

# Colored Dollar: A solution to Token Hacks

Akshit Vig  
[akshit@autonomint.com](mailto:akshit@autonomint.com)

Harsha Surya Abhishek  
[abhishek@autonomint.com](mailto:abhishek@autonomint.com)

July 29, 2024

## **Abstract**

Token hacks are one of the biggest problems in blockchain right now. The open source nature of decentralised applications introduces multiple potential attack vectors. As per defiyield's Rekt's database, around \$20 billion USD has been hacked till now. Out of these, only 10% of the funds worth only \$2 billion were recovered and rest is lost. Moreover, the repercussions of such attacks extend beyond the immediate financial loss, considering the damage from post-attack disruption to the normal course of business, forensic investigation, restoration and deletion of hacked data and systems, and reputational harm. Consequently the total costs escalates even further. This paper presents a solution to solve this issue while preserving the decentralised structure inherent to blockchain technology.

# 1. Introduction

Decentralised applications (like stablecoins) relies on tokens (i.e. fungible tokens) to handle the token economics, utility and payment mechanism for a user friendly and trustworthy design. These tokens are fungible in nature and utilised across different defi applications to do a plethora of composable functions like depositing as collaterals in lending & borrowing protocols, as Liquidity enablement by acting as LP in decentralised exchanges (DEX). These tokens are sitting in the treasuries of multiple protocols and transferred to centralised exchanges by users where multi-sig wallets are responsible for enabling deposits & withdrawals of token from CEX to blockchains. A single fungible token can be part of multiple txns and circulation before it is burned by some action. Whenever a hack occurs, due to some Dapp vulnerability, millions of dollars worth of tokens are hacked away and sent to multiple wallets which then utilises cryptocurrency mixers/tumblers to hide the origin and owners of funds by blending the assets of many users. It becomes difficult for protocols hacked and parties issuing tokens to recover this amount in a decentralised fashion. Freezing wallets or blocking accounts is currently used to is termed as against the decentralised money structure are also not able to pinpoint the wallets and that entire amount is lost.

Crypto hacks are one of the major reasons behind hesitancy in adoption of crypto rails and solutions by mainstream organisations as per Chainalysis. Until & unless, we find a definite solution to this, crypto won't be able to become a true internet money for the global world. The current solution of handling exploits/money laundering/hacking scenarios are freezing of wallets after instruction from Law enforcement [2]. The 'freezing' happens by blacklisting of the msg.sender function in token contracts by token issuers. This prevents the specific user from moving funds out of their balance, or a contract from moving token out of the user's balance.

If these tokens are deposited in some application as collateral then the funds are under the vault address balance and but can only be withdrawn by the liquidator once the debt reaches some threshold. For Example: USDT in MakerDAO vaults are under the vault address' balance. The user is still free to withdraw USDT from the vault if they want to, but the funds will be frozen once it hits their balance as per USDT token contract. In the case that they choose not to withdraw, the liquidation mechanism should still kick in when the DAI debt grows to the threshold. The USDT in the liquidator's balance sheet will be free to move about as the USDT contract doesn't track funds like dollar bills with numbers, they're fully fungible and are now under another address' balance.

As tokens are fungible in nature so freezing of wallets is currently the only solution utilised to tackle the funds from being circulated across multiple wallets. In case the conflicted wallets are identified then these funds are freezed till some conclusive evidence comes up pointing to the innocence of the person in control of the wallet. It's a slow process and easily bypassed by

attackers or money launderers by using crypto mixing services/tumbles like Tornado cash to obfuscate the origin.

However, one of the bigger issues as we are moving towards regulation and as more & more entities are required to have a regulatory wrapper is the possibility of a big systematic attack that can be launched on different crypto applications. Considering the example mentioned above, as USDT is being utilised as a collateral in Maker DAO's contracts to mint DAI. If Tether (issuer of USDT) is instructed by enforcement authorities to freeze 'vault address' in Maker DAO containing some of the conflicted or hacked USDT collateral which was deposited by the hacker then it would freeze the entire funds sitting in vault address balance apart from a specific user's conflicted amount. This would destabilise Maker DAO functions and will restrict their ability to redeem assets and ultimately lead to fall of DAI peg and a lot of protocols relying on DAI as backing/in treasuries/in LPs etc will face major economic consequences leading to instability of the entire system. These attacks can be orchestrated & architected by competitors or regulatory bodies as well to destabilise the on-chain crypto systems.

Attackers easily bypass blacklists by creating new addresses, making it difficult to maintain effective defenses. They use proxy contracts to mask identities and reroute transactions, hiding the original source of funds. Additionally, they exploit flash loans to carry out complex attacks without needing upfront capital, making them hard to detect. To further obscure the trail of funds, attackers often use mixer services like Tornado Cash, complicating efforts to track stolen assets back to their origins.

