

A new proposal of distributing tokens on blockchain: Proof of Mint

Fair Launch Labs(F.L.L.)

Nov.6, 2024

Abstract

The concept of Fair Mint (or Fair Launch) has gained significant traction within blockchain communities, yet it faces critical issues such as Sybil attacks, insufficient time for building community consensus, fraud, and lack of market value management.

This paper introduces a new solution, Proof of Mint (PoM), inspired by Bitcoin's mining difficulty mechanism. PoM aims to address the fairness concerns by transforming hash power into minting participation levels, thereby mitigating the impact of Sybil attacks and providing ample time for community consensus building.

The proposed PoM is designed to approach ensures a stable minting process, discourages speculative and cheating behavior, incentivizes real user. The paper provides a detailed analysis of the PoM mechanism, including core formulas for calculating the difficulty coefficient, and mint size per Epoch etc.

The paper also presents simulation test parameters and results, showcasing the PoM's effectiveness maintaining a stable minting curve. Additionally, it discusses the total supply calculation based on Eras, Epochs, and reduction factors, as well as the estimated total minting time.

The paper proposes a significant advancement in the Fair Mint paradigm, offering a more equitable and community-driven approach to token minting and distribution. The PoM has the potential to reshape the landscape of decentralized finance by ensuring a fairer and more sustainable minting process.

1- Issues

In the past two years, the Fair Mint (or Fair Launch) in the blockchain communities has become extremely popular. Numerous tokens have quickly completed the minting process and were subsequently listed on decentralized exchanges for trading. However, their price trends often exhibit a common pattern: after an initial rapid increase, the prices continue to decline incessantly.

Over two years of observation and research on various Fair Mint platforms and projects, we have identified the following issues:

1.1 - Sybil Attack

According to relevant data, the proportion of real participants of such Fair Mint games is as high as 90%. However, it is surprising that over 95% of the tokens are actually minted by a minority of individuals proficient in blockchain technology. Consequently, the vast majority of ordinary real users can only obtain a very small amount of tokens.

This technical method is commonly known as "Sybil Attack", which robs Fair Mint of its "fairness." A few individuals control a large number of tokens obtained at almost zero cost, and they manipulate the market by rapidly driving up the token prices and then selling them out at high prices to reap substantial profits.

1.2 - Insufficient time to build consensus

The communities have insufficient time to build consensus. Due to the rapid speed of token minting, the price often begins to decline while the community is still in the process of formation, leading to the disintegration of community consensus and ultimately the dissolution of the community.

1.3 - Fraud

Some project parties participate in minting through technical means themselves, creating a false impression of market “excitement” and obtaining low-cost tokens, manipulating the market to gain huge profits, turning Fair Mint into a tool for fraud.

1.4 - Lack of Market Value Management(MVM)

There is a lack of effective market value management measures. Given the exceptionally fast minting speed and the fact that usually 100% of the tokens are directly (or after the completion of minting) put into circulation, market value management becomes crucial. However, the “ownerless” feature of tokens associated with Fair Mint requires the community to spontaneously organize MVM. Unfortunately, due to the rapid minting speed, MVM often cannot be implemented in time before the collapse of the token price.

1.5 - Does liquidity management workable?

Some Fair Mint platforms have introduced liquidity management mechanisms, such as charging a certain fee during minting, which is locked in a smart contract and automatically added to the liquidity pool of decentralized exchanges after certain conditions are met (such as reaching a target amount or completing all minting).

However, we have observed that in the initial rapid price increase of these projects, the funds in the liquidity pool account for a very low proportion of the total liquidity pool; and when the price falls, the funds in the liquidity pool are far from sufficient to withstand the sell-off.

1.6 - Does Time Locker workable?

Some smart contracts introduce Time Locker to prevent Sybil-attacks, which can indeed prevent batch minting using the same account, but they still cannot prevent those who implement batch minting using different accounts.

1.7 - Mempool and MEV

The mainstream blockchains are mostly based on Ethereum EVM, and the mempool of EVM allows a large number of Bots to “peek” at users’ intentions before the new block is packed and take the lead in on-chain actions, such as minting and trading. We have observed that there is a large amount of such behavior on both Ethereum (with a block packing interval of about 12 seconds) and Layer 2 networks like Arbitrum (with a block packaging interval of about 0.25 seconds). This type of behavior is well known as: **Maximal Extractable Value** (MEV). It is estimated that since January 1, 2020, the total amount of MEV has exceeded 730 million US dollars, most of which has flowed to robots and miners. The existence of MEV is technically driven and, although understandable, it greatly affects the fairness of on-chain actions.

We believe that effectively preventing technical means such as Sybil attacks and giving sufficient time for the establishment of community consensus have become the most serious challenges. Although many Fair Mint platforms have realized the seriousness of this issue and have tried to take corresponding measures to prevent various types of cheating, the actual effects achieved are quite limited.

To effectively prevent Sybil attacks, some platforms or projects have adopted tech such as KYC authentication or relying on servers controlled by the project team to provide authorized signatures requiring minters to have this signature to mint. However, these methods are too centralized and therefore difficult to be accepted widely and supported from the community.

2- Proposal

We believe that, given the current blockchain technology, it is very difficult to completely eliminate these issues. However, some new mechanisms can alleviate these problems and enhance the fairness of Fair Mint and build better community.

This paper proposes the following solutions:

2.1 - The Difficulty Mechanism of Bitcoin Mining

This proposal draws on the difficulty mechanism of Bitcoin mining, so before introducing this proposal, let's briefly introduce the Bitcoin mining mechanism.

The difficulty mechanism of Bitcoin mining is a core component of the Bitcoin network. It ensures that the minting progress is maintained at a stable rate of approximately one block every 10 minutes on a decentralized network. This mechanism is designed to adapt to changes in the network hash power at different times, with the aim of keeping the generation speed of new blocks relatively stable regardless of how the nodes joining the network.

The adjustment of Bitcoin mining difficulty is achieved through an automatic algorithm that adjusts the difficulty approximately every two weeks (2016 blocks). This cycle is designed based on the target block generation time of the Bitcoin network, which is one block every 10 minutes. If the average generation time of new blocks in the past 2016 blocks is less than 10 minutes, the difficulty will increase to ensure that the future block generation time returns back to 10 minutes. Conversely, if the block generation time is more than 10 minutes on average, the difficulty will decrease.

