OnChainBunnies Whitepaper
On-chain NFT collection that prevents insider trading at mint

Axelle Moortgat Tom Holvoet

Abstract

Last year, Forbes published an article where the son of millionaire was accused of participating in Non Fungible Token (NFT) market manipulation for his own benefit. This white paper presents a comprehensive solution to address the presence of insider trading in NFT collections. The proposed solution is twofold. The first component involves the complete on-chain implementation of the collection to ensure transparency. This decreases the developer's ability to engage in malicious acts and sharing confidential data to third parties. However, bringing the collection entirely on-chain is expensive. A solution is proposed that leverages the Loot generating algorithm and incorporates additional optimizations. All optimizations are tested with a self created NFT collection, called OnChainBunnies. First a smart contract for this collection without optimizations was executed. After this the optimizations were added, resulting in a 99.95% reduction in gas deployment. The second part exists of an algorithm that selects a sample from the entire set of conceivable combinations of traits using a random number generated on-chain. However, this approach introduces two challenges. The first challenge is the introduction of duplicates in the collection. Verifying the presence of duplicates incurs significant storage expenses. A sampling algorithm without replacement is selected to address this issue. By combining on-chain generation of images together with on-chain sampling, this white paper presents the first NFT collection to tackle insider trading in NFT collections while preventing duplicates and reducing storage costs.

1. Introduction

Even though the blockchain is touted as a trusted environment that excludes trusted third parties, this is not the case with most NFT collections. A significant degree of trust in developers is needed. Last year Forbes published an article where a man was accused of insider trading in NFT collections [1]. Developers possess insider trading information. During the process of generating a collection, specific traits are assigned to each item. Consequently, each unique `tokenId` corresponds to multiple `traitIds`. However, in most collections, the developer possesses the knowledge regarding which tokenId results in rare `traitIds`. Addressing this problem necessitates bringing an entire collection, including images, on-chain and the implementation of an algorithm that employs randomization during the reveal phase. As a result, the developer is no longer in possession of the sensitive information because it is randomly generated on the blockchain.

In fact, when a developer chooses to store the NFT images external, it comes with confidence in a central party. This goes against blockchain's guiding principles of decentralization. A malicious developer may remove the images or the server may be attacked. Furthermore, the external server provider has the option of no longer hosting the image [2]. Previous incidents have demonstrated instances where NFT images have vanished because of the external storage. These incidents include the case where all NFTs stored on the FTX servers disappeared [3] or the Raccoons story. In the latter the founders of the Raccoons collection changed all images to a pile of bones, higlighting the vulnerability of most NFT collections [4]. To address this issue, OnChainBunnies aims to provide a solution by fully bringing the NFT collection onto the blockchain, while ensuring a reasonable deployment cost.

This white paper describes the first trustworthy NFT collection, combining resilience against insider trading, complete transparency, full on-chain functionality, and remarkably low deployment costs. Section two provides an overview of the collection, OnChainBunnies. After this, the problem of insider trading is illustrated. The fourth section describes the methods used to optimize bringing the entire collection onto the blockchain, considering the high cost associated with storing images. A cost-cutting method is presented and described. Finally, a sampling method that selects random combinations of traits from all conceivable combinations during mint is presented. Duplicates are not permitted in this case, and a one-to-one mapping from a `sampleId` to the combination of `traitId`s is required.

2. OnChainBunnies

OnChainBunnies is an NFT collection consisting of 5000 unique items, each characterized by a combination of 9 different traits: purse, bracelet, wizard, whisker, eyebrows, hat, glasses, tie, and background. These traits correspond to 8, 6, 5, 4, 4, 30, 24, 28, and 6 values, respectively, resulting in a staggering total of 77,414,400 possible combinations, excluding the background trait. The collection employs the SVG format, utilizing geometric shapes to minimize storage usage. The base figure itself occupies 1633 bytes, while the traits range from 225 to 671 bytes. The choice of Scalable Vector Graphics format serves a twofold purpose: it ensures that image resolution remains consistent when scaled and allows for easy manipulation through code due to the composition of different elements. For instance, rectangles can be described using the "rect" element combined with width, height, and coordinate parameters (x and y). This collection was generated as part of a thesis that involved broader research into reducing on-chain storage costs in NFT collections. This whitepaper explains the best optimizations discovered in the thesis [5].

3. Insider trading

Insider trading occurs due to the fact that the sampling of the combination of traits is predictable. This predictability can be attributed to two methods. Firstly, in off-chain collections, where the images are stored with a third party, the sampling occurs off-chain in advance. The developer assembles the items before deployment and stores them at a remote server or via a distributed file system. Secondly, in the case of the Loot collection, the code becomes publicly available upon deployment, allowing anyone to calculate the exact images that will be generated. In both scenarios, it is clear which `tokenId` will result in which `traitId`s. Secondly, in the case of the Loot collection, the code becomes publicly available upon deployment, allowing anyone to calculate the exact images that will be generated. Anyone who is aware of the method can perform the calculation and act in a manner that benefits them.