Usually, a lot of high class security solutions as utilised by covert agencies relies on destroying an "asset of value" if the attacker is able to exploit the program [3]. In this incidents, there is usually a security policy which can either be destroying the asset being stolen or destroying or preventing the destination from being able to access or decrypt the asset. Since, there is always a duplicate copy existing in the system or it's easier to re-create the asset so that's why the security policy of destroying the "asset of value" is mostly chosen so as to destroy it before it comes under attacker control. But, it's very difficult and costly to outnumber and recover the asset back from the attacker without any collateral damage to the parties involved. A similar mechanism is currently missing in the blockchain space when some unfortunate hack situation happens.

We feel that the main issue here is not being able to attribute the fungible tokens to the minter while they move across wallets and application. Attributing fungible tokens with different colors (metadata) and tracing them across blockchain is a difficult problem and is more popularly known as the Colored coins problem. Before Ethereum was invented, few papers had come trying to build Colored coins on Bitcoin to attribute the Bitcoin satoshis with different assets apart from Bitcoin [4]. The purpose was to create and attribute physical, monetary or Real world assets in a way such that they can be stored and transferred digitally without reliance on single

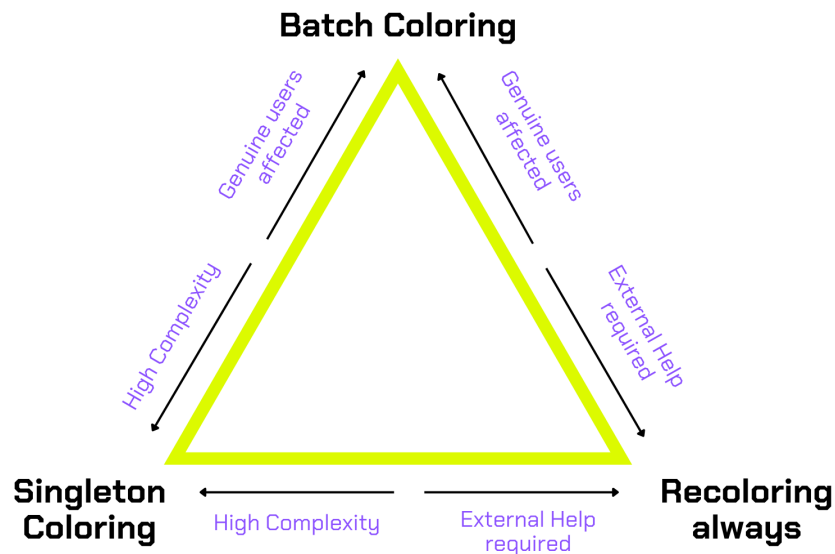
third party just like Bitcoin. This became possible on Ethereum blockchain where one can easily launch a new token by adopting different ERC standards as per the use case.

However, the problem of attributing fungible tokens with different colors or metadata is still not possible due to computationally intensive nature. It is possible to attribute a single token with some metadata (NFT). Currently, solving the 'Colored coins problem' require attributing or mapping the fungible tokens to minters and tracing those mapping losslessly as these mapped /colored fungible tokens are circulated across wallets. This is very difficult computationally as it entails mapping the colored tokens from  $N$  minters across every wallet and debiting & crediting the different  $N$  minted colored tokens whenever a transfer takes places from 1 wallet to another. Mapping  $N$  minters would result in  $O(N)$  storage complexity per wallet and transfer of some of them will entail  $O(N)$  space complexity which makes it a highly expensive solution to be handled in blockchains and hence difficult to track & attribute for different tokens in a decentralised structure.

A solution is needed which can solve this issue with low complexity for particular usecases. Our primary objective is to solve the issue of token hacks. This paper outlines the mechanism that will allow us to solve for this issue by using Colored Coin attribution mechanism to trace the colored tokens across blockchain and take appropriate governance policy through applications of rebasing.

We have created a solution that makes use of color tracing algorithm with  $O(1)$  complexity and utiising concepts like Bloom filtering to create space efficient probabilistic data structures. This will be followed by a unique rebasing mechanism to allow rebasing across specific colors. The objective is to directly attribute the hacked tokens as they move across wallets across blockchains and then use decentralised governance to perform the desired action on them. Our solution will also be able to work in a Mixers/Tornado cash based environment as we are attributing & tracing the tokens and not tracing the wallets which usually gets mixed up in this crypto mixer solutions.

## 2. The Trilemma of Color Tracing Algorithm in countering hacks



If the color tracing algorithm is based on single token coloring where individual tokens are assigned color and the color will remain same always then it will help in tracking every token at any moment irrespective of the attacker transferring tokens across multiple wallets. However this will be a highly complex solution requiring high  $O(N)$  storage complexity for every wallet and  $O(N)$  storage complexity for every token and  $O(N)$  space complexity during every transfer. Thus it is not feasible in current setting.

If the color tracing algorithm is based on batch coloring of the tokens that will always remain same on every transfer then it will be less complex than above and can help in tracing the tokens and imposing restrictions on token transfer for the identified colored coins. But, this will then restrict the transfer ability for genuine users also who might have been holding the same colored coins. Hence it is also not an optimal solution. Also, this will require  $O(N)$  storage complexity and  $O(N)$  space complexity.