The specific calculation of the difficulty is based on the comparison between the generation time of the previous 2016 blocks and the target time (20160 minutes, i.e., two weeks). The difficulty adjustment formula increases or decreases the difficulty based on the ratio of actual time to target time. This adjustment is continuous to ensure that the Bitcoin network can adapt to the constantly changing hash power, whether miners joining or leaving, or high hash power mining hardware involving.

Furthermore, the adjustment of difficulty is also subject to a maximum change limit, that is, the difficulty cannot exceed 4 times the previous difficulty in any single adjustment. This is to prevent sudden large fluctuations in difficulty from impacting the network.

As the Bitcoin network develops, the mining difficulty continues to increase, mainly due to the growing hash power participating in mining. As more and more miners join the network and mining hardware technology improves, the total computing power for mining continues to rise, which increases the difficulty for individual miners or mining pools to discover new blocks. Therefore, the increase in mining difficulty reflects the growth of hash power in the Bitcoin network and also ensures that the issuance rate of Bitcoin remains stable.

The Bitcoin mining difficulty mechanism is key to the adaptability and stability of the Bitcoin network. It ensures that the generation rate of Bitcoin blocks remains at the designed target level by dynamically adjusting the difficulty value, while adapting to changes in network hash power.

```
1  /* Calculate the difficulty for a given block index.  
2  */
```

```

3 double GetDifficulty(const CBlockIndex& blockindex)
4 {
5     int nShift = (blockindex.nBits >> 24) & 0xff;
6     double dDiff =
7         (double)0x0000ffff / (double)(blockindex.nBits & 0x00ffffff);
8
9     while (nShift < 29)
10    {
11        dDiff *= 256.0;
12        nShift++;
13    }
14    while (nShift > 29)
15    {
16        dDiff /= 256.0;
17        nShift--;
18    }
19
20    return dDiff;
21 }

```

Key code of aboved difficulty rules in bitcoin code(c++ version). Bitcoin github repo: Source code

```

1 unsigned int GetNextWorkRequired(const CBlockIndex* pindexLast, const CBlockHeader *pblock, const Consensus
2 {
3     assert(pindexLast != nullptr);
4     unsigned int nProofOfWorkLimit = UintToArith256(params.powLimit).GetCompact();
5
6     // Only change once per difficulty adjustment interval
7     if ((pindexLast->nHeight+1) % params.DifficultyAdjustmentInterval() != 0)
8     {
9         if (params.fPowAllowMinDifficultyBlocks)
10        {
11            // Special difficulty rule for testnet:
12            // If the new block's timestamp is more than 2* 10 minutes
13            // then allow mining of a min-difficulty block.
14            if (pblock->GetBlockTime() > pindexLast->GetBlockTime() + params.nPowTargetSpacing*2)
15                return nProofOfWorkLimit;
16            else
17            {
18                // Return the last non-special-min-difficulty-rules-block
19                const CBlockIndex* pindex = pindexLast;
20                while (pindex->pprev && pindex->nHeight % params.DifficultyAdjustmentInterval() != 0 &&
21                    pindex = pindex->pprev;
22                return pindex->nBits;
23            }
24        }
25        return pindexLast->nBits;
26    }
27
28    // Go back by what we want to be 14 days worth of blocks
29    int nHeightFirst = pindexLast->nHeight - (params.DifficultyAdjustmentInterval()-1);
30    assert(nHeightFirst >= 0);

```

```

31     const CBlockIndex* pindexFirst = pindexLast->GetAncestor(nHeightFirst);
32     assert(pindexFirst);
33
34     return CalculateNextWorkRequired(pindexLast, pindexFirst->GetBlockTime(), params);
35 }

```

Bitcoin github repo: Source code

2.2- Proof of Mint(PoM)

If we replace Bitcoin's hash calculations with the act of minting, effectively transforming hash power into the level of participation in minting, we arrive at our proposed solution, known as **Proof of Mint**, or **PoM** for short.

Assuming we deploy **PoM** on Ethereum, which has updated to a **Proof of Stake (PoS)** consensus with an average block time of approximately **12 seconds**. To achieve stability in the minting process, we have designed the following plan:

- The entire minting progress is divided into several **Eras**, each **Era** containing several **Epochs**.
- Each **Epoch** has a difficulty-coefficient (the initial **Epoch**'s difficulty-coefficient is **1**), which is determined by the actual running time of the previous **Epoch** and determines the current block's mint-size.
- The **target-mint-size-of-epoch** and **base-mint-size-of-epoch** in each **Epoch** within an **Era** decrease according to a fixed **reduction-factor**.
- Each minting requires a fixed fee, which does not change with the decreasing mint-size of **Era** and **Epoch**.

2.3- Example:

In the 1st **Era**, the **target-mint-size-of-epoch** is **100,000 tokens**, the **base-mint-size-of-epoch** is **100 tokens**, the **target-mint-time-per-epoch** is **10 minutes**, and the minting fee is **0.1 ETH**.

On the 10th **Epoch**, if **100,000 tokens** were minted in 3 minutes with a **difficulty-coefficient** of **1.5**, then the 11th **Epoch**'s **difficulty-coefficient** will automatically increase to **1.5105**, and the **mint-size-per-minting** will automatically decrease to $100 / 1.5105 = 66.203243959$ tokens, with the cost per token to **0.1ETH / 66.203243959 tokens = 0.0015105 ETH**.

The increase in cost and decrease in mint-size can affect the enthusiasm for participation in minting.

Affected by the decrease in mint-size and the decline in enthusiasm, assume that completing the 11th **Epoch** takes **600 minutes** (10 hours), then the 12th **Epoch**'s **difficulty-coefficient** will keep same as 12th **Epoch**'s **difficulty-coefficient**, and the **mint-size-per-minting** and the cost per token are same as the 11th **Epoch**.

Then, the 13th **Epoch**'s minting time change to be **5 minutes**, and the **difficulty-coefficient** will automatically increase to **1.5180525**, and the **mint-size-per-minting** will decrease to $100 / 1.5180525 = 65.873874586$ tokens, with the cost per token to **0.1ETH / 65.873874586 tokens = 0.0015180525 ETH**, which is 0.5% higher than the 12th **Epoch**.

Additionally, the **base-mint-size-of-epoch** decreases progressively. Assuming a **reduction-factor** of **3/4**:

- 1st **Era**, the **target-mint-size-of-epoch** is **100,000 tokens**, and the **base-mint-size-of-epoch** is **100 tokens**.

