**Blacklist Solutions**

Initially the blacklist was to be stored in the token account, but this has the following drawbacks:

* This can result in database bloat if many accounts are blacklisted.
* Nodes may have to scan through many entries to confirm whether a particular account is or isn’t blacklisted.

One solution is to simply use a single bit in the token entry stored within an individual account to indicate whether this account is blacklisted for a particular token. However, a token account controller may want to blacklist accounts that have not yet been tied to the token (by being whitelisted or by receiving tokens). In the case of a token account with whitelisting enabled, ‘blacklisting’ untied accounts can be done simply by rejecting their applications for whitelisting. However, token accounts that do not have whitelisting enabled will be unable to effectively blacklist untied accounts since any account can receive such tokens without first being approved for whitelisting. One solution is to allow controllers of token accounts with whitelisting disabled to blacklist any account by creating an entry in that account (the same entry that would be created if this account had a balance for this particular token), and setting the bit for blacklisting. However, this can be exploited by a malicious controller who spams the network by blacklisting every account, permanently increasing the size of each account. To mitigate this, the traditional blacklist will be used for token accounts with whitelisting disabled.

This means that nodes will not need to pull the token account’s blacklist from the database for transactions between accounts that already have the relevant token entries, since the blacklist information will be contained within the individual account. Accounts with whitelisting disabled to not benefit from this optimization since the blacklist for these accounts will be centralized. Additionally, accounts with whitelisting disabled will have to check both the centralized blacklist as well as the blacklist bit in the individual account, since it’s possible for a token account to toggle the whitelist setting from true to false, and accounts that were blacklisted during the prior phase will be identified by their bitfield rather than the central blacklist.

An alternative solution is to treat whitelisting and blacklisting as a combined feature, whereby only two configurations of these options is allowed: Either a token account enables both whitelisting and blacklisting, or disables both. In other words it would not be possible to enable whitelisting and disable blacklisting or vice versa. This would be a conceptually simpler approach, but it requires a since now we could eliminate the need

so that when we want to blacklist accounts that dont have an entry for a particular token, we can do so without updating the individual account which could be abused to cause bloat

However, token accounts with whitelisting can eschew the need to blacklist accounts that do not yet have an entry for the token by simply rejecting such an accounts application for whitelisting.

So now the problem is how to handle token accounts that do not have whitelisting. Since any account can receive this token without first being whitelisted, we need a way to prevent unregistered accounts for these types of tokens from receiving tokens. The solution is to use a traditional blacklist stored inside of the token account.