DeBond: Decentralised Securities Solution

Yu LIU, Songbo WANG, Samuel Gwlanold EDOUMOU, Toufic BATRICE

info@debond.org

debond.org,

Abstract

DeBond is a decentralised securities solution based on Ethereum Virtual Machine [1]. Earlier versions of the ERC-20 [2] Lp token is a simple mathematical proof of the loan, fungible representation of the amount of the asset deposited into a smart contract. In comparison, DeBond uses EIP-3475: Multiple Callable Bonds Standard [3] to identify each single deposit. This means that the liquidity provider signs a derivatives agreement with the issuer. On which, both parties agree certain repayment conditions and interest rate. EIP-3475 can divide a LP into countless small sub-categories. Each part represents a derivative contract. This law of obligation can be then sold on a secondary market, like any derivative product in the current centralised economic system.

Contents

1 Definition

1.1 Formal Definitions

change order of variable introduction

Variable	Description
$B\,(c, n, a)$	Bond balance associated to address a, of class c, and of nonce n. $$\begin{cases} c \subseteq \mathbb{Z} \wedge 0 < a < 2^{256} - 1 \\ n \subseteq \mathbb{Z}\ 0 < a < 2^{256} - 1 \\ a \subseteq \mathbb{Z}_{\text{hex}} \wedge 0 < a < 2^{256} - 1 \end{cases}$$
$B_{total}(\langle c_i \rangle_{i=1}^{C})$	The sum of all the balances of all the nonce of all the classes in the sequence $\langle c_i \rangle_{i=1}^{C}$. $$B_{total}(\langle c_i \rangle_{i=1}^{C}) = B_{sum}(n_l)$$
$B_{\text{sum}}(n)$	The sum of all the balances of all the nonce from the first nonce 0 to the n-th nonce . $$\sum_{i=0}^{n} B(\langle c_i \rangle_{i=1}^{C}, n)$$
$\langle c_i \rangle_{i=1}^{C} = \langle c_1,\, c_2,\, c_3,\, \cdots,\, c_C \rangle$	A sequence of classes, C=the number of elements in the sequence.
$b_{erc20}(a)$	Balance of an ERC-20 token of an address a
F_n	The n-th Fibonacci number, with $F_0 = 0, F_1 = F_2 = 1, F_3 = 2, \ldots$
n_{now}	Nonce of the current block time T_l of a class c
$n_l(c)$	Nonce of the last nonce issued defined by `last_nonce[c]`
$\langle n_i \rangle_{i=1}^{N} = \langle n_1,\, n_2,\, n_3,\, \cdots,\, n_N \rangle$	A sequence of nonce.
T	Number of seconds defined by `block.timestamp`
T_{year}	Number of seconds in a year T_{year}=31536000
T_{epoch}	`time_epoch` Length of a nonce in time

Variable	Description
$T_{\text{start}}(c, n)$	Number of seconds of the starting time of a bond class and of a bond nonce, defined by `bondClass[c]._nonceInfo[n][0]`
. $T_{\text{maturity}}(c, n)$	Number of seconds of the maturity time of a bond class and of a bond nonce, defined by `bondClass[c]._nonceInfo[n][1]`
T_l	Number of seconds of the current block time defined by `time.now`
$\Delta n(dT)$	Difference in nonce of a difference in time dT $$\Delta N(dT) = \frac{dT}{T_{epoch}}$$
ΔT	Time lapse between two timestamps T_1, T_2
$\Delta T(c, n)$	Time lapse from the starting time of a nonce to the start of the next nonce $$\Delta T(c, n) = T_{start}(c, n+1) - T_{start}(c, n)$$
$s_{\text{collateral}}$	All collateralised supply of a toke, locked in LP $$s_{\text{collateral}} = \sum_{i=0}^{n} l_i n$$
s_{total}	Output of ERC-20 `totalSupply()` function, the total supply of a token
s_{max}	Output of ERC-20 `maximumSupply()` function, the maximum supply of a token
t_i, l_i	t_i represents an ERC-20 token, and l_i is the amount of liquidity of the corresponding token
$P\left(t_1, t_2; l_1, l_2\right)$	Price of unit token t_1 of t_2, calculated by $P = \frac{l_2}{l_1}$
$P_{determined}\left(t_1, t_2; l_1, l_2, T\right)$	The arbitrated price of unit token t_1 of t_2, calculated by $P = \frac{l_2}{l_1}$ at a giving time T

Variable	Description
$C_{\text{DBIT}}(s_{\text{total}})$	Minting Cost, the contract determined price of unit DBIT in USD, calculated by: $$1.05^{\log_e s_{\text{total}}}$$
$C_{\text{DGOV}}(s_{\text{total}}, s_{\max})$	Price of unit DGOV in DBIT, calculated by: $$100 + (S_{total}10000/S_{max})^2/100000$$
$k(t_1, t_2)$	Total liquidity of the automatic market maker (AMM) for tokens t_1, t_2, the amount of the tokens l_1, l_2 being fixed by the relation $l_1 l_2 = k(t_1, t_2)$
$p(t_1, t_2)$	The pair liquidity pool of t_1, t_2
$l_{\text{determined}}(t_1, t_2; l_1)$	The output liquidity l_2 of contract determined price (CDP) method
π^*	The principal used in an investment.
I	The face interest from an interest settlement $I = \pi\iota$
ι	The face interest rate of an investment from an interest settlement $\frac{I}{\pi}$
$\iota_{benchmark}$	The benchmark interest rate of an investment from an interest settlement. A variable defined by `IR.benchmarkIR` and can be updated by `governanceContract.updateBenchmarkIR()` $$2\iota_{benchmark} = \iota_{float} + \iota_{fix}$$
$\iota_{float}(t)$	The floating interest rate of an investment from an interest settlement. Based on benchmark IR and the ratio of floating rate bond to fixed rate bond of the same settlement token t. $$\iota_{float}(t) = 2\iota_{benchmark}\frac{B_{fix}(t)}{B_{float}(t) + B_{fix}(t)}$$

Variable	Description
$\iota_{fix}(t)$	The fixed interest rate of an investment from an interest settlement. Based on benchmark IR and the ratio of floating rate bond to fixed rate bond of the same settlement token t. $$\iota_{fix}(t) = 2\iota_{benchmark}\frac{B_{float}(t)}{B_{float}(t) + B_{fix}(t)}$$
ι_{total}	The total nominal interest rate of an investment $$\sum_{i=0}^{n} \iota_i$$
ι_{compound}	The compound interest rate of an investment, with n times of settlement $$(\prod_{i=1}^{n}(1+\iota_i)) - 1$$
$\iota_{\text{remaining}}$	The remaining interest rate of an investment from χ-th interest settlement to n-th $$\sum_{i=\chi}^{n} \iota_n$$
λ	The amount of a loan
Γ	The value of the collateral
ρ	The collateral rate of a derivative $\frac{\lambda}{\Gamma}$
$T_{estimate}(n)$	The estimated redemption time based on the average liquidity flow over last month μ_{month}, and the additional liquidity needed. $$T_{maturity}(n) + \frac{B_{\text{sum}}(n)(1+\iota_{benchmark}) - B_{\text{sum}}(n_l)}{\mu_{month}} T_{epoch}$$

2 Introduction

In this paper, we present DeBond, a novel derivatives based LP proof system that gives liquidity providers and the borrower a customizable agreement over their loan/debt. Also provides tools to managing the redemption conditions and price ranges in which their capital is used. With the method of ERC-3475 Multiple Callable Bonds Standard, the systematic way of the liquidity fragmentation and gas efficient transaction, new financial derivatives can be introduced to decentralised finance.

Bonds tend to be less volatile and less risky than other financial products. Other financial derivatives can be used for Hedge and insurance. Thus reduce the risk involved. Because of those facts, they are some of the most important tools of financial markets. They serve as a stabilizer for the system. They perform well when the market is in decline, as interest rates fall and bond prices rise in turn.

If the main goal of an investor is to always be able to get their principal back from the staking, a stable bond is their best choice. When they stake for a company or a government bond, they pay for their expectation of the future of the borrower. If needed, as debtors, buyers of the bond can always get some of their liquidity back, either through pledging them for loans or selling on the secondary market. The bond holders, being creditors have a higher priority level then the stockholders. If bankrupt, the asset of the borrower will be first used to repay the bondholders. And if there are any left, pay for the stock owner.

