Cryndex: A Decentralised Protocol for Synthetic Assets Pegged to Real-World Values

Alexander Hunter

July 8, 2025

Contents

Abstract

This project presents Cryndex, a decentralised application (dApp) that synthesises stable cryptocurrencies pegged to real-world assets including commodities, stocks, and market indices. Cryndex implements a collateralised borrowing mechanism where users can mint synthetic tokens by depositing collateral, creating a decentralised financial ecosystem that bridges traditional and digital finance. The protocol incorporates elements from established DeFi platforms, borrowing mechanics inspired by Aave's health factor system and a savings rate mechanism modelled after MakerDAO's DAI Savings Rate. This hybridisation creates a comprehensive system where borrowers pay interest to maintain their positions, which in turn subsidises yields for savers who deposit assets into the protocol. Through extensive testing and evaluation, the project demonstrates how blockchain technology can provide access to traditional financial markets without intermediaries. The implementation showcases the technical feasibility of creating synthetic assets with reliable price pegs while maintaining protocol solvency through carefully balanced risk parameters. Cryndex represents a novel approach to democratising access to global financial markets through decentralised technology.

1 Introduction

The traditional financial system presents numerous barriers to entry for average users seeking exposure to diverse asset classes. Geographic restrictions, high entry costs, complex intermediary structures, and regulatory constraints often limit individuals' ability to invest in foreign stocks, commodities, or market indices. Blockchain technology and decentralised finance (DeFi) offer promising solutions to these limitations by enabling permissionless, borderless access to financial instruments.

This project introduces Cryndex—a decentralised application built on blockchain technology that creates synthetic tokens pegged to real-world assets. The name "Cryndex" combines "Crypto" and "Index," reflecting the protocol's capacity to tokenise not just single asset prices, but also a combination of different price feeds. These synthetic assets provide users worldwide with exposure to traditional markets without requiring direct ownership of the underlying assets.

The core functionality of Cryndex revolves around a collateralised debt system where users can deposit cryptocurrency as collateral to mint synthetic tokens representing real-world assets. This mechanism ensures that each synthetic token maintains its peg through over-collateralisation, creating stability within the ecosystem. The borrowing mechanics incorporate key elements from Aave's lending platform, including health factors, liquidation thresholds, and liquidation bonuses that protect the protocol's solvency during market volatility.

Beyond the borrowing functionality, Cryndex implements a savings mechanism inspired by MakerDAO's DAI Savings Rate, allowing users to deposit their synthetic assets to earn yield. This saving rate is funded by the interest payments from borrowers, creating a self-sustaining economic model within the protocol.

The aims of this project are to:

1. Design and implement a scalable architecture for creating and managing synthetic assets pegged to real-world values
2. Develop robust collateralisation and liquidation mechanisms to maintain the peg stability
3. Create an integrated savings protocol that provides yield opportunities for token holders
4. Balance protocol parameters to ensure long-term sustainability and risk management
5. Evaluate the system's resilience against various market conditions and user behaviours

The success of this project will be evaluated through technical stress testing, economic simulations, and analysis of system stability under various market conditions. The evaluation will focus particularly on the reliability of price pegs, the effectiveness of the liquidation mechanisms, and the sustainability of the savings rate model across different market scenarios.

This report presents the theoretical foundations, technical implementation, evaluation findings, and potential future enhancements of the Cryndex protocol, demonstrating how blockchain technology can democratise access to global financial markets through synthetic assets.

2 Background and Theory

2.1 Decentralised Finance and Synthetic Assets

Decentralised Finance (DeFi) has emerged as a significant innovation in the blockchain ecosystem, aiming to recreate traditional financial systems without centralised intermediaries. The rapid growth of DeFi protocols since 2020 has demonstrated substantial market interest in permissionless financial services, with total value locked (TVL) across protocols reaching billions of dollars.

Within this ecosystem, synthetic assets represent tokenised derivatives that mirror the value of real-world assets without requiring direct ownership of the underlying assets. These synthetic assets enable blockchain users to gain exposure to traditional financial markets, commodities, and other assets that would otherwise be inaccessible due to geographic, regulatory, or economic constraints.