In both cases, it is evident that which `tokenId` results in which image. However, even if a person possesses this information, they still need to be able to purchase the correct image. How this works and how easy this is depends on the method of reveal. To illustrate these methods, let's consider an example where a person is aware that `tokenId` 1723 corresponds to a rare and valuable NFT.

A first method that can be implemented is a reveal at mint. In this case, the image is revealed immediately upon mint. The `tokenId` serves as a counter in the minting process, ensuring that the first buyer owns `tokenId` 1, and subsequent buyers are assigned incremental `tokenId`s. So in this case, the person with the insider information has to wait until he is the 1723 buyer and mints at that specific time. This is not always easy due to the fact that there may be other buyers at the same time. A second option for reveal works with a reveal phase. During mint, the `tokenId` is calculated in the same manner as before, but the image and traits are not immediately visible. Nevertheless, the image is already for sale before the reveal. If a person with insider info accidentally minted `tokenId` 1724 because there was another buyer at the same time, the person can still offer a higher price for `tokenId` 1723. The unsuspecting buyer who sees this offer will likely accept it, unaware of the item's value.

To counter the problem of insider trading, an algorithm can be constructed to sample the items from all possible `sampleId`s in a random manner at the time of the mint or at reveal, depending on the method chosen. It was chosen to work with a reveal at mint since it is trivial to convert it to a reveal phase. In this case the sampling algorithm is performed at a different moment in the process. Making the sampling random ensures that neither the developer nor the buyer can calculate which `tokenId` results in which `traitId`s. However, to perform this sampling algorithm, an assembling algorithm like that of Loot needs to be employed. This will be discussed in the next section.

4. Cost reduction on-chain storage

In order to shift from off-chain selection of `traitIds` to on-chain selection, the complete collection, including the images, must be stored on the blockchain. However, storage cost is the most expensive cost on the blockchain. Extensive research efforts have been dedicated to finding effective strategies for reducing these costs.

The Loot generation process, also well known from the OnChainMonkey collection, is used as the foundation for the low-cost method of storing images on-chain [6]. In this technique the traits are stored as storage data in the smart contract opposed to each image individually. On querying the image, the `tokenURI` method will derive the combination of traits from the `tokenId`. Subsequently, the image will be assembled on-chain and displayed to the requester. Generating the image instead of saving the full image ensures that the gas price can be reduced by an average factor of 855.

There are three methods applied sequentially to improve the Loot contract. As the first method, the standard Solidity optimizer can be used as an initial optimisation. This optimizer ensures optimal translation of Solidity code into opcodes. This leads to a reduction of gas cost of 14,90 % for deployment. Second, all of the SVG components are maintained as storage data, which is the most costly data location. Therefore, as a performance improvement, one may choose to move the arrays of SVG strings to the related getter as memory data. As a result, the traits will not be stored as state variables but rather derived from the code each time. Whenever an image retrieval occurs, the traits will be allocated as temporal data and deallocated after the retrieval process. This optimization introduces a reduction of 44,32 % in terms of deployment cost in comparison with the previous optimization. Finally, the application of the sampling algorithm along with bitfield encoding provides a further optimization of 10,65 %. This optimizations is further discussed in the next section. All findings are summarized in Table 1. It can be substantiated that all the optimizations together, including the Loot algorithm, results in a deployment cost decrease of 99,95 %. The Loot algorithm was further optimized with a deployment cost reduction of 57,66 %.

5. Random sampling on-chain

The selection of valid items is facilitated through a sampling algorithm that relies on the utilization of a random number. The random number is generated using the prevrandao value of the current block [7]. The "prevrandao" field refers to a specific field within the Ethereum Beacon Chain that stores the previous random number generated by the RANDAO process. It is used to choose the validator of the block and serves as an input for the next round of the RANDAO process, ensuring a verifiable and transparent generation of random numbers within the Ethereum ecosystem. It is worth mentioning that this approach

Technique	**Deployment cost [gas]**	**Optimization factor**	**Optimization percentage**
One-by-one storing	~13 515 385 000	~855	~99,88%
Loot generation without optimizer	15 806 809	1,17	14,90%
Optimizer with 200 runs	13 452 033	1,17	14,90%
Memory data	7 490 396	1,80	44,32%
Sampling with bitfieldencoding	6 692 382	1,12	10,65%

Table 1: Results optimisations in terms of deployment gas cost

introduces the possibility for block validators to withhold from validating a block if the generated random number does not align with their preferences. In this case, the validator is also a buyer of the NFT colleciton. Furthermore, the validator is not aware of the collection's rare traits at mint, only if the validator mints the last item of the collection. Finally, the hash of the concatenation of the `tokenId`, address of the buyer and the prevrandao field is used to generate the random number. This ensures that the random numbers generated by several mint operations within a single block are distinct. A more in depth study of different methods to generate randomness in NFT collections can be found in the thesis [5].

