Dexy - Simple Stablecoin Design Based On Algorithmic Central Bank

kushti, scalahub

May 19, 2025

Abstract

In this paper, we consider a new stablecoin protocol design called Dexy. The protocol is maintaining the peg with two assets only (base currency and stablecoin), and also two functional components, namely, a reference market (in form of a liquidity pool where anyone can trade base currency against stablecoin), and an algorithmic central bank, which is responsible for new stablecoins issuance, and also for maintaining stablecoin pricing corresponding to a peg in the reference market.

1 Introduction

Algorithmic stablecoins is a natural extension of cryptocurrencies, trying to solve problems with volatility of their prices by pegging stablecoin price to an asset which price is considered to be more or less stable over long periods of time (e.g. gold).

Having an asset with a stable value could be useful in many scenarios, for example:

- securing fundraising; a project can be sure that funds collected during fundraising will have stable value in the mid- and long-term.
- doing business with predictable results. For example, a shop can be sure that funds collected from sales will be about the same value when the shop is ordering goods from warehouses (otherwise, the shop may go bankrupt if its margin is not that big to cover exchange rate fluctuations).
- shorting: when cryptocurrency prices are high, it is desirable for investors to rebalance their portfolio by increasing exposure to fiat currencies (or real-world commodities). However, as fiat currencies and centralized exchanges impose significant risks, it would be better to buy fiat and commodity substitues in form of stablecoins on decentralized exchanges.
- lending and other decentralized finance applications. Stability of collateral value is critical for many applications.

Algorithmic stablecoins are different from centralized stablecoins, such as USDT and USDC, for which there is a trusted party which holds the peg. In case of an algorithmic stablecoin, the pegging is done via rebasement of total supply (as in Ampleforth etc), or via imitating a trusted party which holds reserves and doing market interventions when it is needed for getting exchange rate back to the peg. Imititating the trusted party is usually done by allowing anyone on the blockchain to create over-collateralized financial instruments, such as collateralized debt positions (as in DAI), zero-coupon bonds (as in the Yield protocol), reserve asset (as in Djed), or by issuing stabilizing financial instruments in case of depeg (as in Neutrino). Alex notes : add links to the paragraph above

In this work we present Dexy, a stablecoin protocol where an algorithmic central bank performing interventions in case of depeg is presented explicitly as a contract with few predefined rules. The bank is trying to stabilize stablecoin value on the markets, using a liquidity pool as a reference market, by providing stablecoin liquidity, when stablecoin price is over the peg, or injecting base currency from its reserves, when the stablecoin is under the peg. In extreme case, when bank reserves are depleted and stablecoin is still under the peg, its value is restored by extracting stablecoins from the liquidity pool (to inject back when price will go above the peg again). In some aspects Dexy could resemble Fei or Gyroscope. Alex notes : make comparison subsection

The rest of the paper is organized as follows. In Section 2 we sketch Dexy design in general and introduce notation used in other sections. Section 3 is defining worst-case scenario for bank reserves (and so, for stablecoin stability). Then Dexy is detalized in Section 4. Stability of the rules are discussed in Section 5. Trust assumptions Dexy protocol is based on are stated in Section 6. Section 7 describes concrete Dexy implementation, gold-pegged DexyGold stablecoin and its tokenomics.

2 Dexy Design in General

Unlike popular algorithmic stablecoins based on two tokens (in addition to base currency), such as Djed, Dexy is based on one token but two protocols. In the first place, there is a reference market (done as on-chain protocol, such as automated market maker [1]), where trading of Dexy vs the base currency (ERG) happens. In the second place, to tackle with situations when the reference market price is way too different from a target price (price of a pegged asset, as reported by a trusted oracle), there is an algorithmic central bank which is doing interventions in order to readjust the market price (so make it closer to the oracle's one). The central bank can also mint new Dexy tokens by selling them for ERG. The bank is using reserves in ERG it is buying for interventions.

