Transmuter: An autonomous and modular price stability module for decentralized stablecoin protocols

Picodes[*] Guillaume Nervo Pablo Veyrat

May 2023

Abstract

Here, we introduce Transmuter, a system for a stablecoin backed by a basket of other assets, that comes with guarantees on the maximum exposure the stablecoin can have to each asset in the basket. This system is intended as an improvement over current price stability modules for stablecoin protocols, designed to work autonomously in case of black swan events. It is compatible with other common mechanisms like collateralized-debt position models often used to issue stablecoins.

Contents

[*] Please direct any enquiry to contact@angle.money

1 Motivation

Decentralized stablecoin protocols are sets of smart contracts designed to take volatile (or not) cryptocurrencies as an input to produce a stable output. So far, maintaining peg for over-collateralized decentralized stablecoins [1] often implied relying on price stability modules [2] (PSM).

The first instance of a price stability module was that of Maker [3] which allows the conversion of USDC [4] into DAI at a 1:1 rate. FRAX [5] relies on a similar system that consists in minting FRAX in a FRAX-USDC Curve [6] pool whenever there are more FRAX than USDC in it. Angle Core module [7] is a more advanced mechanism. It is essentially a price stability module that comes with built-in hedging mechanisms allowing it to work with non-pegged assets in the backing.

While the use of PSM was often questioned because of the centralization vectors it created within the concerned stablecoin protocols (at some point more than 60% of the DAI in circulation were backed by USDC in the PSM [8]), designs of such PSMs have rarely been put in question.

In March 2023, the USDC depeg brought back the matter to the spotlight. In the case of Maker, which assumed that 1 USDC was worth 1$, the USDC depeg led people to trade USDC for DAI and DAI for other stablecoins in the PSM that did not depeg like GUSD [9]. In short, the USDC depeg led the reserves of Maker in the PSM to be in majority converted into the weakest asset in the backing. The protocol had some circuit breakers, like maximum daily inflows, optional transaction fees, but these were not enough to prevent the depletion of "healthy" reserves.

As for Angle, which did not consider that 1 USDC was worth 1$, people burnt agEUR against USDC worth 0.9$ to later mint back agEUR when USDC was back at 1$ and arbitrage the protocol.

While everything came back to normal when USDC price raised back to 1$, this situation was a heavy stress test for all price stability modules, and showed that current PSM designs were in the wide majority not adapted to a depeg of a stablecoin in the backing.

In this paper, we present a novel approach to build a price stability module with the following properties:

- It is scalable and allows for mints and burns with low slippage.
- It is able to autonomously withstand unforeseen events, such as collateral depegs or hacks, without requiring any governance intervention.
- It treats all stablecoins holders equally during black swan events, without leaving any bank run possibility.
- It is robust enough to accept any non-volatile backing, including stablecoins, yield-bearing tokens, and real-world assets.
- It is modular and can be generalized to any stablecoin.
- It is able to properly diversify and segregate risks among all the assets comprising its backing, even when the natural tendency for automated systems is quite often to be left holding the weakest assets.

2 Overview and Assumptions

The Transmuter system is conceived as a stablecoin backed by a basket of different assets. Ideally, these assets should be denominated in the stablecoin's base currency, to avoid any exchange or volatility risk. In fact, a strong assumption of the Transmuter system is that backing assets have *target prices* in the stablecoin's base currency. The target value for a collateral could be either absolute or updated relatively frequently.

For every asset in the reserves, we name p_i the current price of the asset and *deviation* the value $\max(1 - \frac{p_i}{\texttt{target price}_i}, 0)$. The Transmuter system basically uses this deviation measure as a circuit breaker.

EXAMPLE 2.1

Throughout this paper, we'll take the example of a Euro (€) stablecoin like agEUR backed through Transmuter by three other assets ($\texttt{EUR}_A$, $\texttt{EUR}_B$ and $\texttt{EUR}_{\texttt{yield}}$). If needed, you can find detailed and illustrated examples here 7.
Let us assume that $\texttt{EUR}_A$ is a centralized Euro stablecoin with a target value of 1€. If the current observed price of $\texttt{EUR}_A$ is 0.95, deviation is $1 - 0.95/1 = 0.05 = 5\%$.

For each asset in its reserves, the Transmuter system also tracks how many stablecoins were issued from this asset. We introduce the *exposure* to an asset i as:

$$\frac{\texttt{stablecoins issued using } i}{\texttt{total stablecoins issued}}$$

EXAMPLE 2.2

To continue the previous example, assume 1000 stablecoins are backed by $\texttt{EUR}_A$, and in total 3100 stablecoins are backed by the Transmuter system. The exposure to $\texttt{EUR}_A$ is then $1000/3100 = 0.323 = 32.3\%$.

The goal of the system is to ensure that exposures stay within reasonable boundaries and that in case of black swan events they do not deviate too much.

Now, 3 primary interactions with the system are possible:

- Stablecoins can be minted 3.2 at oracle value from any of the assets with adaptive fees 3.1, provided that the deviation of *the asset used* is reasonable.
- Stablecoins can be burnt 3.3 at oracle value for any of the assets in the backing with adaptive fees, provided that the deviation of *all assets* are reasonable. The idea is to avoid capital outflows changing the exposures of the system in times of uncertainty.
- Stablecoins can be redeemed 3.4 at any time against a proportional amount of each asset in the backing. Indeed, users should have a way to exit at any time, and as redemption does not change overall exposures, it's fine to allow it in any conditions.

Figures 3 and 4 in appendix 7 illustrate these different interactions with the Transmuter system. Note as well that Transmuter is generalizable and can work for a stablecoin pegged to any asset.

EXAMPLE 2.3

We could imagine a stablecoin pegged to ETH and maintaining its price stability through Transmuter with in the backing different ETH liquid staking derivative tokens.
If in this case wstETH [10] was included in the backing, the target value would be variable and encoded as the conversion rate between wstETH and stETH.

3 Actions

3.1 Asset exposure control

As explained above, the system allows minting and burning the stablecoin against any of the assets in the backing. However, it needs to ensure:

- That the oracles used to mint/burn cannot lead to the system being tricked into holding only the weakest assets
- That the oracles used are not manipulable
- That exposures of the stablecoin to its backing assets remain within acceptable bounds

To control these exposures, minting and burning happen with fees depending on the exposure the system will have to the asset used after the mint or burn operation.