- 2nd Era, the **target-mint-size-of-epoch** is $100,000 * 3/4 = 75,000$ tokens, and the **base-mint-size-of-epoch** is 75 tokens.
- 3rd Era, the **target-mint-size-of-epoch** is $75,000 * 3/4 = 56,250$ tokens, and the **base-mint-size-of-epoch** is 56.25 tokens.
- ...

2.4- Benefits

2.4.1- Benefit 1 If Bots participate in the minting, they will find that the faster the minting speed, the fewer tokens they receive, and the higher the cost. Contrary to many methods that try to prevent Bots from participating minting, this proposal does not prevent Bots from batch minting. However, the high cost and low yield will stop Bots.

2.4.2- Benefit 2 If Bots monitor the minting time of the previous Epoch to calculate the difficulty of the next Epoch and valuate whether to participate or not, then all Bots must have their own strategies. Otherwise, convergent strategies will lead to all Bots crowding and causing a significant decrease in yield and a substantial increase in costs. Because the yield of Bots do not depend on the “speed” of minting, but more on the “guesswork” of the behavior of other Bots, greatly increasing the strategic difficulty for Bots.

2.4.3- Benefit 3 Adopting a mechanism where difficulty only increases and never decreases ensures that the expected mining cost is always on the rise. When the difficulty increases rapidly, users will not anticipate a decrease in difficulty, either continuing to mint or stopping minting. This avoids the halt in minting that can occur when waiting for the difficulty to drop.

2.4.4- Benefit 4 As the difficulty and mining cost increases, until the market deems the cost to have reached a reasonable level, at which point minting will slow down or stop. The mining cost at this time will be an Anchor of the market price of tokens.

2.4.5- Benefit 5 The funds collected from mining are used for the liquidity pool of decentralized exchange. This proposal avoids the issue of some platforms in the past charging a fixed minting fee that was far from sufficient for the market’s liquidity needs. As the difficulty increases, the output per minting will decrease, and the number of minting to complete the target mint size of epoch will increase, thereby raising the minting fee. The increased minting fees are directed into the liquidity pool, providing ample liquidity support for Market Value Management.

2.4.6- Benefit 6 When market prices fall, people will find that minting is no longer cost-effective, leading to a slowdown or cessation of minting activities. The addition of low-cost supply will decrease or stop, avoiding the “death spiral” trap of falling prices coupled with a continuous increase in tokens supply. If new miners come in, the difficulty and the cost will rise again.

2.4.7- Benefit 7 The cost of minting and the level of difficulty are entirely dependent on market participation, with no centralized control, achieving a dynamic balance in a decentralized environment.

This mechanism effectively incentivizes early participants and is very friendly to community members who keep attention, while being unfriendly to speculators and those who cheat through technical tricks.

We have coded PoM mining algorithm on the Ethereum testnet `sepolia(solidity)` and the `Solana(rust/anchor)` Devnet, and will provide front-end `Typescript` scripts for community testing, while also open-sourcing `python` language simulation code.

3. Calculations

3.1 - Formulas

3.1.1 - Difficulty Coefficient of Current Epoch

- d : Difficulty coefficient
- d' : Difficulty coefficient of previous epoch
- δd : Change of difficulty coefficient
- N_e : Elapsed Blocks numbers Of passed Epoch
- N_t : Target Number Of Blocks Per Epoch

The change of difficulty coefficient is calculated based on the ratio of actual time to target time. We only consider the elapsed blocks are higher than the target number of blocks per epoch. The formula is as follows:

$$\delta d = \frac{1 - \frac{N_e}{N_t}}{100}, (N_e < N_t) \quad (1)$$

$$\delta d = 0, (N_e \geq N_t) \quad (2)$$

100 is a factor used to control the proportion of difficulty increase within a certain threshold range. Setting this value to 100 means that the maximum rate of difficulty increase is 1%. If this value is set to 50, then the maximum rate of difficulty increase is 2%.

The difficulty coefficient of current epoch is:

$$d = d' * (1 + \delta d) \quad (3)$$

For blockchains that can accurately obtain block timestamps (such as `Solana`), the above `number of blocks` can be replaced with `timestamp`.

Example:

In the example above, the target minting time of each Epoch is 10 minutes. On the 10th Epoch, minting took 3 minutes with a difficulty-coefficient of 1.5, then the new difficulty-coefficient for the 11th Epoch will be adjusted to: $1.5 * ((1 - 3 / 10) / 100 + 1) = 1.5105$.

And On the 11th Epoch, minting took 600 minutes which is longer than the target minting time of 10 minutes, then keep the same difficulty-coefficient of 1.5105 as 10th Epoch.

3.1.2 - Base Mint size per Minting of the Current Era(Mb)

- M_b : Base mint size per minting of current Era
- M_0 : Base mint size per minting of the Genesis Era
- f : Reduction factor
- e : Current era

$$M_b = M_0 * f^{e-1} \quad (4)$$

Reduction factor

- $f=0.5$ means halving each Era
- $f=2/3$ means a 33.33% reduction each Era
- $f=3/4$ means a 25% reduction each Era
- $f=4/5$ means a 20% reduction each Era
- $f=5/6$ means a 1/6 reduction each Era. etc.

3.1.3 - Target Mint Size per Epoch of current Era

- T : Target mint size per Epoch of the current Era
- T_0 : Target mint size per Epoch of the Genesis Era
- f : Reduction factor
- e : Current era

$$T = T_0 * f^{e-1} \quad (5)$$

3.1.4 - Mint Size per Minting of current Epoch

- M : Mint size per Minting of current Epoch
- M_b : Base mint size per minting of current era
- d : Difficulty coefficient

$$M = \frac{M_b}{d} \quad (6)$$

Example:

In the example above, the base mint size per minting of the current Era is 100 tokens, and the difficulty-coefficient is 2.55, then the minting reward will be: $100 / 2.55 = 39.216$ tokens.

3.1.5 If the T is not an integer multiple of M , adjustments need to be made to the M , that is:

$$M_a = \frac{T}{\lfloor \frac{T}{M} \rfloor + 1}, (T \nmid M) \quad (7)$$

If the T is an integer multiple of the M , no adjustments are needed as described above.

$$M_a = M, (T \mid M) \quad (8)$$

Example If d (difficulty-coefficient) is 1.69, T is 10,000 tokens, and the M_b is 525 tokens. The M would be $525 / 1.69 = 310.650887574$ tokens(see.3.1.4). Based on this mint size, the number of minting instances would be $10,000 / 310.650887574 = 32.19$ times. Since there is a fractional number of instances, we need to round down 32.19 to 32 and add 1, making it 33 times. Consequently, the adjusted mint size per minting(M_a) is: $10,000 / 33 = 303.03$ tokens.