2.2 Health Factor and Liquidation Mechanisms

Building upon Aave's risk management framework, Cryndex implements a health factor system to determine the safety of each loan position. The health factor is calculated as:

$$\text{Health Factor} = \frac{\sum_i (\text{Collateral Value}_i \times \text{Liquidation Threshold}_i)}{\sum_j \text{Debt Value}} \tag{1}$$

When a borrower's health factor falls below 1, their position becomes eligible for liquidation. Each accepted collateral currency has a liquidation threshold and liquidation bonus/discount which is determined by Cryndex governance proposals. This mechanism protects the protocol from insolvency by ensuring that undercollateralised positions are quickly resolved before they can negatively impact the system.

The liquidation process involves third-party liquidators who repay a portion of the outstanding debt in exchange for a discount on the collateral, represented by the liquidation bonus. The liquidation bonus incentivises prompt liquidation of risky positions and can be modelled as:

$$\text{Liquidation Bonus} = \text{Debt Repaid} \times (1 + \text{Bonus Rate}) \tag{2}$$

2.3 Interest Rate Models

Cryndex implements a dual interest rate system consisting of borrowing rates paid by borrowers and savings rates earned by depositors. The relationship between these rates is crucial for protocol sustainability. Cryndex token holders (CRYX) can stake their tokens to vote on proposals that determine both borrowing and savings rates. This decentralised governance model ensures that interest rates dynamically reflect market conditions, as token holders collectively adjust rates to balance protocol stability and user incentives.

2.4 Price Oracles and Asset Valuation

Reliable price feeds are essential for maintaining the stability of synthetic assets. Cryndex leverages Chainlink's decentralised price oracles to obtain accurate and manipulation-resistant price data for both collateral assets and the underlying assets that synthetic tokens represent. To mitigate the impact of short-term price volatility, Cryndex uses Chainlink's price feeds with a time-weighted average price (TWAP) mechanism:

$$\text{TWAP} = \frac{\sum_{i=1}^{n} P_i \times T_i}{\sum_{i=1}^{n} T_i} \tag{3}$$

Where P_i represents price observations from Chainlink oracles and T_i represents the time period for each observation. This approach ensures robust asset valuation while relying on Chainlink's proven infrastructure for data integrity.

2.5 Governance and Parameter Adjustment

Drawing from successful DeFi governance models, Cryndex incorporates mechanisms for decentralised governance to adjust key protocol parameters, including:

- Collateralisation ratios for different collateral tokens
- Liquidation thresholds and liquidation bonuses
- Borrow and savings interest rates

These parameters must be calibrated to balance capital efficiency, system stability, and user experience across different market conditions.

2.6 Challenges in Synthetic Asset Protocols

Previous synthetic asset protocols have faced several challenges that Cryndex aims to address:

1. **Peg Stability:** Maintaining the link between synthetic assets and their real-world counterparts during market volatility
2. **Capital Efficiency:** Balancing collateralisation requirements against capital efficiency
3. **Oracle Reliability:** Ensuring price feeds remain accurate and manipulation-resistant
4. **Liquidation Efficiency:** Optimising liquidation parameters to ensure timely resolution of at-risk positions
5. **Scalability:** Designing systems that can support diverse asset types while maintaining consistent risk parameters

The theoretical foundation of Cryndex builds upon these established models while introducing innovations in multi-asset management, risk parameterisation, and economic incentive alignment to create a more robust and versatile synthetic asset platform.

3 Technical Architecture

3.1 General Overview

Cryndex's architecture comprises six smart contracts, each with distinct roles, ensuring modularity and scalability:

- CryndexAsset.sol: An OpenZeppelin ERC20 contract that handles the core logic for transfering and receiving currency whilst also providing functions for calculating the price of the asset and handling which accounts can mint and burn tokens.
- CryndexFactory.sol: A factory contract for deploying new CryndexAsset instances, facilitating the addition of new synthetic assets without altering existing contracts. This design supports scalability and flexibility, aligning with composable design principles. The initial factory will create standard assets while other factories can be created in the future to handle the creation of different synthetic assets with extra functionality.
- CryndexGovernor.sol: Utilises OpenZeppelin's standard governance contract, enabling CryndexToken holders to lock tokens and vote on proposals. This governs critical parameters like interest rates and liquidation thresholds, ensuring decentralised control.
- CryndexSavings.sol: Manages deposits of synthetic assets, offering savers interest yields subsidised by borrow rates. This mechanism, modeled after MakerDAO's DAI Savings Rate, creates a self-sustaining economic model.
- CryndexToken.sol: The ERC20 governance token, allowing holders to participate in voting and earn rewards by staking their tokens, fostering community engagement and protocol governance.
- CryndexTreasury.sol: Handles deposits of collateral and the borrowing of Cryndex assets. The treasury returns the health factor of an account by taking all collateral held by an account as well as all debt minted by an account. Should an account's health factor drop below 1, the account will be open for liquidation.

The Cryndex protocol adopts a modular architecture that adheres to the principle of separation of concerns, ensuring each smart contract is tasked with a specific function, thereby enhancing maintainability, testability, and scalability. This design isolates critical operations such as asset management, governance, and collateral handling into distinct contracts, minimising the risk of systemic failures and enabling targeted upgrades without disrupting the entire protocol. The deployment sequence is carefully organised to establish inter-contract dependencies: the `CryndexToken` and `CryndexGovernor` contracts are deployed first to establish the governance framework, allowing token holders to vote on protocol parameters, such as interest rates and liquidation thresholds. Next, the `CryndexSavings` and `CryndexFactory` contracts are deployed to facilitate yield generation for depositors and the creation of synthetic assets, respectively. The `CryndexTreasury` contract is deployed last, as it serves as the central repository for all other contract addresses, except for synthetic assets, which are managed by the factory. This sequence ensures all required addresses are available before the treasury is initialised, preventing errors during initialisation and ensuring robust integration across the protocol.

3.2 Synthetic Asset Deployment

To deploy a new synthetic asset, the `createAsset` function is invoked on the `CryndexFactory` contract, as illustrated below:

```
/// @notice Creates new asset contracts
/// @param name The name of the asset
/// @param symbol The symbol of the asset
/// @param assetClass The class of asset (commodity, equity, etc.)
/// @param aggregators Chainlink aggregator addresses for the asset
/// @param multipliers The proportionality of each price feed
/// @param borrowRate The initial borrow rate
/// @param savingsRate The initial savings rate
/// @param governor The address of the governor
function createAsset(
    string memory name,
    string memory symbol,
    string memory assetClass,
    address[] memory aggregators,
    uint256[] memory multipliers,
    uint112 borrowRate,
    uint112 savingsRate,
    address governor
) public onlyOwner returns (address) {
    require(
        aggregators.length == multipliers.length && aggregators.length
            > 0,
        "Invalid price feed lengths"
    );
    address asset = address(
        new CryndexAsset(
            name,
            symbol,
            assetClass,
            aggregators,
            multipliers,
            address(treasury),
            governor
        )
    );
    assets.push(asset);
    treasury.addAsset(asset, borrowRate);
    ICryndexSavings(treasury.getSavings()).addAsset(asset, savingsRate)
        ;
    emit AssetCreated(
        name,
        symbol,
        assetClass,
        aggregators,
        multipliers,
        asset
    );
    return asset;
}
```

This function executes several pivotal tasks to integrate the new asset into the Cryndex ecosystem. It deploys a new `CryndexAsset` contract, specifying the asset's name, symbol, class (e.g., commodity, equity), and the address of a Chainlink price aggregator. The function records the new asset's address in the factory's `assets` array for efficient tracking and assigns the asset a minter and burner role in the `CryndexTreasury`, enabling the treasury to handle loans of the asset. Furthermore, it registers the asset with the `CryndexSavings` contract through `addAsset`, establishing an initial savings rate that allows depositors to earn yield on the asset, subsidised by interest payments from borrowers. This creates a self-sustaining economic model within the protocol. The reliance on Chainlink's decentralised oracles ensures robust and reliable price data, while the `onlyOwner` modifier restricts asset creation to authorised governance entities, bolstering security. This modular, secure, and scalable approach to asset deployment exemplifies Cryndex's capability to support a diverse range of synthetic assets, facilitating seamless access to global financial markets within a decentralised framework.