If the color tracing algorithm is based on a policy that color (metadata) will always update to a new unique color (metadata) on every token transfer then we need to keep a track of history of all metadata updates from the origin wallet where the hack occurred to any proceeding wallet where the hacker transfers the amount. This will require the help of an external solution. Also, we would need to update the different. Also, this kills the ultimate purpose of implementing colored coins which is being able to know the location of colored coins at any moment without requiring to track it.

Another possibility of above algorithm is conditional coloring i.e. If the color tracing algorithm is based on a policy of recoloring the tokens as per some mathematical formula like

1. If Recipient wallet colored tokens  $Y < \text{Originating wallet Colored Tokens } X$   
Then, keep the color  $X$  of the total tokens in Recipient wallet  
Result - Possibility of genuine users getting affected who also have the same conditions
2. If Recipient wallet colored tokens  $Y \geq \text{Originating wallet Colored Tokens } X$   
Then, keep the color  $Y$  of the total tokens in Recipient wallet  
Result - Possibility of genuine users getting affected who also have the same conditions

Hence, this is also not an optimal solution

Thus as you see, we have a trilemma in our hand which means that we need to choose a combination of 2 algorithms that can switch at opportune times and help us fulfill below purposes

1. Exact Identification of Colored coins in hacker wallets
2. No genuine users should get affected
3. Complexity of the solution should be less and external help should be of limited nature

Based on above, the best solution will be a combination of below

1. Adoption of color tracing algorithm that the color will always update to a new unique colour
2. As the hack occurs, then based on the information received we will identify the wallets containing different colored tokens and then change the algorithm to the one where the color will always remain same on every transfer. This will help us not always keeping a tracking of new color tokens as the hacker transfer them across and instead limit the tracking part to a set of colors which will always remain same from that point onward on every transfer. This will help us take action on the colored tokens as per the decentralised governance policy decided and also share this colored tokens blacklist with different Dapps.

### 3. Color Tracing of fungible minted tokens with lossy mechanism

The solution relies on recoloring the fungible tokens as they move across blockchain. Smart contract will assign a unique metadata to the batch of tokens minted at the origin by some user. This metadata is the unique colour assigned to this batch of tokens. As tokens are transferred from one wallet to another wallet then the tokens are recolored by updating the metadata to a new metadata in the recipient wallet.

So, if user A has 100 tokens with Color A, User B has 50 tokens with Color B  
As User A transfers 50 tokens to User B, the total balance of User B increases to 100 tokens and all of these tokens are updated to a new color/metadata.

User A still holds 50 tokens with Color A and smart contract will map this updated colored token balances. In case User A has transferred entire tokens from their wallet to User B then the previous metadata will not be attributing to any tokens in circulation in the blockchain. The smart contract can then delete that metadata attribute after 24 hours.

At any point in time, whenever a token transfer occurs, then a metadata updation happens for the recipient wallet token, but the total number of colored attribution maintained in smart contract  $\leq$  total token mints circulation. Infact, The total color (metadata) attributes required to be stored by the smart contract will be way lesser than the total token mints in circulation because different wallets will be holding multiple tokens so the total metadata attributes to be tracked will be minor %age of the total tokens in circulation.

If some wallet transfers token in decimals to million of wallets then this increases the attribution so this case has to be handled. A way to handle above situation is that whenever someone transfers  $\leq$  \$1 token to a wallet then the smart contract will update the token metadata to a float metadata attribute. Any tokens across any wallet  $\leq$  \$1 are part of this float metadata attribute.

1. If A tokens out of total X tokens in user wallet A with Color G are sent to a User B wallet then smart contract debits A tokens from the total circulation of Color G from User A wallet balance map.
2. If recipient B is currently holding Y tokens of Color R of which the total circulation is also Y. In this case, the Color R tokens are reduced to 0 and the total Y tokens balance mapping in User B wallet are debited to 0. Their will be no credit of these Y tokens elsewhere because their is no metadata of these colored tokens existing now.
3. The earlier User B wallet tokens and new tokens added after transfer from User A is now recolored to a new color O whose circulation increases by  $(A + Y)$ .

Thus, through above mechanism, we are able to keep track of colored tokens as they move across wallets by forgetting the earlier metadata (after 24 hours) and recoloring to a new metadata. This helps ensure  $O(1)$  space complexity and the storage complexity reduces if the tokens are transferred to a pool like Liquidity Pool or to a vault address of a particular Defi application or to any exchange. As all the tokens are pooled in some application address with the same metadata.

We have implemented this Colored Token mechanism for a specific use case in Autonomint's \$USDa stablecoin project [5]. At Autonomint, users minting stablecoins against collateral are offered this ABOND tokens which are yield bearing assets and backed by deposited user collateral. The metadata/color associated with the originating mint of these ABOND tokens are

the cumulative rates applicable at that time.

Cumulative rates are the global variable rates which helps normalise or discount the user deposits to the genesis state. Anytime there is a %age change in deposited value due to yields accrued on the same then in order to reflect those yields back to user balances require a global cumulative rate mechanism. As different users arrive at different times with different deposit amounts so the accrual of yields can happen in 2 ways.