3.1.6 - Token cost Although the cost per minting remains unchanged, as the difficulty increases, the number of Tokens obtained per minting will decrease, so the cost of the Tokens will increase.

Here is how to calculate the cost of token.

- P_0 : Minting fee
- p : Token cost

$$p = \frac{P_0}{M_a} \quad (9)$$

If the T is not an integer multiple of M , price will be:

$$p = \frac{P_0 * (\lfloor \frac{T_0}{M_0} * d \rfloor + 1)}{T_0 * f^{e-1}}, (T \nmid M) \quad (10)$$

$$p = \frac{P_0}{M_0 * f^{e-1}} * d, (T \mid M) \quad (11)$$

As P_0 , T_0 , M_0 , f and e are constant in an era, we know that: $p \propto d$.

The token cost depends on the difficulty coefficient, which varies with the level of participation, more minting, higher difficulty, more supply and higher increasement of cost, less minting, lower difficulty, less supply and lower increasement of cost. Thereby forming a community-driven dynamic balancing mechanism.

3.2 - Explanation

3.2.1 T and M are decreasing exponentially with each **Era**, but the ratio between the two remains constant, regardless of the **Era**.

$$T = T_0 * f^{e-1} \quad (12)$$

$$M = \frac{M_b}{d} = \frac{M_0 * f^{e-1}}{d} \quad (13)$$

$$\frac{T}{M} = \frac{T_0}{M_0} * d \quad (14)$$

3.2.2 If the N_t is set to 10 minutes, then the theoretical target interval time for each minting is: 600 seconds / 33 = 18.18 seconds per instance.

If the interval time is longer, it means that completing all mints in an **Epoch** takes longer than planned (10 minutes), which implies fewer miners, and in this case, the difficulty decreases, and the reward per minting increases.

Conversely, if the interval time is shorter, it indicates that completing an **Epoch** of mining takes less time than planned (10 minutes), which implies more miners, and in this case, the difficulty increases, and the reward per minting decreases.

3.3- Key code for calculating mint size of epoch (rust)

```

1 fn get_mint_size(config_data: &ConfigData, era: u32, elapsed_seconds_epoch: i64, previous_difficulty_coef
2   let delta_difficulty_coefficient = if (elapsed_seconds_epoch as f64) < (config_data.target_seconds_per
3     (1.0 - (elapsed_seconds_epoch as f64) / (config_data.target_seconds_per_epoch as f64)) / 100.0
4   } else {
5     0.0
6   };
7
8   let mut difficulty_coefficient = previous_difficulty_coefficient * (1.0 + delta_difficulty_coefficient);
9   let base_mint_size = config_data.initial_mint_size * config_data.reduce_ratio.powf((era - 1) as f64);
10
11   let mut mint_size = base_mint_size / difficulty_coefficient;
12   if target_mint_size_epoch % mint_size > 0.0 {
13     mint_size = target_mint_size_epoch / ((target_mint_size_epoch / mint_size).trunc() + 1.0);
14     difficulty_coefficient = base_mint_size / mint_size;
15   }
16
17   (mint_size, difficulty_coefficient)
18 }

```

4. Deployment Parameter

4.1 Total Supply

The total supply of PoM scheme is different from the manual setting of the total supply of many tokens. It is calculated based on Era, Epoch, and the **reduction factor**.

There are four parameters that determine the total supply:

- E : Total Eras
- C : Epoches per Era
- T_0 : Initial target mint size per epoch
- f : Reduce factory by era, $f \in (0, 1)$

Calculation

$$TotalSupply = \sum_{i=1}^E (C \cdot T_0 \cdot f^{i-1}) = C \cdot T_0 \cdot \frac{1 - f^E}{1 - f} \quad (15)$$

Example: $E=15$, $C=10$, $T_0=100,000$ tokens, $f=0.75$

The total yield is: 3,946,546.155959368 tokens

Click [here](#) to open Wolfram calculation, you can use **wolfram** to calculate the total supply easily.

As E approaches infinity, meaning that mining can continue indefinitely, **TotalSupply** will converge to a value, which is:

$$\lim_{E \rightarrow \infty} (C \cdot T_0 \cdot \frac{1 - f^E}{1 - f}) = C \cdot T_0 \cdot \frac{1 - f^\infty}{1 - f} = \frac{C \cdot T_0}{1 - f} \quad (16)$$

4.2 Estimated Total Minting Time:

Note: The actual total minting time will differ from the estimated total minting time.

The following formula is for calculating the estimated total minting time.

- E : Eras
- C : Epoches per Era
- N_t : Target Number Of Blocks Per Epoch
- t : Seconds per block

Calculation

$$TotalEstimatedTime = E \cdot C \cdot N_t \cdot t \quad (17)$$

Example: $E=15$, $C=10$, $N_t=50$, $t=12$ seconds

The estimated total minting time: $15 * 10 * 50 * 12 = 90,000$ seconds = 25 hours

Combining the two formulas above, we can calculate the value of E_r when 80% of the total supply (r) of tokens has been minted.

$$C \cdot T_0 \cdot \frac{1 - f^{E_r}}{1 - f} = \frac{C \cdot T_0}{1 - f} * r \quad (18)$$

From this equation, we get:

$$E_r = \log_f(1 - r) \quad (19)$$

Example:

$r=80\%$, $f=0.75$, then $E_r = \log_{0.75}(1 - 0.8) = \frac{\ln 0.2}{\ln 0.75} = 5.59$.

That means in the middle of the 5th Era, 80% of the total supply will be minted.

If $C = 10$, on the 56th epoch, 80% of the total supply will be minted.

4.3 Total Minting Fee

Each minting requires a fixed fee, however, due to the increase in difficulty, the minting times within the same epoch will increase, and the number of tokens obtained per minting will decrease, thus the total fee will increase accordingly.

Note: These minting fees are added to Liquidity pool automatically.