Decentralised finance is one of the most volatile markets. Where the trust is not needed, and the risk is omnipresent. What the market really needs is a decentralised bonds system based on trust-less smart contracts, and derivatives that can be used to hedge against the risk. The huge bond LP will be used to stabilize the market. The bond holders can then get relatively stable income from the expansion of the market. During a declining phase, The overly sold bond can be burned by DBGP. like bad debt being relieved by the state intervention. The economy can then go into the next expansion phase [5].

- **LP token:** fungible mathematical proof of a loan to a liquidity pool $p\left(t_1, t_2\right)$.
- **ERC-3475:** replacement of the ERC-20 LP token. ERC3475 is non-fungible, computationally structured and gas-efficient.
- **Bond:** instrument of indebtedness of the bond issuer to the holders.

- **Derivative:** A contract between two or more parties, whose value is based on an agreement of some market index or financial asset.
- **DeBond:** decentralised bond market system implemented using ERC-3475.
- **Derivative:** Contract between two or more parties whose value is based on an agreed-upon underlying financial asset (like a security) or set of assets (like an index). Common underlying instruments include bonds, commodities, currencies, interest rates, market indexes, and stocks.
- **Bond:** instrument of indebtedness of the bond issuer to the holders.
- **Bond DEX:** decentralised bond exchange, allowing any ERC-3475 bond to be traded on a secondary market.
- **Fixed interest rate DeBond:** DeBond that pays the same level of interest over its entire term.
- **Floating interest rate DeBond:** DeBond that does not have a fixed rate of interest over the life of the instrument.
- **Coupon:** interest rate that the issuer pays to the holder. For fixed rate bonds, the coupon is fixed throughout the life of the bond. For floating rate notes, the coupon varies throughout the life of the bond and is based on the movement of a money market reference rate.
- **Yield:** rate of return received from investing in the bond.
- **Securitized Debt:** is the tradable security that represents the right over an income-producing asset, or a debt.
- **DeBond:** decentralised bond market system implemented using ERC-3475.
- **AMM:** Automated market makers, agents that pool liquidity and make it available to traders according to x*y=k.
- **CDP:** Contract Determined Price, nonlinear function that arbitrarily controls the minting cost of a certain Token.
- **APM**(t_1, t_2, l_1)**:** Automated Pair Maker method, method that automatically pairs a single token to a LP. LP providers need only one token t_1 to provide liquidity for a pair $p\,(t_1, t_2)$.
- **DeBond Bank,** set of smart contracts, trustee and bond issuer. Settlement currencies, currency to be used to repay securities or interests in securities.

- **DBIT,** DeBond Index Token (DBIT), ERC-20 token, a quantified representation, measuring the investment performance and characteristics of the DeBond market. DBIT is also the settlement currency of the DeBond ecosystem.
- **DBIT Bond:** The redemption of the DBIT bond is paid in DBIT.
- **Minting Cost:** is the price per token of the bond in the first market defined by CDP.
- **QTM:** Quantity Theory of Money tells that the general price level of goods and services is proportional to the money supply in an economy. Debond protocol uses this theory to control the market supply and velocity of the bond settlement token DBIT.
- **DeCapital:** decentralised model of a funds management entity, allowing the trusted liquidity to be more efficiently used, and generate income for the creditor.
- **Proposal Contract:** Logical subject of a VM recognisable proposal. Converted from nature language to a computational representation.
- **DBGP:** Decentralised Bond Governance Platform. Current governance platforms such as snapshot, provide only a place for counting votes. No obligations are imposed to the private key owner. Where DBGP obliges the creator of the proposal to convert his idea into a proposal contract. The access to the governance contract can then be given to this proposal contract. This platform is also used in the managements of DeCapital.
- **DGOV:** Governance token of DBGP.

3 DEBOND

DeBond is a financial tool to securitize any form of digital debt or asset into a bond class. This class can be then fragmented into sub-categories. We use the term "nonce" to represent such sub-categories. Nonces are not interchangeable, We can consider each nonce as a separate ERC-721 derivative contract.

3.1 Features

- **Non-fungible:** Not like any form of ERC-20 LP token, a DeBond with different nonce is not mutually interchangeable, hence not fungi-

ble.

- **Derivatives:** Each DeBond can be regarded as a smart contract based derivative.
- **Fragmented:** DeBond can fragment liquidity pool, debt and any form of the digital asset into many pieces. Each fragment represents a separate sub-category of the object in question.
- **Multi-dimensional:** Each ERC3475 DeBond stores an array of integers, not like a ERC-20 token can store only a mapping from an address to a balance. This feature can be used to store redemption conditions and different interest rates.
- **Gas-efficient:** Current existing fragmentation of LP or NFT use ERC-20 token standard. This means each fragmentation of NFT or LP needs the deployment of a new smart contract. Thus generate unnecessary gas expense,
- **Multiple fragments:** Because of the high gas spent, the current fragmentation systems must limit the number of classes and sub-categories generated. ERC-3475 can in turn generate almost countless classes and sub-categories without the need to publish a new smart contract.
- **Callable:** the debts that a DeBond represents must be fulfilled when the conditions are met. It can be a fixed amount of repayment or any floating interest based on a predetermined index factor.
- **Obligated:** ERC-3475 obliges both the borrowers/debtors and the creditors to fulfil their promise. This feature can be used to make sure that the LP will not be emptied suddenly. In return the borrower will ensure the promised yield.
- **Marketable:** DeBond is itself a financial product that can be exchanged and speculated on a secondary market.
- **Securitised:** An derivative issuer can design a marketable financial instrument by merging or pooling various derivative contract into one group. The issuer can then sell this group of repackaged assets on the secondary market.

3.2 EIP-3475

A standard interface for contracts that manage multiple callable bonds. A single contract includes any given number of bond classes, bond nonce, bond balance of an address. This standard provides independent functions to

read, transfer any collection of bonds, as well as allow bonds to be redeemed from the bond issuer if certain conditions are met. This token standard can replace current ERC-20 LP token. ERC-3475 has a more complex data structure, which will allow the LP token to store more information, and allow the developer to build more sophisticated logic for the redemption and reward system of the DEFI project in question.

This API standard allows for the creation of any number of bond types in a single contract. Existing LP token standards like ERC-20 require deployment of separate factory and token contracts per token type. The need of issuing bonds with multiple redemption data can't be achieved with existing token standards. ERC-3475 Multiple Callable Bonds Standard allows for each bond class ID to represent a new configurable token type, and for each bond nonce to represent an issuing date or any other forms of data in uint256. Every single nonce of a bond class may have its own metadata, supply and other redemption conditions.

Current LP token is a simple ERC-20 token, which has not much complicity in data structure. To allow more complex reward and redemption logic to be built, we need a new LP token standard that can manage multiple bonds, store much more data and is gas efficient. ERC-3475 standard interface allows any tokens on solidity compatible block chains to create its own bond. These bonds and derivatives with the same interface standard can be exchanged in the secondary market. And it allows any 3rd party wallet applications or exchanges to read the balance and the redemption conditions of these tokens. ERC-3475 bonds and derivatives can also be packed into separate packages. Those packages can in their turn be divided and exchanged in a secondary market.

New functions built in ERC-3475 Multiple Callable Bonds Standard, will allow the users to economize their gas fee spend. Trading and burning of ERC-3475 Bonds will also multiply tokens market cap, helping it to recover from recession period. Existing structures, such as AMM exchanges or lending platforms can be updated to recognize ERC-3475 bonds or derivatives.

ERC-3475 is a tokenized exchangeable asset. A bond contract can store the data of 2^{256} bond classes and for each class, 2^{256} bond nonces. Unlike a typical ERC-20 or ERC-721 token, an ERC-3475 bond is semi-fungible and multi-dimensional. Any bond nonce of a class is non-fungible from another nonce or class. It has an identified uint256 array, in which we can store the redemption time, interest rate, redemption conditions and other info. For example:

3.2.1 Balances

need to be more clear

`"bondClass[class]._balances"` is the array of bond class to a nonce, and to the balance. `mapping(address =>mapping(uint256=> uint256)) _balances;` is the balance of an address are associated with the following 3d array: *B (a,c,n)*, where a is the vector of address, c is the vector of class and n is the vector of nonce.

$$\begin{cases} c \subseteq \mathbb{Z} \wedge 0 < a < 2^{256} - 1 \\ n \subseteq \mathbb{Z} \wedge 0 < a < 2^{256} - 1 \\ a \subseteq \mathbb{Z}_{hex} \wedge 0 < a < 2^{256} - 1 \end{cases}$$

e.g.