As a simple solution for the *reference market*, we are using constant-product Automated Market Maker (CP-AMM) liquidity pool, similar to Spectrum and UniSwap. The pool has ERG on one side and Dexy on another. For CP-AMM pool, multiplication of ERG and Dexy amounts before and after a trade should be preserved, so $e * u = e' * u'$, where e and u are amounts of ERG and Dexy in the pool before the trade, and e' and u' are amounts of ERG and Dexy in the pool after the trade, correspondingly. As for any CP-AMM pool, it is possible to put liquidity into the pool, and remove liquidity from it, however, there are some limitations here we are going to uncover further.

The bank has two basic operations. It can mint new stablecoin tokens per request, using trusted oracle's price, by accepting ERG in its reserves. It also can intervene into markets by providing ERG from reserves when needed (namely, when price in the pool $\frac{u}{e}$ is significantly different from price on external markets p which reported by oracle).

Now we are going to consider what are possible failure scenarios for such design and how to put restrictions and design rules for the system to ensure stable pricing for stablecoin tokens.

2.1 Notation

We are introducing notation used further:

- T - period before intervention starts. After one intervention the bank can start another one only after T time units passed.
- p - price reported by the oracle at the moment (for example, 20 USD per ERG)
- s - price which the bank should stand in case of sudden price crash. For example, we can assume that $s = \frac{p}{4}$ (so if p is 20 USD per ERG, then s is 5 USD per ERG, means the bank needs to have enough reserves to save the markets when the price is suddenly crashing from 20 to 5 USD per ERG)
- R - ratio between p and s, $R = \frac{p}{s}$
- r - ratio between p, and price in the pool, which is $\frac{u}{e}$, thus $r = \frac{p}{\frac{u}{e}} = \frac{p*e}{u}$
- e - amount of ERG in the liquidity pool
- u - amount of stablecoin in the liquidity pool
- O - amount of stablecoin outside the liquidity pool. The distribution in O is not known for the Dexy protocol, but the bank can easily store how many stablecoin tokens were minted and then get O by deducting u from it.
- E - amount of ERG in the bank.

3 Worst Case Scenario and Bank Reserves

The bank is doing interventions when the situation is far from normal in the markets, and enough time passed for markets to stabilize themselves with no interventions. In our case, the bank is doing interventions based on stablecoin price in the liquidity pool in comparison with oracle provided price. The bank's intervention then is about injecting its ERG reserves into the pool.

First of all, let's assume that the oracle price crashed from p to s sharply and stands there, and before the crash there were e of ERG and u of stablecoin in the liquidity pool, with pool's price being p. The worst case then is when no liqudity put into the pool during the period T. With large enough T and large enough R this assumption is not very realistic probably: some traders will buy ERGs with their stablecoins anyway, price is usually failing with swings where traders could mint stablecoins thus increasing ERG bank reserves, etc. However, it would be reasonable to consider worst-case scenario, then in the real world Dexy will be even more durable than in theory.

In this case, the bank must intervene after T units of time, as the price differs significantly, and restore the price in the pool, so set it to s. We denote amounts of ERG and stablecoin in the pool after the intervention as e' and u', respectively. Then:

- $e * u = e' * u'$
- as the bank injects E_1 ergs into the pool, $e' = e + E_1$
- $\frac{u'}{e'} = s$, thus $u' = s * (e + E_1)$
- from above, $E_1 = \sqrt{\frac{e*u}{s}} - e$

So by injecting E_1 ERG, the bank recovers the price. However, this is not enough, as now there are O stablecoin units which can be injected into the pool from outside. Again, in the real life it is not realistic to assume that all the O stablecoin would be injected, as some of them are simply lost, some would be kept to buy cheap ERG at the bottom (we remind that stablecoin often used as a bet for ERG price decline), etc. However, we need to assume worst-case scenario again. We also unrealistically assume that all the O tokens are being sold in very small batches not significantly affecting price in the pool, and after each batch seller of a new batch is waiting for a bank intervention to happen (so for T units of time), and sells only after the intervention. In this case all the O tokens are being sold at price close to s, so the bank should have about $E_2 = \frac{O}{s}$ ERGs in reserve to buy the tokens back.