1. Looping over every deposit and applying the yield accrual logic as per the deposited amount and time gone by since deposit. This is a very costly operation in blockchain and not scalable as more users take positions in the application
2. Instead of above, a simple cumulative mechanism will just capture the yields accrued per sec in a global cumulative rate. Any user deposits can be normalized/discounted by dividing with this cumulative rate and anytime a user can check the total yields accrued by multiplying the normalized amount with this cumulative rate. This mechanism is widely used across defi applications.

We assign this cumulative rates as the metadata to any user minting ABOND tokens and mapping the user ABOND balance to a unique cumulative rate which makes it a Colored coin. Below is the process of recoloring of ABOND tokens.

1. User A transfers “a” ABOND token with Cumulative rate X/Color X from their wallet to User B who holds “b” ABOND tokens with Cumulative rate Y/Color Y
2. The recoloring mechanism identifies the new metadata by taking weighted averages of both the cumulative rates or colors.

So, New Color Z or Cumulative Rate  $Z = (a * X + b * Y) / (a + b)$

This new color Z is assigned to all the ABOND tokens in User B wallet and the updated color is mapped in the smart contract balances.

3. Whenever user wants to redeem the ABOND tokens, then we check the color or cumulative rate of ABOND tokens in the user wallet and accordingly does inhouse calculation on the amount of yield accrued.

So, above is another usecase where colored tokens can be utilised to attribute yields back to the users as per their participation with the stablecoin minting process. Higher the stablecoins minted then higher the ABOND tokens received with specific colors/cumulative rates and higher the



yields accrued to them. A similar mechanism will be applied to trace the color tokens in solving the token hack issue when hacked tokens are circulated across multiple wallets.

### **Enriching Metadata with relevant information**

The color (metadata) assigned on every new token batch mint or every token transfer should convey multiple data points without increasing the bit size.

The necessary information to be conveyed through metadata are

1. Timestamp of color origination
2. Order of transaction/transfer
3. Identification of pattern in metadata for proactive hacking analysis

The best way to present all the above information in a single number in a gas efficient manner is “Global Cumulative Rate”

The system relies on market participants to call the “Global cumulative rate” function rather than, say, automatically calling it by the protocol. The function will be called every time there is a token transfer or batch minting of tokens by a single user.

Process of calculation of metadata i.e. Global Cumulative Rate

1. Discretize time in 1-second intervals, starting from  $t_0$
2. Let the initial value of the “Global cumulative rate” be denoted by  $R_0$  at genesis of protocol
3. Select a percent change figure which needs to be discretized in 1-second intervals & Let the (per-second) rate at time  $t$  have value  $F_i$  (this generally takes the form  $1+x$ , where  $x$  is small)

Then the “Global Cumulative Rate”  $R$  at Time  $T$  is given by:

$$R(t) = R_0 \prod_{i=t_0+1}^t F_i = R_0 \cdot F_{t_0+1} \cdot F_{t_0+2} \cdot \dots \cdot F_{t-1} \cdot F_t$$

At  $t = 0$ , assume the following values:

$$\text{GCR (Metadata)} = 1 ; \text{Per-second rate} = f$$

In a block with  $t = 28$ , when a token transfer is made so a new color (metadata)

$$\text{GCR} = f^{28}$$

Thus, similarly we will calculating these metadata at every colored token event

#### 4. Use of Bloom Filtering

As mentioned above, the existing color tracing mechanism allows us to keep track of the attribution of particular metadata or colored tokens across the blockchain at a particular time but we are not able to keep track of the history of these transfers which is usually required to pinpoint the hacked tokens trail. The moment a hack occurs on a particular Dapp then the tokens are moved across multiple wallets with the metadata constantly updating across these transfers. To tackle above issue, we use Boom filters. Ethereum does not currently store token metadata in bloom filters within block headers. Ethereum's log bloom filters are used primarily for efficiently indexing and searching logs generated by contract events, not for tracking token metadata.

At its most basic level, a bloom filter is a large bit-string. Data to be added is hashed with a series of hash operations, whose outputs are interpreted as unsigned integers (modulo the number of bits in the bloom filter) are treated as indices for bits to be set [6]. To test if a value was previously added to the bloom filter, the same hash functions are run and those bit indices are checked. If any of the checked bits are not set, then the value was definitely not previously added to the bloom filter. If all of the bits are set, it is assumed the value was previously added, though the likelihood that it was depends heavily on the size of the bloom filter and the number of values that have been added.

Since the raw bloom filter data is transmitted between nodes as a part of the filterload message, it is important that all of the parameters are well-defined for a given bloom filter. In Bitcoin, the hash function earlier used was always 32-bit Murmur Hash version 3. The seed value used to initialize the Murmur hash is dependent on a tweak (or nonce), chosen at random by the creator of the bloom filter, and the number of hashes being performed (max of 50). For each hash of the input data, the seed is calculated as:  $\text{hash\_number} * 0\text{x}\text{FBA4C795} + \text{tweak}$ . So the seed for the first hash operation is  $0 * 0\text{x}\text{FBA4C795} + \text{tweak}$ , the second is  $1 * 0\text{x}\text{FBA4C795} + \text{tweak}$ , and so on, up to the number of hashes selected when the bloom filter was created.