Practically, the mint fees can be set to a high value (100%) when the exposure is above a target exposure, while burn fees can be made low in this case. Conversely, when exposure is below the target window, mint fees can be set low and burn fees high so the exposure does not drift below a certain threshold. With such parameters, the exposure should naturally converge to a target window.

EXAMPLE 3.1

Here is an example of what the fees could look like for $\texttt{EUR}_A$, $\texttt{EUR}_B$ and $\texttt{EUR}_{yield}$. With such values, the exposure for each asset should autonomously live within the red and blue vertical lines.

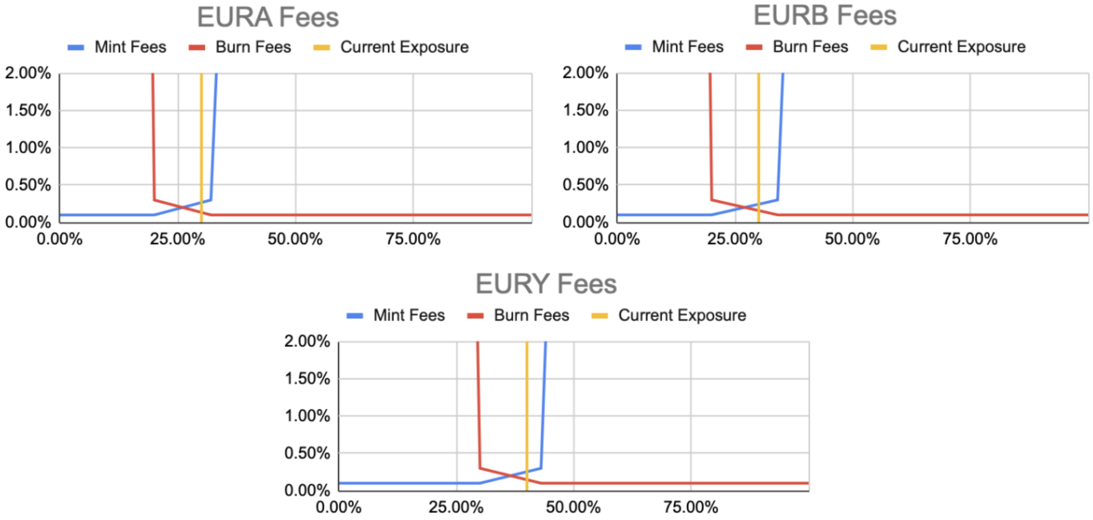


Figure 1: Examples of fee depending on the exposure settings

Such fee functions enable the system to encourage users to mint from one asset in the backing and to burn for another specific asset. Over time, provided that the Transmuter system is well routed by aggregators like 1inch [11], exposures are meant to converge to areas where fees are low, thereby incentivizing users to maintain an exposure close to what is desired.

The scope and modularity of what can be implied by this fee mechanism system in terms of reserve dynamics is immense. While fees could be set to define ranges where exposures should naturally converge, fees could also be put in order to implicitly define a liquidity buffer to be depleted before an investment buffer when there is a sell pressure on the stablecoin.

To control exposures to assets in the backing, some protocols like Maker enforce volume limitations for what can be traded. These are absolute figures that need to be updated as liquidity in the PSM grows. With the Transmuter design, not only can volume limitations be enforced through 100% mint or burn fees beyond some exposures, there is also no need to update the fee values when liquidity in the system scales as these are based on relative exposures.

Note that it is still possible that exposures go over the bounds where for instance mint fees reach 100%. Reason is that when you burn for an asset, you're mathematically increasing the exposures to all other assets in the system.

EXAMPLE 3.2

> Assume the system is targeting a 33% maximum exposure for $\texttt{EUR}_A$, and so far 30 agEUR have been issued with $\texttt{EUR}_A$, 30 with $\texttt{EUR}_B$ and 40 with $\texttt{EUR}_{\texttt{yield}}$, then someone burning 15 agEUR for $\texttt{EUR}_{\texttt{yield}}$ would bring the exposure to $\texttt{EUR}_A$ to above 35%. It should be at this point impossible to mint agEUR with $\texttt{EUR}_A$, but burning agEUR for $\texttt{EUR}_A$ should come at a low cost.

The fact that exposures may periodically go over the desired ratio can only be detrimental to the system if an asset in the backing depegs after having gone over the target exposure, which supposes that someone burnt for "safe" assets before oracles actually noticed the depeg. Having minimum fees at the oracle deviation thresholds are a way to counter this.

In the Transmuter system, there can be negative fees to incentivize people to come with a certain asset. The system however verifies that this does not open arbitrage loops. It is impossible to set negative mint fees if these are in absolute value bigger than the positive burn fees for all the other assets in the system.

3.2 Mint

Denoting by d_i the deviation of asset i, the Transmuter system enables minting 1 stablecoin against a specific asset i by bringing:

$$\max(\frac{1}{\texttt{target price}_i}, \frac{1}{p_i}) = \frac{d_i}{p_i} + \frac{1}{t_i}$$

Precisely speaking, as the system is designed to take fees, the amount to bring is:

$$(1 + \texttt{fee})(\frac{d_i}{p_i} + \frac{1}{t_i})$$

This guarantees that if an asset in the backing depegs then it is not profitable to mint with this asset.

EXAMPLE 3.3

> Assume $\texttt{EUR}_A$ trades at 0.95€ and there are 0 fees, then $\frac{1}{0.95}$ $\texttt{EUR}_A$ are needed to get 1 agEUR.

3.3 Burn

The stablecoin can be burnt for any asset in the backing. Contrarily to the mint case, the price at which the stablecoin is burnt does not only depend on the price of the asset for which it is burnt, it also depends on the price of all the other stablecoins in the backing.

The system looks into the deviation d_i for all assets i in the backing with respect to their target price and then applies to the burn price a penalty equal to the largest deviation possible.

The burn price for an asset j among N assets is then: $\frac{1-\max_{i\in N} d_i}{p_j}$.

In its normal state, the stablecoin can be burnt for any of the assets in the system at their fair value which guarantees a small slippage for burning the asset. UX wise, this system is thus intended to be the most advantageous burning method if all p_i are close to their target value.

In case of a depeg of one of the asset in the backing or of a black swan, this mechanism is meant to preserve the system's exposures to all assets. As the stablecoin can be burnt for the same value of assets regardless of the asset it is burnt for, it disincentivizes stablecoin holders from rushing to exit towards the safest asset.