`bondClass[1].0x2d03B6C79B75eE7aB35298878D05fe36DC1fE8Ef=> (5 => 500000000));`
this example gives the balance of:
address 0x2d03B6C79B75eE7aB35298878D05fe36DC1fE8Ef possess 500000000 of bond class 1, bond nonce 5.

3.2.2 Multidimensional data structure

```
mapping (uint256=> uint256[]) _nonceInfo;
string[] nonceInfoDiscription;
bondClass[class]._nonceInfo[n][];
bondClass[class].nonceInfoDiscription[];
```

`_nonceInfo` is the mapping from a bond nonce to a list of uint256 variables, list of bond nonce to bond info. In this list, the 0th variable MUST be $T_{start}(c, n)$, the starting time of the nonce in question. the 1st variable MUST be $T_{\text{maturity}}(c, n)$ the maturity time of the nonce in question. Other variables can be described by `bondClass[class].nonceInfoDiscription[];` and defined by code.

e.g.

`["1615584000","1616584000",(uint256)...]`

this example gives the starting time of the bond nonce of the bond class, which is 1615584000; the maturity time, 1616584000...

3.2.3 Bond transfer

"transferBond()" allows the transfer of any number of bond types from an address to another.
The""_from" argument is the address of the holder whose balance is about to decrees.
The ""_to" argument is the address of the recipient whose balance is about to increase.
The "class" is the list of classes of bonds or derivatives, the first bond class created will be 0, and so on.
The "nonce" is the list of nonce of the given bond class. This param is for distinctions of the issuing conditions of the bond.
The "_amount" is the list of amounts of the bond that will be transferred from "_from"address to "_to" address.

$$\begin{cases} B\left(a_1, c, n\right) := B\left(a_1, c, n\right) - \_amount, \\ B\left(a_2, c, n\right) := B\left(a_2, c, n\right) + \_amount, \\ \qquad\qquad if \_amount \leq B\left(a_1, c, n\right) \end{cases}$$

$$\begin{cases} B\left(a_1, c, n\right) := B\left(a_1, c, n\right), \\ B\left(a_2, c, n\right) := B\left(a_2, c, n\right), \\ \quad if \_amount > B\left(a_1, c, n\right) \end{cases}$$

```
function transferBond(address _from, address _to, uint256[]
calldata class, uint256[] calldata nonce, uint256[] calldata
_amount) external returns(bool);
```

e.g.
`transferBond(0x2d03B6C79B75eE7aB35298878D05fe36DC1fE8Ef,`
`0x82a55a613429Aeb3D01fbE6841bE1AcA4fFD5b2B,`
`[1,2,4], [42,61,25],[500000000,60000000,150000000]);`
This example shows the transfer from "_from" address, to "_to" address, "500000000" of bond class "1" nonce "42", "60000000" of bond class "2" nonce "61", "150000000" of bond class "3" nonce "25". Returns a bool. "true" if passed.
*NOTE: This transfer is atomic i.e., If one transfer in the transaction fails, the function will revert all the previous passed transfers within the transaction.

3.3 Automated Pair Maker (APM)

Method that automatically build pairs from a single token. By using this method, LP providers need only one token to add liquidity. Compared with the current AMM pair model, APM has many architectural changes, some of which are necessitated by the DeBond redemption mechanism.

3.3.1 Adding liquidity

When a liquidity provider is adding liquidity to a pair $p(t_1, t_2)$ with only token t_1, APM send first Δl_1 to the pair $p(t_1, t_2)$; then uses CDP as an index to calculate the amount of liquidity $l_{\text{determined}}(t_1, t_2; l_1)$ needed for t_2. By minting t_2 needed, APM can add both the t_1 and t_2 to the pair $p(t_1, t_2)$. Adding liquidity:

$$\begin{cases} b_{erc20}(a) := b_{erc20}(a) - \Delta l_1 \\ l_1 := l_1 + \Delta l_1, \\ l_2 := l_2 + l_{\text{determined}}(t_1, t_2; l_1) \end{cases}$$

DBIT and DGOV bond are typical European bond. The interest is directly added to the face value of the bond issued.When using `buyBondFromToken()` method:

$$B(a, i, n) := B(a, i, n) + l_{\text{determined}}(1 + \iota)$$

When using `stakingForBondFromToken()` method:

$$\begin{cases} B(a, 2, n) := B(a, 2, n) + l_{\text{determined}}(\iota), \\ B(a, 1, n) := B(a, 1, n) + \Delta l_1 \end{cases}$$

Then use `update()` method of the pair contract to update the k of the AMM pair contract, and `mint()` method to reclaim the LP token.

$$k(l1, l2) := k_{new}(l1, l2)$$

APM can then use the LP token as a pledge to issue ERC-3475 bond to the liquidity provider. By applying this method, DeBond is fragmenting the existing LP into sub-categories. Each sub-categories possessed a non-fungible data-set. Which means that each sub-categories is a loan contract that both parties agree on.

How?

Is'nt it determined by the AMM formula xy=k?

minting? not swaping?

do not see the link between what is explained in intro (token, governance plateform) and european bond. Link?

what is iota? l_determined is called two times but does not take the same structure of arguments, need to be better defined

3.3.2 Redemption

During the redemption of the ERC-3475 bond, the interest will be paid with the native token t_2. He can in his turn, choose to keep the t_2 or swap it into any token existing on AMM. The liquidity pool of token $p\left(t_1, t_2\right)$ will be served as the liquidity provider is trading t_2 on AMM.
Interest payment:

$$\begin{cases} B\left(a,i,n\right) := B\left(a,i,n\right) - \Delta B, \\ k(l1,l2) := k_{new}(l1,l2), \\ b_{erc20}(a) := b_{erc20}(a) + \Delta b_{erc20} \end{cases}$$

Return of the principal in the case of using `stakingForBondFromToken()` method:

$$\begin{cases} B\left(a,i,n\right) := B\left(a,i,n\right) - \Delta B, \\ l_2 := l_2 - P(l_1 - \Delta b_{erc20}, l_2)\Delta b_{erc20}, \\ l_1 := l_1 - \Delta b_{erc20}, \\ k(l1,l2) := k_{new}(l_1,l_2), \\ b_{erc20}(a) := b_{erc20}(a) + \Delta b_{erc20} \end{cases}$$

This system will work well when the LP is big enough. APM introduces multiple sub-pools for each APM paired LP. All sub-pools are created and controlled by the APM contract. APM contract also keeps all the ERC-20 LP token generated from adding liquidity. With those LP tokens, APM can transfer the liquidity between the sub-pairs and the main pair, when the APM method is being called.

4 APM Toufic version

Our apm is constitued by only one address. This means that whatever pool you are adding liquidity to, you always send tokens to the same address. So, to track the price, we cannot use the token balance in this contract, because tokens can belong to several pools. Instead, we use virtual pools : given two tokens address, using mappings, we are able to track the token reserve for each pair. The way we track the reserves is inspired from uniswap mechanism, which will be explained in section 4.1 .

Our APM will handle 3 main functions :

1. adding liquidity to a pool. Although it is true that a user only adds

one token, we mint a equivalent amount of another token to be paired with, so it is the same as classic AMM.

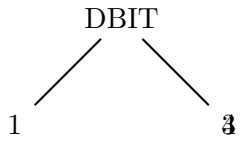
2. for swapping, it is the same as classic AMM.

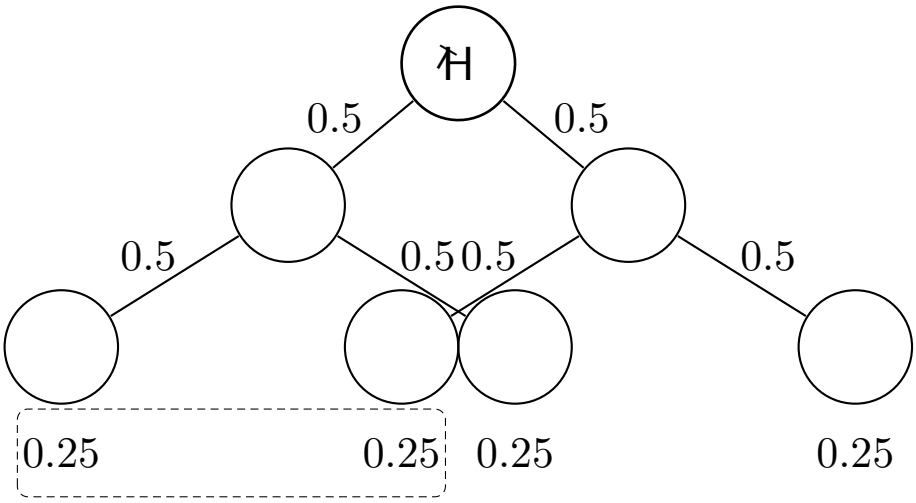
3. For redeeming liquidity, the user will only redeem one token. Instead of taking this token from a particular pool, we take remove token from the total pool, which is a special mecanism that needs to be explained

4.1 How we use Uniswap mechanism

Our price mechanism in the APM is based on Uniswap LP token mechanism. They have a pool of two tokens, (T_1, T_2). Every User who adds liquidity to the pool gets LP token. Once someone else adds liquidity, we give that person LP token, but we do not change the amount of LP token that previous users hold. To know how much liquidity each user holds, we divide their number of LP token by the total amount of LP token issued (which increase every time someone adds liquidity), which gives the ratio of assets they hold.