Summing up E_1 and E_2, we got ERG reserves the bank should have to be ready for worst-case scenario: $E_w = E_1 + E_2 = \sqrt{\frac{e*u}{s}} - e + \frac{O}{s}$.

It is simple to see why this scenario is worst-case for the bank. In this scenario, the bank (and only the bank) is buying all the O of external *stablecoin* at the worst possible price s, and also set the price by burning its own reserves only.

4 Bank and Pool Rules

Based on needed reserves for worst-case scenario estimation, we can consider minting rules accordingly. Similarly to SigUSD (a Djed instantiation), we can, for example, target for security in case of 4x price drop, so to consider $R = \frac{p}{s} = 4$, and allow to mint stablecoin while there are enough, so not less than E_w, ERG in reserves. However, in this case most of time stablecoin would be non-mintable, and only during moments of ERG price going up significantly it will be possible to mint Dexy. As worst-case scenario is based on unrealistic assumptions, unlikely a realistic protocol can be built on top of it.

Thus we leave worst-case scenario for UI, so dapps working with the Dexy may show e.g. collateralization for the worst-case scenario. Having on-chain data analysis, more precise estimations of reserve quality can be made (by considering stablecoins locked in DeFi protocols, likely forgotten, etc).

Instead, we always allow for cautios minting. That is, we whether allow minting when oracle price is above pool's price (in this way the bank is providing liquidity for arbitrage when stablecoin pice is above the peg), or we allow to mint a little bit (per some time period) when liquidity pool is in good shape. In details, we have two following minting operations, with minting price being the oracle's price p:

- Arbitrage mint: if price reported by the oracle is higher than in the pool, i.e. $p > \frac{u}{e}$, we allow to print enough stablecoin tokens to bring the price to p. That is, the bank allows to mint up to $\delta_u = \sqrt{p * e * u} - u$ stablecoin by accepting up to $\delta_e = \frac{\delta_u}{p}$ ERGs into reserves.

  To instantiate the rule, we can allow for arbitrage minting if the price p is more than $\frac{u}{e}$ by at least 1% for time period T_{arb} (e.g. 1 hour), also, the bank is charging 0.5% fee for the operation. After arbitrage mint it is not possible to do another one within 30 minutes (to prevent aggressive liquidity minting via chained transactions etc).

- *Free mint: we allow to mint up to $\frac{u}{100}$ stablecoins within time period T_{free}.* To instantiate the rule, we propose to allow for free mint if $0.98 < r < 1.02$. Minting fee in this case is also 0.5%. We propose to set T_free to 12 hours, then bank reserves can grow by 2% of LP volume per day when the peg is okay..

In addition to minting actions, which increase bank reserves, we define following two actions which decrease them:

- Intervention: if price reported by the oracle is lower than one in the pool by significantly enough margin, i.e. $\frac{p*e}{u}$ is less than some constant which is hard-wired into the protocol, then the bank is providing ERGs. to restore the price. *To instantiate the rule, we propose to allow the bank to intervene if $r <= 0.98$ for time period $T_{int} = 1$ day. During intervention tracker is reset, so another intervention will be after $T_{int} = 1$ day at least. The* *bank is getting price to 99.5% of the oracle price max, but also spending no more than 1% of its reserves for that.*

We also state following rules for the liquidity pool (which, otherwise, acts as ordinary CP-AMM liquidity pool):

- Stopping withdrawals: if r is below some threshold, withdrawals are stopped, so only swaps are possible. *To instantiate the rule, we propose to stop withdrawals immediately if* $r <= 0.98$.
- *Second stabilization mechanism: what if the bank is out of reserves, but stablecoin is still below the peg? In this case we restore price in the pool by removing liquidity, and there are two possible options here:*