EXAMPLE 3.4

In the example with agEUR, users can burn 1 agEUR and redeem an amount of $\texttt{EUR}_A$ equal to:

$$\frac{1-\max(d_{\texttt{EUR}_A}, d_{\texttt{EUR}_B}, d_{\texttt{EUR}_{\texttt{yield}}})}{p_{\texttt{EUR}_A}} \times (1-\texttt{fee})$$

Now, consider the case where $\texttt{EUR}_A$ depegs by 5% but not $\texttt{EUR}_B$ or $\texttt{EUR}_{\texttt{yield}}$, then people burning stablecoins can choose to get 1 $\texttt{EUR}_A$ worth 0.95€. They can also choose to get 0.95 $\texttt{EUR}_B$ worth 0.95€, or 0.95 $\texttt{EUR}_{\texttt{yield}}$ worth 0.95€. In all cases, it's never profitable to burn agEUR for the assets that remain safe in the system.

3.4 Redeem

The third mechanism with which users can interact in Transmuter is the redemption. Inspired from Gyroscope [12], it allows users to redeem the stablecoin for a proportion of reserve assets minus a fee. This action is meant to be the privileged method of interaction with the protocol during black swan events (like a depeg of one of the stablecoins in the system) without affecting the risk or underlying exposure of any other stablecoin holder.

A fee can be applied depending on the collateral ratio with a peak that should be set just below 100% CR. This is meant to deter users from starting a bank run as they would have to pay slightly higher fees in case of small transitory depegs.

The collateral ratio is defined here as:

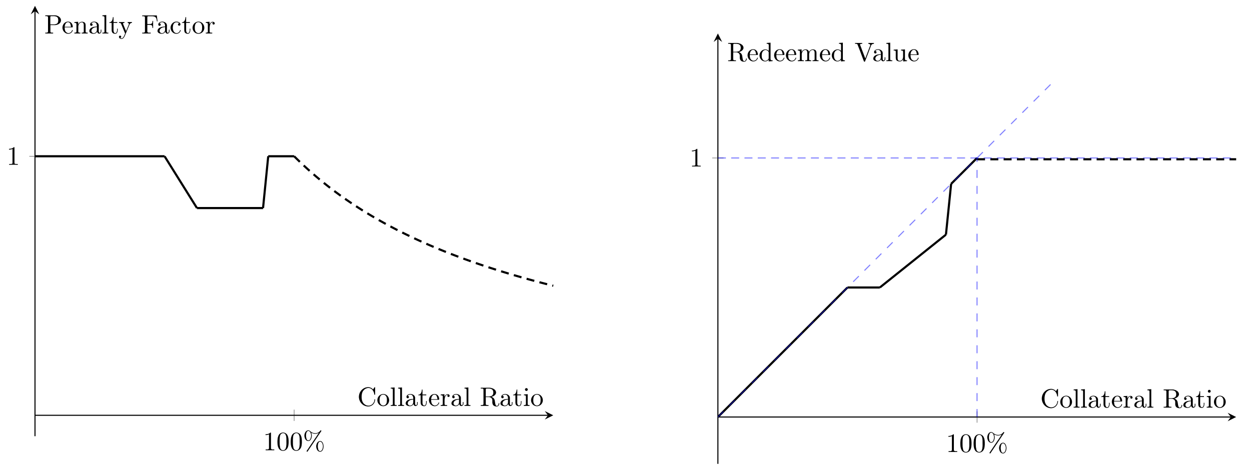
$$\texttt{CR} = \frac{\sum_{i\in N} \texttt{balance}_i \times p_i}{\texttt{total stablecoins issued by the system}}$$

Apart from the fees applied which depend on the collateral ratio and thus on prices, the amount redeemed of each token does not depend on the prices of the backing assets. The system simply exchanges a portion of the stablecoins issued against a portion of the reserves minus a penalty factor depending on the collateral ratio.

The amount of what can be redeemed of an asset i when redeeming amount in the reserves is computed as:

$$\frac{\texttt{amount}}{\texttt{total stablecoins issued}} \times \texttt{balance}_i \times \texttt{penalty factor}$$

The following figures show an example of evolution of the penalty factor depending on the collateral ratio, and the redeemed value that corresponds with this factor:



(a) Penalty Factor as a function of `CR`

(b) Redeemed Value depending on the `CR`

The goal of the decreasing redemption quotes as the collateral ratio gets a bit below 100% is to disincentivize bank runs and reward users who wait for a transitory downturn to pass in a sustainable way. Users who still want to exit when the protocol gets badly collateralized may still exit, but by doing so they are re-collateralizing the protocol, thus leaving better off the users who remained in the system.

Below a certain level of collateral ratio, the penalty factor goes back to 1. This is to enable all users to exit at the fair value of the reserves when the protocol is in an irremediably bad health.

Note that the reason why the penalty factor curve is hyperbolic when the collateral ratio gets over 100% is to prevent users from redeeming any excess collateral from the protocol.

EXAMPLE 3.5

Consider a state where 100 stablecoins have been issued, and there are 30 $\texttt{EUR}_A$, 30 $\texttt{EUR}_B$ and 40 $\texttt{EUR}_{\texttt{yield}}$ in reserves and $\texttt{EUR}_A$ depegs by 5%, the collateral ratio becomes 98.5%. If in this case the penalty is set to be 0.98, a user redeeming 10 agEUR gets 2.94 $\texttt{EUR}_A$ worth 2.793€, 2.94 $\texttt{EUR}_B$ worth 2.94€, and 3.92 $\texttt{EUR}_{\texttt{yield}}$ worth 3.92€.

The user got 9.653€ in value, which is actually less than the 9.85€ they should have obtained if they had been settled at the value of the collateral ratio: the protocol is partly re-collateralized by the redemption, and the redemption increased the overall health of the system. This is the kind of mechanism that could have been applied by Maker during the USDC depeg.

In case of a black swan event or a depeg of a stablecoin in the backing, this redemption feature guarantees that all users are treated fairly. The fact that with a redemption the backing decreases in proportion to the reserves and that no governance is needed to activate this feature ensures that there cannot be any form of sequentiality in place where the first arrived for redemptions end up better off than the others. It makes the system highly predictable in the face of downturns and enables every market participant to have a high degree of clarity of what the optimal behavior with respect to the protocol should be. Protocol stakeholders do not have to anticipate what would be the outcome of governance decisions and how they should position themselves in the wake of a depeg of a reserve asset regarding a potential settlement.

Note that Transmuter could just work with the mint and redemption functionalities. The burn function is more to be seen as a UX facilitator for users who do not want to end up with a multitude of tokens when redeeming. In most circumstances, and when exposures to different assets are close to their targets (see section 3.1), the protocol is better off with users redeeming rather than burning for one asset. To this extent, the redemption method should be the cheapest and the privileged method of interaction for advanced traders who can handle multiple tokens at once.

