Frankencoin

Luzius Meisser[*]
University of Zurich

Basile Maire[**]
Desma Eight, LLC

May 23, 2022

[*]`luzius.meisser@uzh.ch`
[**]`basile.maire@desma8.io`

Frankencoin

Abstract

We present an open architecture to create a decentralized, fully collateralized stablecoin that does not rely on external price oracles. Collateralization is maintained through a novel liquidation mechanism that is based on auctions and flexible with regards to the collateral used. The long-term fundamental value that mirrors that of the reference currency is obtained via collateralization and trust in the stakeholders to align the return of holding Frankencoin with that of the Swiss franc. Market forces, rather than an oracle-based conversion mechanism, maintain the short term peg. We assess the risks of the system through the lens of a bank regulator, using historical data. The name Frankencoin is both a reference to our initial reference currency, the Swiss franc, as well as to the system's self-governing nature.

JEL Classification Codes: D40, D44, E42, G23, G32

Key Words: digital currencies, stablecoins, auctions, risk management, decentralized systems

1 Introduction

We start from the use-case of a Swiss franc Lombard loan, collateralized with tokens such as crypto-currencies or tokenized securities. The borrower deposits collateral into the system and thereby gains the possibility to mint a token, the “Frankencoin” ZCHF, that is pegged to the Swiss franc. If the value of the borrower’s collateral falls below a certain threshold, the loan can be liquidated and the borrower incurs a haircut. If the value of the collateral falls below the loan value before the liquidation ends, the loss eats into the system’s capital reserve. Implementing this system in a fully decentralized way on the blockchain leads us to our main contributions:

– A modular approach that allows anyone to propose new minting mechanisms in the form of a "mint plugin", resulting in a diversified loan collateral mix

– Mint plugins allow for oracle-free collateralized loans, based on a auction-based liquidation mechanism that is suitable for both, liquid and illiquid collateral

– Capital requirements inspired by traditional banking rules and clearing house practices

In comparison to other decentralized stablecoins, the Frankencoin stands out with its simplicity and versatility, which is achieved by having a basic but extensible setup that relies on the overarching economic incentives of the system participants instead of strictly enforcing a narrow peg with technical means.

1.1 Existing Systems

This section describes existing stablecoins and how they solve the challenge of having a stable value compared to a reference currency. In comparison to the Frankencoin, most stablecoins are more directly concerned about steering the short-term peg to the reference currency at small deviation, whereas the Frankencoin aims at creating a credible fundamental value in the long term and relies more on market forces to keep the exchange rate close to the fundamental value in the short term.

1.1.1 Stabilization Methods

The largest two stablecoins, the Tether (USDT) and the USD Coin (USDC) are based on the promise of an issuer to always sell and repurchase them at the price of the reference currency, thereby enforcing a 1:1 peg for as long as the issuer is solvent.[1] There is also a Crypto Franc (XCHF) tracking the Swiss franc using this approach.[2] In the background, the issuer can apply any stabilization mechanism the law allows. Currently, regulators are prohibitively restrictive when it comes to using cryptocurrencies as a collateral. Creators of decentralized stablecoins can be much more creative as their coins usually fall outside the scope of banking regulation, at least for now. At first sight, this might be seen as unfair, but there are also fundamental reasons why a fully transparent blockchain-based stablecoin can be inherently more stable than an opaque issuer-backed stablecoin, justifying less strict regulation.

[1] `https://tether.to/` and `https://www.circle.com/en/usdc`
[2] `https://www.bitcoinsuisse.com/cryptofranc`

At an initial version of this paper, the third largest stablecoin was the TerraUSD (UST), belonging to the class of *algorithmic stablecoins*.[3] The TerraUSD was built on the Terra blockchain with a built-in oracle at the protocol level. The blockchain validators had to provide quotes for the supported currencies and were punished if they failed to do so or if their quotes were too far from the quotes of the other validators. The basis currency is the Luna, and the stabilization mechanism was based on a mechanism to print Luna tokens to buy TerraUSD in case it falls below the price of the reference currency, and to do the opposite if the price of the TerraUSD is too high. The stability of the TerraUSD rested on the assumption that the fundamental value of the Luna token does not fall too fast when more have to be printed to stabilize the system. Similarly to the currency of a country, this works well for as long as the economy around the Luna token, namely everything that happens on the Luna blockchain, is strong enough to support its value. However, it can collapse quite fast once the market loses trust in its stability, as we have seen with the spectacular collapse in May, 2022. Another similarly designed token, the Iron Coin, faced the same fate. The Iron coin that was supposed to track the USD dollar was supposed to be stabilized by printing Titanium tokens whenever needed. For a critical perspective on algorithmic stablecoins, written before the crash of Luna, see for example Clements (2021).

An additional class of stablecoins are based on a blockchain-based collateral, known example being the DAI and the Liquity USD (LUSD).[4] Typically they let anyone deposit a collateral and then mint a certain quantity of the stablecoin. If the value of the collateral falls below a critical threshold, the deposit is liquidated and the proceeds are used to repay the minted coins. This is the principle that we apply to the Frankencoin, with the major difference being that the Frankencoin is much more adaptable to different collateral types, as it introduces a liquidation mechanism that does not rely on the presence of an oracle.

1.1.2 Problems with Oracles

A blockchain oracle is a service that connects blockchain-external data feeds, such as e.g., price data or temperature data, with smart contracts. When a decentralized stablecoins relies on a price oracle, this introduces an external dependency and potentially leads to centralization. For example, when analyzing the currently most popular oracle in the Ethereum system, Chainlink, one finds that its administrators could collude to manipulate prices. While the price feeds offered by Chainlink are the median of often more than a dozen independent oracles, configuring the feed only requires the signatures of three of its administrators, enabling them to potentially manipulate prices if they would like to do so.[5] But even if the administration was more decentralized, one would still have to trust the independent price sources to not collude. History shows that sometimes even the most reputable institutions cannot resist doing some price manipulation when the incentive to do so is big enough. One famous example is the LIBOR scandal, see for example Wheatley (2012).

A more robust type of oracles is exchange data based on actual trades. Here, the challenge is to find an exchange that can not only be relied on to have high liquidity today, but also for the full life-time of the Frankencoin or the respective bridge contract. Building a robust mint plugin based on the price-feed of a decentralized exchange would require a fallback mechanism in case liquidity

[3]`https://www.terra.money/`

[4]`https://makerdao.com/en/` and `https://www.liquity.org/`

[5]For example, the Chainlink price feed for the ETH / USD price pair found at `data.chain.link/ethereum/mainnet/crypto-usd/eth-usd` is based on smart contract 0x5f4ec3df9cbd43714fe2740f5e3616155c5b8419, which is currently controlled by a multi-signature contract that requires 3 out of 19 signers to approve transactions.

on that exchange falls below an acceptable level. This is easier said than done as trading volumes can be manipulated. That is why in this article, we prefer other approaches for price discovery.

1.2 Structure of the Paper

The paper is structured as follows. In Section 2, the reader is introduced to the cornerstone of the paper, namely a method for Oracle-free collateralized minting based on auctions. We describe how minters can mint their own Frankencoins in exchange for plugin-specific collateral. In Section 3, we introduce a reserve pool to address the residual risk of the system not being able to liquidate collateral at a price high enough to repay an open position. This pool is capitalized by the minters as well as voluntary contributors seeking a return on their Frankencoin holdings. With these tools, it is possible to create a system that can withstand adverse market events. Next, we turn our attention to how the peg to the Swiss franc is maintained in Section 4. This completes the overall design of the Frankencoin, allowing us to move our attention to the real-world implementation in Section 5 and concluding with an extensive risk analysis in Section 6, in which we apply the typical capital requirements found in banking regulation to the system.

2 Oracle-free Collateralized Minting

This section specifies and analyzes the proposed oracle-free collateralized minting plugin from a game-theoretical perspective. There are two relevant games to be analyzed. The first is the initialization game between the minter and the system when proposing to mint new Frankencoins against a collateral. The second is the liquidation game to ensure the stability of the resulting collateralized negative Frankencoin position. The latter game has four participants, namely the minter, a challenger, various bidders, and the system, whereas the challenger and the bidders are the actors that take relevant decisions. For simplicity, market prices are assumed to be constant for the duration of the game.

2.1 Definitions

Letter	Name	Description
M	minter	Deposits a collateral of C_M units of an asset and mints Z_M Frankencoins.
C	challenger	Challenges the minter offering C_C units of the collateral asset for sale.
B	bidder	Bids Z_B Frankencoins for C_C units of the collateral asset.
S	system	Vetoes plugins. Bears the liquidation profits and risks. Pays reward.

Table 1: Actors. The four relevant actors in the two discussed games.

2.2 Initialization Game

The initialization game is about initiating a new collateralized Frankencoin position.

Specification: Two actors, a minter M and the system S take part in this sequential game with two rounds as depicted in Figure 1. In the first round, the minter chooses a strategy $(Z_M, C_M) \in$

Letter	Name	Description
$C_M \in \mathbb{R}_{>0}$	minter collateral	Amount of the collateral provided by the minter.
$Z_M \in \mathbb{R}_{>0}$	minted currency	Frankencoins (ZCHF) created by the minter.
$C_C \in (0, C_M]$	challenger collateral	Collateral asset quantity offered by the challenger.
$\tilde{Z}_M = Z_M \frac{C_C}{C_M}$	challenged part	The challenged part when $C_C < C_M$.
$Z_B \in \mathbb{R}_{>0}$	bid	Frankencoins offered by the bidder for C_C.
$D_S \in noop, veto$	governance	The system's approval decision.

Table 2: Strategies. The decision variables chosen by the actors during the discussed games.

Letter	Name	Description
$l = \frac{1}{1+h}$	loan-to-value	The loan-to-value threshold above which the collateral is liquidated.
$p > 0$	price	The market price of the collateral in Frankencoin.
$0 < k < h$	reward	Fraction of the challenged position that is paid to successful challengers as a reward.
$v > 0$	value	The utility value the minter derives from the minting.

Table 3: Parameters. The relevant parameters and their meaning.

$\mathbb{R}^2_{>0}$. The variable C_M is the amount of the collateral to deposit and the variable Z_M is the amount of the currency to be minted. In the second round, the system chooses a strategy $D_S \in \{noop, veto\}$, whereas *noop* is a common abbreviation for 'no operation' and represents the choice of doing nothing. The other possibility is to *veto* the proposal. It is assumed that the minter has a desire to obtain a Frankencoin position, represented by a positive utility v. If that utility was zero, no game would take place at all.

Theorem 1 (Valid Initialization)**.** *Assuming the liquidation game always ends with positions being well-collateralized (Theorem 2), the initialization game will never end with a position being opened that is not well-collateralized, whereas well-collateralized is defined as* $Z_M \leq lpC_M$*.*

Proof. Theorem 1 can be shown by analyzing the initialization game as depicted in Figure 1.

Case 1 (left path, $Z_M \leq lpC_M$)**.** *The collateralization is sufficient to prevent a liquidation. The system S is indifferent between veto or noop, as there is no loss or gain for either decision. The resulting position, if any, is well collateralized.*

Case 2 (middle path, $Z_M > lpC_M$ and $Z_M \leq pC_M/(1+k)$)**.** *The collateralization is not sufficient and would result in a liquidation, and the collateralization is sufficient to pay the challenger fee k. With a veto, the system S is not making any gain or loss. If the system S chooses noop, a liquidation game results in a gain of* $\frac{1}{1+k}pC_M - Z_M \geq 0$*. Hence the system chooses noop, and with Theorem 2, the resulting position (if any) is well collateralized.*

Case 3 (right path, $Z_M > lpC_M$ and $Z_M > pC_M/(1+k)$)**.** *The collateralization is not sufficient and would result in a liquidation, and the collateralization is not sufficient to pay the challenger fee k. With a veto, the system S is not making any gain or loss. If the system S chooses noop, a liquidation game results in a loss of* $\frac{1}{1+k}pC_M - Z_M < 0$*. Hence the system chooses to veto, so there is no resulting position.*

To conclude, the game results in either no position, or a well collateralized position. ■