3.3 Deposit

Users may deposit accepted tokens into the treasury which can then be used as collateral for the synthetic assets they wish to borrow. This is the simplest but first action a user must take if they wish to mint tokens themselves. The technical process can be outlined as follows:

1. Transfer tokens from the caller's account
2. If the account is depositing BNB, wrap the BNB into WBNB
3. Update collateral mapping amount for the specified account to be credited
4. Update the total reserves of the collateral asset
5. Emit Deposit event

```
/// @notice Allows an account to deposit ERC20 or native tokens into
    the treasury
/// @param account The account to be credited with the deposit
/// @param token Address of the token to deposit
/// @param amount Amount of tokens to deposit
function deposit(
    address account,
    address token,
    uint256 amount
)
    public
    payable
    amountCheck(amount)
    isAcceptedCollateral(token)
    isCollateralDisabled(token)
{
    if (token == address(0)) {
        require(msg.value == amount, "Invalid amount");
        IWBNB(wbnb).deposit{value: amount}();
    } else IERC20(token).transferFrom(msg.sender, address(this), amount
        );
    collateral[account][token] += amount;
    totalReservesAmount[token] += amount;
    if (!isTokenCollateralised[account][token]) {
        tokensCollateralised[account].push(token);
        isTokenCollateralised[account][token] = true;
    }
    emit Deposit(msg.sender, account, token, amount, block.timestamp);
}
```

3.4 Withdraw

If a user would like to withdraw their collateral, they must make sure that the amount they withdraw would mean that their health factor remains above 1. If not, then their transaction would be reverted. The steps for withdrawls are as follows:

1. Decrement the caller's collateral mapping amount
2. Decrement the total reserves of the token mapping
3. Check that the caller's health factor remains above 1 after withdrawl
4. Calculate fee of 0.1% of the withdrawn value
5. Transfer withdrawn tokens to specified recipient, the Cryndex protocol and the staking contract for rewards
6. Emit the Withdraw event

```
/// @notice Withdraws an ERC20 token stored in the treasury to an
    address
/// @param account The account to receive the withdrawn tokens
/// @param token Address of the ERC20 token
/// @param amount Amount to withdraw
function withdraw(
    address account,
    address token,
    uint256 amount
) public amountCheck(amount) isAcceptedCollateral(token) {
    require(
        collateral[msg.sender][token] >= amount,
        "Insufficient balance"
    );
    collateral[msg.sender][token] -= amount;
    totalReservesAmount[token] -= amount;
    if (collateral[msg.sender][token] == 0) {
        delete collateral[msg.sender][token];
        isTokenCollateralised[msg.sender][token] = false;
        for (uint8 i; i < tokensCollateralised[msg.sender].length; i++)
            {
            if (tokensCollateralised[msg.sender][i] == token) {
                tokensCollateralised[msg.sender][i] =
                    tokensCollateralised[
                     msg.sender
                ][tokensCollateralised[msg.sender].length - 1];
                tokensCollateralised[msg.sender].pop();
                break;
            }
        }
    }
    require(calculateHealthFactor(msg.sender) >= 1e18, "HF < 1");
    uint256 fee = (amount * CryndexUtilsLibrary.FEE_PCT) /
```

```
        CryndexUtilsLibrary.PRECISION;
    token = token == address(0) ? wbnb : token;
    IERC20(token).transfer(account, amount - fee);
    IERC20(token).transfer(owner(), fee / 2);
    if (ICryndexStaking(staking).getTotalStakedBalance() == 0) {
        IERC20(token).transfer(owner(), fee / 2);
    } else {
        IERC20(token).approve(staking, fee / 2);
        ICryndexStaking(staking).depositReward(token, fee / 2);
    }
    emit Withdrawal(msg.sender, account, token, amount, fee);
}
```

3.5 Borrow

When a user has deposited collateral into their account, they may mint new Cryndex assets backed by their collateral. If the requested borrow amount would cause the borrower's health factor to drop below 1, then the transaction would revert. The steps for borrowing are as follows:

1. Get current debt amount of the token including accrued fees
2. Add the current debt amount and the extra debt to be borrowed and assign it to the borrowers base debt amount
3. Update debt indices and timestamps both on the borrower and globally
4. Check that the caller's health factor remains above 1 after borrow
5. Mint 99.9% of the requested borrow amount to the specified recipient, 0.05% of the amount to the Cryndex protocol and 0.05% for staking rewards
6. Emit the Borrow event

```
/// @notice Borrows new asset tokens
/// @param account The account to receive the borrowed tokens
/// @param asset Address of the asset to be borrowed
/// @param amount Amount of tokens to be borrowed
function borrow(
    address account,
    address asset,
    uint256 amount
) public amountCheck(amount) isCryndexAsset(asset) {
    bool found;
    uint256 debtAmount = getDebtBalance(msg.sender, asset);
    debt[msg.sender][asset] = debtAmount + amount;
    (, uint256 lastBorrowRateIdx) = getCurrentBorrowRate(asset);
    lastDebtUpdateIdx[msg.sender][asset] = lastBorrowRateIdx;
    lastDebtUpdateTimestamp[msg.sender][asset] = block.timestamp;
    incrementTotalDebtAmount(asset, amount);
    for (uint8 i; i < assetsBorrowed[msg.sender].length; i++) {
        if (assetsBorrowed[msg.sender][i] == asset) {
            found = true;
            break;
        }
    }
    if (!found) assetsBorrowed[msg.sender].push(asset);
    require(calculateHealthFactor(msg.sender) >= 1e18, "HF < 1");
    uint256 fee = (amount * CryndexUtilsLibrary.FEE_PCT) /
        CryndexUtilsLibrary.PRECISION;
    ICryndexAsset(asset).mint(account, amount - fee);
    fee = fee / 2;
    ICryndexAsset(asset).mint(owner(), fee);
    if (ICryndexStaking(staking).getTotalStakedBalance() == 0) {
        ICryndexAsset(asset).mint(owner(), fee);
    } else {
        ICryndexAsset(asset).mint(address(this), fee);
        ICryndexAsset(asset).approve(staking, fee);
        ICryndexStaking(staking).depositReward(asset, fee);
    }
    emit Borrow(msg.sender, account, asset, amount, fee);
}
```

3.6 Repay

If a user wants to unlock their collateral from the treasury or pay back some interest to manage their health factor, they muse use the repayment function. By repaying borrowed assets, a borrower's health factor would increase which would decrease the likelihood of their liquidation. The steps for repayments are as follows:

1. Get current debt amount of the token including accrued interest
2. Decrement the debt amount to the specified account
3. Update the debt indices on the specified account and globally
4. Burn the amount specified from the caller
5. Emit the Repayment event

```
/// @notice Repays the asset and burns the tokens
/// @param account The account who's debt will be repaid
/// @param asset Address of the asset to be repaid
/// @param amount Amount of tokens to be repaid
function repay(
    address account,
    address asset,
    uint256 amount
) public amountCheck(amount) isCryndexAsset(asset) {
    uint256 debtAmount = getDebtBalance(account, asset);
    amount = amount > debtAmount ? debtAmount : amount;
    debt[account][asset] = debtAmount - amount;
    (, uint256 lastBorrowRateIdx) = getCurrentBorrowRate(asset);
    lastDebtUpdateIdx[account][asset] = lastBorrowRateIdx;
    lastDebtUpdateTimestamp[account][asset] = block.timestamp;
    if (debt[account][asset] == 0) {
        delete debt[account][asset];
        for (uint8 i; i < assetsBorrowed[account].length; i++) {
            if (assetsBorrowed[account][i] == asset) {
                assetsBorrowed[account][i] = assetsBorrowed[account][
                    assetsBorrowed[account].length - 1
                ];
                assetsBorrowed[account].pop();
                break;
            }
        }
    }
    decrementTotalDebtAmount(asset, amount);
    ICryndexAsset(asset).burn(msg.sender, amount);
    emit Repayment(account, msg.sender, asset, amount, block.timestamp)
        ;
}
```