Figure 5 in appendix 7 compares the effect of burning with respect to redeeming in different collateral ratio settings.

4 Implementation

4.1 Collateral management and re-collateralization

While collateral assets in Transmuter remain by default idle in the contract, the system supports the possibility to plug manager contracts or strategies to generate a yield on the assets in the backing.

This is something which is optional and that can be activated on a per asset level. Investing a portion of the reserves of Transmuter in yield strategies can enable it to get over-collateralized over time and to build a safety buffer for future depegs of assets in the system. If the protocol gets badly collateralized, this kind of mechanism may also help the protocol slowly re-collateralize itself.

This comes with an extra composability risk if the underlying protocols in which the strategies invest get hacked or if the strategies make a loss. To maintain the invariant about redemptions and eliminate bank run effects in case of a loss in a strategy, such manager contracts must send the underlying tokens they are controlling during a redemption. The oracles for assets which are invested in yield strategies should also take into account the state of the strategies in which they are invested.

EXAMPLE 4.1

> For instance if $\texttt{EUR}_A$ is invested on Aave [13], the protocol will control $a\texttt{EUR}_A$. During a redemption, the protocol should send users a basket of $\texttt{EUR}_A$ and $a\texttt{EUR}_A$ based on what is invested. If Aave gets hacked, this guarantees that first users will not be getting more than the ones coming afterwards. The deviation value in the oracle for $\texttt{EUR}_A$ would also need to take into account the fact that the system does not only have $\texttt{EUR}_A$ but also $a\texttt{EUR}_A$ which decreased in value.

4.2 Reactive fees

Fees for a specific action are defined by the values they should take at certain exposures.

EXAMPLE 4.2

> For instance, the system may be set such at exposure 0% for $\texttt{EUR}_A$ after a mint, mint fees must be $f = 0.1\%$, at 30% $f = 0.3\%$, at 32% $f = 0.34\%$ and at 40% $f = 100\%$.

The tendency would be to define the fee curve as a piecewise linear function of the exposure and take the mean rate over the amounts between the start and the end exposure. Yet in this case, fees would not be linear on the amounts swapped, the system would be path dependent and users may be incentivized in some circumstances in splitting their orders in multiple sub-orders.

The Transmuter system is implemented such that in the same block (and putting gas cost considerations aside), splitting an order in multiple sub orders involving the same asset gives exactly the same output as making one single order. On top of that, it also enables people to specify when minting or burning whether they want to get an exact amount of tokens in output or bring an exact amount of tokens in input, with both methods being purely equivalent.

EXAMPLE 4.3

> Let's say that someone wants to burn 100 agEUR. Fees in Transmuter are computed such that in a

> single block, burning 100 agEUR yields exactly the same as burning 50 and then 50.

For more details on how fees are computed and path independence is achieved, check Appendix (section 6).

4.3 External reserve updates

Transmuter supports external systems that can update the value of the amount of stablecoins issued by each asset in the backing. This is useful if a loss or a profit is made in some other place of the protocol and the global collateralization ratio needs to be updated.

EXAMPLE 4.4

> Imagine a system for an ETH stablecoin that takes liquid staking derivative tokens as an asset in the backing. In normal conditions, the value of the reserves increases with respect to the value of the stablecoins in circulation. An idea can then be to distribute the upside in collateral value by minting more of the ETH stablecoin through an ERC4626 [14] savings contract. As the stablecoins given as a profit are not issued through Transmuter but by another module in this case, the collateralization metrics of Transmuter must be updated to preserve the accounting of the system.

4.4 Composability with other stablecoin systems

Transmuter is a stablecoin system. It can be a full protocol working on its own, but it can also be used as a module (or facilitator [15]) within a bigger stablecoin protocol.

Transmuter has its own internal accounting metrics, and it never looks into the `totalSupply` of the stablecoin it is minting. This means that it is possible to have other smart contracts with a minting right on the stablecoin handled by Transmuter without breaking the logic of the system.

Transmuter is particularly well suited to work in parallel with borrowing modules (i.e systems relying on collateralized-debt positions), with flash-loan systems [16] enabled at the stablecoin level, or with direct deposit modules (often referred to as algorithmic market operations [17]). It is even possible to adapt Transmuter so that the accounting of direct deposit modules is taken into account in the logic of burns and redemptions.

EXAMPLE 4.5

> Imagine a USD stablecoin protocol that accepts DAI and USDT as collateral in Transmuter and that mints through another module the stablecoin in a Curve pool whenever there are more USDC than of the stablecoin in the pool (like what FRAX is doing). If USDC was to depeg and if the module minting on Curve exists in parallel to Transmuter, Transmuter would guarantee that the protocol keeps fixed exposures with respect to DAI and USDT relatively, but it could not do anything with respect to the USDC exposure of the stablecoin. Onboarding within Transmuter direct deposit modules like the ones that invest liquidity on AMMs can help avoid this kind of situation and maintain the guarantee that protocols do not end up with only "bad" collateral in reserves in case of a black swan event. In this setting, having "onboarded" direct deposit modules would mean that each of them would have a maximum exposure above which they cannot mint more stablecoins and that users redeeming would also get in the process LP tokens from the direct deposit modules on top of the other supported collateral assets.

4.5 Gas optimizations

4.5.1 Diamond proxy pattern and oracles

The Transmuter system works with external oracles to price the reserve assets. To avoid exploits, fees for each asset should be set with the oracle deviation thresholds of all other assets in mind.

Because the burn and redemption operations imply reading into the oracles of all the assets in the system, gas costs scale linearly with the amount of assets accepted in the system. Transmuter is implemented as a single contract relying on a diamond proxy pattern [18] so the logic associated to the oracle for each asset can all be taken into account in the same contract thus reducing gas costs for calling oracles for each asset.

Beyond this, the diamond proxy pattern opens some composability and flexibility in how the system is implemented. Transmuter could read into several oracles (like UniswapV3 TWAPs [19] and Chainlink [20]) for a single asset, and take the value that is most at its advantage.

In fact, any oracle type can be supported provided that the values given for both the target price and for the current price are both non manipulable.

4.5.2 Smart accounting

To track the exposure to each asset in the backing and compute the value of the adaptive fees, the system needs to store the amount of stablecoins that have been issued from each of the assets in the backing.