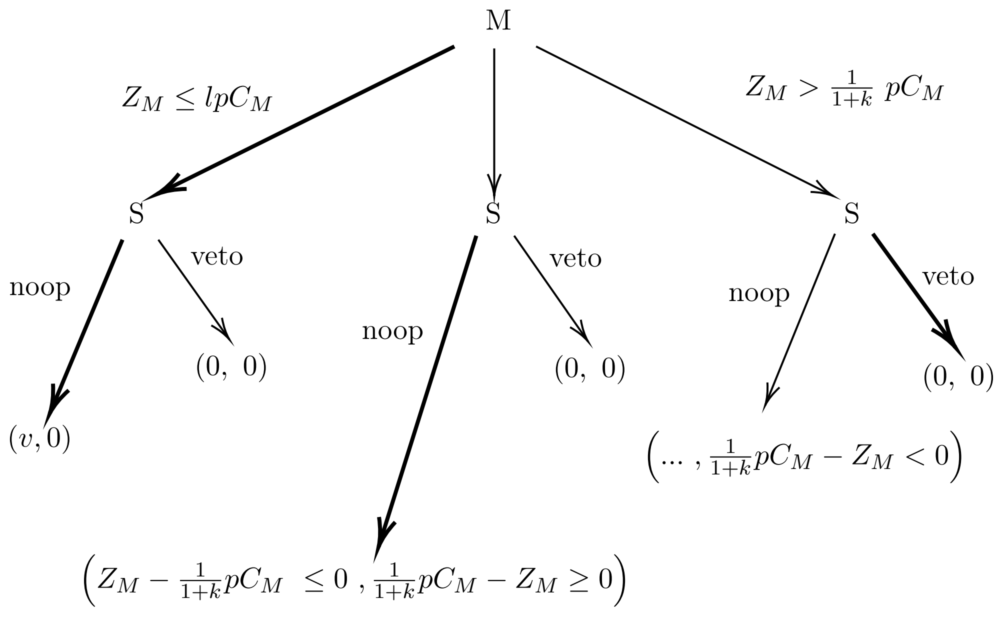

M
$Z_M \leq lpC_M$
$Z_M > \frac{1}{1+k}\, pC_M$
S
S
S
noop
veto
noop
veto
noop
veto
$(v, 0)$
$(0,\ 0)$
$(0,\ 0)$
$(0,\ 0)$
$\left(\ldots\ , \frac{1}{1+k}pC_M - Z_M < 0\right)$
$\left(Z_M - \frac{1}{1+k}pC_M\ \leq 0\ , \frac{1}{1+k}pC_M - Z_M \geq 0\right)$

Figure 1: Extensive form graph of the initialization game between the minter (M) and the system (S), with the preferred choices in bold. The first element in the round brackets is the minter's payoff and the second element that of the system S. A rational minter M goes for the leftmost path where they can successfully mint the desired number of Frankencoins and derive a value v from the obtained liquidity. In the other two cases, the minter is either liquidated in the subsequent liquidation game and suffers a loss from that, or the position is so undercollateralized that it is immediately vetoed by the system to avert a loss.

In practice, casting a veto requires marginally more effort than doing nothing (noop), and proposing a new collateralized Frankencoin position requires more effort than vetoing one. Revisiting the proof above, we see that these effort relations are beneficial for the game (e.g., in case 1, the system would choose noop, instead of being indifferent). However, we do not require these relations for Theorem 1 to hold and therefore do not formalize these efforts for simplicity.

Proposition 1 (No undercollateralized minting)**.** *A rational minter does not initiate a new position that is undercollateralized.*

Proof. In the proof of Theorem 1, we saw that in case 3, noop will never be chosen, and in case 2, noop is always chosen, finally in case 1, either path is chosen. Hence, for all viable nodes, we see that case 1 is the most profitable for the minter, the other choices result in either a utility of zero, or a loss. ■

2.3 Liquidation Game

The purpose of the liquidation game is to provide a mechanism to prevent an undercollateralization of a mint plugin by liquidating the collateral before its price falls below the value of the Frankencoins it has been used to print. It is designed such that it makes sense for challengers to start a round of the liquidation game as soon as the loan-to-value ratio exceeds the loan-to-value threshold, i.e. the value of the collateral has fallen so low such that $lpC_M < Z_M$ with p denoting the market price of the collateral at the point in time at which the liquidation game takes place. For simplicity, the market price of the collateral is assumed to be constant for the duration of the game.

2.3.1 Base Scenario

The liquidation game consists of two stages: a decision to challenge the collateralization and subsequent competitive decisions to bid on the challenge. Anyone can start a challenge and thereby become the challenger. Also, anyone can place a bid. In the base scenario, it is assumed that the minter, challenger, bidder and system are distinct persons. In subsequent sections, we show that the game still works as intended if identities overlap, for example when the bidder and the minter are the same person.

SPECIFICATION: Two groups of actors, the challenger and a group of competitive bidders take place in this sequential game as depicted in Figure 4. The minter is not allowed to repay the outstanding balance and reclaim the collateral for as long as a challenge is pending and is therefore no actor in this game, neither is the system. The liquidation game starts with the challenger's decision to start a challenge with quantity $C_C \in [0, C_M]$.[6] We assume there is enough collateral in existence for all positions to be challenged, and the challenger is indifferent between holding a given amount of Frankencoin, or an *equal* value in collateral asset. The bidders' set of available strategies is embodied by choosing the bid amount for the challenged collateral. The bidding process with multiple rounds of bidding by the competing bidders ends either after a fixed duration τ, or if the currently highest bid fulfills the collateral requirement, whichever occurs earlier. The highest bid at the end of the bidding process is Z_B. Challenges that show the position to not be well-collateralized are considered successful challenges. If the challenge is successful, the best bidder receives the minter's collateral C_c, the challenger is given a reward of $k\tilde{Z}_M$, and the rest of the bid is used to erase the outstanding position $\tilde{Z}_M$, with the remaining (potentially negative) amount of $Z_B - (1+k)\tilde{Z}_M$ going to the Frankencoin reserves. If the challenge is not successful, the best bidder receives the challenger's collateral C_c, and the challenger receives the bid amount Z_B.

Hence, depending on how high the bid is, the bidders are either bidding for the quantity C_C of the collateral asset provided by the challenger, or for a same quantity C_C of the collateral asset taken out of the minter's collateral deposit. This ensures that a malicious challenger cannot force the minter to sell the collateral. Once the highest bid reaches a level high enough to show that the collateral is valuable enough for the position to be well-collateralized, i.e. $Z_B = (1+h)\frac{Z_M}{C_M}C_C = \tilde{Z}_M$, the challenge is averted and ended immediately. Table 4 presents the payoffs. The allocation of the highest bid as implied by the payoff table is shown in Figure 3.

Actor		Payoff if challenge successful	Payoff if challenge averted
B	bidder	$pC_C - Z_B$	$pC_C - Z_B$
C	challenger	$k\tilde{Z}_M$	$-pC_C + Z_B$
M	minter	$\tilde{Z}_M - pC_C$	0
S	system	$Z_B - k\tilde{Z}_M - \tilde{Z}_M$	0

Table 4: Payoff table. The payoff for each actor in the liquidation game with $\tilde{Z}_M = Z_M \frac{C_C}{C_M}$ representing the minted amount adjusted to the size of the challenge. See also Figure 3.

Theorem 2 (Successful Liquidation)**.** *Given rational actors, the challenged part of the collateral is liquidated if and only if $Z_M > lpC_M$, where p is the market price.*

[6]The quantity of the asset C_C used for the challenge is of secondary importance and for most purposes one can simply assume $C_C = C_M$.

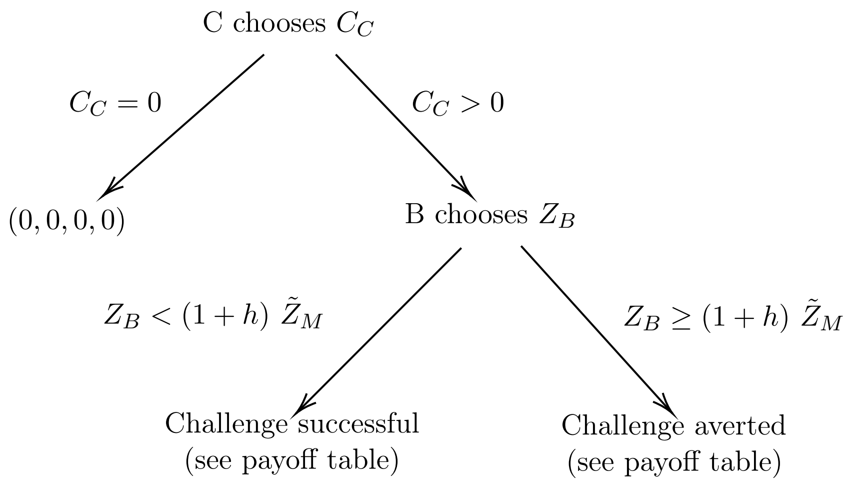

C chooses C_C
$C_C = 0$
$C_C > 0$
$(0, 0, 0, 0)$
B chooses Z_B
$Z_B < (1 + h)\ \tilde{Z}_M$
$Z_B \geq (1 + h)\ \tilde{Z}_M$
Challenge successful (see payoff table)
Challenge averted (see payoff table)

Figure 2: Extensive form graph of the liquidation game. It only makes sense to start a challenge if the highest bid can be expected to imply a violation of the loan-to-value threshold, i.e. if the market price of the collateral is low enough.

Proof. Being rational, the bidders will never bid higher than the market value, i.e., $Z_B \leq pC_C$. Considering that the challenge ends immediately once $Z_B \geq (1+h)\tilde{Z}_M$ is reached, the bidders also have no incentive to bid above that trigger point. Under perfect competition, the highest bid is:

$$Z_B = \min(pC_C, (1+h)\tilde{Z}_M) \tag{1}$$

From that point, all that is left to prove the theorem is connecting the dots.

Case 1 (Successful challenge)**.** *The challenge is defined as successful when $Z_B < (1+h)\tilde{Z}_M$. In this case, the highest bid must be $Z_B = pC_C$, allowing us to restate the inequality as:*

$$Z_B = pC_C < (1+h)\tilde{Z}_M = \frac{1}{l} Z_M \frac{C_C}{C_M}$$

from which immediately follows $Z_M > lpC_M$.

Case 2 (Averted challenge)**.** *In case of an averted challenge, i.e. $Z_B \geq (1+h)\tilde{Z}_M$, the highest bid must be $Z_B = (1+h)\tilde{Z}_M \leq pC_C$ as the challenge is immediately ended at that point. Plugging the definitions of l and $\tilde{Z}_M$ leads us directly to $Z_M \leq lpC_M$.*

So in both cases, the theorem holds. ■

Foreseeing the bidders' strategy, the challenger can assume the challenge to succeed whenever the value of the collateral has fallen low enough to make the position not well-collateralized any more, i.e., whenever:

$$p < (1+h)\frac{\tilde{Z}_M}{C_C} = (1+h)\frac{Z_M}{C_M} \tag{2}$$

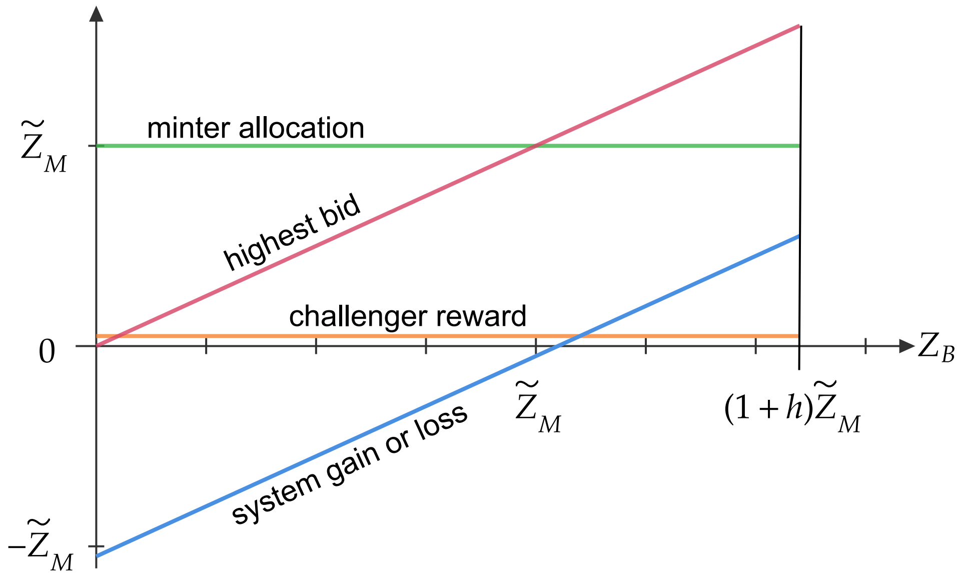


Figure 3: Bid allocation after a successful challenge. How the winning bid Z_B (red) is allocated to the minter (green), challenger (orange), and the system (blue) in the case of a successful challenge. Given an efficient market, the winning bid is equivalent to the market value of the collateral. The four lines sum up to zero. It is pivotal for the system to ensure that challenges happen before the price has fallen too low in order to avoid losses. There is no way to take away more from the minter than the collateral. In the extreme case of the collateral having become worthless, the system has no choice but to forgive the outstanding amount and to pay for the challenger reward out of its reserves.

Proposition 2. *It is only rational to start a challenge as long as inequality* (2) *is fulfilled, i.e. if the market price is so low that the value of the collateral does not suffice anymore to reach the desired loan-to-value ratio.*

Proof. If (2) holds, the challenger can earn payoff $k\tilde{Z}_M$. If (2) does not hold, a challenger would make a loss of $Z_B - pC_C = (1+h)\tilde{Z}_M - pC_C \leq 0$. ■

2.3.2 Overlapping Identities Scenario

A crucial assumption of the base scenario was that all actors are distinct. However, in practice, the bidder and the minter might be the same person, or there might be side-payments between them to tilt the incentives. This section shows how theorem 2 is affected and how it can be preserved when the bidder does not act on its own.