In practice, the size of the bloom filter and the number of hash operations are calculated based on a target false-positive rate and an expected number of values to store. Given a target false-positive probability,  $P$ , and number of items to add,  $N$ , the corresponding bloom filter size,  $S$ , is calculated as  $(-1 / \text{pow}(\log(2), 2) * N * \log(P)) / 8$ . The number of hash operations is then calculated as  $S * 8 / N * \log(2)$ .







## Bloom Filters are also used in Ethereum

Events in the ethereum system must be easily searched for, so that applications can filter and display events, including historical ones, without undue overhead [7]. At the same time, storage space is expensive, so we don't want to store a lot of duplicate data - such as the list of transactions, and the logs they generate. The logs bloom filter exists to resolve this. When a block is generated or verified, the address of any logging contract, and all the indexed fields from the logs generated by executing those transactions are added to a bloom filter, which is included in the block header. The actual logs are not included in the block data, to save space. When an application wants to find all the log entries from a given contract, or with specific indexed fields (or both), the node can quickly scan over the header of each block, checking the bloom filter to see if it may contain relevant logs. If it does, the node re-executes the transactions from that block, regenerating the logs, and returning the relevant ones to the application.

### 5. Tracking history of colored tokens with Bloom filters

As mentioned earlier, the ultimate purpose of implementing colored coins is being able to know the location of colored coins at any moment by just their color. However, to make the colored coin algorithm work in a decentralized setting without the high complexity, will require a small component of tracking of history of colored coins. This will allow the Dapps to proactively handle the hacking situation.

Address	Tokens	Colour
0x34.....21	1000	1.014758494732
0x34.....21	2000	1.028004784299
0x34.....21	450	1.035628567789
0x34.....21	789	1.048937563920
0x34.....21	10000	1.089047880243
0x34.....21	1000000	1.09543860987

Colour	Bloom Filter
1.014758494732	
1.028004784299	
1.035628567789	
1.048937563920	
1.085628567789	
1.098937563920	

Currently the most optimal solution is

1. We start with a Color Tracing algorithm where color always updates on transfer as discussed above

Currently the Ethereum ERC20 smart contract keeps the track of token balances for users at any particular moment.

The smart contract also need to keep a track of color of tokens for users (addresses)

Every existing color will have a bloom filter to help the protocol identify if a particular element was part of a set in history.

2. Now, assuming some Dapp got hacked and tokens stolen from their address

All the tokens were of some particular color considering the address will have the same color of tokens and the same has been mapped to that address in our smart contract

The color of tokens stolen/hacked updates to a new color (metadata) as per the color tracing algorithm due to token transfer to hacker's wallet

3. The algorithm will have a conditionality logic for recipient wallet with no colored tokens

At first instance of interacting with recipient wallet with no colored tokens, the color(metadata) updates to a new color(metadata)

The Bloom filter mapped to new color, stores the previous color in the bit array.

Now, if these tokens are again transferred to a consecutive recipient wallet with no colored tokens then

—> The Bloom filter of the existing origin color will store the upcoming new color along with the previous color.

—> The Bloom filter of the new color after transfer will store the previous color

The new upcoming color is decided based on below condition

—> Current color Bloom filter has 1 color by checking the Bit arrays

If above is Yes, then it means the current wallet held no colored tokens before the transfer  
So, the upcoming color remains the same and no Bloom filter is mapped to this color ( or Bloom filter will be 0)

If above were No, then it meant that the previous wallet held some colored tokens

4. The algorithm has now updated so that the color remains consistent with every transfer. This makes it easier for us to identify hacked colored tokens, even if an attacker transfers them across multiple addresses. The decentralized governance policy can then take appropriate action on the identified colored tokens.

### Example Scenario

1. Hack Occurrence:
  - Dapp's blue-colored tokens are hacked and transferred to a hacker's wallet.
  - Tokens change color to red upon transfer.
  - The Bloom filter for red records that these tokens were previously blue.
2. Further Transfers:
  - Hacker transfers tokens to another wallet, changing the color to green.
  - The Bloom filter for green records that these tokens were previously red.
  - The Bloom filter for red already indicates that the tokens were blue.

Thus, we see here that the Bloom filters of every color mostly will be requiring to store previous color (metadata) and can also store upcoming color(metadata) during transfer. Also, if color remains same on transfer then there is no need to maintain a Bloom filter. However, for security purposes we will be able to maintain a Bloom filter for past 24-48 hrs of past colors i.e the ones with no tokens mapped to them as there are no existing addresses holding this colored tokens making the color extinct. These colors will be deleted after 24-48 hours.

The color tracking mechanism can be seen as having some similarities with hierarchical structure of Merkle Tree where the history of colors (as tracked by Bloom filters) forms a chain of transitions.