  1. *Burn: if the bank if empty, and r is below some threshold, it is allowed to burn stablecoins in the pool.* To instantiate the rule, we propose to burn stablecoin if $r <= 0.95$ for time period T_{burn}. T_{burn} must be quite big, e.g. 1 *week*. We burn enough to return to the state of stopped withdrawals, so to $r = 0.98$. After burning the tracker is reset, so another burn will be done sfter T_{burn} at least.
  2. Extract for the future: if the bank is empty and r is below some threshold, it is allowed to extract stablecoins from the pool and lock by a contract which is releasing stablecoins in the future when stablecoin price is above the peg. *To instantiate the rule, we propose to extract stablecoin if* $r <= 0.95$ *for time period* T_{burn} *(so the same as in burn). To prioritize extracted funds over arbitrage mint, we do not have delay in releasing contract. We burn enough to return to the state of stopped withdrawals, so to* $r = 0.98$. *After extraction the tracker is reset, so another burn will be done sfter* T_{burn} *at least.*

In addition, we introduce 2% redemption fee to avoid excessive liquidity hopping.

5 Stability

With second stabilization mechanism (burning or extraction) in place, the price in the reference market will be eventually stabilized. However, for liquidity holders burning is painful, extraction not so but still not desirable, thus Dexy protocol is trying to avoid it (unlike other protocols, such as Gyroscope or Fei, where redemption rate fails below 1 immediately as collateralization falls under 100%), by giving markets time to self-stabilize, and then doing interventions. This could mean slower stabilization, in comparison with other protocols. Slower stabilization helps the bank to play with possible adversaries in the environment with information assymetry (humans always have access to more information than algorithmic bank, with more flexible decision making as well).

Dexy is also cautios about minting new stablecoins, which could mean slow growth of number of tokens issued. This could be inconvient, especially for big players, but the protocol is focused on stability in the first place.

Please note, that liquidity pool is disincentivizing massive bank runs due to its constant-product nature. Actually, for the bank massive bank runs are simpler for bank to resolve, in comparison with slow drain as in worst-case scenario.

6 Trust Assumptions

It is important to explicitly state assumptions the protocol is based on, so a user can choose whether to trust it or not.

- Oracle: the biggest trust issue in the protocol is oracle delivering gold price. This trust issue is unavoidable but can be relaxed by using not a single entity but a federation of oracles with an average price (after noise filtering) to be delivered to the Dexy central bank. It is important to reward oracles from protocol activities (such as minting), otherwise, they will be incentivized to attack the protocol. See Section 7.1 for details.

- No governance: unlike most of stablecoin protocols, we do not consider governance, as usually it is another trust issue. Real world instantiations can consider governance, we suppose that such instantiations should clearly inform about governance-related trust assumptions.

7 Dexy Gold

In this section we consider concrete implementation of the Dexy protocol, gold-pegged stablecoin named DexyGold.

7.1 Oracle incentivization and tokenomics

Oracles security is the most important issue for our stablecoin protocol, as this is the only component which is not working autonomously on-chain. Thus the protocol needs to reward oracles.

Oracle Pools 2.0 framework [2] has support for paying rewards in custom tokens. So we will create GORT (Gold Oracle Reward Token), which will be used to reward oracles and also developers.

X oracles wil get up to $2 \cdot X$ GORTs per epoch (one datapoint update), per Oracle Pool v2 design [2] ($2 \cdot X$ if all the oracles are active). We consider $X = 30$ oracles with one hour long epoch, which means up to 60 GORTs will be released to oracles.

In addition to oracles, GORT is used to reward developers. To do that, we consider a flat emission contract, which is releasing the same amount per one hour, so 2 GORT per block during initial period (2 years).

7.2 Implementation

7.3 Bank Contract

Alex notes : put contracts here

7.4 Simulations

We made simulations, you can find them in Dexy repository. Alex notes : finish

Acknowledgments

Authors would like to thank Ile for his inspiring forum posts.

References

[1] J. Xu, K. Paruch, S. Cousaert, and Y. Feng, "Sok: Decentralized exchanges (dex) with automated market maker (amm) protocols," *ACM Computing Surveys*, 2021.

[2] scalahub, "Eip-0023 oracle pool 2.0." Available at `https://github.com/ergoplatform/eips/pull/41`.