 Smart-contract comparison

We previously discussed how contracts natural language contracts are provide a wide range of possibilites but this also comes with some limitations. A natural comparison is one with smart-contracts. 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 third-parties. While we could naïvely argue that our token model contracts can theorically do anything that a smart-contract can do as both are Turing-complete by design, we should note that in practice this is not true. Smart-contracts excel when it comes to manipulate complex data and do heavy computations which would be too much of a burden for a human being to keep track of. However, as of time of writing have serious flaws when it comes to requesting data external to the blockchain itself. Oracles tries to tackle this issue but this still relies third-party data that can potentially be compromised in the process of pushing them on-chain. This is where our token contracts can come handy to solve some of this issues by implicitly exploiting some kind of large-scale Transactive Memory which allow anyone to establish the current state of a contract asking for external data.

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 being an asset itself but which can also be used for anything else. One area where such value shine is generative art. It is trivial to use such value as a seed to initialize a Pseudo-random number generator and 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 consider a bijective mapping between the two sets allowing any token to claim that it owns any such algorithmic arts built from its value, hence 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] 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/`.

[5] 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`.

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