### Token Contract

The token contract handling the minting, transfer of these colored fungible tokens will have the below functions

1. Struct & Mappings
2. Token Transfer Function
3. Wrap Color
4. Update Bloom Filters function

## 5.. Bloom Filter check function

“Wrap Color” is invoked when the colour of the token needs to be updated to a new metadata while token transfer. Whenever a token transfer occurs then the smart contract needs to maintain a debit & credit of colored tokens.

### How do we use Bloom filters?

So, we need to track 2 things after the hack has occurred

1. Whether a particular color held a different color (metadata) of tokens in the previous transfers
2. Whether a particular set of colors held a different set of of color (metadata) in their previous transfers

So, that's where we use Bloom filters to probabilistically track the presence of specific metadata across. Bloom filters offer efficient membership checking with reduced on-chain storage requirements. Bloom filters are designed to be space-efficient probabilistic data structures that allow for fast membership tests [8]. They are particularly useful for applications where the potential for false positives is acceptable, as they never produce false negatives. However, In the context of token tracking and metadata, careful tuning of the Bloom filter parameters can minimize false positives to negligible levels.

### Algorithm to Maintain the Bloom Filter

1. Initialization:
  - Create a bit array of a predefined size  $m$  with all bits set to 0.
  - Select  $k$  independent hash functions.
2. Adding Previous colors (Metadata):
  - For each color, generate the metadata hash of the previous colors using the selected hash functions.
  - Update the bit array by setting the bits at the positions determined by the hash functions.
3. Transferring Tokens:
  - When tokens are transferred, update the new metadata in the Bloom filter of new color.
4. Querying Tokens:
  - To check if a specific color (metadata) is likely present, calculate the positions in the bit array using the hash functions.
  - Check if all the bits at these positions are set to 1

We identify the list of wallets when the Wallets that return a positive query result for `metadata_hack` are flagged.

Due to the nature of Bloom filters, there might be false positives, but no false negatives.

### How to minimize false positives?

To minimize false positives, we need to carefully choose the size of the Bloom filter (m) and the number of hash functions (k) [9]. The false positive rate (P) is given by:

$$P \approx (1 - e^{-kn/m})^k$$

where:

- n is the number of items inserted into the Bloom filter.
- m is the number of bits in the Bloom filter.
- k is the number of hash functions.

By adjusting m and k, we can achieve a desired false positive rate.

## 6. Practical Implementation in Token Tracking

For a decentralized colored token tracking system with potentially millions of colors and token transfers, we need to choose parameters that ensure a very low false positive rate. For instance:

- Assume we want to track up to 1 million unique metadata entries.
- We desire a false positive rate of less than 0.1%.

Using the formula for optimal k (number of hash functions):

$$k = \frac{m}{n} \ln(2)$$

For n = 1,000,000 and P = 0.001

$$m = - \frac{n \ln(P)}{(\ln(2))^2}$$

Plugging in the values

$$m = - \frac{1000000 \ln(0.001)}{(\ln(2))^2} \approx 14,377,586 \text{ bits} \approx 1.7 \text{ MB}$$

And for K

$$K \approx \frac{m}{n} \ln(2) \approx \frac{14377586}{1000000} \times \ln(2) \approx 10$$

The smart contract enforces the Bloom filter update by making it a part of the transfer function. The transfer will not be successful unless the Bloom filter of the receiving wallet colored tokens is updated with the new metadata

We need to use atleast a minimum 24-hour rolling window for metadata tracking to significantly reduce storage and computational overhead while maintaining the ability to respond quickly to hacking incidents. However, Since hacks can occur at any time, a fixed 24-hour period may not always be practical so it's good to keep a rolling window of 48 hours with previous 24 hours getting deleted after every 48 hours.

## 7. Setting the Rebasing policy for compromised tokens

Rebasing as a concept was first initiated by Ampleforth through their elastic supply mechanism [10] where tokens are designed with a variable supply which can be increased or decreased through positive or negative rebases. This is normally used to tackle price fluctuations for Ampleforth to create a price stable currency.

Rebasing has been now primarily used by staking derivatives protocols like Lido to share the yields back to the users holding their stETH token [11].

The mechanism used here is mainly allocating the increased value accrued to assets backing the rebasing token to users. This is done by tracking the shares of users in the contract.

The mechanism works in below ways

1. Submission of assets to be rebased - A user can send some token to the contract address and the same amount of positive rebased ERC20 tokens will be minted to the sender address.
2. Deposit of asset as per specific policy - User-submitted tokens are stored either in the buffer and can be later used for withdrawals or passed further to explore the specific usecase. So, in Lido's case the submitted ETH is either store in buffer or passed further to their Staking router contract to be used as validator's deposits for securing ETH consensus.
3. Redeem - The token might be redeemed back to token deposited through the protocol using some withdrawal contract.
4. Rebase - When an oracle report occurs, the supply of the token is increased or decreased algorithmically, based on the use case. In case of LIDO, when the oracle report occurs then the rebasing happens as per the staking rewards (or slashing penalties) on the Beacon Chain, execution layer rewards or fulfilled withdrawal requests A rebase happens when oracle reports beacon stats.