Proposition 3 (The challenger as bidder)**.** *Theorem 2 still holds when allowing the challenger to be the bidder.*

This can be shown by again looking at both cases, the case of the successful and the case of the averted challenge. In case of the averted challenge, the cumulative payoff of the challenger and the bidder is zero:

$$pC_C - Z_B - pC_C + Z_B = 0$$

This implies that a challenger can cancel a challenge at any time without incurring any costs, simply by bidding high enough. Still, as long as the position is not well-collateralized, the challenger gets a higher payoff by not cancelling the challenge, namely:

$$pC_C - Z_B + k\tilde{Z}_M > 0$$

Therefore, a bidding challenger will not behave any differently than any other bidder and theorem 2 is preserved.

Proposition 4 (The minter as bidder)**.** *Theorem 2 is violated when allowing the minter to be the bidder in an isolated game, but still holds when allowing the challenger to repeat the game under the same market price p.*

This is the most interesting proposition of the three. The minter might want to avert a looming liquidation by placing bids above the market price, i.e. $Z_B > pC_C$. For the minter, bidding above the market price to avert a challenge yields a loss of $pC_C - Z_B < 0$. However, this loss might still be the better option than taking the liquidation loss of $\tilde{Z}_M - pC_C < 0$. So in such a case, a rational minter would bid above the market price, violating theorem 2!

To solve this violation, one needs to expand the scope from a one-shot game to a repeated game. When the minter averts the challenge by bidding higher than the market price, the challenger makes a profit when selling C_C to the bidder. If the challenger is given the opportunity to restock the collateral asset C_C at market price p and then repeat the game before the minter gets a chance to close the open position, theorem 2 still holds. In that case, the challenger can threaten to infinitely repeat the game, causing an infinite loss to the minter. Therefore, a rational minter would prefer to not interfere with the bidding and take the smaller than infinite loss of the liquidation, which can only be repeated as long as there is collateral left to liquidate.

Proposition 5 (The system as bidder)**.** *Theorem 2 is violated when allowing the system to be the bidder in an isolated game, but still holds when allowing the challenger to repeat the game under the same market price p.*

In case of the market price being high enough that other bidders already avert the challenge, the incentive of the system acting as bidder are identical to that of a bidder alone, as the payoff for the system in that case is zero.

In case of a successful challenge, the system is indifferent with regards to the size of the bid when acting as a bidder, as Z_B drops out of the equation for the cumulative payoff:

$$pC_C - Z_B + Z_B - k\tilde{Z}_M - \tilde{Z}_M = pC_C - k\tilde{Z}_M - \tilde{Z}_M$$

This means that the system has no incentive to move the bid as long as we stay within the range of bids that would imply a successful challenge. However, since the overall payoff for the system in case of a successful challenge can be negative, the system might have an incentive to overbid in order to avert the challenge. This is similar to the problem of the minter being the bidder and the solution of looking at a repeated game is also applicable here.

2.4 Example

As an example, let us say the minter minted 1,000 ZCHF providing 10 ABC tokens as collateral, with $h = 0.2$ and $k = 0.02$. Then, a challenge was started with 4 ABC tokens, and the highest bid was 440 ZCHF. The required collateral value would have been $(1 + h)400 = 480$ ZCHF, so the challenge is successful since $440 < 480$. The bidder gets 4 ABC tokens from the minter. The minter's position is reduced to 600 ZCHF and 6 ABC tokens. From the 440 ZCHF bid, 400 ZCHF are burned, $k * 400 = 8$ ZCHF are given to the challenger as a reward, and the remaining 32 ZCHF flow into the reserve.

Figure 4 illustrates this auction mechanism graphically.

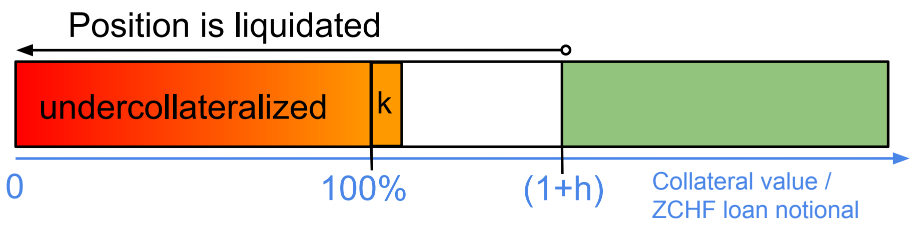


Figure 4: Auction. Any participant can challenge a loan position consisting of the ZCHF loan and the deposited collateral. The challenger deposits collateral of the size of the loan. Auction participants bid for the collateral (here BTC) by depositing ZCHF. If the best bid price in ZCHF is above (1+h) times the notional ZCHF challenged, the challenger receives the best bidder's ZCHF and the bidder the challenger's collateral (green zone). If the best bid price is below (1+h), the borrower loses their collateral (position is liquidated). In this case the challenger gets back their collateral and earns a fee of k times the challenged loan amount. The bidder gets the collateral deposited by the borrower. The ZCHF posted by the bidder are used to pay a reward k to the challenger, $(1 + h - k)$ times the challenged loan amount go to the contributor pool, and an amount equal to the loan amount is burned. If the ZCHF are not sufficient to burn an amount equal to the loan amount after paying k, the capital contributors' ZCHF of this amount of ZCHF is burnt/the reserve requirement is increased. Hence the unshaded area in the center is where contributors earn liquidation rewards, the area to the left is costly for contributors.

3 Reserve Pool

For clearing houses, each clearing member is required to contribute to a default fund based on the risk they bring to the clearing house and on the utilization of the clearing services. In the same vein, borrowers in the Frankencoin system contribute to the system reserves. This is done by adding a slight twist to the collateralized minting mechanism from the previous section.

3.1 Pool Specification

Frankencoin has a reserve pool that received all the excess proceeds from liquidations and has to cover the loss if these proceeds are negative. Anyone can contribute to the reserve, getting a proportional share in the pool in return. The assigned pool share $S(k_c)$ from contributing capital k_c to the pool is:

$$S(k_c) = \frac{k_c}{K_t + k_c} = \frac{k_c}{K_{t+1}}$$

whereas K_t denotes the capital already in the pool before the contribution. Likewise, the capital $K_r(s_k)$ returned from redeeming pool share s_k at time T is:

$$\kappa_T(s_k) = \frac{s_k}{\sum_i s_i} K_T.$$

Whether the returned capital $\kappa_T(s_k)$ is lower or higher than the paid-in capital k_c depends on the inflows and outflows of the reserves that happened in the period due to liquidations. Other contributors adding or removing capital does not have an impact. The amount of capital attributable to a share unit is invariant as others contribute or redeem capital.

The incentive for contributing to the reserve pool is the expectation that more money flows in than out when liquidations happen. Furthermore, pool contributors get the power to participate in the governance, i.e., they can act on behalf of the system and veto new mint plugins if the have enough shares.

3.2 Compulsory Minter Contribution

In order to make the minters contribute to the stability of the overall system, the minting mechanism is adjusted. When minting, an additional hZ_M ZCHF are minted on top of the already minted Z_M. These additional funds are contributed to the reserve pool with a pool share of $s_M = S(hZ_M)$ being assigned to the minter in return. This increases the stability of the system without binding additional capital, but by imposing some additional risk onto the minters.

With this adjustment, the amount of $ZCHF$ needed to pay back the loan depends on the value of the pool share s_M. That is, instead of having to repay Z_M, the minter will have to repay $Z_M + hZ_M - \kappa_T(s_M)$. Depending on the success of the system, this might be above or below the original loan Z_M.

When a position is successfully challenged, the challenged proportion of the pool shares $\tilde{s}_M = s_M \frac{C_C}{C_M}$ are redeemed and added to the liquidation proceeds, but an additional $h\tilde{Z}_M$ burnt to close also that part of the position. This leads to a new payoff for the minter and the system in case of a liquidation, as shown in the adjusted payoff Table 5.

Actor		Payoff if challenge successful	Payoff if challenge averted
B	bidder	$pC_C - Z_B$	$pC_C - Z_B$
C	challenger	$k\tilde{Z}_M$	$-pC_C + Z_B$
M	minter	$(1+h)\tilde{Z}_M - pC_C - \kappa_T(\tilde{s}_M)$	0
S	system	$Z_B - (1+h+k)Z_M + \kappa_T(\tilde{s}_M)$	0

Table 5: Adjusted payoff table. The payoff for each actor in the liquidation game including compulsory reserve contributions, affecting the minter and the system, but not the outcome of the liquidation game.

In case of a normal repayment of the loan, all the additional value and risk of having participated in the reserve goes to the minter. In case of a successful challenge, that additional value and risk is carried by the system that collectively represents the other minters and the contributors. If the value of the pool shares has not changed since the minter opened the position, the payoff is identical to the situation without compulsory minter contributions.

4 Fundamental Value

We now address the conditions under which the *peg to the Swiss franc* should hold. The fundamental value of the Frankencoin comes from a combination of three elements: first, it is necessary to have enough collateral such that in the long run, each Frankencoin is worth at least one Swiss franc as the minters need to buy them if they want to get their collateral back. Second, contributing Frankencoin to the reserve should come with a yield such that the established fundamental value at an unknown point in the future translates into the same fundamental value today. Third, one needs some assurance for the minters to ensure that they cannot be made subject to a short-squeeze.

4.1 Sufficient Backing

On the supply side of the Frankencoin, contributors should not allow any mintings that create a risk of the value of all collateral falling below the Frankencoins in circulation. Assuming that each Frankencoin is always backed with at least one Swiss franc worth of collateral, the Frankencoin is fundamentally sound and one can assume that the minters are willing to pay at least one Swiss franc per Frankencoin at the point in time when they want to get their collateral back again.

4.2 Comparable Yield

On the demand side, we approach the valuation of the Frankencoin from the perspective of a *perpetual bond*. A perpetual bond, or consol, is a bond with coupon payments but no redemption date, see, e.g., Jorion et al. (2010). We price this perpetual along the lines of Jarrow and Turnbull (2000), by discounting the interest payments on a risky term structure. Let's assume that interest payments happen at discrete time-steps $0, ..., \infty$ and we have corresponding risky rates of the term-structure so that the date-0 value of a promised Swiss franc at time t of a risky Franc promise is equal to $\exp(-r_t t)$. Let the constant coupon rate per Frankencoin be c. Now, the value of the perpetual can be written as

$$\begin{aligned} v(0) &= \sum_{t=0}^{\infty} c e^{-r_t t} && (3) \\ &= \sum_{t=0}^{\infty} c e^{-yt} && (4) \\ &= \frac{c}{1-e^{-y}}, && (5) \end{aligned}$$

where the second line replaces the time-specific discount rates by a yield, and the last line is an application of geometric series. For the value to be at par, $v(0) = 1$, we have to choose the coupon rate accordingly: $c = 1 - e^{-y}$. Hence, if the interest earned from contributing ZCHF are in line with discounting, the present value of one ZCHF is equal to one Swiss franc.

The risky term-structure corresponds to the Swiss franc risk-free term-structure plus a spread that compensates the investor for the risks. Whenever the risk-free term-structure, or the Swiss franc risk changes, c has to be adapted for the value $v(0)$ to be equal to one. In the Frankencoin system, the yield is implicitly set by the capital contributors collectively as they can veto minters that do not yield the right risk-adjusted return for the system. In effect, the contributors act as an oracle, but for long term interest rates instead of short term prices.

For this to work, minters should either be forced to pay an adjustable interest rate on their open positions, or the positions should be limited in time with a fixed interest rate that can change as they apply for a renewal of their collateralized positions. The shorter the average term of the open positions is, the faster the contributors can push the yield that can be earned from contributing to the reserve towards the correct c. Generally, contributors should seek to increase the yield if there is too little demand for holding ZCHF and vice versa.

The comparison with the consol shows nicely that it is not necessary for a stablecoin to be directly convertible to the reference currency in order to have the same fundamental value. It suffices if the coin offers the same expected return. Thereby, the problem is reduced to the ability to pay out an equivalent interest.

4.3 Price Ceiling

While it is generally not possible to exchange Frankencoins directly into collateral residing in the mint plugins, the minters will have to buy back their minted Frankencoins before they can get their collateral back. Here, the minters face a risk of a short squeeze. By minting and selling Frankencoins, they are short ZCHF and might be forced to pay more than one Swiss franc per Frankencoin to unlock their collateral. So while savers face the risk of the Frankencoin falling below the peg, minters face the risk of the Frankencoin departing upwards from the peg.

In the proposed setup, we start with a very simple mechanism to avert the risk of an overvaluation: we provide a bridge plugin that allows holders of other Swiss franc based stablecoins to convert them 1:1 into Frankencoins. As long as such bridge plugins exist, minters can be confident that they do not need to overpay for the unlocking of their collateral. However, while relying on other stablecoins can help in practice, it is not desirable to depend on them.