During a redemption, stablecoins are burnt for a portion of each asset. If there are N assets, then it means that N values in storage must be updated. To avoid the multiplication of storage updates, Transmuter relies on a `normalizer` variable.

If x_i stablecoins are issued from asset i, then Transmuter tracks the amount of stablecoins issued from i by keeping in storage: $r_i = \frac{x_i}{\texttt{normalizer}}$.

If someone redeems y stablecoins out of the Y that had been issued by the system, the `normalizer` variable becomes: $\texttt{normalizer} \times (1 - \frac{y}{Y})$.

After the `normalizer` update, the amount of stablecoins issued from asset i becomes:

$$\frac{x_i}{\texttt{normalizer}} \times \texttt{normalizer} \times (1 - \frac{y}{Y}) = x_i(1 - \frac{y}{Y})$$

4.6 Whitelisted collateral assets

Transmuter is compatible with assets for which holders may be permissionned. There is the option to make some specific collateral assets whitelisted, in the sense that only some addresses are eligible to burn for this asset or receive it during a redemption. This could typically apply to the case of security tokens which cannot be permissionlessly sent to any address.

While this remains a feature that should not be used by default, the consequence of introducing such whitelisted assets in the system is that the full expressivity and fairness of Transmuter is only available to addresses who are whitelisted.

If there is a whitelisted collateral in the system representing 20% of the reserves and the protocol is collateralized at 100%, then a non whitelisted address will only be able to get 80% of the value of the stablecoin during a redemption (as it will not be eligible to receive the whitelisted token), while a whitelisted address will get 100% of it. Note that non-whitelisted addresses can still burn in this case for collateral assets that do not require the whitelist.

As the redemption feature is mostly designed for advanced market makers and arbitrageurs who should be whitelisted anyway, having whitelisted collateral assets in the system will induce more friction in the redemption process but this will not impact the Transmuter's ability to get the stablecoin priced at the fair value of its reserves on the secondary market.

Different types of whitelists may apply depending on the assets. And the management of whitelists may be delegated to protocol governance or to external trusted systems.

4.7 Governance considerations

The main functionalities of the Transmuter system can work autonomously with no governance involved. As soon as fees are set, the system can naturally converge close to the desired exposures without any manual intervention from anyone. In case a stablecoin in the backing depegs from its target value, the redemption curve ensures an autonomous and natural fallback mechanism.

Transmuter is however not a governance-free system. A governance must be involved to add new collateral assets, to adjust the fee parameters that determine the target and maximum exposures to each asset, to change the different oracle parameters, to modify the settings of the redemption curve, or to simply pause minting or burning from an asset. To keep the system predictable to all stakeholders as it is designed, governance modules plugged to Transmuter should implement proper timelocks before making any change.

Given an initial setup with preset parameters and collateral types, the Transmuter system can still be made fully immutable.

5 Conclusion

The Transmuter system introduced here, thanks to its mint, burn and redemption functionalities has among other things the following desired properties:

- **UX**: The system enables minting and burning with limited fees from a wide range of assets.
- **Scalable**: It works similarly with \$1m TVL as with \$1bn TVL
- **Resilient**: Its underlying mechanisms are all fully autonomous and hence predictable by all type of stakeholders. In case of a black swan event, Transmuter provides reasons to bet on the stablecoin returning to its target price.
- **Trustless**: In the same vein, no governance is needed to activate the features protecting the system in case of a depeg.
- **Fair**: There cannot be any bank run as redemptions are thought to break sequentiality between users.
- **Robust**: Transmuter can be used as a basket of different stablecoins or assets allowing collateral risk to be well diversified.
- **Safe**: If fees are properly set, the design provides incentives to bring the basket of reserves to a target desired allocation. In case of a depeg, it is unprofitable to perform trades that leave the protocol holding the weakest asset (contrary to what happened to Maker during the USDC depeg).
- **Gas-efficient**: Thanks to a range of different optimizations, the system is able to minimize the gas needed to interact with it.
- **Modular**: The system can not only accept any type of asset in the backing, its implementation is such that it can work for any type of stablecoin. For agEUR, it could for instance work with in the backing assets like EUROC, EURe [21] or EUROe [22]. At the same time, it could also be used to make a basket of different ETH liquid staking derivative tokens, conceived as safer than any of the tokens in the backing.

6 Appendix A: Fee computation and path independence

Here we explain how the Transmuter system manages to derive mint and burn fees efficiently, and we show how some properties like path independence 4.2 can be derived from the computing methodology used.
We assume that:

- There are N supported assets in Transmuter.
- A normalized amount of r_j stablecoins have been issued from each asset j.
- Governance has specified a set of exposures $(e_i)_j$ for each asset j for which mint fees should be equal to $(f_i)_j$.
- Governance has specified another set of exposures $(e'_i)_j$ for each asset j for which burn fees should be equal to $(f'_i)_j$.

6.1 Deriving Mint and Burn fees

Whenever a mint or burn occurs, the contract recomputes a piecewise linear function g of the amount minted or burnt such that when the exposures for which fees are known are reached, the fees are exactly as specified.

Let us assume we need to compute fees for a given asset j. For simplicity, we'll write r instead of r_j, f_i instead of $(f_i)_j$, etc. We furthermore assume that the exposure is e with $e_i \leq e \leq e_{i+1}$, and that the length of the set of exposures (e_i) is l. Let us introduce the different parts g_i of the mint fee function, and we'll denote by b_i the change in the number of stablecoins minted using the asset required to change the exposure from e to e_i.

- b_i will be negative as $e \geq e_i$. It corresponds to a burn using the asset.

$$\frac{r + b_i}{\sum_k r_k + b_i} = e_i \iff b_i = \frac{e_i \sum_k r_k - r}{1 - e_i} = \frac{e_i \sum_{k \neq j} r_k}{1 - e_i} - r \tag{1}$$

- b_{i+1} will be positive as $e_{i+1} \geq e$. It corresponds to a mint using the asset.

$$\frac{r + b_{i+1}}{\sum_k r_k + b_{i+1}} = e_{i+1} \iff b_{i+1} = \frac{e_{i+1} \sum_k r_k - r}{1 - e_{i+1}} = \frac{e_{i+1} \sum_{k \neq j} r_k}{1 - e_{i+1}} - r \tag{2}$$

- The b_i values are all increasing.
- For $b_i \leq x \leq b_{i+1}$, the mint fee function is linear and such that:

$$g_i(x) = \frac{x - b_i}{b_{i+1} - b_i}(f_{i+1} - f_i) + f_i \tag{3}$$

The overall mint fee function of the amount x of stablecoins minted writes:

$$g(x) = \sum_{j=i}^{l} \left(\frac{f_{j+1} - f_j}{b_{j+1} - b_j}(x - b_j) + f_j\right) \mathbf{1}_{b_j \leq x < b_{j+1}} \tag{4}$$

In the case of a burn operation, a similar g function can be defined, if we compute the values b_i considering that the amounts swapped are positive for a burn and negative for a mint and that the exposures (e'_i) given by governance are decreasing, such that $\forall i \leq l', e'_{i+1} \leq e'_i$.