3.7 Liquidations

Liquidations are an essential part of the Cryndex protocol as it allows Cryndex assets to justify their value through backing of cryptocurrency. If the value of borrowers' collateral were to drop or the value of the debt asset were to rise, then the tokens a borrower has minted can no longer justify its value. In this case, a liquidator can come along and pay back 50% of the debt owed by a borrower in exchange for a 1:1 value ratio of a borrower's collateral plus a liquidation bonus determined by the governance system and which is set for each collateral token. The logic for liquidations can be seen in the following function:

```
/// @notice Computes liquidation parameters and updates debt position
/// @param token Collateral token to seize
/// @param asset Debt asset to repay
/// @param debtToCover Amount of asset to repay
/// @param account Account to liquidate
/// @return Amount Liquidated
function liquidate(
    address token,
    address asset,
    uint256 debtToCover,
    address account
)
    public amountCheck(debtToCover) isAcceptedCollateral(token)
       isCryndexAsset(asset) returns (uint256)
{
    require(calculateHealthFactor(account) < 1e18, "HF >= 1");
    uint256 debtBalance = getDebtBalance(account, asset);
    uint256 maxDebtToCover = debtBalance / 2;
    require(debtToCover <= maxDebtToCover, "Covering too much debt");
    require(
        ICryndexAsset(asset).balanceOf(msg.sender) >= debtToCover,
        "Insufficient balance"
    );
    uint256 debtPrice = ICryndexAsset(asset).getAssetPrice();
    uint256 liquidationBonus = getLiquidationBonus(token);
    uint256 collateralPrice = latestAnswer(token);
    uint256 collateralToSeize = (debtPrice *
        debtToCover *
        (CryndexUtilsLibrary.PRECISION + liquidationBonus)) /
        (collateralPrice * CryndexUtilsLibrary.PRECISION);
    require(
        collateral[account][token] >= collateralToSeize,
        "Insufficient collateral balance"
    );
    (, uint256 currentBorrowRateIdx) = getCurrentBorrowRate(asset);
    collateral[account][token] -= collateralToSeize;
    debt[account][asset] = debtBalance > debtToCover
        ? debtBalance - debtToCover
        : 0;
    lastDebtUpdateIdx[account][asset] = currentBorrowRateIdx;
    lastDebtUpdateTimestamp[account][asset] = block.timestamp;
    totalReservesAmount[token] -= collateralToSeize;
    decrementTotalDebtAmount(asset, debtToCover);
    ICryndexAsset(asset).burn(msg.sender, debtToCover);
    token = token == address(0) ? wbnb : token;
    IERC20(token).transfer(msg.sender, collateralToSeize);
    emit Liquidation(
        msg.sender,
        token,
        asset,
        debtToCover,
        collateralToSeize,
        account
    );
    return collateralToSeize;
}
```

3.8 Debt Accuration Algorithm and Savings Contract

Borrowers are also subject to a borrow rate which increases the debt that they owe to the protocol. This borrow rate is essential to subsidise the savings rate in the savings contract. The algorithm for calculating the interest owed by a borrower works as follows: a loop is started at the index of array of interest rates set by the governance module that a borrower last borrowed or made a repayment on their loan as well as the timestamp of this last update. For each iteration of the loop borrow rate at each index is taken as well as the start time and end time so that the period in seconds can be calculated. The debt amount is then added to the total debt amount by a simple principle interest calculation (not compounding interest). At each iteration of the loop the interest is accumulated onto the original debt amount and can be described by the following equation:

$$D_{\text{new}} = D_0 + \sum_{i=\text{startIdx}}^{\text{lastIdx}} \frac{D_i \cdot R_i \cdot T_i}{Y \cdot 10^{18}} \tag{4}$$

where:

- D_0 is the initial debt amount,
- D_i is the debt amount at the start of period i,
- R_i is the borrow rate for period i,
- T_i is the time elapsed for period i (in seconds),
- Y is the number of seconds in a year ($365 \times 24 \times 60 \times 60 = 31,536,000$),
- 10^{18} is the precision constant,
- startIdx and lastIdx are the indices of the first and last borrow rate periods.