In the absence of bridge plugins, minters have to trust the contributors to always allow the minting of new Frankencoins at competitive terms, such that a short-squeeze can be averted by simply minting additional Frankencoins and repaying the open position with those. In effect, the system relies on good governance by the contributors at both the supply and demand side. On the supply side, contributors must allow economically sensible mint plugins and disallow irresponsible ones. On the demand side, the contributors must ensure that the risk-adjusted interest rate tracks that of the Swiss franc.

4.4 Long-term Incentives

Like many blockchain-based systems, the Frankencoin relies on the rationality of its participants. There is no technical barrier for the contributors to mismanage the Frankencoin, for example by letting a buggy mint plugin pass. It is in the economic interest of the contributors to look after the system and to make it work as expected. In particular, contributors have to be aware that they would destroy the long-term value of the Frankencoin by letting it deport too far from the peg. Since their reserve is at stake and they are the last who could cash out in case of a crash due to a loss of trust, contributors have a strong incentive to not let the price fall. At the same time, contributors also have a strong incentive to make sure that there is a wide range of available mint plugins such that the risk of a short-squeeze for minters is under control.

The reason why contributors have such a strong interest in making sure the Frankencoin tracks the value of the Swiss franc is that it enables them to earn net interest income to the extent that the Frankencoin is used as a transactional currency. For example, if the open market interest rate is 3% without any margin between borrowers and lenders, but only half of the outstanding Frankencoins are in the reserve pool, then the pool contributors can effectively earn a yield 6% on their contributions. This is only possible if the Frankencoin is valuable enough as a transactional currency that half of the holders do not care about foregoing their interest. And the transactional value is highest when the Frankencoin reliably tracks the value of the reference currency.

5 Implementation

This section describes the software architecture of the Frankencoin, with each subsection describing a separate module of the system. It also serves as a guide for adventurous readers wishing to acquiring some Frankencoins or otherwise taking part in the system. The proposed minting mechanisms are referred to with the technical *plugins*, emphasizing the extensible and open architecture of the system.

5.1 Frankencoin Contract

At the core, there is a contract that represents the Frankencoin with all the standard functionality of an ERC-20 token. It also stores a list of mint plugins. Anyone can add new mint plugins to this list and if their addition is not vetoed within three weeks, they can start minting Frankencoins to the extent that the plugin allows. It is the responsibility of the minter to implement the mint plugin such that it respects all capital and other requirements.

It seems important to us that new mint plugins are automatically approved if there is no veto from the contributors. This adds to the decentralized nature of the system and minimizes the necessary number of interventions. If proposing a new plugin is sufficiently costly and there is a credible threat of undesired plugins being rejected, it will only very rarely be necessary to take action and veto a plugin.

Anyone can also try to veto a proposed plugin, but for the veto to go through, one must have the support of at least 3% of the outstanding reserve pool shares. Owner of pool shares can delegate their voting power to anyone, allowing small share holder to organize themselves as a group with voting power.

An example implementation can be found in Appendix D.

5.2 Reserve Pool Contract

The reserve pool allows anyone to contribute to the reserve and receive reserve pool shares in exchange as specified in Section 3. Pool shares themselves are also freely transferable ERC-20 tokens.

An example implementation can be found in Appendix E.

5.3 Bridge Plugin

The simplest possible mint plugin is one that is based on a stablecoin with the same reference currency. For the Frankencoin, this could for example be the CryptoFranc (XCHF) issued by Bitcoin Suisse or the Digital Swiss Franc (DCHF) issued by Sygnum. This mint plugin allows anyone to deposit the specified stablecoin and to get Frankencoins in return. Also, the minting contract would allow anyone to convert Frankencoins back into the specific stablecoin for as long as there are any left.

An example implementation can be found in Appendix F.

5.4 Oracle-free Collateralized Mint Plugin

The oracle-free collateralized mint plugin is one of the cornerstones of this article. Once an instance of the plugin is initialized and approved, it enables the owner to mint Frankencoins backed by the specified collateral. It neither depends on an oracle nor on an external market. Each minter must create their own instance of a mint plugin for each collateral asset they desire to use. Future implementations might allow for more flexible use, e.g. by multiple different minters in parallel in order to reduce the overhead and waiting time when opening a new position.

An example implementation can be found in Appendix G.

6 Risk Analysis

We first analyse the risk to capital contributors in the next section. As part of this analysis we estimate the challenge success probability for a given starting level of the loan-to-value in Section 6.1.1, and size the minting fee required to compensate for the risk of the capital contributors in Section 6.1.2.

These results serve as foundation to calibrate risk-based capital requirements. Section 6.2 proposes rules for capital requirements based on banking regulation. In Section 6.3 we propose a risk based capital requirement based on modified Basel iii rules. If the capital drops below the minimal capital requirement, no more Frankencoins can be issued through standard plugins. An exception are mint plugins that are classified as risk-free, most notably bridge plugins to trusted stablecoins of the same denomination, if any.

6.1 Contributor Risk from Collateralized Mint Plugin

The Frankencoin system is at risk of losing equity capital if a liquidation ends at a price below the outstanding ZCHF amount. This section analyses this risk given the challenge mechanism defined in Section 2.3. We use Bitcoin as an example collateral. As established in Section 2.3, the rational bids imply prices close to (exogenous) market prices. We do not account for slippage incurred when selling the collateral in exogenous markets, so we can use Bitcoin market prices for the exercise at hand.

The challenge starting level h' is the level at which a rational system participant initiates a challenge. Hence h' is implicitly defined as

$$p_0 = (1+h')\frac{Z_M}{C_M} \tag{6}$$

$$= (1+h')\frac{\tilde{Z}_M}{C_C} \tag{7}$$

$$\tag{8}$$

where p_0 is the exogenous market price at the time the challenge is started, and the second line follows by definition of $\tilde{Z}_M$. From Section 2.3 we have the *condition for liquidation*:

$$Z_B < (1+h)\tilde{Z}_M, \tag{9}$$

to which we add the time dimension now. We consider the asset return $\tilde{r}_t$ over the liquidation horizon τ. The challenge is averted, if during the liquidation period a bid above the liquidation threshold is realized. We can now express the condition for liquidation as a function of the asset return, and the values h and h':

$$e^{\tilde{r}_t} < \frac{1+h}{1+h'} \quad \forall 0 \leq t \leq \tau. \tag{10}$$

Proof. By definition, the liquidation ends at price $p_{t'} = Z_B/C_C$, where t' is the time the challenge was averted or τ if it wasn't. We established that $p_{t'}$ is close to the exogenous market price, and hence $p_{t'} = p_0 \exp(\tilde{r}_{t'})$. We use this to rewrite Equation (9)

$$p_{t'}C_C < (1+h)\tilde{Z}_M. \tag{11}$$

Using Equation (7) we replace $\tilde{Z}_M$ to get

$$p_{t'}C_C < (1+h)\frac{p_0 C_C}{1+h'} \tag{12}$$

$$e^{\tilde{r}_{t'}} < \frac{1+h}{1+h'}. \tag{13}$$

We can now see that replacing t' with t and requiring $0 \leq t \leq \tau$ Equation (10) is equivalent. ■

To prevent being outbidden, the rational bidder will bid only either towards the end of the bidding horizon, or if the challenge can be averted. We can therefore assume the liquidation happens at the price p_τ unless it has been averted.

The profit $\tilde{P}$ to the capital contributors is zero in case the challenge is averted, positive in case of a liquidation at a price $p_\tau \geq (1+k)\frac{\tilde{Z}_M}{C_C}$, negative if the price is below, and zero if the challenge is averted and no liquidation takes place. We define a dummy variable D that is exactly 1 in case a challenge is successful and 0 otherwise:

$$D = \mathbf{1}_{e^{\tilde{r}_t} < \frac{1+h}{1+h'} \, \forall 0 \leq t \leq \tau} \tag{14}$$

We can now express the profit $\tilde{P}$ per unit of the challenged amount as the following random variable:

$$\tilde{P} = \left[(1+h')e^{\tilde{r}_\tau} - (1+k)\right] D \tag{15}$$

Note that if the challenge starts at $h' \geq h$, the indicator function is always $D = 0$ as the challenge is immediately ended again. We now determine the value $\tilde{P}$ to decide whether capital contributors need to be compensated by a minting fee.

To tackle the valuation of $\tilde{P}$, we resort to the arbitrage free pricing principle, which states that the value of a contingent claim is given by its discounted expected value under the risk-neutral probability measure, see for example Björk (2009). We assume that the risk-free rate used for discounting is equal to zero, which we consider adequate especially since the period τ is very short. Let $f_\tau(x)$ be the density function for the return distribution over the period τ. Now, we can value $\tilde{P}$ conditional on the starting level of liquidation h' as follows

$$\mathbb{E}_\tau^{\mathbb{Q}}\left[\tilde{P}|h'\right] = D\int_{-\infty}^{\log\frac{1+h}{1+h'}} \left[(1+h')e^x - (1+k)\right] f_\tau(x)dx \tag{16}$$

where the subscript τ emphasizes that the distribution depends on the liquidation period.

We now discuss h'. If we knew the distribution of starting levels h', we could integrate out h' to arrive at $\mathbb{E}_\tau^{\mathbb{Q}}\left[\tilde{P}\right]$. As established, challengers trade off the probability of collecting the reward given they issue a challenge versus the probability of not being able to be front-run by other challengers. To investigate this trade off, we assume that challengers issue a challenge when the collateral value reaches a level for which there is a given probability α that they receive the challenger reward. We then investigate what minting fees each h' (or equivalent, each α) implies.

Equation (16) uses the risk-neutral probability measure, often referred to as $\mathbb{Q}$, rather than the objective measure $\mathbb{P}$. In practice, parameters for the measure $\mathbb{P}$ are extracted directly from market data (e.g., sample volatilities and expected returns), whereas parameters for the measure $\mathbb{Q}$ have to be extracted from option data under the same model assumptions (e.g., option implied volatilities). With risk averse investors, the $\mathbb{Q}$-measure puts more weight on adverse market events, see for example Breeden and Litzenberger (1978), leading to a higher risk-neutral price of $\tilde{P}$, compared to the value obtained when integrating Equation (16) under the objective measure. We therefore proceed by calibrating a probability distribution to observed market data and use this as a lower bound for the price of $\tilde{P}$, or, equivalently the minting fee should be at least equal to the negative value of $\tilde{P}$ under the measure $\mathbb{P}$:

$$\Theta_F^{(i)} \geq -\mathbb{E}_\tau^{\mathbb{P}}\left[\tilde{P}\right], \tag{17}$$

where we add the negative sign because $\tilde{P}$ is a profit. For brevity, we omit the superscript $\mathbb{P}$ in the sequel.

6.1.1 Challenge success probability

We are now ready to calibrate the parameters. To do so, we evaluate the probability of liquidation for different challenger levels h'. We then locate reasonable challenger levels and investigate the lower bound of the minting fee, $\Theta_F^{(i)}$, for the given level of h'. In Appendix C we describe the BTCCHF candle-data that we use to calibrate the fees. We use 24h candle data (Open, Low, High, Close) and use the bootstrap method introduced by Efron (1992) for estimation.

With candle data, we can readily determine whether for a given period, the challenge started at the beginning of the candle would have been averted, $D = 0$, or succeeded $D = 1$, by

$$\hat{D}_\tau = \mathbf{1}_{\left\{\frac{P_H}{P_O} < \frac{1+h}{1+h'}\right\}}, \tag{18}$$

where P_O is the open price, P_H the high price over period τ (without indexing the candle for brevity), which follows directly from definition of D in Equation (14). For the bootstrap, we define the following variables. Let N be the number of candle observations, B the number of bootstrap replications, and $\hat{\mathbf{D}}_b = \{\hat{D}_\tau^{(b,1)}, ..., \hat{D}_\tau^{(b,N)}\}$ the b^{th} bootstrap replication for a period-τ dummy that equals one if the challenge was not averted. The bootstrap estimate for the probability of liquidation follows directly from Equation (18):

$$\hat{\mathbb{E}}\left[D|h'\right] = \frac{1}{B}\sum_{j=0}^{B-1}\frac{1}{N}\sum_{n=1}^{N}\hat{D}_\tau^{(j,n)}. \tag{19}$$

For comparison, we also evaluate the probability of liquidation conditional on h', assuming the challenge cannot be averted prior to time τ. For this assumption we use both, a bootstrap evaluation (replacing P_H/P_O by P_C/P_O, with P_C equal to the close price, in Equation (18)) and a closed-form evaluation assuming the returns are normally distributed:

$$\hat{\mathbb{E}}\left[\mathbf{1}_{\{\tilde{r}_\tau < \log\frac{1+h}{1+h'}\}}\middle| h'\right] = \Phi\left(\frac{\log(1+h) - \log(1+h') - \mu_\tau}{\sigma_\tau}\right), \tag{20}$$

where μ_τ and σ_τ are location and scale parameters of the normal distribution, and $\Phi(\cdot)$ represents the standard normal cumulative distribution function. We use the sample mean and standard deviation of the observed returns to estimate μ_τ and σ_τ respectively. Figure 5 shows the results. We see that allowing the challenge to be averted early significantly reduces the probability that a challenge is successful when it is issued at a price close to the liquidation threshold. At $(1+h')$ about 2% below the liquidation threshold $(1+h)$, there is a fifty percent chance of the challenge being successful.