- Fee : Total Fee
- P_0 : Minting fee per minting
- d : Difficulty coefficient
- Q : Mint times in an epoch
- T_0 : Target mint size per epoch of the Genesis Era
- M_0 : Base mint size per minting of the Genesis Era
- C_e : Elapsed epoches, $C_e = E * C$

Mint times in an epoch:

$$Q = \lfloor \frac{T_0}{M_0} * d \rfloor + 1, (T_0 \nmid M_0) \quad (20)$$

$$Q = \frac{T_0}{M_0} * d, (T_0 \mid M_0) \quad (21)$$

For simplicity, let's assume $T_0 \mid M_0$, the total fee:

$$TotalFee = \sum_{i=0}^{C_e} (P_0 \cdot Q_i) = \sum_{i=0}^{C_e} (\frac{P_0 \cdot T_0}{M_0} * d_i) \quad (22)$$

As $d_i = d_{i-1} \cdot (1 + \delta d_i)$, $\delta d \in [0, 0.01]$ (see. 3.1.1), and $d_0 = 1$, the total fee range will be:

$$TotalFee \in [\frac{P_0 \cdot T_0}{M_0} \cdot \sum_{i=0}^{C_e} 1^i, \frac{P_0 \cdot T_0}{M_0} \cdot \sum_{i=0}^{C_e} 1.01^i] \quad (23)$$

Simplified:

$$TotalFee \in [\frac{P_0 \cdot T_0}{M_0} \cdot (C_e + 1), \frac{P_0 \cdot T_0}{M_0} \cdot 100 \cdot (1.01^{C_e+1} - 1)] \quad (24)$$

Example

$P_0 = 1$ USD, $T_0 = 9000$, $M_0 = 100$, $d = 1.5$, $C_e = 300$, Minimum total fee is \$27,090, and max is \$170,877.

This indicates that if minting is done quickly, the actual minting time in epoch (N_e) is less than the target time(N_t), it will lead to a continuous increase in difficulty and mint cost. Consequently, the total fees collected will be 6.3 times higher than if the difficulty remained constant, and the difference becomes more significant as the epoch number increases.

4.4 Use cases

4.4.1 If the target total supply is 21 million(around), there can be (but not limited to) the following parameter combinations:

C	T_0	M_0	f	N_t	t	Total Supply	Eras(95%)	Epochs(95%)	Days(95%)	Total Fee(min)	Total Fee(max)
600	9000	1000	0.75	2000	12	21.6 million	10.413	6248	1735.56	56,241	9.088e29
500	11000	1000	0.75	500	12	22 million	10.413	5206	361.57	57,277	3.49e25
2500	1000	100	0.75	1000	0.4	10 million	10.413	26032	120.5	260,320	3.15e115

Note: * $Eras = \log_f(1 - 0.95)$ * $Epochs = Eras * C$ * $days = Epochs * N_t * t / 3600 / 24 = \log_f(1 - 0.95) * C * N_t * t / 86400$ * As the f, t are fixed, the total days is depend on C and N_t

4.4.2 How to calculate all parameters We will try to calculate all parameters according to the following conditions:

- Total Supply
- Target minting days
- Minimum total fee collected

The constants are: * Total Supply: 10,000,000 * Total minting days: 180 days, 95% of the total supply will be minted * Target minting fee: 30,000 USD * $f = 0.75$ * $T_0 = 10,000$ * $M_0 = 1,000$ * $t = 0.4$ seconds(for Solana blockchain)

1- $\therefore TotalSupply = \frac{C \cdot T_0}{1-f}$, so: $C = 10,000,000 * (1 - 0.75)/10000 = 250$

2- From the following formula, we can get $N_t = 14935$

$$C * N_t * t * \log_f(1 - 0.95) = 180days * 86400seconds/day \quad (25)$$

3- From the Following formula, we can get $C_e = 2603$

$$C_e = Eras * C = \log_f(1 - 0.95) * C = 10.413 * 250 = 2603 \quad (26)$$

4- We already know: $T_0 = 10,000$, $C_e = 2603$, $M_0 = 1000$, the minimum total fee formula:

$$\frac{P_0 * T_0 * (C_e + 1)}{M_0} = P_0 * 10,000 * 2604/1000 = 300,000 \quad (27)$$

According the minimum total fee formula, we can get $P_0 = 11.52$ USD, which means the lowest price of each token is $P_0/M_0 = 11.52/1000 = 0.01152$ USD

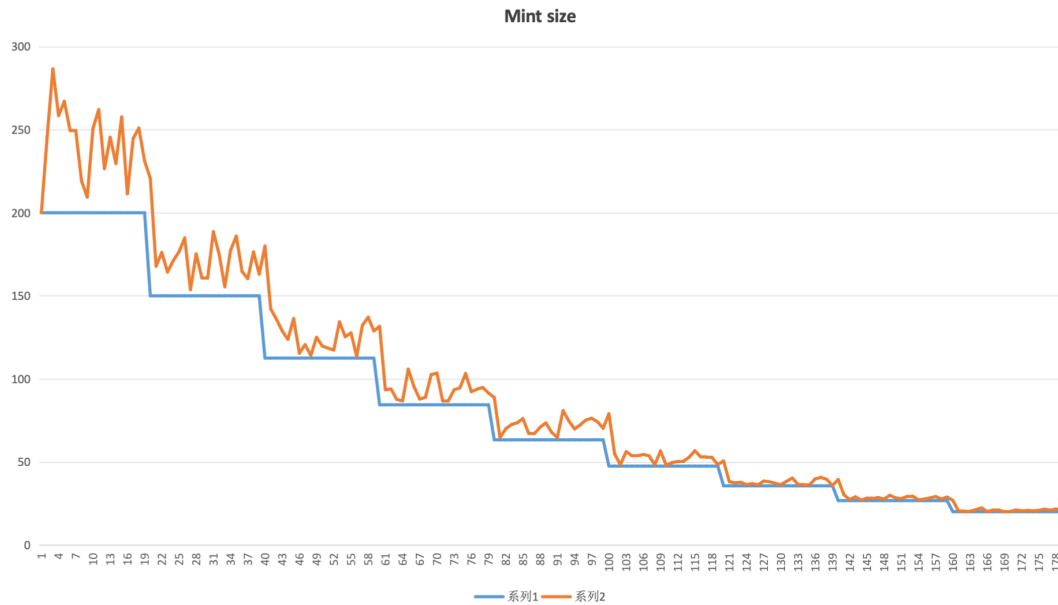
[Click here to open online calculator](#)

5. Testing and Evaluation

Below are the parameters for the simulation test:

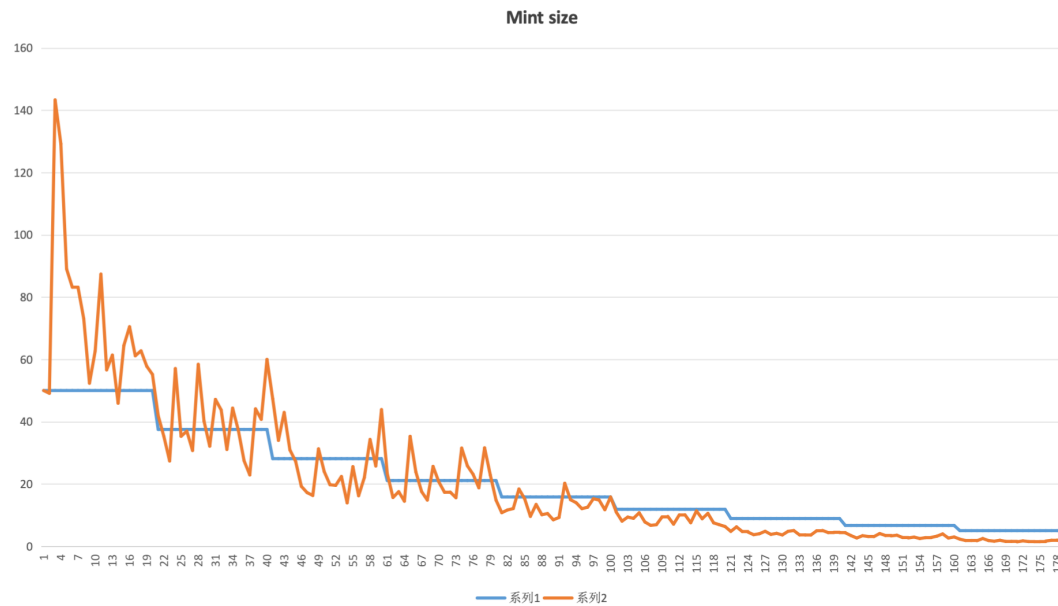
- Minting interval time range: Randomly between 0-30 seconds
- Total number of Eras: 10
- Number of Epochs per Era: 20
- Minimum difficulty coefficient: 0.2
- Reduction coefficient: $3/4$
- Base mint size per minting in the Genesis Era: 50
- Target mint size per Epoch in the Genesis Era: 200

Minting Rewards



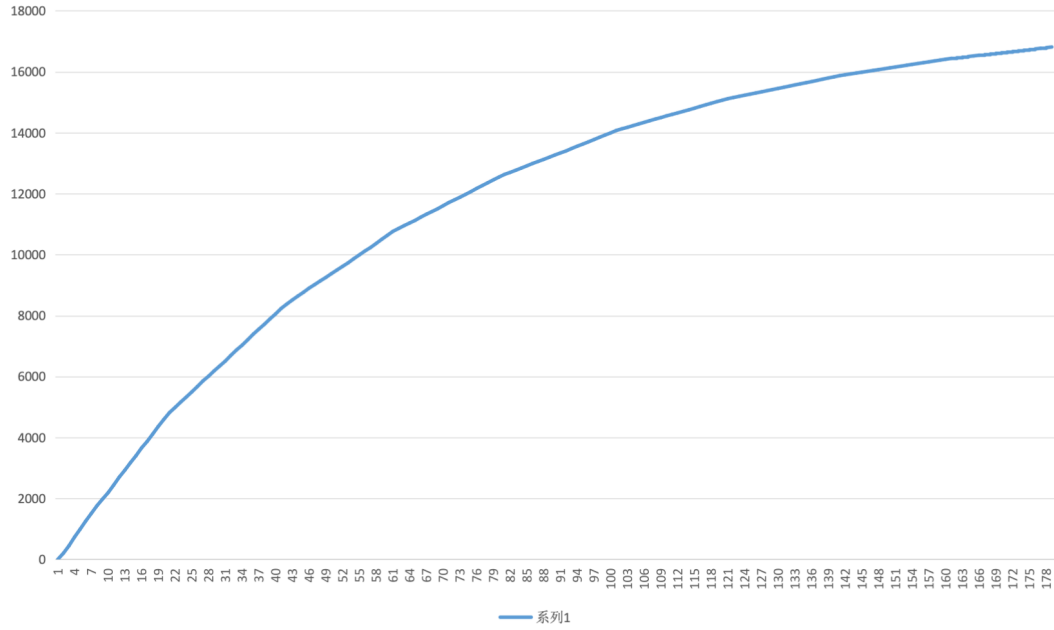
Epoch Actual Minting Yield vs. Target Minting Yield

The orange line is the **actual mint size of the current Epoch**, and the blue line represents the **target mint size per minting**, showing a trend of a 25% reduction per Era.

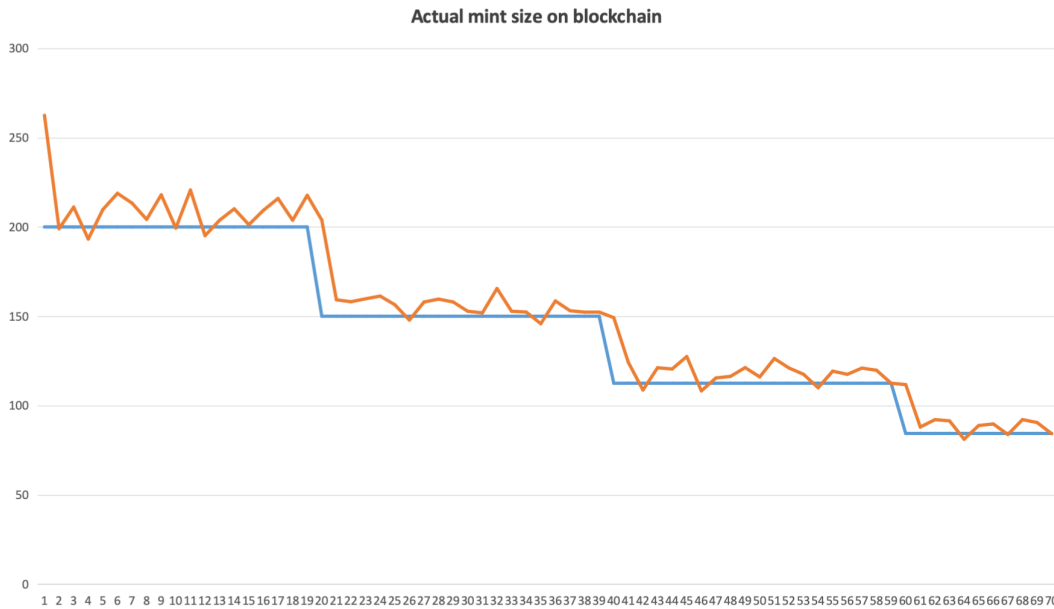


Actual Minting Reward vs. Target Minting Reward

The orange line is the **actual mint size**, and the blue line is the **target mint size**.