Instead of transferring paid interest from repayments made by borrowers to savers, the savings contract can mint the synthetic asset on demand as it has been assigned the MINTER_ROLE. There could be a situation where there is only one generator of a synthetic asset that has deposited into the savings contract. The borrower's own interest payments are meant to subsidise the savings interest payments which would not yet be available for a withdraw from the savings contract. By minting the synthetic asset within the savings contract, the borrower would have immediate access to the interest payments which they would have to pay back to the protocol regardless. When a borrower wants to unlock their collateral, they must pay back some or all of the loan whilst retaining a health factor above 1. The debt accuration alogrithm is the same for calculating interest earnt by depositers in the `CryndexSavings` contract. This is to ensure that the same amount of borrowing interest payments accurately subsidise the savings contract. The debt accuration algorithm is a simple principle interest rate calculation that accumulates each time a new interest rate is set by the governance system. A compound interest rate algorithm was considered for Cryndex but was rejected for its extra complexity and for an MVP system, a simpler algorithm provides the necessary yield generations to encourage adoption whilst also reducing gas fees further encouraging adoption.

3.9 Flash Loans and Flash Minting

Flash loans, inspired by Aave, is a DeFi feature allowing smart contracts to borrow all available collateral in the Cryndex treasury. This enables operations like arbitrage or liquidations without needing collateral for borrowing. In order for a flash loan to be successful, a borrower must repay the full loan amount plus a fee otherwise the transaction is reverted and it is as if the funds never left the treasury. Given that there will be idle capital backing all Cryndex Assets in the treasury, this is a great way to increase capital efficiency whilst also minimising risk for lenders.

Flash minting is the process of minting an infinite amount of tokens for any given Cryndex Asset, similar to a flash loan. Flash minting is essential to the Cryndex ecosystem as it allows arbitragers to keep the price of Cryndex Assets pegged to the correct value across different decentralised exchanges.

Flash minting works in exactly the same way as flash loans, the only difference to flash loans is that the funds will not be transferred back to the treasury and will instead be burned from the borrower's account. If the borrower doesn't have the funds they borrowed plus the fees, the transaction will be reverted.

4 Arbitrage for Price Stability

Cryndex Assets maintain their price stability through incentives of arbitrage. For example, each CRAU token represents a specific amount of gold, with its value tied to the market price of gold rather than a fiat currency. To ensure price stability, Cryndex employs a mechanism akin to MakerDAO's: if CRAU's market price on exchanges rises above the gold peg, users are incentivised to create more CRAU by depositing collateral, increasing supply and pushing the price down. Conversely, if CRAU trades below the peg, users can repay their CRAU debt to unlock collateral, reducing supply and raising the price back toward the peg. This arbitrage mechanism, supported by transparent crypto reserves and smart contract governance, mirrors DAI's supply-demand dynamics but anchors CRAU's value to physical gold, blending blockchain efficiency with gold's stability. These market dynamics work for all Cryndex assets meaning all price feeds available can be tokenised.

5 Cryndex (CRYX) Tokenomics

The Cryndex token (CRYX) allows stakers to vote on governance proposals on the protocol, this includes the borrow and savings rates on each Cryndex Asset as well as the liquidation threshold and bonus of each collateral token. Stakers will also earn rewards based on how many tokens they stake vs how many total tokens are staked. Rewards will be deposited via the treasury contract from protocol fees. In upcoming versions of the Cryndex protocol, we intend to add reduced borrow rates for Cryndex stakers as well as increased savings rates. Stakers will also have a cooldown period of 14 days after staking or unstaking tokens where they will not be able to unstake any tokens over the period. Staking more tokens will mean that the cooldown period restarts. The CRYX token has a maximum supply of 200 million tokens and will never increase to prevent inflation. This will encourage the long term usage of the Cryndex application as the price of the token will increase with the popularity of the protocol.

6 Conclusion

In conclusion, Cryndex successfully delivers a decentralised platform for synthetic assets, achieving its aim of democratising financial market access through robust collateralisation, savings, extra asset denominations and risk management mechanisms. Its modular design, comprehensive testing, and innovative features establish it as a competitive player within the BNB network. Critical reflection underscores the need to address oracle, governance, and scalability risks, while proposed further work, including layer-2 integration, cross-chain bridges, and dynamic interest rate models, offers transformative potential to elevate Cryndex's impact in DeFi. By aligning technical innovation with user-centric design, Cryndex paves the way for a more inclusive, efficient, and resilient financial ecosystem, fulfilling its vision of bridging traditional and digital finance.