The normal distribution overestimates the probability of liquidation for values h' above the liquidation threshold h, and underestimates the liquidation probability for value of $h' < h$ compared to the bootstrap result. We use the variance resulting from the bootstrap replications to estimate confidence intervals.

6.1.2 Minting fee valuation

Finally, we use use Equation (15) to arrive at our bootstrap point estimate for the value of the liquidation profit P. Let $\hat{\mathbf{r}}_b = \{\hat{r}_\tau^{(b,1)}, ..., \hat{r}_\tau^{(b,N)}\}$ the b^{th} bootstrap replication for a period-τ return, now

$$\hat{\mathbb{E}}_\tau\left[\tilde{P}|h'\right] = \frac{1}{BN}\sum_{j=0}^{B-1}\sum_{n=1}^{N}\hat{D}_\tau^{(j,n)}\left[(1+h')e^{\hat{r}_\tau^{(j,n)}} - (1+k)\right]. \tag{21}$$

We use $B = 10,000$ bootstrap replications and calculate confidence intervals at the 1%-level by applying the central limit theorem.[7] The confidence interval result in ranges in the 0.01%-area. Figure 6 shows the results. At low challenge levels, the contributors have to burn ZCHF and there is a loss for them. At very high challenge levels, often the challenge does not result in a liquidation and hence there is no gain or loss for contributors. Capital contributors make the most profit for challenges that start at a level of about $1 + h' = 107\%$.

In Figure 6 we see that capital contributors do not lose on average if the challenge starts above $1+h' = 102\%$. From Figure 5 we learn that starting the challenge process at $1+h' = 102\%$, there is about a 90% probability that the challenge ends in a liquidation, hence the challenge is expected to

[7] Having a vector of B bootstrap estimates $\mathbf{x}$, we report the point estimate as the mean of $\mathbf{x}$, and the error at the a-level as $z_{1-a}\sqrt{V[\mathbf{x}]/B}$ with z_{1-a} the standard normal quantile.

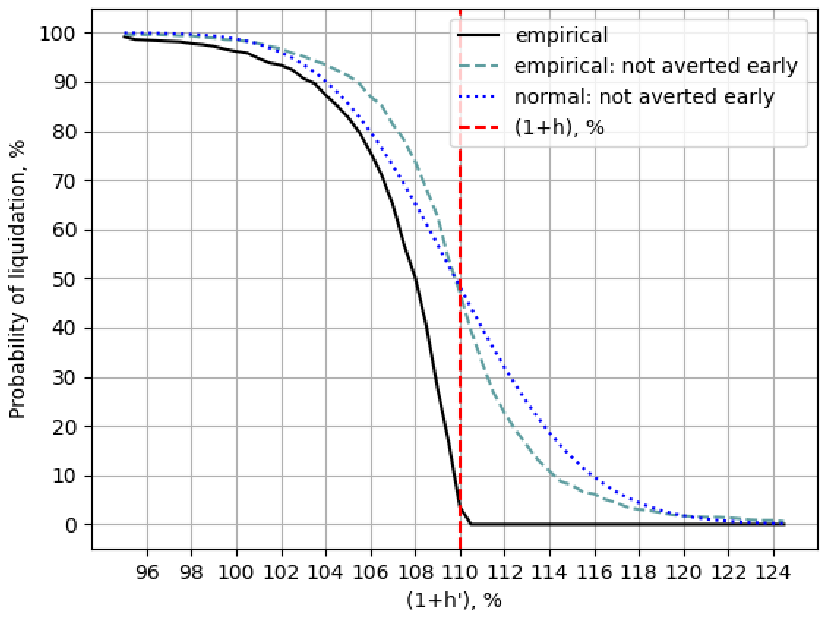


Figure 5: Probability of Liquidation given h'. This figure plots the probability of the challenge ending in a liquidation, when the challenge is started at a level h', with the liquidation level at $h = 0.10$ and a challenge duration of 24 hours (solid black line). The probabilities are estimated via the bootstrap ($B = 5,000$ samples). Confidence intervals at the 1%-level remain below 0.05% around the point estimates for each h'. For comparison, we hypothetically assume the challenges cannot be averted before the end of the liquidation horizon (dashed and dotted curves). We compare the bootstrap estimate (dashed line termed "empirical: not averted early"), with the theoretical probability using normally distributed returns (with sample mean and standard-deviation).

start above the 102% level. We conclude that the minters do not need to be charged to compensate the capital contributors, unless risk-aversion would be very high.

6.2 Capital Requirements

Banking regulation prescribes three types of capital requirements for which we find a meaningful analogue in the Frankencoin system, see for example the Basel III banking regulation Basel Committee on Banking Supervision (2010) or the Dodd-Frank Act Acharya et al. (2010).

1. *Risk-based capital requirements.* In the Frankencoin system, each mint plugin defines its own reserve requirement. For each mint plugin, we define a reserve requirement. The fact that the Frankencoin system uses shared capital, with plugin-specific reserve requirements adds a diversification benefit, that is, adding capital requirements of two plugins together leads to a greater or equal reserve than when estimating the capital requirement based on the two plugins taken together.

2. *Leverage limits* restrict the overall amount of leverage in the balance sheet. The balance sheet of Frankencoin is presented in Appendix A. This limit has two main benefits. First, leverage limits are largely model-free and are therefore a safeguard against model risk (e.g., model risk

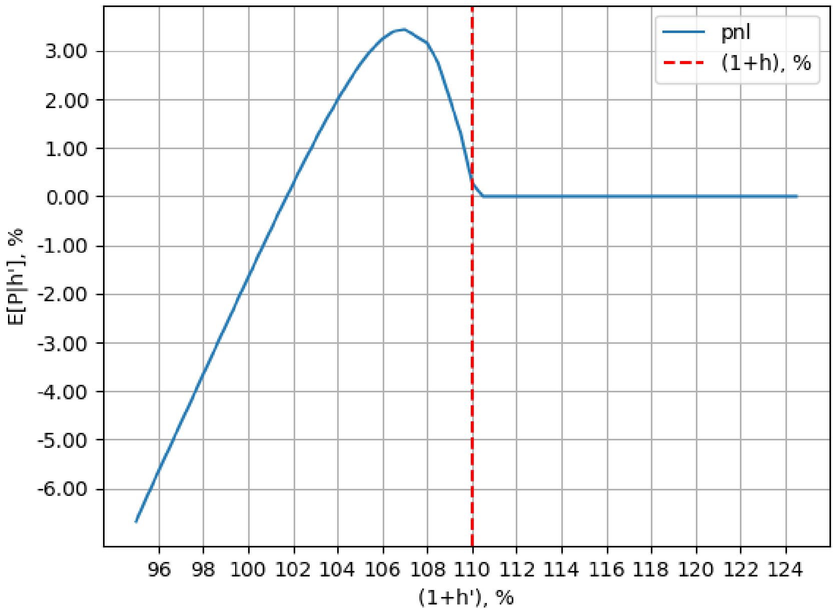


Figure 6: Collateral mint plugin fee. We plot the value of profit and loss, given the liquidation level start h'. Negative values observed for approximately $1 + h' < 102\%$ correspond to a loss for the capital contributors. Challengers are likely to start the challenge at a level above $h' = 0.02$ which corresponds to a liquidation probability of over 90% as we see in Figure 5. Therefore we conclude that with these parameters, we do not need to compensate capital contributors with a minting fee, unless there is a high risk-aversion.

arising from the risk-models used to calibrate plugin-specific capital requirements). Second, a leverage limit provides us a global (i.e. Frankencoin-system-wide) capital limit.

3. *Concentration limits.* If the collateral of Frankencoin was mainly exposed to the price of one asset, Frankencoin could more easily collapse following a large price drop of that asset, compared to a more diversified collateral in the system. We therefore aim at limiting the concentration of collateral-type.

Of these three measures, only the risk-based capital requirements are explicitly implemented in the provided reference implementation. Furthermore, similar to clearing houses, where each clearing member is required to contribute to the guaranty fund based on the risk of their position, borrowers provide a compulsory minting contribution as detailed in Section 3.2.

A simple way to measure *concentration* is the one-firm concentration ratio, see Curry and George (1983), which equals the percentage of market share held by the largest firm. Let K be the number of mint plugins. Each mint plugin tracks the amount of ZCHF issued and not burned, z_j.[8] The relative amount issued by plugin j is given by $s_j = z_j / \sum_i^K z_i$. We aim to prevent that the largest s_j is beyond a threshold. However, when applying this restriction on a system-wide level, we run into the problem that the system could consist of multiple different plugins with the same collateral, or collateral with high price correlation. We therefore suggest that each mint plugin comes with its own concentration threshold $\theta_i^{(C)}$, if such a concentration limit is to be implemented. Once a mint

[8]If collateral is liquidated, the ZCHF burnt are subtracted from z_j.

plugin reaches its threshold, $\theta_i^{(C)}$, the plugin cannot issue any more ZCHF until the concentration is reduced.

In the spirit of the creators of the *leverage limits*, we aim to have the least possible assumptions to define leverage limits in our system. We therefore do not want to rely on any collateral valuations or distributional assumptions. Bridge plugins should not count towards the leverage ratio, because their value is stable and diversified through the concentration limits, and because they provide a convenient way to issue new ZCHF that are subsequently used as capital contribution. However, we want to limit the amount of ZCHF issued through collateralized mint plugins relative to the ZCHF held as a reserve in the capital contributor pool. We therefore define the leverage ratio as

$$\lambda = \frac{1}{R} \sum_{i \in \mathcal{C}} z_i, \tag{22}$$

where z_i is the amount of tokens issued by mint plugin i, $\mathcal{C}$ is the set of plugin indices that are to be included in the leverage calculation (i.e., not the bridge plugins), and R is the total amount of contributed ZCHF in the reserve. The leverage ratio could be enforced by tracking the total amount of relevant outstanding Frankencoins and then denying additional minting if the reserves are not high enough, i.e. if the leverage ratio is larger than a governance set threshold $\theta^{(L)}$.

We examine the *risk-based capital* requirement more deeply in the next section. Risk-based capital requirements may involve relatively complex calculations for current blockchain capabilities, however, the calculations can be done off-chain and only the resulting parameters need to be stored on-chain.

Despite the comprehensive capital management measures, there are only a few parameters and summary statistics that need to be stored on-chain, which we summarize in Table 6.

Table 6: Summary of parameters. This table lists the parameters and data that needs to be stored to implement the mentioned risk mitigation measures.

Element To Store	Type	Description
z_i	Data	Each mint plugin i keeps track of ZCHF issued by the plugin and not burned
z, $z^{\mathcal{C}}$	Data	The governance contract keeps track of the sum of z_i and the sum over the z_i issued by leverage constraint relevant contracts
R	Data	Amount of ZCHF in capital reserves
$\ell_i = \mathbf{1}_{\{i \in \mathcal{C}\}}$	Parameter	Each of the K mint plugins defines whether they are part of the leverage ratio calculation or not
h, τ, k	Parameter	Parameters for collateralized mint plugin
$\Theta^{(L)}$	Parameter	Maximal leverage ratio
$\Theta_i^{(C)}$	Parameter	Maximal relative amount of ZCHF issued by mint plugin i to limit concentration risk
$\Theta_i^{(F)}$	Parameter	Each mint plugin has a minting fee
$\Theta_i^{(R)}$	Parameter	Each mint plugin sets its own risk-based reserve requirement, relative to the issued ZCHF
$\Theta_i^{(I)}$	Parameter	Each mint plugin can define an interest rate paid to capital contributors (can be zero, e.g., in case of bridge plugins)

6.3 Risk-based Capital Requirements

Lombard loans are loans extended by banks to their customers, secured by the customers' securities that are held in bank custody. In the Lombard loan agreement, bank and customer agree on the terms of the loan, including that additional assets have to be provided in case the value of the collateral falls below a margin call level. The bank has the right to sell the pledged securities, if the value falls below an agreed level. In the Frankencoin system, there is no agreement beyond what is defined in the smart contract, and hence there will not be any bankruptcy litigation. As a consequence, there is a loss to the system that has to be covered with Frankencoin reserves, as soon as the collateral drops below the loan amount. Not surprisingly, this difference renders the Basel iii capital requirements for collateralized loans inadequate for the Frankencoin system. We therefore propose a more conservative approach to capital reserves.

Each mint plugin defines the required reserves of ZCHF to be held against the issued volume of ZCHF. We now assess how to specifically determine the risk-based capital requirements for our illustrative setup. We do not require risk-based capital reserves for the bridge plugins, so we focus entirely on the collateralized mint plugin, collateralized in Bitcoin.