In our model, the pool (T_1, T_2) is the analog of the balance T_1 in the apm, and users are virtual pools. We want to know how much T_1 each virtual pool has. To do so, we consider that each time T_1 is added trough a pool, it is as if a user adds liquidity in uniswap. But instead of giving him LP token, we will give him virtual Lp token (VLP) :VLP are not really tokens because virtual pools are not users, VLP are an amount stored in a mapping, corresponding to the amount of LP token a user would get if he adds liquidity to a pool.





4.2 definitions

We consider our DBIT token, which can be paired with a set of tokens (T_i) for $i \in n$, $n \in \mathbb{N}$. We have n pools $(T_i, DBIT)$. Let:

- $r_{i,DBIT}$ be the reserve of Token i in the $(T_i, DBIT)$ pool.
- $r_{DBIT,i}$ the reserve of DBIT in the $(R_i, DBIT)$ pool.
- $R = \sum_{i=1}^{n} r_{DBIT,i}$ the total reserve of DBIT in every $(T_i, DBIT)$ pool.
- vlp_i the amount of VLP token for DBIT in pool i.
- $totalVLP = \sum_{i=1}^{n} vlp_i$ the total amount of LP token for DBIT

4.3 adding liquidity

Let's suppose that someone adds liquidity to pool one : $r_{DBIT,i} := r_{DBIT,i} + \Delta_{DBIT}$ and $r_{i,DBIT} := r_{i,DBIT} + \Delta_i$. We now have $R := R + \Delta_{DBIT}$, and $l_i = l_i + \frac{L}{R}\Delta_{DBIT}$ and NOT $l_i = l_i + \Delta_{DBIT}$. (explain why not R/L (it's L/R) with examples).

4.4 removing liquidity

Let's suppose that somone removes DBIT from the whole pool : $R := R - \Delta_{DBIT}$, l DOES NOT CHANGE (and so L), so we have :

$$r_{DBIT,i} = \frac{l_i}{L}(R - \Delta_{DBIT})$$

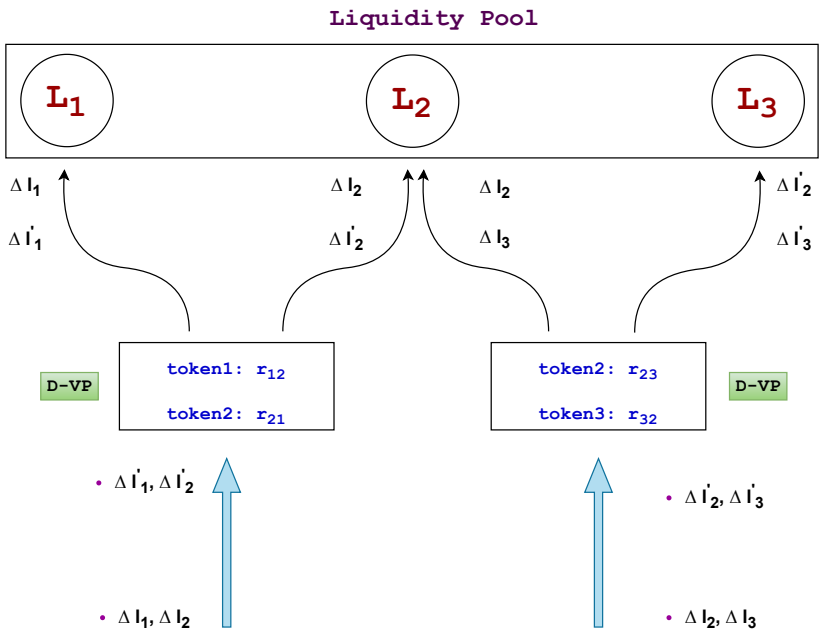


Figure 1: Debond virtual pools

4.5 Swap

Let's say we want to sell DBIT to buy T_1. (update R) We have $r_{i,DBIT} * r_{DBIT,i} = k$, with k constant. If i have to sell Δ_{DBIT}, I will receive back the following amount of T_1 :

$$r_{i,DBIT} - \frac{k}{r_{DBIT,i} + \Delta_{DBIT}}$$

We have : $r_{DBIT,i} = r_{DBIT,i} + \Delta_{DBIT}$, $R = R + \Delta_{DBIT}$

So :

$$\frac{l_i}{L} = \frac{r_{DBIT,i} + \Delta_{DBIT}}{R + \Delta_{DBIT}} = c$$

Let's define $L_{-i} = \sum_{j \neq i} l_j$, which we suppose is constant.

So :

$$\frac{l_i}{L} = \frac{l_i}{l_i + L_{-i}}$$

And :

$$l_i = c(l_i + L_{-i})$$

Finally :

$$l_i = \frac{cL_{-i}}{1 - c}$$

5 APPLICATIONS

We can regard DeBond as a set of applications built on top of Ethereum block-chain. The first category is the debt accounting system, providing users with more powerful ways of managing their staking, loan and interact with smart contracts using their pledged assets. This includes the securitisation of LP token, financial derivatives, bonds etc.. The second category is the P2P landing application, where a loan is issued based on the specified terms and pledged assets involved. There is also an off-chain possibility open to the transformation of an off-chain debt or pledge to on-chained cryptographic data. Finally, there are applications such as hedging contracts or Decapital, a smart contract based hedging or venture funds. With this application, the pledged asset involved, will be managed by by Decapital. It will be used to fund on-chian and off-chain projects, hedging funds and other supercilious investment.

DeBond systems have many applications ranging from financial derivatives to fixed-rated bond products. Individual bond represents the debt of a digital currency. DeBond products are easy to implement. Using the same ERC-3475 interface, anyone can create and build his own bond product. Those bonds or derivatives are, in theory, the database structured object, intelligible by other smart contracts. The array of integers is the basic building module of an ERC-3475 bond. Every bond contract has the method listed below:

```
function balanceOf (address account, uint256 class, uint256 nonce)
external view returns (uint256);
```

```
function batchBalanceOf (address account, uint256 class) external
view returns(uint256[] memory);

function totalBatchBalanceOf (address account, uint256 class) external
view returns(uint256);

function getBondSymbol (uint256 class) view external returns (string
memory);

function getBondClassInfo (uint256 class) external view returns
(string memory BondSymbol, uint256[] memory info);

function getBondNonceInfo (uint256 class, uint256 nonce) external
view returns (uint256[] memory info);

function getBondProgress (uint256 class, uint256 nonce) external
view returns (uint256[2] memory);

function getBatchBondProgress (uint256 class, uint256 nonce) external
view returns (uint256[] memory, uint256[] memory);
```

With the methods listed above, a front-end interface or DeBond wallet can read the balances, class, nonce, progress, redemption time and other info from a bond contract.

5.1 DeBond Wallet

DeBond wallet allows users to manage accounts and their ERC-3475 assets in a variety of ways. Built on top of a hot-wallet such as Metamask[8], DeBond wallet allows users to read, buy, redeem, exchange and transfer any ERC-3475 standard asset. DeBond wallet is a web app which was designed specially for ERC-3475 assets. It allows any user to have an easier access to Web3 infrastructure, interact with the Ethereum ABI, and communicate with the EVM smart contract. DeBond wallet is user friendly and compatible with all web3 browsers (like Chrome with Metamask plug-in).

