IceBlock Whitepaper

IceBlock Team
v1.1

April 2021

Abstract

IceBlock introduces a block-based timelock on selling tokens from an address that recently bought or sold. With the recent surge in low-liquidity fair launches, many projects have experienced frustration of the ability of early investor whales to completely crash the price of the token when the decide to cash out. By introducing this timelock, whales will be forced to slowly sell their holdings over a longer period of time.

1 Introduction

Recently there has been a surge in deflationary currencies, growing rapidly since the success of projects such as SafeMoon, PooCoin, and others. Many have tried to follow in their success, but with a significant issue. Early investors in these projects quickly make huge returns, often on the magnitude of 10x-100x. This leads to early investors selling large quantities of coins soon after project launch, crashing the token into obscurity before it even has a chance to mature.

IceBlock remedies this, by introducing Freeze, a blockchain based timelock that prevents addresses from selling too quickly. By limiting selling to say, 0.1% of supply per day per address, early investors are forced to slowly cash out their positions. This prevents the price from being crashed by whale addresses, and provides smaller addresses with a more equal standing in the market.

This is particularly important with the rise of low-liquidity fair launches, where the initial listing of a token in a liquidity pool may only be paired with a few hundred dollars, implying a tiny marketcap. This is advantageous to developers, as they will not require to burn as much capital upon launch. The downside of this however, is that keen individuals can build up large positions in the token rather quickly and easily, giving them large control over the token price, known as "whales".

By introducing Freeze, while these whales may still exist, they will be on an equal playing field to smaller addresses, and will have a lessened ability to manipulate price.

2 Freeze in Iceblock

With the IceBlock token, a freeze 24 hour freeze will be applied to your address after you buy or sell. In the IceBlock protocol, you will only be prevented from selling while frozen, but can continue to sell. This adds another level of bullish tokenomics where there will be a constrained supply depending on the number of token holders who are currently unfrozen, matched with an unbounded demand.

2.1 Multiple wallets and why they are no concern

2.2 Technical Implementation

IceBlock is based around the PooCoin smart contract, which allows us to harness existing transaction size limits to 0.1% of supply.

We introduce a new variable, $uint256\ \_boughtBlock[address]$ which tracks the block at which each address most recently bought or sold:

```
1 // Check if spender is still frozen and is not excluded from
     freezing. (uniswap contract)
2 if(_blocksSinceBuy(sender) < _blockFreeze && !_isExcluded[sender])
     {
3     require(_blocksSinceBuy(sender) >= _blockFreeze, "Call
          _blocksSinceBuy() on BSCSCAN! Transfer attempt before
          blockfreeze timeout.");
4 } else { // If they are allowed to spend
5     // Freeze or refreeze buyer if they are not excluded
6     if (!_isExcluded[recipient]) {
7         _refreeze(sender, recipient, amount);
8     }
9 }
```

2.3 Refreezing: Buying Iceblock whilst Frozen

To make things more interesting, IceBlock implements a "refreezing" mechanic. If you are frozen and receive some more IceBlock (ICE) through a transaction (e.g. purchase), then you will lose 25% of the progress you had made towards being unfrozen.

$$\_boughtBlock[recipient] = \_boughtBlock[recipient] + 3 * \_blocksSinceBuy(recipient)/4; \tag{1}$$

For example, if I bought some ICE and waited 8 hours, and then bought again. I would keep 75% of my 8 hours of unfreezing progress, which is 6 hours. This means that instead of being 8 hours into a 24 hour freeze, I would now be set back to being 6 hours into a 24 hour freeze.

This will make almost no difference to those who are making multiple buys in quick succession, as they are only losing negligible progress. Whereas someone who buys closer to the end of their freeze will lose a few hours progress.

It is important to note that the choice of **losing** 25% **of progress** is an important one. It ensures that **freezes are never longer than 24 hours since your last buy or sell**. This would not be the case had we implemented a 25% increase in freeze time after a buy or sell.

3 IceBlock Predictions & Prospect

IceBlock will pioneer improvements in low liquidity deflationary currencies, increasing their lifespan and perhaps allowing for longer term burning and taxes to take effect. Iceblock strives to lesser the significance of whales and to create a more egalitarian token where small hodlers have the same power as the large.

We also introduce a novel random burn or fee, as opposed to a fixed split between the two. There is not much behind this, particularly with low transaction limits it will not affect the token supply significantly. Other than it being a bit of fun!

```
1  // ICEBLOCK V1
2  //    .-------,
3  //  ,'-------,'|
4  // |       //| |
5  // |         | |
6  // |  //     | ;
7  // |--------|'.-
8  //    '.-        ';
9  //         '-.-.'
10
11 function _getTValues(uint256 tAmount, uint256 taxFee, uint256
       burnFee) public view
12                                              returns (uint256, uint256,
                                                    uint256, uint256) {
13     uint256 tFee = 0;
14     uint256 tBurn = 0;
15     uint256 tBurnOrFee = 0;
16
17     if (tAmount > 1000 * 10 ** uint256(_decimals)) {
18         tFee = 0;
19         tBurn = 0;
20     } else {
21
22         // Combine both fees and randomly make them a fee or burn
23         tBurnOrFee = ((tAmount.mul(burnFee+taxFee)).div(100)).div
               (100);
24
25         // Odd or even?
26         if (random(tAmount) % 2 == 0) {
27             tFee = tBurnOrFee;
28         } else {
29             tBurn = tBurnOrFee;
30         }
31     }
32     uint256 tTransferAmount =  tAmount.sub(tFee).sub(tBurn);
33     // return (tTransferAmount, tFee, tBurn);
```

```
34          return (tTransferAmount, tFee, tBurn, tBurnOrFee);
```