5.1. No duplicates

The choice to select in a random fashion may cause a combination of traits to be accidentally selected twice. However, the problem of duplication can be effectively addressed through the implementation of a sampling technique without replacement [8]. That means that the selected item is not placed back in the pool after the sampling, thereby eliminating the possibility of duplicates. In this case, this is achieved by selecting the items in an monotonically ascending order. The pseudo code of the algorithm is represented in 1 and will be carried out for every new mint operation. Here, n represents the sample size, which correspond with the number of items there will be selected. For the onChainBunnies collection this is set to 5000. N corresponds to the population size, which equals the number of possible combinations. In the OnChainBunnies collection this is fixed to 268,435,456 and will be further discussed in the next subsection. Finally, m represents the number of items that are already minted and t the last sampled id. Each time an item is sampled and thus a new mint operation is carried out, a random number u is generated. The random number has a value ranging from 1 to n/N. The new `sampleId` is formed by the sum of the random integer and the previous `sampleId` t. The parameters t and m will be stored as storage data. The parameters n and N are fixed and thus can be stored as memory data.

Algorithm 1 Sampling tokenIds without replacement

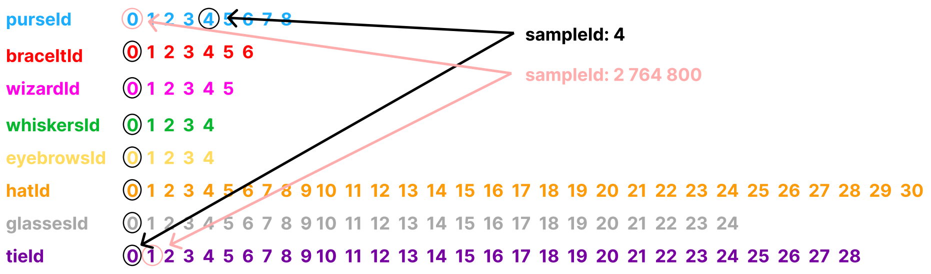
```
$n \leftarrow 5000$
$N \leftarrow 268435456$
$t \leftarrow 0$
$m \leftarrow 0$
$u \leftarrow u \sim (1, \frac{n}{N})$
$id \leftarrow t + u$
$m \leftarrow m + 1$
$t \leftarrow id$
**return** id
```

5.2. One-to-one mapping

The algorithm described above guarantees the exclusion of duplicate `tokenIds` in the selection process. However, it is essential to ensure that different `tokenIds` do not map to the same combination of `traitIds`, as this would reintroduce duplicates. Consequently, a unique mapping between the `sampleId` and distinct `traitIds` is required. Moreover, this mapping must ensure that it will not be possible to predict the next picked traits. For example, a mapping based on modulo calculations makes it possible to predict single traits. This mapping is illustrated in Figure 1. In the example the `purseId` is calculated as `sampleId` % 9, while the `tieId` is calculated as `sampleId` % 2 764 800 (8*6*5*4*4*30*24). Now, the buyer will know that `sampleIds` less than 2 764 800, will yield in an item with `tieId` = 0. If the buyer wants a tie with id 7, he just waits long enough until the `sampleId` is higher than 2 764 800 * 7. To mitigate this issue, a mapping using bitfield encoding was adopted.

In bitfield encoding traits are extracted from a single `sampleId`, with each trait represented by a specific number of bits. Figure 2 is an illustration of an example. The `tieId`, in this case, exists of 5 bits, which correspond to $2^5(32)$ distinct values. In the example the bits that assemble the `tieId` are located at index 8, 19, 20, 21 and 28 respectively. Since the next `sampleId` is incremented within the range of 1 to n/N, which in the case of OnChainBunnies corresponds to 1 to 25,687, it becomes impossible to predict the specific values assigned to the last 14 bits (equivalent to 16,384 or 2^14). As long as a trait has at least one presentable bit within these last bits, the corresponding `traitId` cannot be predicted. However, it is possible to determine the range within which these `traitIds` will fall. Consequently, the `tieId` can differ for each consecutive `sampleId`, ensuring uniqueness. Furthermore, the mapping ensures a unique mapping from `sampleId` to `traitIds`. Moreover, this method guarantees low-cost storage. Instead of seven separate `traitIds`, only one id has to be stored for each NFT.