DeBond wallet supports any ERC-3475 bond or derivatives. Those financial products can be displayed, traded and read on a DeBond wallet without any development needed. This feature of the "Bond system" displays the transition function described further below in this document.

```
function transferBond (
```

```
address _from,
address _to,
uint256[] calldata class,
uint256[] calldata nonce,
uint256[] calldata _amount
) external returns(bool);
```

With this EVM interface, DeBond wallet will use web3 to interact with the smart contract. DeBond-based DEFI project act as a derivative issuer. They can potentially include other feature and method. By doing so, the market will explore all the possibilities until it finds several most well-designed derivatives. DeBond wallet provides a derivative infrastructure for this potential market.

Existing liquidity pool can be adapted to support ERC-3475. The implementation of ERC-3475 bond to replace the existing ERC-20 LP token, will be gradually taking place. The DeBond wallet will be a practical way to display all your staking. In the form of bonds or derivatives, the infrastructure will be adapted to the needs of other DEFI project developers. DeBond wallet serves as a way of displaying all ERC-3475 compatible DEFI project's LP data. Using ERC-3475 interface, a DEFI project can simply use DeBond wallet to interact with their smart contract, without any adaptation or front-end developments needed.

5.2 DeBond DEX

A open secondary market is the most easy way to exchange DeBond. Common AMM method [9] can't be adapted to ERC-3475 bonds. The derivatives created with ERC-3475 need the implantation of a specially designed auction system. The main challenge in building such secondary bond market is that the majority of the bonds need to be bound together before putting into an open order. The DeBond DEX uses dutch auction method. Buyers will compete with other parties to find the real market price of a derivatives.

5.2.1 DeBond Auction Object

```
struct AUCTION {
bool auctionStatus;
address seller;
uint256 startingPrice;
uint256 endingPrice;
```

```
uint256 auctionTimestamp;
uint256 auctionDuration;
address bondAddress;
uint256[] bondClass;
uint256[] bondNonce;
uint256[] bondAmount
}

function addAuction (AUCTION calldata _auction) external returns
(bool);
```

The code cited above shows an auction object. It's a package of any given number of bond object bounded together, as long as they are from the same DeBond contract. For example, a user can bound several high risk sub-prime bonds with some stable prime bond. Using this method, the user is creating a new derivative product.

5.2.2 Nash equilibrium

Using dutch auction method, DeBond DEX implements Nash equilibrium[10] to allow the market to negotiate the right price of the derivative. The DeBond DEX is built for any ERC-3475 bond, and uses DBIT as the only settlement currency. Because of the index nature of DBIT, the price of DBIT to USD will be less volatile then other ERC-20 tokens. All the on-going orders and the pledged asset behind those bonds will assure that the DBIT will reflect the weighted average performance of de DeBond and the DEFI market in question.

let us consider that $P_{equilibrium}(i)$ here be a set of all possible price ranges for bidder i. Which $i = 1, 2, 3 \dots N$ from the first bidder to the last one. Let us suppose that each bidder knows the set of possible price ranges of each other player. With $p^* = (p_i^*, p_{-i}^*)$ be a set of possible price ranges for each bidder; knowing that the $n-1$ sets of prices mean that all other bidder except i. Let $u_i(s_i, s_{-i}^*)$ be the auctioneer's payoff as a function of knowing each other player's set of prices. The set of price p^* is a Nash equilibrium if:

$$u_i(p_i^*, p_{-i}^*) \geq u_i(p_i, p_{-i}^*) \text{ of each } p_i \in P_{equilibrium}(i)$$

As a set of DeBond objects is being auctioned, the auctioneer sets a starting price p_{start} and a ending price p_{end}, remained that $p_{start} > p_{end}$

When an auction starts,

$$T_{start} := T_l$$

Auctioneer set the duration ΔT of the auction at the start of an auction. We know

$$T_{end} := T_{start} + \Delta T$$

Therefor the change of price in time can be expressed in a linger function:

$$c = \frac{\Delta P}{\Delta T}$$

$$\begin{cases} & T \in [T_{start}, T_{end}], \\ P_T = f'(T) = P_{start} - (P_{start} - P_{end}) \dfrac{T - T_{start}}{\Delta T} \end{cases}$$

With this in mind, the real closing price of a set of DeBond is:

$$P_{real} = \begin{cases} P_T, if P_T \leq max P_{equilibrium}(i) \\ max P_{equilibrium}(i), if P_T > max P_{equilibrium}(i) \end{cases}$$

5.2.3 Price range and interest velocity

The issuer of derivatives promises to fulfil their duties written on the contract, when the conditions are met. This mechanism allows any one to find a solution to their liquidity or risk management problems.

Provided of course, that the market dose go to the direction that we expected. In practice, however, is the other way. In this scenario their derivatives will shortly lose most of the face value because the market is losing confidence. But because of the pledged asset locked in the bank contract, the credibility of their bond will touch a lower limit, based on the value of the pledged asset. This means that during the recession period, velocity of money decreases, less people will trust in their abilities of repayment. This will affect the value of derivatives on the secondary market.

The price of the underlying derivatives (typically a bond) follows a geometric Brownian motion. That is:

$$\frac{dS}{S} = \mu \, dt + \sigma \, dW$$

where W is a stochastic variable (Brownian motion). Note that W, and consequently its infinitesimal increment dW, represents the only source of uncertainty in the price history of the derivative. Intuitively, $W(t)$ is a process that "wiggles up and down" in such a random way that its expected change over any time interval is 0. (In addition, its variance over time T is equal to T; see Wiener process § Basic properties); a good discrete analogue for W is a simple random walk. Thus the above equation states that the

infinitesimal rate of return on the price of the bond has an expected value of μdt and a variance of $\sigma^2 dt$.

Using a binomial lattice, one calibrates the model parameters to fit both the current term structure of interest rates (yield curve), and the volatility structure for interest rate caps (usually as implied by the Black-76-prices for each component caplet); see aside. Using the calibrated lattice one can then value a variety of more complex interest-rate sensitive securities and interest rate derivatives.

Although initially developed for a lattice-based environment, the model has been shown to imply the following continuous stochastic differential equation:

$$dr(t) = [\theta(t) - \alpha(t)r(t)]\ dt + \sigma(t)\, dW(t).$$

r = the instantaneous short rate at time t
θ_t = value of the underlying asset at option expiry
σ_t = instant short rate volatility
W_t = a standard Brownian motion under a risk-neutral probability measure;
dW_t= its differential.
θ and α are both time-dependent — the extended Vasicek model.

For constant (time independent) short rate volatility, σ, the model is:

$$d\ln(r) = \theta_t\, dt + \sigma\, dW_t$$

For the rest of this article we assume only θ has t-dependence. Neglecting the stochastic term for a moment, notice that for $\alpha > 0$ -th change in r is negative if r is currently "large" (greater than $\theta(t)/\alpha$ and positive if the current value is small. That is, the stochastic process is a mean-reverting Ornstein–Uhlenbeck process.

θ is calculated from the initial yield curve describing the current term structure of interest rates. Typically α is left as a user input (for example it may be estimated from historical data). σ is determined via calibration to a set of caplets and swaptions readily tradeable in the market.

When α , θ , and σ are constant, Itô's lemma can be used to prove that

$$r(t) = e^{-\alpha t} r(0) + \frac{\theta}{\alpha}\left(1 - e^{-\alpha t}\right) + \sigma e^{-\alpha t} \int_0^t e^{\alpha u}\, dW(u),$$

which has distribution:

$$r(t) \sim \mathcal{N}\left(e^{-\alpha t} r(0) + \frac{\theta}{\alpha}\left(1 - e^{-\alpha t}\right), \frac{\sigma^2}{2\alpha}\left(1 - e^{-2\alpha t}\right)\right),$$

Where $\mathcal{N}(\mu, \sigma^2)$ is the normal distribution with mean μ and variance σ^2. When $\theta(t)$ is time-dependent,

$$r(t) = e^{-\alpha t} r(0) + \int_0^t e^{\alpha(s-t)} \theta(s) ds + \sigma e^{-\alpha t} \int_0^t e^{\alpha u} \, dW(u),$$

which has distribution

$$r(t) \sim \mathcal{N}\left(e^{-\alpha t} r(0) + \int_0^t e^{\alpha(s-t)} \theta(s) ds, \frac{\sigma^2}{2\alpha} \left(1 - e^{-2\alpha t}\right) \right)$$

It turns out that the time-S value of the T-maturity discount bond has distribution:

$$P(S,T) = A(S,T) \exp(-B(S,T) r(S))$$

, where

$$B(S,T) = \frac{1 - \exp(-\alpha(T-S))}{\alpha},$$

$$A(S,T) = \frac{P(0,T)}{P(0,S)} \exp\left(-B(S,T) \frac{\partial \log(P(0,S))}{\partial S} - \frac{\sigma^2 (\exp(-\alpha T) - \exp(-\alpha S))^2 (\exp(2\alpha S) - 1)}{4\alpha^3} \right).$$

Note that their terminal distribution for $P(S,T)$ is distributed log-normally.