Total Minting Curve



Simulation Results on Ethereum Chain

The above are simulation results on the Ethereum chain.

6. Affiliate minting program

6.1 - Description

The **Affiliate Minting Program (AMP)** allows each user to generate an **Unique Referral Code (URC)** and share it with others. When others use this **URC** to mint, they can get a discount on the minting fees, and the referrer can get some rewards. **AMP** is designed to be decentralized and community-driven, incentive for building bettercommunity.

1. Decentralization and Community Driven

- All minting must use an **URC**, that every minting is associated with a member of the community.
- The progress and speed of minting are influenced by the community members in sharing **URC**, not by the core team or some whales.
- Through **AMP**, the community can form a consensus more easily and attract more members to participate in the community.

2. Incentivizing Community Building

- Dual incentives: Both the Code Sharers (referrer) and the users who uses the **URC** (minter) can benefit, and this dual incentive mechanism helps the growth and activity of the community.

3. Mint discount and Rewards for referr

- The mint fee discount is linked to the token balance of the account providing the **URC**. The more referrer's balance, the more discount of minting, the less mint fee. That will encouraging participants to hold more tokens rather than selling them all.
- **URC** sharers can get stable rewards, this rewards are automatically distributed to the referrer's account by the chain coins(ETH for Ethereum, or SOL for Solana) when the minting is happened.

4. Minting Fees and Difficulty Adjustment

- Dynamic difficulty: The total cost and difficulty of minting will be dynamically adjusted based on the community; the more people mint, the faster the minting speed, the higher the total minting cost, and the higher the extra minting fee.
- Part of the minting fees is redistributed as discounts to minters and as rewards to **URC** sharers.

$$ExtraMintFee = \frac{P_0 \cdot T_0}{M_0} \cdot (\sum_{i=0}^{C_e} d_i - C_e - 1) \quad (28)$$

The formula indicates that the higher d_i is (approximately above 1), the higher the extra minting fee.

Since the minting fee goes into the Liquidity Pool to support exchanging in the marketplace, therefore, **AMP** will have a positive impact on community development and token exchanging.

6.2 - Calculations

6.2.1 - Mint discount Discount is decided by the ratio of the balance of code sharers to the total supply of tokens.

- r : The ratio of the balance of code sharers to the total supply of tokens.
- k : The discount rate.

r	k
< 0.2%	0
0.2-0.4%	5%
0.4-0.6%	10%
0.6-0.8%	15%
0.8-1%	20%
> 1%	25%

6.2.2 - Minting Fee after discount

- Fee : Minting fee after discount
- P_0 : Original minting fee
- p_0 : Price of minted token for the first Epoch
- p : Price of minted token before discount
- d : Difficulty coefficient
- k : The discount rate

$$\frac{Fee}{M_b} \cdot d = p_0 + (p - p_0) \cdot (1 - k) = p \cdot (1 - k) + p_0 \cdot k \quad (29)$$

$$p = \frac{P_0}{M_b} \cdot d, p_0 = \frac{P_0}{M_b} \quad (30)$$

From aboved equivalent formula, we can get the minting fee after discount:

$$Fee = P_0 \cdot (1 + \frac{k}{d} - k), (d \geq 1, k \leq 0.25) \quad (31)$$

The balance between Fee and original P_0 is:

$$P_0 - Fee = P_0 - P_0 \cdot (1 + \frac{k}{d} - k) = P_0 \cdot k \cdot (1 - \frac{1}{d}) \quad (32)$$

The discount rate is:

$$\frac{P_0 - Fee}{P_0} = k \cdot (1 - \frac{1}{d}) \quad (33)$$

Example:

$P_0 = 8$ USD, $d = 12.3$, \$Balance=26,000, $TotalSupply = 5,000,000$ USD

$r = 26,000/5,000,000 = 0.52$, the discount(k) is 10%.

The minting fee is: $8 * (1 + 0.1/12.3 - 0.1) = 7.265$ USD

Comparing with the original minting fee, the discount is: $1 - 7.265/8 = 9.19\%$

NOTE: The amount of minted tokens is not changed, see 3.1.4.

6.2.3 - Mint code Mint code is a unique code that is generated by the sharer's account and timestamp.

```
1 // Generate the code by concatenating the Solana address and the current timestamp
2 pub fn generate_referral_code(solana_address: &Pubkey, unix_timestamp: u64) -> u64 {
3     let address_u64 = u64::from_le_bytes(solana_address.to_bytes()[..8].try_into().unwrap());
4     let timestamp_u64 = unix_timestamp;
5     (address_u64 << 32) | (timestamp_u64 & 0xFFFFFFFF)
6 }
```

6.2.4 - limitations of mint code

- The number of times that the code can be used is not unlimited, with a default of 20. That means after **20** mints by this code, the code will be invalid and need the sharer re-activate it.
- Code sharers can not re-activate the code at anytime, there's an interval between each activation. Default interval is **24 hours**.

6.2.5 - Benefits of code sharers The code sharer can got 20% of the balance fee the miner saved.

$$CodeSharerReward = 0.2 \cdot (P_0 - Fee) = 0.2 \cdot P_0 \cdot k \cdot (1 - \frac{1}{d}) \quad (34)$$

The higher the difficulty, the larger the k , the greater the benefits for the Code Sharer.

Because $max(k) = 25\%$, when $d = \infty$, Code Sharer receives 5% of P_0 .

Example:

From the previous example, the reward for code sharer is:

$$0.2 * 8 * 0.10 * (1 - 1/12.3) = 0.147 \text{ USD}$$

6.3 - Evaluation

Let's make some change on the formula of minting fee, and see how it affects fee.

$$\frac{MintFee}{P_0} = 1 + \frac{k}{d} - k, (d \geq 1, k \leq 0.25) \quad (35)$$

6.3.1 - Difficulty(d) Effects on Minting Fee As the formula of minting fee, the higher the difficulty is, the lower minting fee and the more discount.