With this in mind, the fee to pay when minting or burning M stablecoins can be computed with a basic integral:

$$G(M) = \int_0^M g(x)dx \tag{5}$$

where the g function depends on the initial exposure to the asset j, and on whether the operation is a mint or a burn.

Assuming that the oracle value is 1, someone minting M stablecoins with the asset j must then bring $M + G(M)$. And someone burning M stablecoins for asset j when all assets are at their target price may get $M - G(M)$.

The computation of the integral G can be done efficiently and for cheap on-chain with the following algorithm.

Let us place ourselves in the case of a mint transaction of M stablecoins with asset n for which initial exposure $e \in [e_i, e_{i+1}]$. We start by initializing a counter $c = M$ for the amount of assets to bring. Then:

- If $0 \leq M \leq b_{i+1}$: the minting fee counter c needs to be updated by the integral of the g function between 0 and M. As this function is linear in this segment, it can be easily computed such that $c \mathrel{+}= M\frac{g(0)+g(M)}{2}$.
- Else: The minting cost c must be increased by the cost of minting b_{i+1} stablecoins, that is: $c \mathrel{+}= b_{i+1}\frac{g_i(0)+g_i(b_{i+1})}{2}$. After this, the *if* condition above can be called with $M \mathrel{-}= b_{i+1}$ and $i \mathrel{+}= 1$. What this step practically means is that if the amount of asset brought makes the amount of stablecoins minted so big that it brings the exposure into the next segment: $[e_{i+1}, e_{i+2}]$, the swap can virtually be swapped in two: one swap to fill this segment, and another swap for what remains.

With this method, both mint and burn fees can be computed iteratively on-chain with a computation cost that is linear in the length of the set of exposures given by governance when specifying fees.

6.2 Path independence

This method enables mints and burns within Transmuter to be path independent. In our case, the path independence property consists in saying that, putting gas costs aside, the fees paid for doing a swap (mint or burn) of M stablecoins are the same as the fees paid for doing a swap of u stablecoins followed by a swap of v with $u + v = M$.

Mathematically, this translates into $G(M) = G^{(0)}(u) + G^{(1)}(v)$, where $G^{(1)}$ represents the new fee structure after the swap of u stablecoins.

Let us consider here the simple case of a mint where $M < b_{i+1}$ and $e \in [e_i, e_{i+1}]$, meaning the operation doesn't make the exposure bigger than e_{i+1}. We have:

$$G^{(0)}(u) + G^{(1)}(v) = \int_0^u g^{(0)}(x)dx + \int_0^v g^{(1)}(x)dx \tag{6}$$

And:

$$g^{(1)}(x) = \frac{f_{i+1} - f_i}{b^1_{i+1} - b^1_i}(x - b^1_i) + f_i = \frac{f_{i+1} - f_i}{b^0_{i+1} - u - b^0_i + u}(x + u - b^0_i) + f_i = g^{(0)}(x + u) \tag{7}$$

This gives:

$$\begin{aligned} G^{(0)}(u) + G^{(1)}(v) &= \int_0^u g^{(0)}(x)dx + \int_0^v g^{(0)}(x+u)dx \\ &= \int_0^u g^{(0)}(x)dx + \int_u^{u+v} g^{(0)}(x)dx \\ &= G^{(0)}(M) \end{aligned}$$

Any swap can be split in multiple infinitesimal swaps and still yield the exact same output.

6.3 Dealing with reflexivity

The algorithm detailed above 6.1 is applicable when the quantity of stablecoins intended for acquisition or burning is predetermined. As explained here 4.2, Transmuter is designed in a manner that it is also possible to specify a precise amount of assets for minting or to request an exact amount of assets when burning. Both options are perfectly analogous to their respective equivalent, that is: requesting a specific quantity of stablecoins during a minting operation, or providing a designated amount of stablecoins for a burning process.

Achieving equivalence in the cases of someone coming with (mint) or requesting (burn) an amount of assets of value M implies computing an amount m_t of stablecoins such that:

- For a mint: $m_t + G(m_t) = M$
- For a burn: $m_t - G(m_t) = M$

To compute the amount of stablecoins m_t to mint if $e \in [e_i, e_{i+1}]$, we start by initializing m_t as a counter to 0, and then:

- If $M \leq b_{i+1} \times (1 + \frac{g_i(0)+g_i(b_{i+1})}{2})$, or in other words if M is lower than the value of assets we'd need to make the exposure greater than e_{i+1}, then the m_t counter can be increased by the solution $m^*_{t_+}$ to:

$$M = m_t \times (1 + \frac{g_i(0) + g_i(m_t)}{2}) = m_t \times (1 + g_i(0) + \frac{f_{i+1} - f_i}{b_{i+1} - b_i}\frac{m_t}{2})$$
$$\iff m^*_{t_+} = \frac{-1 - g_i(0) + \sqrt{(1 + g_i(0))^2 + 2M\frac{f_{i+1}-f_i}{b_{i+1}-b_i}}}{\frac{f_{i+1}-f_i}{b_{i+1}-b_i}}$$

- Else, we can increase m_t by b_{i+1} and test again the *if* condition above with a value of assets brought equal to $M - b_{i+1}(1 + \frac{g_i(0)+g_i(b_{i+1})}{2})$ and with i += 1.

In the burn case, the algorithm is slightly different. To compute the amount m_t of stablecoins to burn if $e \in [e_i, e_{i+1}]$, we initialize $m_t = 0$ and:

- If $M \leq b_{i+1} \times (1 - \frac{g_i(0)+g_i(b_{i+1})}{2})$, or in other words if M is lower than the value of assets we'd need to get to make the exposure smaller than e_{i+1}, then the m_t counter can be increased by the solution $m^*_{t_-}$ to:

$$M = m_t \times (1 - \frac{g_i(0) + g_i(m_t)}{2}) = m_t \times (1 - g_i(0) - \frac{f_{i+1} - f_i}{b_{i+1} - b_i}\frac{m_t}{2})$$
$$\iff m^*_{t_-} = \frac{1 - g_i(0) - \sqrt{(1 - g_i(0))^2 - 2M\frac{f_{i+1}-f_i}{b_{i+1}-b_i}}}{\frac{f_{i+1}-f_i}{b_{i+1}-b_i}}$$

- Else, we can increase m_t by b_{i+1} and test again the *if* condition above with a value of assets obtained in the end equal to $M - b_{i+1}(1 - \frac{g_i(0)+g_i(b_{i+1})}{2})$ and with i += 1.