By selecting as numeraire the time-S bond (which corresponds to switching to the S-forward measure), we have from the fundamental theorem of arbitrage-free pricing, the value at time t of a derivative which has payoff at time S.

$$V(t) = P(t,S) \mathbb{E}_S[V(S) \mid \mathcal{F}(t)].$$

Here, $\mathbb{E}_S$ is the expectation taken with respect to the forward measure. Moreover, standard arbitrage arguments show that the time T forward price $F_V(t,T)$ for a payoff at time T given by $V(T)$must satisfy $F_V(t,T) = V(t)/P(t,T)$, thus:

$$F_V(t,T) = \mathbb{E}_T[V(T) \mid \mathcal{F}(t)].$$

Thus it is possible to value many derivatives V dependent solely on a single bond $P(S,T)$ analytically when working in the Hull–White model. For example, in the case of a bond put

$$V(S) = (K - P(S,T))^+.$$

Because $P(S,T)$ is lognormally distributed, the general calculation used for the Black–Scholes model shows that

$$E_S[(K - P(S,T))^+] = KN(-d_2) - F(t,S,T)N(-d_1),$$

where

$$d_1 = \frac{\log(F/K) + \sigma_P^2 S/2}{\sigma_P \sqrt{S}}$$

and

$$d_2 = d_1 - \sigma_P \sqrt{S}.$$

Thus today's value (with the$P(0,S$ multiplied back in and$t$ set to 0) is:

$$P(0,S)KN(-d_2) - P(0,T)N(-d_1).$$

Here σ_P is the standard deviation (relative volatility) of the log-normal distribution for $P(S,T)$. A fairly substantial amount of algebra shows that it is related to the original parameters
via

$$\sqrt{S}\sigma_P = \frac{\sigma}{\alpha}(1 - \exp(-\alpha(T-S)))\sqrt{\frac{1-\exp(-2\alpha S)}{2\alpha}}.$$

Note that this expectation was done in the S-bond measure, whereas we did not specify a measure at all for the original Hull–White process. This does not matter — the volatility is all that matters and is measure-independent.

Because interest rate caps/floors are equivalent to bond puts and calls respectively, the above analysis shows that caps and floors can be priced analytically in the Hull–White model. Jamshidian's trick applies to Hull–White (as today's value of a swaption in the Hull–White model is a monotonic function of today's short rate). Thus knowing how to price caps is also sufficient for pricing swaptions. In the even that the underlying is a compounded backward-looking rate rather than a (forward-looking) LIBOR term rate, Turfus (2020) shows how this formula can be straightforwardly modified to take into account the additional convexity.

5.3 DBIT and DGOV Bond

Both DBIT and DGOV bonds are zero coupon bond. In which the face value is repaid at the time of maturity. That definition assumes a positive time value of money. It does not make periodic interest payments or have so-called coupon.

There are buying method and the staking method use APM and CDP features. Saying that the liquidity pool of DBIT can be added by using only

one type of token. The APM system will use the CDP function to automatically mint the needed amount of DBIT to be used to pair up with the staking asset. The difference between buying and staking is that the buying method actually mint new DBIT according to the value of the pledged asset. But the staking method uses the function to transfer the according amount of DBIT from the DBIT LP to the investor.

At the redemption of the DBIT bond, investors get the amount of DBIT bonds redeemed.The graph below shows the common procedure of buying a DBIT bond. The interest and principal will be paid off by DBIT.

5.3.1 Market types

We define here, the interest payment of the redemption of DBIT bonds is the first market. As the DBIT earned from selling bonds or borrowing loans is the second market.

- **First market:** Buying DeBond. Is the method of using a digital asset to buy the bond. This means that the principle will only be paid back in the form of settlement token. Where staking for DeBond, Is the method of using a digital asset as a staking collateral to get the bond, In this way, the staking asset will be able to be redeemed and returned to the investor once the bond is redeemed. The price of DeBond in the first market is defined by CDP method introduced earlier in this article.
- **Secondary market:** On a secondary market, the price of the derivatives, typically bond is heavily influenced by the Pull to Par effect. In which the price of a bond converges to face value as time passes. At maturity the price of a debt instrument in good standing should equal its par (or face value). It results from the difference between market interest rate and the nominal yield on the bond.

5.3.2 Classes

The face interest rate can be modified by the governance contract. In terms of the interest calculation method, there are 2 type and in total 10 classes of DeBond:

- **Floating rate bond:** It's a high risk subprime bond. Though the face interest rate is higher than the fixed rate bond, the exact mature date

varies according to the liquidity flow of the pledged asset. Meaning that in terms of the annual interest rate, the IR is variable. During a crisis, the floating rate bond holder may suffer from impairment losses.

- **Fixed rate bond:** It's a low risk bond. Investing in such bonds will guarantee the face interest rate and safeguard the pledged asset from impairment losses. The face interest is determined on the date of the purchase of the bond. An algorithm will consider the ratio of floating rate, fixed rate bonds and several other factors, and gives a deterministic face IR. Once the bond is issued, the face interest rate will never be changed. Fixed rate bond holders have also a higher priority of creditors. During a crisis, their pledged asset will be safeguarded and returned properly.

Each class `int256` constant. In the case of DBIT and DGOV bond, there are minimum 6 digits from the right to left.

- The fist three digits represent the maturity period. Defined by:

$$T_{\text{maturity}}(c, n) - T_{\text{start}}(c, n)$$

- The forth digit represents the type of the bond. By 0=`floating_rate`, by 1=`fix_rate`.
- The fifth digit represents the issuing method. By 0=`buyBondFromToken()`, by 1=`stakingForBondFromToken()`.
- The sixth digit and after represents the i-th settlement token t_i.

Classes

Type & Maturity	Class(c)	Nonce (N)	Nominal IR	Preferred$*$
Floating rate 0.5 year	$t_i, m, 0, 000$	(1,5)	ι_{float}	2nd
Fix rate 0.5 year	$t_i, m, 1, 000$	(1,5)	ι_{fix}	1st

$*$Preferred Creditor is the order in which the creditor has the priority of the claim, when the debtor don't have enough fund to repay the all the debt.

5.3.3 Nonce

In DBIT and DGOV bond, nonce marks the difference in maturity date. For example, the nonce `15` represents the maturity date in 1st December 2021,

nonce `16` represents the maturity date in 2nd December 2021. The interval of two bond is defined by T_{epoch}. By a sequence of nonce $\langle n_i \rangle_{i=1}^{N}$ = mean that the investor will receive bonds of N different maturity date.

When purchasing the bond from first market, investor can choice either one maturity date or multiple. In the case of multiple maturity dates, the total nominal interest rate of an investment is the sum of the nominal interest rate of every bond nonce. In the case of one maturity date, the protocol will add together the nominal interest rate of each possible nonce. This mechanism is described in detail in the section 4.4.4 Interest rate of this article.

If the investor choice multiple maturity dates, he will receive bonds with N nonce $\langle n_i \rangle_{i=1}^{N} = \langle n_1, n_2, n_3, \cdots, n_N \rangle$. The number of nonce N is based on the maturity time. We arbitrarily fixed the multiple nonce of 0.5 year bond to 5. For the rest of maturity time is quadratic of the times of maturity time in year Y over 0.5. N must be an integer $\mathbb{Z}$.

$$\begin{cases} \langle n_i \rangle_{i=1}^{N} = \langle n_1, n_2, n_3, \cdots, a_N \rangle, \\ N = 4 + \sqrt{\dfrac{Y}{0.5}}, \\ N := \lceil N \rceil = \min\{x \in \mathbb{Z} \mid N \leq x\} \end{cases}$$

The sequence is based on several factor listed below:
N = the length of sequence, described above.
n_l = the nonce of time now T_l, defined by `time.now`.
$\Delta n_{\text{maturity}}$ = difference in length from the issuing time to the maturity date.
F_n= the n-th number of a Fibonacci sequence.
x= a variable measuring the length of each Fibonacci unit in $\Delta n_{\text{maturity}}$

$$\langle n_i \rangle_{i=1}^{N} = \begin{cases} x = \dfrac{\Delta n_{\text{maturity}}}{F_{N+1}}, \\ \langle (n_l + F_2 x), (n_1 + F_3 x), (n_l + F_4 x), \cdots, (n_l + F_{i+1} x) \rangle \end{cases}$$

5.3.4 Interest rate

There are several factors involved in the calculation of the real interest rate ι_{real}.