The assessment we perform now differs from the one we made to determine the minimal fee in two main aspects, (1) we want to estimate a loss for the whole collateralized mint plugin not only for an individual position, and (2) we are not looking for an insurance valuation but for the capital required. The latter point implies that we do not need any assumption on risk-neutral measures but we can directly work with the observed data and use real world probabilities.

We start by applying the Basel iii rules for risk based capital.

6.3.1 From Basel Rules to Frankencoin Capital Requirements

The Basel Committee of Banking Supervision defines capital rules for collateralized loans in the Basel iii regulatory framework. In view of the authors, the collateralized mint plugin best meets the conditions to be considered a "repo-style transaction", see 22.66 in Basel Committee on Banking Supervision (2019a), henceforth [CRE22]. In this framework, the bank first weights assets according to the risk to obtain the risk-weighted assets (RWA). The bank is then required to hold a certain amount of capital, for example Common Equity Tier 1 must be at least 4.5% of RWA and total capital must be at least 8% of RWA, for details see Basel Committee on Banking Supervision (2019b). For collateralized loans, the rules are detailed in [CRE22]. Banks have two options: *"Banks may opt for either the simple approach, which substitutes the risk weighting of the collateral for the risk weighting of the counterparty for the collateralized portion of the exposure (generally subject to a 20% floor), or for the comprehensive approach, which allows a more precise offset of collateral against exposures, by effectively reducing the exposure amount by the value ascribed to the collateral"*, see 22.12 in [CRE22].

We focus on the comprehensive approach. First, the bank determines the exposure amount after risk mitigation, E^*, which depends on the loan and the collateral. Both are subject to a haircut, see 22.40 in [CRE22]:

$$E^* = \max[0, E(1 + H_e) - C(1 - H_c - H_{fx}), \tag{23}$$

where E^* is the exposure amount after risk mitigation, E the current value of the exposure (the loan), H_e haircut appropriate to the exposure, C the current value of the collateral received, H_c the haircut appropriate to the collateral, H_{fx} the haircut appropriate for currency mismatch between the collateral and exposure. The exposure amount E^* is to be multiplied by the risk weight of the counterparty to obtain the risk weighted asset amount for the collateralized loan. Collateral haircuts are either determined based on the type of collateral (e.g., 25% for 'other equity' and a holding period of 10 days). The exposure is then multiplied by the risk weight of the counterparty to obtain RWA. If the counterparty is not rated, a weight of 1.5 is applied. Figure 7 summarizes this approach. In Appendix B we calculate the Basel iii capital requirements using the comprehensive approach and using the approach with own haircut estimates based on a 99%-VaR as per Basel iii. Table 7 shows the results: per Basel, the required capital reserve results to only about 1% of outstanding loans, even when assuming a conservative loan-to-value ratio of $1/(1+h) = 1/1.1$ (which is always below current exposure assuming liquidations are effective).

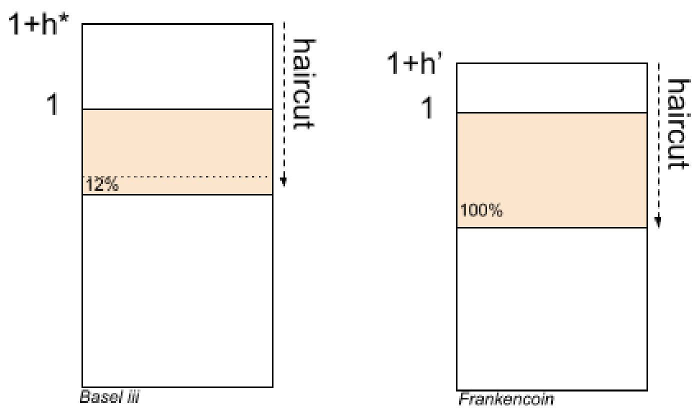


Figure 7: Basel iii vs own approach. Basel iii subtracts a haircut from the current collateralization value $(1+h^*)$ to obtain the exposure after risk mitigation (shaded area). This quantity is multiplied by 1.5 for unrated counterparties. Total capital is to be 8% of this. However, in the Frankencoin system there is no loan agreement beyond the smart contract, and hence no litigation. We therefore propose a more strict approach, in which (1) the calculation of the haircut starts at a conservative liquidation level $(1+h')$, (2) the entire exposure after risk mitigation is to be covered, as opposed to $1.5 \cdot 8\% = 12\%$.

To render the Frankencoin system resilient, the capital requirements must not assume that litigation is possible, hence the whole exposure after risk mitigation should be covered by capital reserves, as opposed to 12% per Basel iii. Second, instead of applying the haircut on the current exposure value, $(1+h^*)$ as per Basel iii, we apply the haircut on a conservative liquidation level $(1+h')$. This is illustrated in Figure 7. To calculate the haircut, we estimate an empirical VaR at the 99%-level (a VaR level in line with Basel iii). We call this the Frankencoin Proposal in Table 7. Second, we look at the Basel iii requirements for collateralized lending

6.3.2 Capital Estimation per Frankencoin Proposal

We construct empirical samples of loss data by applying the 24h log-returns to the loss function given by Equation (15), which we repeat here:

$$\tilde{P} = \left[(1+h')e^{\tilde{r}_\tau} - (1+k)\right] D,$$

Method	Capital
Frankencoin Proposal (empirical VaR)	10%
RWA based, Basel iii Comprehensive	1.1%
RWA based, Basel iii Comprehensive (own haircuts)	1.3%

Table 7: Risk-based Capital Requirements. Results of capital reserves required using different approaches. For the setup at hand, we recommend that the system should hold 10% of the outstanding loans as Frankencoin reserves.

and we multiply the samples by -1 to have a positive number for a loss. Figure 8 gives an overview of the loss data for different challenger levels h' via boxplots. We see that the median level of losses are benign, even for very low challenger levels of $h' = -1\%$ (meaning the liquidation is only started when the collateral has a value of 99% of the loan notional, although the position would liquidate if the collateral is at 110% of the loan notional). However, we also see that there are high losses for all levels of h' depicted, driven by two extreme returns in the data, where the loss exceeds 20% of the loan notional. These losses are at a loan level and apply if the loan challenge starts at a collateral coverage ratio of $(1 + h')$. On a plugin-wide level, we would only lose the same percentage of the outstanding loan notional, if all loans started at $(1 + h')$ simultaneously. Realistically, collateral coverage ratios are more spread out and hence, the loan-level losses observed in Figure 8 are upper bounds to the plugin-wide loss.

Table 8 presents the 99-percentiles for daily losses (using the loss empirical loss distribution seen in Figure 8, given the challenge start level h'. A conservative starting level is $h' = 2\%$ as we have seen in the previous chapter. The associated 10.73% loss at the 99%-level would assume that all loans have an equally low collateralization of h'. To conclude, using our proposed approach that deviates from the Basel iii rules towards the conservative side, we propose to set the capital requirement to 10% of the outstanding loans.

h'	VaR99, %
-0.01	13.42
0.00	12.52
0.01	11.63
0.02	10.73
0.03	9.84
0.04	8.94
0.05	8.05
0.06	7.15
0.07	6.26
0.08	5.36
0.09	3.77
0.10	0.00

Table 8: Empirical VaR. This table shows the empirical VaR in percent of the outstanding loan amount given the challenge starting level h'.

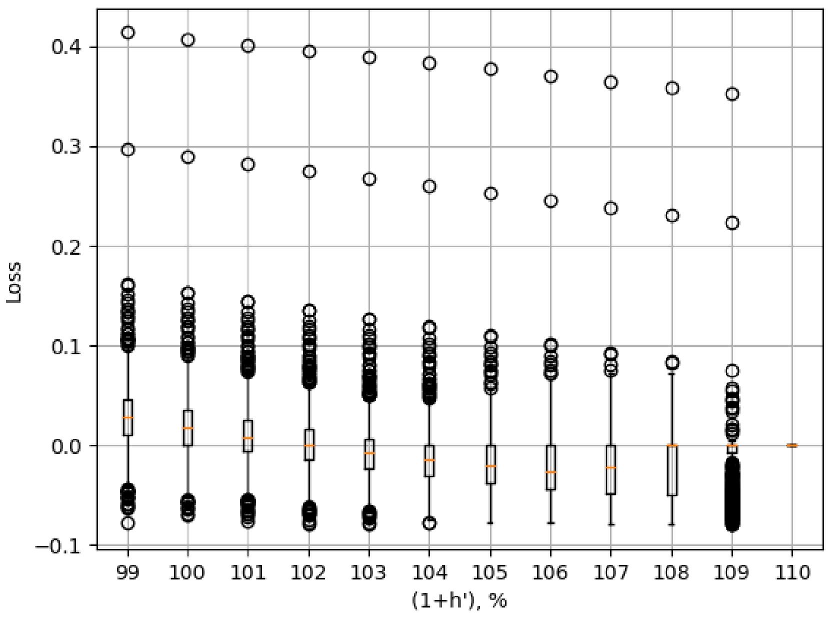


Figure 8: Empirical Loss Boxplot. This figure shows standard boxplots for different challenger levels h', given that a liquidation is triggered when the challenge results in a collateral value of below 110% of the loan, and a challenger fee of $k = 2\%$. The vertical bar depicts the median, the boxes reach from the first quartile to the third quartile, the whiskers extend to the max/min or at most 1.5 times the range of the box, circles are plotted outside the span of the whiskers. For values h' closer to $h = 0.10$, the loss more often ends in a gain (negative number). At $h = 0.10$ the challenge is averted. Losses stay on average benign but we also observe a few very high losses that exceed 20% of the loan notional due to two very negative returns (-49%, -31%).

7 Conclusion

Blockchain technology opens the possibility to create new kind of monetary institutions that are governed in a decentralized way. Unlike bank loans that are issued by a centralized entity, the presented Frankencoin can be minted by the borrowers themselves, given that they provide suitable collateral, and the system has sufficient capital in its reserve. In the presence of a central bank digital currency (CBDC), see for example Chaum et al. (2021) or Brunnermeier and Niepelt (2019), the presented Frankencoin could come with a bridge to that CBDC, in which case the Frankencoin system could be seen as a fractional reserve multiplier just like the established banks are for traditional central bank money. With that perspective, it is no surprise that risk mitigation strategies from the current banking regulation can serve as a starting point to risk mitigation in the Frankencoin system.

References

Acharya, V. V., Cooley, T. F., Richardson, M. P., Walter, I., of Business, N. Y. U. S. S., and Scholes, M. (2010). *Regulating Wall Street: The Dodd-Frank Act and the new architecture of global finance*, volume 608. Wiley Hoboken, NJ.

Basel Committee on Banking Supervision (2010). Basel iii: A global regulatory framework for more resilient banks and banking systems.

Basel Committee on Banking Supervision (2017). High-level summary of basel iii reforms. `https://www.bis.org/bcbs/publ/d424_hlsummary.pdf`. Accessed: 2022-05-01.

Basel Committee on Banking Supervision (2019a). Cre calculation of rwa for credit risk. `https://www.bis.org/basel_framework/chapter/CRE/22.htm`. Accessed: 2022-05-01.

Basel Committee on Banking Supervision (2019b). Rbc risk-based capital requirements. `https://www.bis.org/basel_framework/chapter/RBC/20.htm`. Accessed: 2022-05-01.

Björk, T. (2009). *Arbitrage theory in continuous time.* Oxford university press.

Breeden, D. T. and Litzenberger, R. H. (1978). Prices of state-contingent claims implicit in option prices. *Journal of business*, pages 621–651.

Brunnermeier, M. K. and Niepelt, D. (2019). On the equivalence of private and public money. *Journal of Monetary Economics*, 106:27–41.

Chaum, D., Grothoff, C., and Moser, T. (2021). How to issue a central bank digital currency. *arXiv preprint arXiv:2103.00254.*

Clements, R. (2021). Built to fail: The inherent fragility of algorithmic stablecoins. *Wake Forest L. Rev. Online*, 11:131.

Curry, B. and George, K. D. (1983). Industrial concentration: a survey. *The Journal of Industrial Economics*, pages 203–255.

Efron, B. (1992). Bootstrap methods: another look at the jackknife. In *Breakthroughs in statistics*, pages 569–593. Springer.

Jarrow, R. A. and Turnbull, S. M. (2000). *Derivative securities.* South-Western Pub.

Jorion, P. et al. (2010). *Financial Risk Manager Handbook: FRM Part I/Part II*, volume 625. John Wiley & Sons.

Wheatley, M. (2012). The wheatley review of libor. *Final report.*

A Frankencoin Balance Sheet

Figure 9 presents a stylized balance sheet view of the Frankencoin system. When central banks print and issue currency, the outstanding amounts appear on the liability side of the balance sheet. Similarly, the minted Frankencoins also appear on the same side of the balance sheet. To see how they end up there, one first needs to consider the balance sheet of an individual minter. When a minter mints new Frankencoins, the freshly minted coins appear on the asset side of the minter's balance and at the same time, a repayment obligation is created on the liabilities side. For the Frankencoin system, the opposite happens: the minter's repayment obligation appears on the asset side and the minted ZCHF among the liabilities.