The rebasing mechanism is implemented via the "shares" concept. Instead of storing a map with account balances, The contract stores which share of the total pool is owned by the account. The balance of an account is calculated as follows:

$$\text{balanceOf}(\text{account}) = \text{shares}[\text{account}] * \text{totalPooledToken} / \text{totalShares}$$

- Shares - map of user account shares. Every time a user deposits token, it is converted to shares and added to the current user shares amount.
- totalShares - the sum of shares of all accounts in the shares map
- totalPooledToken - a sum of three types of token owned by protocol contract
  - buffered balance - ether stored on contract and hasn't been deposited or locked for withdrawals yet
  - transient balance - ether submitted to the official Deposit contract but not yet visible in the beacon state
  - beacon balance - the total amount of ether on validator accounts. In case of LIDO, This value is reported by oracles and makes the strongest impact on Lido stETH total supply change

Example of rebasing in Lido stETH

Total shares = 5

TotalPooledETH = 10

Shares of (Alice) = 1

Shares of (Bob) = 4

Thus, balance of (Alice) = 2

Balance of (Bob) = 8

On each rebase totalPooledEther normally increases, indicating that there were some rewards earned by validators, that ought to be distributed, so the user balance gets increased as well automatically, despite their shares remaining as they were. So, if the totalPooledEther were to increase due to rewards earned by validators on the ETH delegated to be staked to secure Ethereum consensus

totalPooledEther = 15 ETH

Here, the balance of user increases

balanceOf(Alice) = 3 tokens which corresponds to 3 ETH now

balanceOf(Bob) = 12 tokens which corresponds to 12 ETH now

But, the shares remain still

sharesOf(Alice) -> 1

sharesOf(Bob) -> 4

Above rebases usually occur every 24 hours through an oracle the oracle report, that usually (but not guaranteed) once a day provides the protocol with the data that can't be easily accessed on-chain, but is required for precise accounting.

### **Rebasing during Slashing**

There can occur a slashing penalty to the validator which has been delegated to stake the pooled ETH on behalf of the users. This slashing is incorporated in the following rebase as all the rebase operation does is comparing the previous balance to the current balance.

If slashing penalty of 10 ETH has occurred then

totalPooledEther = 5 ETH

Here, the balance of user decreases

balanceOf(Alice) = 1 tokens which corresponds to 1 ETH now

balanceOf(Bob) = 4 tokens which corresponds to 4 ETH now

But, the shares remain still

sharesOf(Alice) -> 1

sharesOf(Bob) -> 4

Depending on the protocol slashing policy, there can be staker compensation that can happen afterwards by burning stETH that is held by the DAO.

All of the current rebasing mechanisms are designed to affect every user balance equally. But with the color tracing mechanism combined with Bloom filters allow us to segregate the colored tokens. If some of those segregated colored tokens are originating from hacker's address then the protocol impose a slashing mechanism in rebasing to slash the balances of those specific addresses holding the colored tokens. The DAO policy will be created to slash the balances of these specific users holding the colored tokens to 0 and that will decrease the share of these users in total supply to 0. The DAO can then decide as per other policy to mint the tokens back to the users or protocol at the hack

## 8. Using AVS for decentralised consensus on Token provenance

As mentioned earlier, colors will be updating their Bloom filters whenever a colored token transfer is made to new wallet. These colors can just keep a record of previous color and upcoming new color in their Bloom filter. This alone will help us solve for token hack and identify the color of hacked tokens at any moment without relying on past history of colored tokens transfers. However, if we were to keep a record of long list of previous colors for any color then this will be requiring tracking of history of colored tokens on transfers. There might be a need for a global decentralised ledger to store all these metadata updates for all of the colors with decentralised consensus.

Ethereum does not currently store token metadata in bloom filters within their block headers. Ethereum's log bloom filters are used primarily for efficiently indexing and searching logs generated by contract events, not for tracking token metadata.

We need a separate transient ledger which can store all of these metadata updates across all colors in Block headers through Bloom filter mechanism. The data in this ledger can be of transient nature as any color update transfer history is mainly required for 24-48 hours for being taken action by governance policy. The feasible approach will be to use Actively Validated services (AVS) on EigenLayer [12] to manage this ledger with the strong decentralised consensus of Ethereum network.

Below are the 2 components needed

1. Token Contracts on Ethereum: These contracts handle the main logic of colored token transfers, minting, and rebasing, and communicate with the AVS for metadata management.
2. AVS on EigenLayer: The AVS will be responsible for maintaining global bloom filters and tracking token metadata. Validators on EigenLayer will secure and validate the AVS operations.

We will be utilising a hybrid approach where colors update their Bloom filters with the token metadata and AVS to maintain global Bloom filters.