6.3.2 - k Effects on Minting Fee As the formula of minting fee, the higher the k is, the lower minting fee and the more discount.

6.3.3 - Risk We have to consider one situation that **AMP** probably lead to self-minting(using self **URC** to mint), but we believe that: once a person has a certain number of Tokens, they will prefer to build community and share the Code to others.

Additionally, even if everyone use the maximum discount to mint(which is impossible), the minimum total minting fee would be as follows:

$$MintFee = P_0 \cdot \lim_{d \rightarrow \infty} [1 + \frac{k}{d} - max(k)] = P_0 \cdot (1 - max(k)) = 0.75P_0 \quad (36)$$

So, the minimum total minting fee is 75% of the plan. Considering the community activity that **AMP** may bring, and the increase of difficulty will bump the minting fee, this reduction is worthwhile.

6.4 - Initialization

6.4.1 - System Referrer Why need for a system/default referrer? * All minting requires **URC**, if someone can not obtain an **URC** from community members, he/she can use the default **URC** provided by the system. * When using the default **URC**, there is no discount (cause the default referrer's account balance is 0).

6.5 - Program Implementation

```

1 // Calculate the fee value and the referrer reward
2 pub fn get_fee_value(fee_rate: u64, difficulty_coefficient: f64, referrer_ata_balance: u64, total_supply: u64) {
3     let balance_ratio = referrer_ata_balance as f64 / total_supply;
4     let discount_rate = if balance_ratio < 0.002 {0.0}
5     else if balance_ratio < 0.004 {0.05}
6     else if balance_ratio < 0.006 {0.1}
7     else if balance_ratio < 0.008 {0.15}
8     else if balance_ratio < 0.01 {0.2}
9     else {0.25};
10    let fee = fee_rate as f64 * (1.0 + discount_rate / difficulty_coefficient - discount_rate);
11    let code_sharer_reward = 0.2 * fee_rate as f64 * discount_rate * (1.0 - 1.0 / difficulty_coefficient);
12    (fee as f64, code_sharer_reward)
13 }

```

7 - Liquidity

The minting fee will ultimately be injected into the liquidity pool of the Decentralized Exchange, where transaction fees can be earned.

7.1 - Tokens for the Liquidity Pool

A portion of the total Token supply will be allocated to the liquidity pool as SOL/Token pair. This allocation is distributed into a dedicated liquidity account during minting. Once certain conditions are met(Ex. Era#1 is done), these Tokens, along with transaction fees(SOL), are added into the liquidity pool automatically.

- M_a : The number of Tokens minted per instance (see 3.1.4, 3.1.4)
- r_l : The proportion of the initial liquidity pool relative to the total issuance, where $r_l < 1$
- L : The number of Tokens entering the liquidity-specific account per minting event

$$L = \frac{M_a \cdot r_l}{1 - r_l} \quad (37)$$

Cause the total supply of tokens:

$$TotalSupply = \sum_{i=1}^E (C \cdot T_0 \cdot \frac{1 - f^E}{1 - f}) = C \cdot T_0 \cdot \frac{1 - f^E}{1 - f} \quad (38)$$

The tokens for initializing Liquidity pool:

$$InitLiquidity = TotalSupply \cdot \frac{r_l}{1 - r_l} = C \cdot T_0 \cdot r_l \cdot \frac{1 - f^E}{(1 - f) \cdot (1 - r_l)} \quad (39)$$

7.2 - Minting Fees

The minting fee is described in section 6.2.2. The distribution of the minting fee is as follows:

- URC supplier : 0-5%
- Protocol : 4.75%
- Liquidity Pool : 90.25-95.25%

7.3 - Estimated Price at Liquidity Pool Initialization

According to the **AMM** of Decentralized Exchanger, the token price at the time of initializing the liquidity pool is:

$$Price = \frac{0.9025 \cdot TotalFee}{InitLiquidity} \quad (40)$$

As the total mint fee is a range:

$$TotalFee \in \left[\frac{P_0 \cdot T_0}{M_0} \cdot (C_e + 1), \frac{P_0 \cdot T_0}{M_0} \cdot 100 \cdot (1.01^{C_e+1} - 1) \right] \quad (41)$$

So, the lowest price when initializing the Liquidity Pool will be:

$$P_{low} = \frac{P_0 \cdot (C_e + 1)(1 - r_l)(1 - f)}{M_0 \cdot C \cdot r_l \cdot (1 - f^E)} * 0.9025 \quad (42)$$

And the highest price will be:

$$P_{high} = \frac{100 \cdot P_0 \cdot (1.01^{C_e+1} - 1)(1 - r_l)(1 - f)}{M_0 \cdot C \cdot r_l \cdot (1 - f^E)} * 0.9025 = \frac{100 \cdot (1.01^{C_e+1} - 1)}{C_e + 1} \cdot P_{low} \quad (43)$$

In the aboved formula, $C_e = E \cdot C$ (C is Epoches per Era, E is Eras)

8 - Complete Case Study

The following is a comprehensive case study demonstrating the deployment of the Proof of Mint(PoM) mechanism on **Solana** blockchain

8.1 - Parameters

- Total supply: 1,000,000,000 tokens
- Target mint size per minting for 1st Era(M_0): 10,000 tokens
- Target mint size per epoch for 1st Era(T_0): 1,000,000 tokens
- Minimum minting instances per Epoch: 100
- Target Era(E): 1, after the first Era is completed, holders can transfer tokens and the liquidity pool is initialized.
- Epoches per ear(C): 250
- Target minting days: 30 days
- Reduction factor(f): 0.75
- Tokens percentage for Liquidity(r_l): 10%
- Fee per minting(P_0): 2 USD per minting (0.0002USD/token)
- Target minting time per Epoch: 173 minutes (around 3 hours)
- How to handle of LP tokens generated when initializing the liquidity pool: **All destroyed**

- Upon reaching Target Era#1:
 - Total Supply: 250,000,000 tokens (25% of max supply)
 - Total number of Epochs: 250
 - Number of minting: $\geq 25,000$
 - Total minting fee: 50,200 USD to 223,050 USD
 - Price at the time of initializing the liquidity pool: 0.001807 USD to 0.008030 USD

8.2 - Mint workflow(Technically)

8.3 - Initialize the liquidity pool workflow (Technically)

Anyone can initialize the liquidity, the only condition is $\text{Era} > 1$.