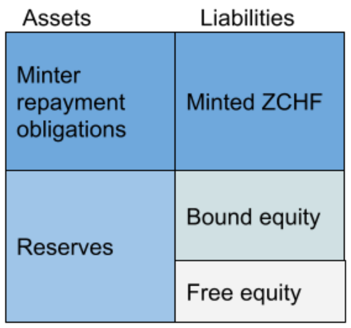


Figure 9: Balance Sheet. This diagram schematizes the balance sheet of the Frankencoin system.

One should note that the total balance sheet of the Frankencoin system is larger than the total number of ZCHF in circulation. That is because the same Frankencoin can appear multiple times in the balance sheet. If a minter sells his ZCHF and the buyer deposits them with the reserves, these Frankencoins appear a second time, this time as reserves with a corresponding equity position on the liabilities side. Equity capital stems from accumulated plugin fees as well as capital contributions paid in by the contributors.

In case a minter cannot meet the repayment obligation, the ZCHF to close the minter's position have to be (partially) taken out of the reserve and equity is reduced. In the depicted balance sheet, bound equity and free equity is distinguished, with the bound equity representing the amount of equity that is needed according to the capital requirements to absorb collateralization failures. The provided collateral itself does not appear on the balance sheet as it belongs to the minter until it is either auctioned off or returned. Free equity could in theory be withdrawn by the contributors or otherwise paid out, for example as interest to the savers.

B Applying Basel Rules for Collateralized Lending

Haircuts are determined based on the type of exposure, see 22.4 in [CRE22]. For main equity indices and gold, the haircut is 15%, for other equities 25% (10 day holding period). Supervisors may also permit the banks to calculate their own haircuts.

Application of the comprehensive Approach

For repo-style transactions, supervisors may choose not to apply the haircuts specified in the comprehensive approach and may instead apply a haircut of zero, if 'the counterparty is a core market participant' as determined by the regulator. This would result in zero RWA because the collateralized mint plugin requires over-collateralization of all loans.

If the Frankencoin system is not exempt from the requirement that the counterparty is to be a core market participant, we may choose the parameters as follows.

$H_e = 0$	The exposure is in CHF (cash), hence no valuation risk
$H_c = 25\%/\sqrt{2}$	The 'other equity' haircut seems to be the most appropriate to reflect collateral in BTC. The 25% are for a 10-day holding period, hence we scale by $1/\sqrt{2}$ to have a 5-day haircut (which is the minimum allowed – our actual holding period is 1 day)
$H_{fx} = 0\%$	The asset is denominated in CHF, hence no additional currency risk

The above numbers are based on a 10-day holding period. If holding periods differ, the percentages are to be scaled by the square-root of time formula. However, for repo-style transactions, a minimum holding period of five days applies. Now, assuming a current loan to value ration of $1/(1 + h^*)$, and an (artificial) holding period of 5 days, we use Equation (23) and set $E = 1$, $C = (1 + h^*)$:

$$E(h^*) = \max[0, 1 - (1 + h^*)(1 - H_c)], \tag{24}$$
$$= \max[0, H_c(1 + h^*) - h^*], \tag{25}$$

where the second line follows from algebra. For example, if $h^* > 0.21$, the resulting exposure is zero (using $Hc = 25\%/\sqrt{2}$, we set the second term in the max-term to zero and solve for h^*). Finally, the exposure E^* needs to be weighted by the counterparty risk weight. The counterparty in the Frankencoin system has generally no rating and cannot be forced to make up for losses in the system, hence the worst weight has to apply, which is w=150%, see Basel Committee on Banking Supervision (2017). Assuming that the loans are just at the (previously defined) liquidation level of $h = 10\%$, we arrive at RWA of

$$E(h)w = 0.09445 \cdot 1.5 = 14\%$$

of the loan amount, of which 8% are required as total capital. Hence, the capital requirement amounts to 1.1% of outstanding loans.

Comprehensive Approach: own Estimates for Haircuts

To calculate the haircuts, a 99th percentile, one-tailed confidence interval is to be used (see 22.50 of [CRE22]), and the choice of historical observation period (sample period) for calculating haircuts shall be a minimum of one year (see 22.53). The minimum holding period is to be set to 5 days. The VaR results in

$$VaR_q = (1 - \exp{(r_q)}) + k, \tag{26}$$

where q is the quantile (99% per Basel iii), r_q is the quantile return, and k the challenger fee. Using the historical data detailed in Appendix C, the 5-day loss at the 99% quantile (using 1-day overlapping returns to calculate historical VaR at 1%-level) amounts to $19.44\% + k = 21.44\%$ (this compares to a haircut of $25\%/\sqrt{2} \approx 17.68\%$ used above). The resulting RWA, using Equation (25) and multiplying by the risk weight, at $h^* = 10\%$ is 20.38%, of which 8% are required as capital.[9] Hence, the capital requirement amounts to 1.6% of outstanding loans.

C Data

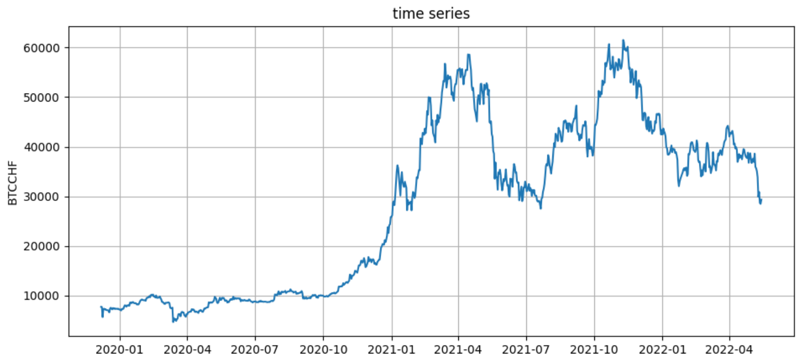


Figure 10: BTCCHF Level Data. This figure plots the level data of the BTCCHF time-series. We have 890 observations of daily candle data without gaps from 2019-12-07 to 2022-05-14.

We gather 1-hour candle data from Kraken, consisting of a timestamp, open, low, high, close, number of trades, and volume.[10] From each open and close price we calculate 24h log-returns. Our return data has the following summary statistics. Table 10 tests the time-series of 24h log-returns

Table 9: Summary Statistics for 24h log-return data.

num. observations	890
min, max	[-49.09%, 25.13%]
mean	0.18%
variance	0.0018
skewness	-1.90
kurtosis	25.93

for stationarity via Augmented Dickey-Fuller test and rejects the null-hypothesis of non-stationarity.

Table 10: Stationarity Test. The ADF test suggests that the log-return data is stationary.

ADF Statistic:	-13.8
p-value:	0.000000
Critical Value for 1%:	-3.4

[9] $\text{RWA} = E(h^*)w = (\text{VaR}(1 + h^*) - h^*)1.5$

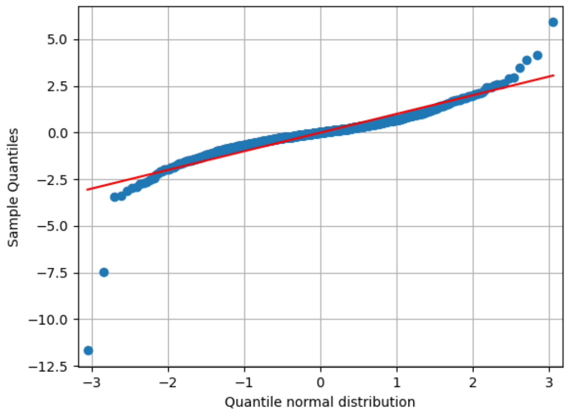
[10] See `https://support.kraken.com/hc/en-us/articles/360047124832-Downloadable-historical-OHLCVT-Open-High-Low-Close-Volume-Trades-data`

Figure 11 present a quantile-quantile plot against the normal distribution and a histogram. Figure 11 also analyses serial correlation of the 24h log-returns. The Autocorrelation Function shows a significant negative correlation at lag one, which however turns out to be driven from extreme returns, as we can see on plot (d).

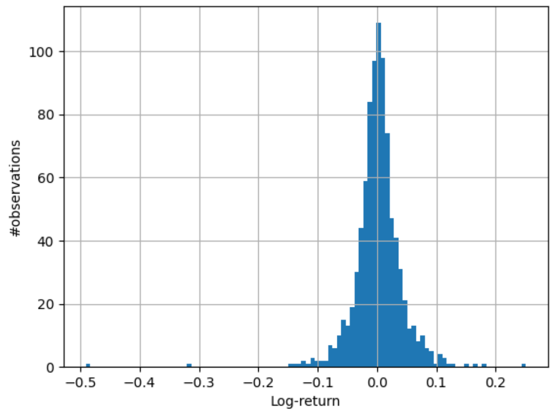
D Frankencoin ERC20 Contract

Listing 1: The Frankencoin token contract, inheriting the standard ERC20 features from an external source.

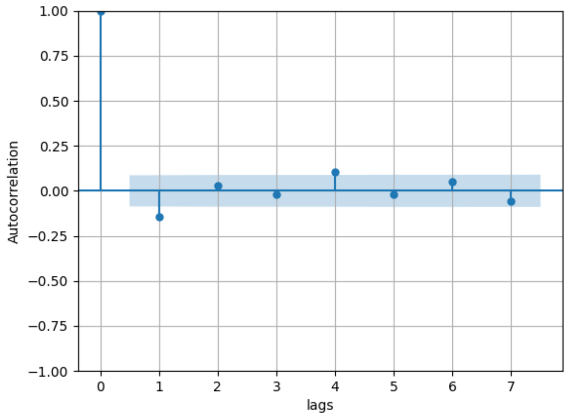
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "./ERC20.sol";
import "./IReservePool.sol";
import "./IFrankencoin.sol";

contract Frankencoin is ERC20, IFrankencoin {

   uint256 private constant MINTER_REMOVED = 1;

   uint256 public constant MIN_FEE = 1000 * (10**18);
   uint256 public constant MIN_APPLICATION_PERIOD = 1000 * (10**18);

   address public immutable reserve;
   uint256 public reserveRequirement;

   mapping (address => uint256) public minters;

   event MinterApplied(address indexed minter, uint256 applicationPeriod, uint256
       applicationFee);
   event MinterDenied(address indexed minter);

   constructor(address _reserve) ERC20(18){
      reserve = _reserve;
   }

   function name() external pure returns (string memory){
      return "Frankencoin V1";
   }

   function symbol() external pure returns (string memory){
      return "ZCHF";
   }

   function suggestMinter(address minter, uint256 applicationPeriod, uint256
       applicationFee) external {
      require(applicationPeriod >= MIN_APPLICATION_PERIOD || totalSupply() == 0, "
          period too short");
      require(applicationFee >= MIN_FEE || totalSupply() == 0, "fee too low");
      require(minters[minter] == 0);
      _transfer(msg.sender, reserve, applicationFee);
      minters[minter] = block.timestamp + applicationPeriod;
      emit MinterApplied(minter, applicationPeriod, applicationFee);
   }

   function denyMinter(address minter, address[] calldata helpers) external {
```



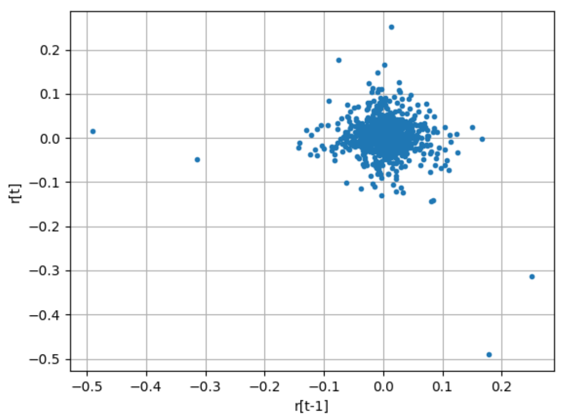
(a)



(b)



(c)



(d)

Figure 11: Return Data from Kraken. Plot (a) shows a quantile-quantile plot of the 24h log-return data against a normal distribution, (b) plots a histogram of the 24h log-returns. From (a) we see that we have more extreme returns than what a normal distribution would suggest. In the center of the distribution, the returns move less than the normal distribution would imply. Plot (c) shows the autocorrelation function and a confidence band at the 0.95-level. Chart (d) plots the returns against the previous day return. The figure indicates that the significant autocorrelation at level 1 must be driven by rare extreme returns.