In the case of the OnChainBunnies collection, 28 bits are utilized to represent the OnChainBunnies, accounting for 268,435,456 possible combinations.



purseId 0 1 2 3 4 5 6 7 8
braceltId 0 1 2 3 4 5 6
wizardId 0 1 2 3 4 5
whiskersId 0 1 2 3 4
eyebrowsId 0 1 2 3 4
hatId 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
glassesId 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
tieId 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
sampleId: 4
sampleId: 2 764 800

Figure 1: Modulo mapping

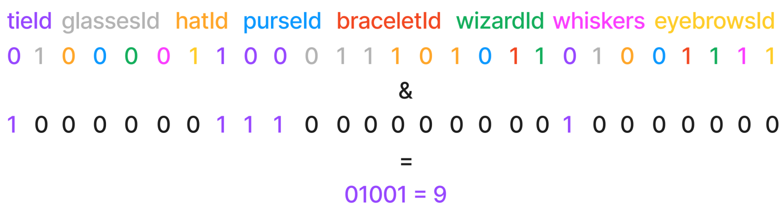


Figure 2: BitfieldEncoding of tieId

However, not all combinations are valid. For instance, the bracelet trait is represented by 3 bits, resulting in 8 distinct values. Value 8 is considered invalid. To address this, a check method has been implemented to verify the existence of all values. If any invalid values are encountered the token is re-sampled.

To enforce rarity, the n/N factor and the distribution of bitfields can be adjusted when generating the random number. Lowering the n/N factor will predominantly result in lower `sampleIds`, causing the higher bits to be set less frequently. As a consequence, traits associated with higher values will be selected less frequently. This effect primarily impacts the trait corresponding to the highest bit.

5.3. Proof

The proof of the algorithm exists out of two parts and is based on the proof provided by Knuth [8]. As a first part there can be verified that there occur no duplicates in the collection when using the sampling algorithm. This can be simply justified by the fact that the `sampleId`s are picked in ascending order and there is one-to-one mapping from `sampleId` to `traitId`s. As a second part there can be proven that at most N records are input. This means that a `sampleId` higher than the sample size will never be picked. This can be confirmed by the fact that the maximum difference between two `sampleId`s will be the sample size divided by the population size. If the maximum difference is always chosen at random, the maximum `sampleId` will be equal to sample size.

6. Conclusion

In this paper, the first on-chain collection that prevents insider trading during the reveal phase, while assuring the abscense of duplicates and low deployment cost, is presented. In order to achieve a cheap deployment cost various optimizations have been implemented. These optimizations include, the Loot assembling method, the usage of the Solidity optimizer, memory data and bitfield encoding. An overall reduction of 99,95% has been achieved. In a second step a sampling algorithm without replacement was presented to select the combinations of traits random and on-chain during the mint phase. The random number is generated by leveraging the prevrandao field of the current block, the `tokenId`, and the address of the buyer. Finally, the bitfield encoding provides a one-to-one mapping from the `tokenId` to the `traitIds`, ensuring fairness during the mint process.

References

[1] J. Kauflin, Did the son of the world's third-richest person trade nfts with inside information?, forbes.com (March 2022).

[2] T. Hepp, M. Sharinghousen, P. Ehret, A. Schoenhals, B. Gipp, On-chain vs. off-chain storage for supply-and blockchain integration, it-Information Technology 60 (5-6) (2018) 283–291.

[3] E. Reguerra, Nfts minted on ftx break highlighting web2 hosting flaws, URL: `https://cointelegraph.com/news/nfts-minted-on-ftx-break-highlighting-web2-hosting-flaws`, last accessed 24 May 2022 (2022).

[4] S. Dickens, Nft collection raccoon secret society "kills" entire project to prove a point, URL: `https://finance.yahoo.com/news/nft-collection-raccoon-secret-society-154509442.html?`, last accessed 2 November 2022 (2021).

[5] A. Moortgat, On-chain storage of nft colletions on the ethereum network (2023).

[6] Loot smart contract, URL: `https://etherscan.io/address/0xff9c1b15b16263c61d017ee9f65c50e4ae0113d7#code`, last accessed 23 february 2023 (2023).

[7] K. Chatterjee, A. K. Goharshady, A. Pourdamghani, Probabilistic smart contracts: Secure randomness on the blockchain, in: 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), IEEE, 2019, pp. 403–412.

[8] D. E. Knuth, Seminumerical algorithms, 2nd Edition, The art of computer programming 2, Addison-Wesley, Reading (Mass.), 1971.