$\iota_T = \iota_{weighted}(T)$ the instantaneous weighted nominal short rate at time T
$p_{determined}(T - \Delta T)$ = the CDP price when the bond is issued
p_T = value (in USD) of the settlement token received at maturity date T
σ_T = instant short rate volatility

W_T = a standard Brownian motion under a risk-neutral probability measure;
dW_T= its differential.

First we calculate the instantaneous nominal short rate ι at time T.This nominal interest rate is calculated from benchmark IR and the ratio of floating rate bond to fixed rate bond of the same settlement token. Then we consider the difference in the price of settlement token over difference in time dT. Finally we add the random walk factor W_T.

$$d\iota_{real}(T) = [\frac{p_T}{p_{determined}(T - \Delta T)}(1 + r_T) - 1]dT + \sigma_T \, dW_T$$

The benchmark interest rate is the interest defined by `IR.benchmarkIR` and can be updated by `governanceContract.updateBenchmarkIR()`. It's the basic interest rate from which we can calculate the floating rate interest ι_{float} and fixed rate interest ι_{fix}.

$$2\iota_{benchmark} = \iota_{float} + \iota_{fix}$$

However this changes is not entirely proportional:

$$\begin{cases} \dfrac{\iota_{float}}{\iota_{fix}} = 1, \\ when \dfrac{B_{float}(t)}{B_{fix}(t)} = 2 \end{cases}$$

For $0 < c < 1$, define:

$$\varphi_c\left(x\right) = \frac{2^{-1/(1-c)x}}{2^{-1/(1-c)x} + 2^{-1/c(1-x)}}$$

This$\varphi_c : [0, 1] \to [0, 1]$ is bijective, sigmoid (sigma-function like) and infinitely differentiable. Particularly, it satisfies the following:

$$\begin{cases} \varphi_c(0) = 0, \\ \varphi_c(c) = \dfrac{1}{2}, \\ \varphi_c(1) = 1 \end{cases}$$

The interest rate $\iota_{float}(t)$ of an floating rate bond is based on benchmark IR and the ratio of floating rate bond to fixed rate bond of the same settlement token t.

$$\iota_{float}(t) = 2\iota_{benchmark} \cdot \varphi_{1/3}(\frac{B_{fix}(t)}{B_{float}(t) + B_{fix}(t)})$$

The interest rate $\iota_{fix}(t)$ of an fix rate bond is based on benchmark IR and the ratio of floating rate bond to fixed rate bond of the same settlement token t.

$$\iota_{fix}(t) = 2\iota_{benchmark} - \iota_{float}(t)$$

Bond investor can choice either one maturity date or multiple.
In the case of one maturity date $N = 1$, the protocol will calculate automatically the compound interest rate $\iota_{compound}$ of each bond nonce ι_i

$$\iota_{compound} = (\prod_{i=1}^{n}(1 + \iota_i)) - 1$$

In the case of multiple maturity dates $N > 1$, the total nominal interest rate of an investment is the sum of the nominal interest rate of every bond nonce ι_i:

$$\iota_{\text{total}} = \sum_{i=0}^{n} \iota_i$$

The nominal interest rate of a 0.5 year bond:

$$\iota_{nominal} \begin{cases} (\prod_{i=1}^{n}(1 + \iota_i)) - 1, if N > 1, \\ \sum_{i=0}^{n} \iota_i, if N = 1 \end{cases}$$

The nominal interest rate $\iota_{nominal}$ of maturity period $\Delta T_{maturity}$:

$$\begin{cases} \Delta T_{maturity} = T_{\text{maturity}} - T_{\text{start}}, \\ \iota_{nominal}(\Delta T_{maturity}) = (1 + \iota_{nominal})^{\frac{\Delta T_{maturity}}{0.5}} - 1 \end{cases}$$

Finally we can calculate the weighted nominal interest rate $\iota_{weighted}$:

$$\begin{cases} \iota_{weighted} = \iota_{nominal}(1 - 1/2^{\kappa}), \\ \kappa = \lceil \frac{\Gamma}{\Lambda} \rceil, \\ \Gamma := 1, \\ \Lambda = 1 - \frac{\mu_{total}}{\mu_{month}} \end{cases}$$

with:
μ_{month}= the average total balances of all the nonce in last month $\frac{T_{month}}{\Delta T_{epoch}}$.
μ_{total}= the average total balances of all the nonce from the fist nonce 0 to the last n_l.

5.3.5 Bonds maturity

The fixed rate bond has only one redemption condition, the maturity time $T_{maturity}$ on the bond. After this mandatory redemption date, the bond will be redeemed without any other questions. Fixed rate bonds are matured once:

$$T_{maturity}(n) \leq T_l$$

Since floating rate bonds have an estimated redemption date $T_{estimate}(c, n)$, which is not fixed. Thus the dT factor is not fixed. Although the IR of fixed rate bond is constant after purchase, the IR may vary according to the different ratio between the fixed rate bond and the Floating rate bond.

$$\begin{cases} B_{\text{sum}}(n)(1 + \iota_{benchmark}) \leq B_{\text{sum}}(n_l), \\ and, T_{maturity}(n) \leq T_l \end{cases}$$

$T_{estimate}(n)$ is the estimated redemption time based on the average liquidity flow over last month μ_{month}, and the additional liquidity needed.

$$T_{maturity}(n) + \frac{B_{\text{sum}}(n)(1 + \iota_{benchmark}) - B_{\text{sum}}(n_l)}{\mu_{month}} T_{epoch}$$

5.3.6 Preferred Creditor

When the redemption crises appears, the collateralised liquidity is not enough to fulfil all the requirement of liquidation. In that case, a mechanism called Preferred Creditor will take place. The bond with longer redemption period will have a priority of creditors.This means if the pledged asset is no longer enough to pay all the debt during a period, the pledged asset will be firstly used to repay the Preferred Creditor.

Bondholder can choice to liquidate his or her bond before the redemption date. But by doing so, the bondholder will suffer from different degree of losses based on their preferred creditor order.

A crisis by definition is when:

$$B_{sum}(n)(1 + \iota_{benchmark}) - B_{sum}(n_l) > 0$$

There are 10 different classes of DeBond (see section 4.4.2 Classes for more information). In this scenario, first we calculate the deficit D of the bond

product,

$$D = B_{sum}(n)(1 + \iota_{benchmark}) - B_{sum}(n_l)$$

Then we attribute the deficit to each bond class from the less preferred to the most preferred. The preference order is defined by:

$$O_{preferred} = \langle o_i \rangle_{i=1}^{C} = \langle 1,\, 2,\, 3,\, \cdots,\, C \rangle$$

According to this order we can calculate the rate of how much remains $R_{rate}(o_i)$.

$$\begin{cases} \sum_{k=1}^{C-1} k = 45, if C = 10, \\ R_{rate}(o_1) = 1 - 0, \\ R_{rate}(o_2) = 1 - 1(\frac{1}{45}), \\ R_{rate}(o_3) = 1 - 2(\frac{1}{45}), \\ R_{rate}(o_4) = 1 - 3(\frac{1}{45}), \\ ... \\ R_{rate}(o_C) = 1 - (C-1)(\frac{1}{45}) \end{cases}$$

Finally we can calculate the remaining value of the bonds R_{o_i} after the

bankruptcy .

$$
\begin{cases}
\frac{R_{o_1} = R_{rate}(o_1)B_{total}(o_1)}{1+\iota_{weighted}(o_1)} \\
\frac{R_{o_2} = R_{rate}(o_2)[\frac{\sum_{i=2}^{C} B_{total}(o_i)}{B_{sum}(n_l)}]B_{total}(o_2)}{1+\iota_{weighted}(o_2)} \\
\frac{R_{o_3} = R_{rate}(o_3)[\frac{\sum_{i=3}^{C} B_{total}(o_i)}{B_{sum}(n_l)}]B_{total}(o_3)}{1+\iota_{weighted}(o_3)} \\
\frac{R_{o_4} = R_{rate}(o_4)[\frac{\sum_{i=4}^{C} B_{total}(o_i)}{B_{sum}(n_l)}]B_{total}(o_4)}{1+\iota_{weighted}(o_4)} \\
... \\
\frac{R_{o_C} = R_{rate}(o_C)[\frac{\sum_{i=C}^{C} B_{total}(o_i)}{B_{sum}(n_l)}]B_{total}(o_C)}{1+\iota_{weighted}(o_C)}
\end{cases}
$$