Note that the computations needed in both the mint and burn algorithm can be further simplified by taking into account the fact that:

$$\frac{f_{i+1} - f_i}{b_{i+1} - b_i} = \frac{f_{i+1} - g_i(0)}{b_{i+1}}$$

In all cases, the resolution methods introduced here guarantee that, it is the same to specify an amount of tokens (stablecoin or collateral) to get or to specify an amount of tokens (collateral or stablecoin) to bring when performing a mint or burn operation with Transmuter.

6.4 Direct resolution for the mint/burn fee computation

We introduced at section 6.1 an iterative algorithm to compute the mint/burn fees depending on the amount of stablecoins minted. Here, we are providing more theory on why the algorithm effectively works to compute $G(M)$.

If $b_j \leq M \leq b_{j+1}$, and if $e \in [e_i, e_{i+1}]$, starting from 4 and 5:

$$\begin{aligned}
G(M) &= \int_0^M \sum_j \left(\frac{f_{j+1} - f_j}{b_{j+1} - b_j}(x - b_j) + f_j\right) \mathbf{1}_{b_j \leq x < b_{j+1}}\, dx \\
&= \sum_{k=i}^{j-1} \left(\int_{b_k}^{b_{k+1}} g_k(x)dx\right) + \int_{b_j}^{M} g_j(x)dx \\
&= \sum_{k=i}^{j-1} \frac{g(b_k) + g(b_{k+1})}{2}(b_{k+1} - b_k) + \frac{g(b_j) + g(M)}{2}(M - b_j) \\
&= \sum_{k=i}^{j-1} \frac{f_k + f_{k+1}}{2}(b_{k+1} - b_k) + \frac{f_j + g(M)}{2}(M - b_j)
\end{aligned}$$

The computation of these integrals/sums above is what is performed in the *if*/*else* condition of the algorithm 6.1 to compute mint and burn fees.

7 Appendix B: Examples and Illustrations

A detailed spreadsheet with various examples from this whitepaper can be found here. You may fork it to try out your own examples.