Hybrid Approach: AVS and Wallets with Bloom Filters

1. Colors Maintaining Bloom Filters:
  - Each color maintains its own Bloom filter to track the previous metadata of the tokens it has received.
  - When tokens are transferred to a wallet, the metadata is updated and added to the color's Bloom filter.
2. AVS Maintaining Global Bloom Filters:

- The AVS maintains a global Bloom filter that aggregates metadata updates from all Colors on every block mined [13].
- The AVS uses these Bloom filters for efficient querying and validation

## **Token Provenance and History Tracking**

Track the history of each token to ensure accurate rebasing.

1. Token History Struct:
  - Store the complete history of each color in a struct.
  - Include details such as origin, previous holders, and transfers.
2. Transaction Logging:
  - Log each transaction, including the sender, receiver, amount, metadata, and unique token identifiers.
3. History-Based Rebase:
  - Use the color history to identify compromised tokens.
  - Rebase only those tokens that can be traced back to the hack.

For efficient data management, The AVS maintains metadata for the last 24 hours by keeping track of the block numbers within this time frame. We can use a dynamic tracking window since the hacks can occur at any time or the hack can be reported or identified with a delayed time period. For this, we will be keeping a 48 hour tracking window. The 24-hour tracking window begins once a hack is detected. At regular intervals, the AVS removes metadata older than 48 hours to ensure the storage remains efficient.

As metadata updates occur, they are added to the Bloom filter associated with the current block or transaction batch. We can then trace back the origin of colored tokens till the past 48 hours through their metadata lineage.

Also, using Bloom filters in Block headers enable selective querying. During a rebase operation or hack investigation, specific Bloom filters can be queried to quickly identify which metadata entries (and thus which tokens) are affected.

## **Using decentralised consensus to manage attackers from discerning Bloom filter patterns**

There is a possibility that by observing the AVS and the structure of metadata, an attacker might attempt to infer patterns or generate metadata that could pollute the Bloom filter by introducing false positives [14].

This will be managed through a decentralised verification mechanism where we distribute the verification process across multiple AVS nodes. Each node maintains its own Bloom filter, and

the final validation is based on a consensus among nodes. We will be using multiple Bloom filters with different salt values (random seeds) for hashing. This reduces the likelihood of a single false positive affecting multiple checks. Introduce a salt to the metadata before hashing. The salt can be periodically changed, making it harder for an attacker to predict or recreate the hash values.

Here is the process

**Generate Salt:** Each time metadata is added, a salt value is generated and concatenated with the metadata before hashing.

**Hashing with Salt:** Hash the concatenated value (metadata + salt) and update the Bloom filter.

**Storing Salt:** The salt value needs to be stored or derivable so that the same salt can be used for checking the metadata.

We can explore some other mitigation strategies as well

**Hash Function Variability:**

- Use a diverse set of hash functions to further randomize the Bloom filter. This makes it harder for an attacker to craft inputs that generate false positives.

**Threshold-based Validation:**

- Implement a threshold mechanism where a token metadata must match in multiple Bloom filters before being considered valid. This adds a layer of redundancy.

**Metadata Encryption:**

- Encrypt the metadata before storing it in the Bloom filter. This adds an additional layer of security, making it harder for an attacker to infer patterns or generate false metadata

## Conclusion

We have proposed a colored token solution for solving token hacks from protocols. A combination of different color tracing algorithms are combined to identify the compromised tokens residing inside hacker wallets. We utilise the application of Bloom filters to add the capability of maintaining color update transfer for every colour in a minimum storage structure. The identified compromised colored tokens are not able to get transferred across wallets due to smart contract transfer restrictions placed on them. These colored tokens are then rebased to 0 through slashing their share in Rebasing contract. The decision to slash to 0 is taken by decentralised governance policy committee. We talk about using the application of Actively Validated Services (AVS) on EigenLayer to keep a record of long list of previous colors for any color in a decentralised ledger and have a decentralised consensus on their provenance through rehypothecation of trust enabled by underlying Ethereum network.

## Reference:

- [1] <https://de.fi/rekt-database>
- [2] <https://go.chainalysis.com/crypto-crime-2024.html>
- [3] <https://hackernoon.com/the-role-of-data-destruction-in-cybersecurity>
- [4] <https://allquantor.at/blockchainbib/pdf/rosenfeld2012overview.pdf>
- [5] <https://docs.autonomint.com/autonomint>
- [6] <https://reference.cash/protocol/spv/bloom-filter>
- [7] <https://www.npmjs.com/package/ethereum-bloom-filters>
- [8] <https://docs.infura.io/api/networks/ethereum/concepts/filters-and-subscriptions>
- [9] <https://medium.com/@naterush1997/eth-goes-bloom-filling-up-ethereums-bloom-filters-68d4ce237009>
- [10] <https://docs.ampleforth.org/>
- [11] <https://docs.lido.fi/contracts/lido/#rebasing>
- [12] <https://docs.eigenlayer.xyz/eigenlayer/avs-guides/how-to-build-an-avs>
- [13] <https://github.com/rsksmart/RSKIPs/blob/master/IPs/RSKIP45.md>
- [14] <https://onlinelibrary.wiley.com/doi/10.1155/2021/2067137>