```
        require(block.timestamp <= minters[minter], "too late");
        require(IReservePool(reserve).isQualified(msg.sender, helpers), "not
            qualified");
        delete minters[minter];
        emit MinterDenied(minter);
    }

    function mintAndCall(address target, uint256 amount, uint256
        reserveRequirementIncrement) external {
        mint(target, amount, reserveRequirementIncrement);
        IERC677Receiver(target).onTokenTransfer(msg.sender, amount, new bytes(0));
    }

    function mint(address target, uint256 amount, uint256
        reserveRequirementIncrement) public {
        require(isMinter(msg.sender), "not approved minter");
        reserveRequirement += reserveRequirementIncrement;
        _mint(target, amount);
    }

    function burn(address target, uint256 amount, uint256
        reserveRequirementDecrement) external {
        require(isMinter(msg.sender), "not approved minter");
        reserveRequirement -= reserveRequirementDecrement;
        _burn(target, amount);
    }

    function burn(uint256 amount) external {
        _burn(msg.sender, amount);
    }

    function notifyLoss(uint256 amount) external {
        require(isMinter(msg.sender));
        _transfer(reserve, msg.sender, amount);
    }

    function isMinter(address minter) public view returns (bool){
        return block.timestamp > minters[minter];
    }

    function reserves() external view returns (uint256) {
        return balanceOf(reserve);
    }

    function hasEnoughReserves() external view returns (bool){
        return balanceOf(reserve) >= reserveRequirement;
    }

}
```

E Reserve Pool Contract

Listing 2: The contract representing the reserve pool.

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.8.0 <0.9.0;
```

```
import "./IFrankencoin.sol";
import "./IERC677Receiver.sol";
import "./ERC20.sol";

/**
 * @title Reserve pool for the Frankencoin
 */
contract ReservePool is ERC20 {

    uint32 private constant QUORUM = 300;

    mapping (address => address) private delegates;

    IFrankencoin public zchf;

    constructor() ERC20(18){
    }

    function initialize(address frankencoin) external {
        require(address(zchf) == address(0x0));
        zchf = IFrankencoin(frankencoin);
    }

    function name() external pure returns (string memory) {
        return "Frankencoin Pool Share";
    }

    function symbol() external pure returns (string memory) {
        return "FPS";
    }

    function price() public view returns (uint256){
        uint256 balance = zchf.balanceOf(address(this));
        if (balance == 0){
            return 0;
        } else {
            return balance / totalSupply();
        }
    }

    function isQualified(address sender, address[] calldata helpers) external
       returns (bool) {
        uint256 votes = balanceOf(sender);
        for (uint i=0; i<helpers.length; i++){
            address current = helpers[i];
            require(current != sender);
            require(canVoteFor(sender, current));
            for (uint j=i+1; j<helpers.length; j++){
                require(current != helpers[j]);
            }
            votes += balanceOf(current);
        }
        return votes * 10000 >= QUORUM * totalSupply();
    }

    function delegateVoteTo(address delegate) external {
        delegates[msg.sender] = delegate;
    }
    function canVoteFor(address delegate, address owner) public returns (bool) {
        if (owner == delegate){
            return true;
        } else if (owner == address(0x0)){
            return false;
        } else {
            return canVoteFor(delegate, delegates[owner]);
        }
    }

    function onTokenTransfer(address from, uint256 amount, bytes calldata)
        external returns (bool) {
        require(msg.sender == address(zchf));
        uint256 total = totalSupply();
        if (total == 0){
            // Initialization of first shares at 1:1
            _mint(from, amount);
        } else {
            _mint(from, amount * totalSupply() / (zchf.balanceOf(address(this)) -
                amount));
        }
        return true;
    }

    function redeem(uint256 shares) external returns (uint256) {
        uint256 proceeds = shares * zchf.balanceOf(address(this)) / totalSupply();
        _burn(msg.sender, shares);
        zchf.transfer(msg.sender, proceeds);
        require(zchf.hasEnoughReserves() || zchf.isMinter(msg.sender), "reserve
            requirement");
        return proceeds;
    }

    function redeemableBalance(address holder) public view returns (uint256){
        return balanceOf(holder) * zchf.balanceOf(address(this)) / totalSupply();
    }


}
```

F Stablecoin Bridge Contract

Listing 3: A mint plugin to create a bridge to an existing Swiss franc stablecoin

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "./IERC20.sol";
import "./IFrankencoin.sol";

/**
 * A minting contract for another CHF stablecoin that we trust.
 */
contract StablecoinBridge {

    IERC20 public immutable chf;
    IFrankencoin public immutable zchf;
    uint256 public immutable horizon;

    constructor(address other, address zchfAddress){
        chf = IERC20(other);
        zchf = IFrankencoin(zchfAddress);
        horizon = block.timestamp + 52 weeks;
    }

    function mint(address target, uint256 amount) external {
        require(block.timestamp <= horizon);
        chf.transferFrom(msg.sender, address(this), amount);
        zchf.mint(target, amount, 0);
    }

    function burn(address target, uint256 amount) external {
        zchf.burn(msg.sender, amount, 0);
        chf.transfer(target, amount);
    }

}
```

G Collateralized Minting Contract

Listing 4: A sample implementation of the presented collateralized minting mechanism. For simplicity, it only allows one minter and one challenge at a time.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "./IERC20.sol";
import "./IReservePool.sol";
import "./IFrankencoin.sol";
import "./Ownable.sol";
import "./IERC677Receiver.sol";

/**
 * A simple collateralized minting contract.
 * This is a proof of concept that only allows one challenge at a time
 * and does not support fractional challenges.
 */
contract CollateralizedMinter is Ownable, IERC677Receiver {

    uint32 public constant BASE = 1000000;
    uint32 public constant COLLATERALIZATION = 1250000; // 125%
    uint32 public constant CHALLENGER_REWARD = 20000; // 2%

    uint256 public constant MIN_CHALLENGE = 100 * 10**18;

    uint256 public constant CHALLENGE_PERIOD = 7 days;

    IERC20 immutable collateral;
    IFrankencoin immutable zchf;
    IReservePool immutable reserve;

    bool private initialized;
    uint32 public challengeCount;
    uint32 public closedChallenges;
    uint256 public coolDownEnd; // cool down after averted challenges
```

```
uint256 public minted; // excluding the reserve contribution
uint256 public mintingLimit; // excluding the reserve contribution
uint256 public depositedCollateral;

mapping (uint32 => Challenge) private challenges;

struct Challenge {
    address challenger;
    uint256 size;
    uint256 end;
    address bidder;
    uint256 bid;
}

constructor(address zchfAddress, address collateralAddress) Ownable(msg.sender
   ) {
    collateral = IERC20(collateralAddress);
    zchf = IFrankencoin(zchfAddress);
    reserve = IReservePool(zchf.reserve());
}

function initialize(uint256 _mintingLimit, uint256 _collateralAmount, uint256
    period, uint256 fee) public {
    require(!initialized);
    initialized = true;
    mintingLimit = _mintingLimit;
    depositedCollateral = _collateralAmount;
    collateral.transferFrom(msg.sender, address(this), _collateralAmount);
    zchf.suggestMinter(address(this), period, fee);
    reserve.delegateVoteTo(msg.sender);
}

function mint(address target) public {
    uint256 amount = mintingLimit; // todo: be more flexible
    require(minted + amount <= mintingLimit);
    uint256 capitalReserve = amount * (COLLATERALIZATION - BASE) / BASE; //
        25% of the minted amount
    zchf.mint(target, amount, 0);
    zchf.mintAndCall(address(reserve), capitalReserve, capitalReserve);
    minted += amount;
}

function payback() external noChallenge {
    uint256 amount = mintingLimit; // todo: be more flexible
    zchf.transferFrom(msg.sender, address(this), amount);
    processPayback(amount);
}

// return Frankencoins without reducing collateral
function onTokenTransfer(address from, uint256 returnedCurrency, bytes
    calldata data) external noChallenge returns (bool) {
    require(msg.sender == address(zchf));
    require(returnedCurrency == mintingLimit);
    processPayback(returnedCurrency);
    return true;
}

function processPayback(uint256 returnedCurrency) internal {
    uint256 poolshare = reserve.redeemableBalance(address(this));
    uint256 mintedIncludingReserve = minted * COLLATERALIZATION / BASE;
    if (poolshare >= mintedIncludingReserve) {
        // our pool share has become so valuable that the loan has paid for
            itself, return everything
        reserve.redeem(reserve.balanceOf(address(this)));
        zchf.burn(address(this), mintedIncludingReserve,
            mintedIncludingReserve - minted);
        zchf.transfer(owner, returnedCurrency + poolshare -
            mintedIncludingReserve);
        minted = 0;
    } else {
        uint256 fundsToRepay = mintedIncludingReserve - poolshare; // that's
            how much the minter must pay to close the position
        if (returnedCurrency > fundsToRepay){
             zchf.transfer(owner, returnedCurrency - fundsToRepay);
             returnedCurrency = fundsToRepay;
        }
        reserve.redeem(reserve.balanceOf(address(this)) * returnedCurrency /
            fundsToRepay);
        uint256 burnAmount = mintedIncludingReserve * returnedCurrency /
            fundsToRepay;
        uint256 reserveAmount = burnAmount  * (COLLATERALIZATION - BASE) /
            COLLATERALIZATION;
        zchf.burn(address(this), burnAmount, reserveAmount);
        minted -= minted * returnedCurrency / fundsToRepay;
    }
}

function withdrawCollateral(address target, uint256 amount) onlyOwner public
    noChallenge {
    internalWithdrawCollateral(target, amount);
}

function internalWithdrawCollateral(address target, uint256 amount) internal {
    uint256 freeCollateral = (depositedCollateral * (mintingLimit - minted)) /
         mintingLimit;
    require(amount <= freeCollateral);
    collateral.transfer(target, amount);
    mintingLimit -= mintingLimit * amount / depositedCollateral;
    depositedCollateral -= amount;
    require(minted <= mintingLimit);
}

modifier noChallenge(){
    require(closedChallenges == challengeCount && block.timestamp >
        coolDownEnd);
    _;
}

function launchChallenge(uint256 challengeSize) external returns (uint32) {
    require(challengeSize >= MIN_CHALLENGE);
    collateral.transferFrom(msg.sender, address(this), challengeSize);
    uint32 number = challengeCount++;
    challenges[number] = Challenge(msg.sender, challengeSize, block.timestamp
        + CHALLENGE_PERIOD, address(0x0), 0);
    return number;
}
```

```
function bid(uint32 challengeNumber, uint256 amount) external {
    Challenge memory challenge = challenges[challengeNumber];
    require(block.timestamp < challenge.end);
    require(amount > challenge.bid);
    if (challenge.bid > 0){
        zchf.transfer(challenge.bidder, challenge.bid); // return old bid
    }
    if (amount * depositedCollateral >= mintingLimit * COLLATERALIZATION /
       BASE * challenge.size){
        // bid above Z_B/C_C >= (1+h)Z_M/C_M, challenge averted, end
           immediately
        zchf.transferFrom(msg.sender, challenge.challenger, amount);
        collateral.transfer(msg.sender, challenge.size);
        closedChallenges++;
        coolDownEnd = block.timestamp + 1 days;
        delete challenges[challengeNumber];
    } else {
        zchf.transferFrom(msg.sender, address(this), amount);
        challenge.bid = amount;
        challenge.bidder = msg.sender;
    }
}

function end(uint32 challengeNumber) external {
    Challenge storage challenge = challenges[challengeNumber];
    require(block.timestamp >= challenge.end);
    // challenge must have been successful, because otherwise it would have
       immediately ended on placing the winning bid
    uint256 challengedCollateral = challenge.size >= depositedCollateral ?
       depositedCollateral : challenge.size;
    uint256 challengedMintings = mintingLimit * challengedCollateral /
       depositedCollateral;
    uint256 challengerReward = CHALLENGER_REWARD * challengedMintings / BASE;
    collateral.transfer(challenge.challenger, challenge.size); // return the
       challenger's collateral
    collateral.transfer(challenge.bidder, challenge.size); // bidder gets
       collateral of owner
    depositedCollateral -= challenge.size;
    if (challengerReward >= challenge.bid) {
        // pay out the reward
        zchf.transfer(challenge.challenger, challengerReward);
    } else {
        // highest bid was lower than challenger reward, this is an
        // edge case, solved slightly differently than in paper
        zchf.transfer(challenge.challenger, challenge.bid);
    }
    uint256 moneyLeft = challenge.bid - challengerReward;
    if (minted == 0){
        // nothing minted yet, return rest of the collateral
        collateral.transfer(owner, depositedCollateral);
        if (moneyLeft > challengedMintings){
            zchf.transfer(owner, challengedMintings);
            zchf.transfer(address(reserve), moneyLeft - challengedMintings);
        } else {
            zchf.transfer(owner, moneyLeft);
        }
        mintingLimit = 0;
        depositedCollateral = 0;
    } else {
            uint256 amountToBurn = challengedMintings * COLLATERALIZATION / BASE;
            if (moneyLeft >= amountToBurn){
                zchf.transfer(address(reserve), moneyLeft - amountToBurn);
            } else {
                zchf.notifyLoss(amountToBurn - moneyLeft);
            }
            zchf.burn(address(this), amountToBurn, amountToBurn -
                challengedMintings);
        }
        closedChallenges++;
        delete challenges[challengeNumber];
    }

}
```