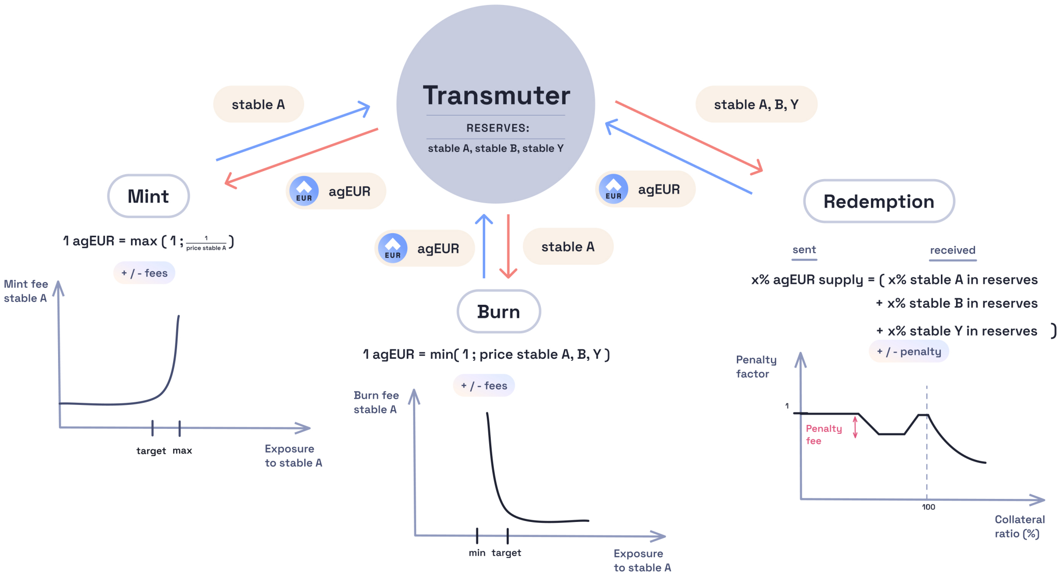


Figure 3: The different features of the Transmuter system

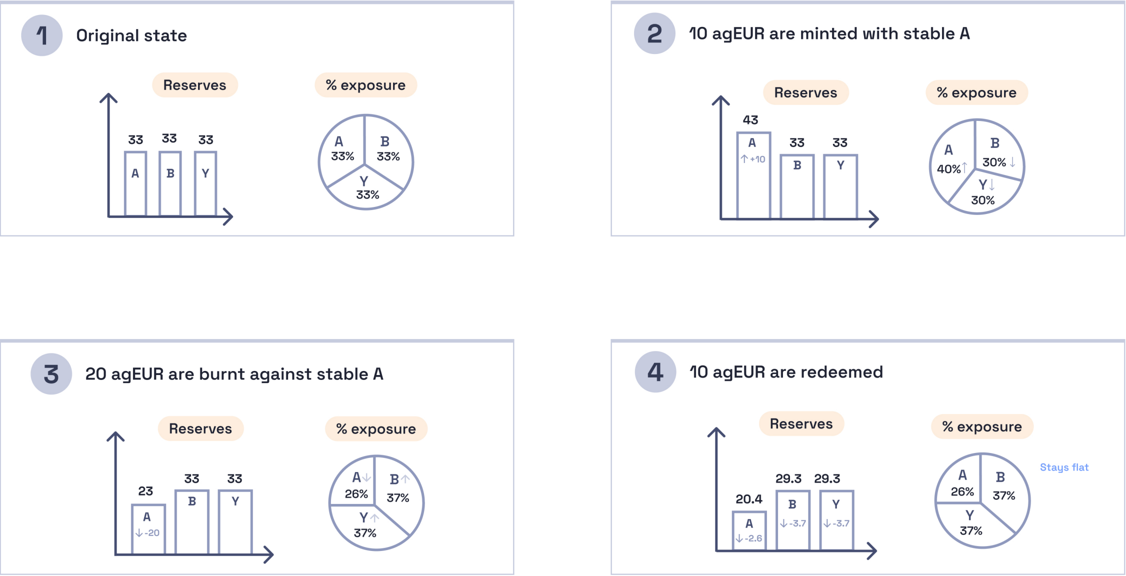


Figure 4: Impact of the different Transmuter actions on the system's reserves

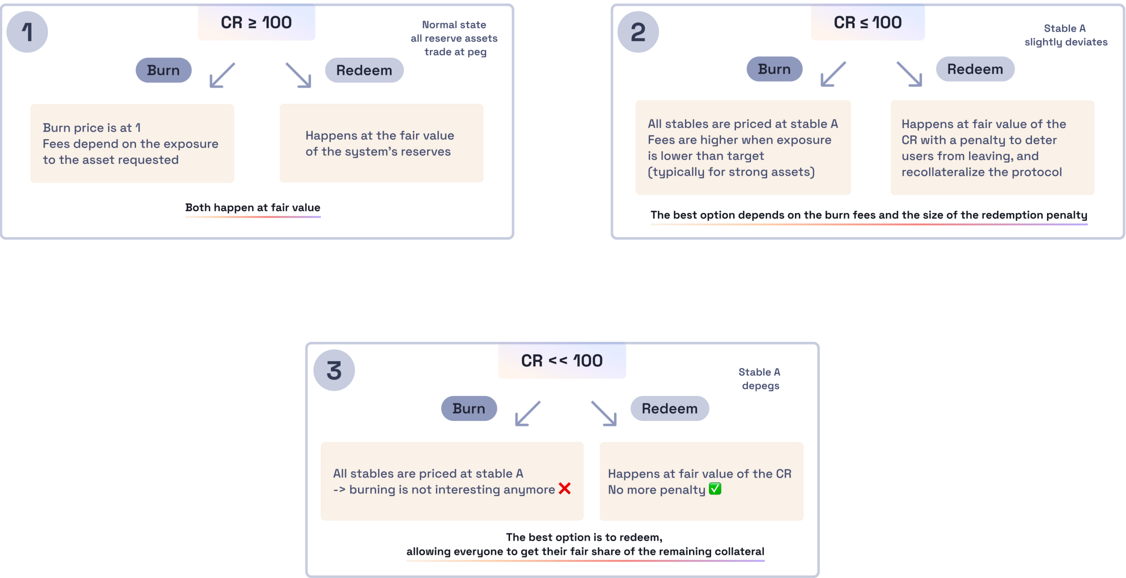

1
CR ≥ 100
Normal state
all reserve assets
trade at peg
Burn
Redeem
Burn price is at 1
Fees depend on the exposure
to the asset requested
Happens at the fair value
of the system's reserves
Both happen at fair value
2
CR ≤ 100
Stable A
slightly deviates
Burn
Redeem
All stables are priced at stable A
Fees are higher when exposure
is lower than target
(typically for strong assets)
Happens at fair value of the
CR with a penalty to deter
users from leaving, and
recollateralize the protocol
The best option depends on the burn fees and the size of the redemption penalty
3
CR << 100
Stable A
depegs
Burn
Redeem
All stables are priced at stable A
-> burning is not interesting anymore ❌
Happens at fair value of the CR
No more penalty ✅
The best option is to redeem,
allowing everyone to get their fair share of the remaining collateral

Figure 5: Comparing burning and redeeming in different situations

8 Acknowledgements

With special thanks to Sarah Duclent, Tuta, GreedyThib, Sébastien Derivaux, Adrien Husson, Dave Montali, Julien Thévenard and 0xValJohn.

References

[1] Maker DAO's Multi-Collateral DAI (MCD) System. https://makerdao.com/en/whitepaper/#abstract. [Online; accessed 4-May-2023].

[2] Maker Peg Stability Module. https://mips.makerdao.com/mips/details/MIP29. [Online; accessed 4-May-2023].

[3] Maker DAO. https://makerdao.com/en/. [Online; accessed 4-May-2023].

[4] Circle. https://www.circle.com/en/. [Online; accessed 4-May-2023].

[5] FRAX Finance. https://frax.finance. [Online; accessed 4-May-2023].

[6] M. Egorov. Curve: StableSwap Whitepaper. https://berkeley-defi.github.io/assets/material/StableSwap.pdf. [Online; accessed 4-May-2023].

[7] Angle Core Module. https://docs.angle.money/overview/whitepapers#angle-core-module. [Online; accessed 4-May-2023].

[8] Maker DAO Analytics. https://daistats.com/#/overview. [Online; accessed 4-May-2023].

[9] Gemini Dollar. https://www.gemini.com/dollar. [Online; accessed 4-May-2023].

[10] Lido: Liquidity for staked tokens. https://lido.fi. [Online; accessed 4-May-2023].

[11] 1inch: Decentralized Exchange Aggregator. https://1inch.io. [Online; accessed 4-May-2023].

[12] Gyroscope Documentation. https://docs.gyro.finance/gyroscope-protocol/readme. [Online; accessed 4-May-2023].

[13] Aave Protocol. https://aave.com. [Online; accessed 4-May-2023].

[14] J. Santoro, t11s, J. Jadeja, A. Cuesta Cañada, and S. Doggo. EIP4626: Tokenized Vaults. https://eips.ethereum.org/EIPS/eip-4626. [Online; accessed 4-May-2023].

[15] Stablecoin facilitators by Aave GHO. https://docs.gho.xyz/concepts/how-gho-works/gho-facilitators. [Online; accessed 4-May-2023].

[16] Flash-Loans by Aave. https://docs.aave.com/developers/guides/flash-loans. [Online; accessed 4-May-2023].

[17] Algorithmic Market Operations by FRAX. https://docs.frax.finance/amo/overview. [Online; accessed 4-May-2023].

[18] N. Mudge. Diamond Proxy EIP. https://eips.ethereum.org/EIPS/eip-2535. [Online; accessed 4-May-2023].

[19] H. Adams, N. Zinsmeister, M. Salem, R. Keefer, and D. Robinson. Uniswap V3 Whitepaper. https://uniswap.org/whitepaper-v3.pdf. [Online; accessed 4-May-2023].

[20] Chainlink: Decentralized Data Feeds. https://chain.link/data-feeds. [Online; accessed 4-May-2023].

[21] Monerium: Euros for Web3. https://monerium.com. [Online; accessed 4-May-2023].

[22] EUROe: A modern European stablecoin. https://www.euroe.com. [Online; accessed 4-May-2023].