5.4 Securitised loan

The creditor of a securitized loan is no longer a smart contract, but a bond purchaser. The debtor has to first pledge his/her digital assets (ERC-20, ERC-721, ERC-3475 etc.) to a smart contract, which then issues the loan bond representing the securitised loan. Based on the creditor's settings of the amount of the loan, repayment method, interest rate, due date, etc. These bonds are going to be placed on the secondary market and traded using the Dutch auction method. If a prospective debtor believes that the bond is set up in a rational manner, he/she will purchase the bond. Upon the completion of the transaction, the creditor receives the loan and the debtor obtains the bond of the collateralised assets. If the creditor repays the loan before the repayment date, he or she can reclaim his or her pledged asset. If not, the bond will be redeemable after the marked repayment date by the bondholder. Until then, the debtor can split or pack the bond into packages and sell them on the secondary market (Bond DEX). The creditor agreed to repay the principal and interests by the specified date in accordance with the established rules. After all the principal and interests are repaid to the creditor(s), the relevant bond of the pledged asset will be invalidated.

5.4.1 Loan contract creation

The method below shows how a loan order is created. By using a custom structure called ERC20LOAN, a user can create their own loan agreement. The object of a loan is the custom structure demonstrated below. A user can create such a structured object by inputting parameters into the EVM smart contract.

```
struct ERC20LOAN {
bool auctionStatus;
address seller;
uint256 startingPrice;
uint256 endingPrice;
uint256 auctionTimestamp;
uint256 auctionDuration;
address bondAddress;
uint256 interestRate;
uint256 loanDuration;
uint256[] bondClass;
uint256[] bondNonce;
uint256[] bondAmount;
}
```

```
function createERC20Loan (ERC20LOAN memory _ERC20Loan) public returns(bool)
require(contract_is_active=true,"contract is not active");
_ERC20Loan.auctionTimestamp=now;
require(_addERC20Loan(_ERC20Loan)==true,"can't create more auction");
require(IERC3475(bond_contract).writeInfo(_ERC20Loan));
require(_addPledgedERC20Asset(_ERC20Loan)==true,"can't move to custody");
return(true);
}
```

5.4.2 P2P securitised loan

Securitised loan has several advantages over traditional DEFI lending protocol, as shown below:

- **No enforced liquidation:** There is a risk of liquidation in the traditional DEFI lending protocol [14], since the collateral will be liquidated if their value falls below a certain level. In contrast, given that the securitised loans can be traded in the secondary market, their price will also decrease along with the collateral. This allows the market

to find the most appropriate price for the bonds without resorting to liquidation.

- **Customized loan agreement:** Interest rate, loan amount, repayment method and date are customizable. Unlike traditional loan agreements, securitised loans allow the creditor to set their own interest rate, loan amount, and repayment schedule. The creditors, debtors and the market will use the game theory to find a If the creditor's settings are reasonable.

- **Transferable securitised loan:** Securitised loans can be divided and assigned to others in the form of bond, turning the loan into a type of speculative investment. The fluctuation of the bond price in the secondary market can therefore create a new speculative market.

- **Higher usage rate of the pledged assets:** Traditional lending protocols often ask for over-collateralization while offering a loan of roughly half of the collateral value. This percentage is fixed in most protocols and cannot be adjusted according to the market conditions accordingly. In the securitised loan system, the creditors and the debtor are engaged in a game theory scenario where they try to find a Nash equilibrium. This allows the collateral rate to be aligned with the market supply and demand.

- **NFT pledged lending:** conventional lending agreements cannot accurately estimate the value of NFT collateral through an oracal. Because of the non-fungible nature of NFTs, NFT pledged loans have been inefficient. In a securitised loan, it is the creditor who appraises the NFTs and competes with each other to determine a best valuation before giving the loan to the creditor.

- **Loan bond splitting and bundling:** both NFT bonds and ERC20 token bonds can be split or bundled for trading. Riskier bonds can be bundled together for trading in the secondary market at a higher value. The same can be done to NFT bonds to spread the debtor's risk.

5.5 DeBond Derivatives

A part from being used as a tool of issuing bond, DeBond can also be used to create all kinds of financial derivatives. This include anything from option, future, swaps, etc..

DeBond can also be implemented for off-chain derivatives. By projecting the securitised contract to the block-chain network and using oracle machines

to feed the latest market data [11], DeBond can also be used to facilitate the transaction of off-chin bonds. This application requires an audited entity as the portal between the DeBond and off-chin derivatives. The use cases of the DeBond derivatives are:

- **Hedging:** Hedge or to mitigate investment risk, by entering into a derivative contract whose value moves in the opposite direction of their position and cancels part of the potential lost.
- **Option:** Create options with the value of the derivative is linked to a predetermined condition or event (e.g., the underlying asset reaching a specific price level).
- **Leveraging:** Provide leverage, such that a small movement in the underlying value can cause a large difference in the value of the derivative.
- **Speculation:** Make profit from speculating the market movements. (e.g. moves in a given direction, stays in or out of a specified range, reaches a certain level).
- **Arbitraging:** Allowing a riskless profit by simultaneously entering into transactions into two or more markets.

Some of the common variants of derivative contracts are as follows:

5.5.1 Forwards

Tailored contract between two parties, where payment takes place at a specific time in the future at today's pre-determined price.

5.5.2 Futures

Contracts to buy or sell an asset on a future date at a price specified today. A futures contract differs from a forward contract in that the futures contract is a standardized contract written by a clearing house that operates an exchange where the contract can be bought and sold; the forward contract is a non-standardized contract written by the parties themselves.

5.5.3 Options

Contracts that give the owner the right, but not the obligation, to buy (in the case of a call option) or sell (in the case of a put option) an asset. The price at which the sale takes place is known as the strike price, and is specified at the time the parties enter into the option. The option contract also specifies a maturity date. In the case of a European option, the owner has the right to require the sale to take place on (but not before) the maturity date; in the case of an American option, the owner can require the sale to take place at any time up to the maturity date. If the owner of the contract exercises this right, the counter-party has the obligation to carry out the transaction. Options are of two types: call option and put option. The buyer of a call option has a right to buy a certain quantity of the underlying asset, at a specified price on or before a given date in the future, but he has no obligation to carry out this right. Similarly, the buyer of a put option has the right to sell a certain quantity of an underlying asset, at a specified price on or before a given date in the future, but he has no obligation to carry out this right.

5.5.4 Binary options

Contracts that provide the owner with an all-or-nothing profit profile.

5.5.5 Warrants

Apart from the commonly used short-dated options which have a maximum maturity period of one year, there exist certain long-dated options as well, known as warrants. These are generally traded over the counter.

5.5.6 Swaps

Apart from the commonly used short-dated options which have a maximum maturity period of one year, there exist certain long-dated options as well, known as warrants. These are generally traded over the counter.

6 Governance

Although the idea of a decentralised governance has been mentioned oftenly. The general concept is that the community will be banded together forming a kind of "decentralized autonomous organization" or DAO. Is that of a virtual entity that has a certain weight over the voice of an address. The shareholders behind this address can then express their point of view through voting. In practice, any manipulation of the governance is through a centralized process, no obligation being imposed to the possessor of the master key owner. Platform such as snapshot[17] serves only as a vote counting service which imposes no obligation on the execution of the proposal. In the same sense, decentralised bond governance platforms allow the community to participate in the governance of the DEFI project. The members would collectively decide on how the project should invest its funds or where the project should do next.

6.1 Proposal

Decentralised governance platform uses an on-chain program called "proposal" as the object of a voting procedure. This means that the proposal in question is in the form of an open-sourced smart contract. The inputs of every single method called from the proposal contract to the governance contract will be written in the open-sourced contract. Meaning that an underlying obligation is imposed on the governance system to execute the proposal as expressed in the content of code. These methods can be used for spending the funds, issuing allocations, updating a smart contract, etc. We can see in the code below, the access to a governance function is given to a passed proposal contract, which is defined by a `proposal_class` and a `proposal_nonce`.

```
function migratorLP(
uint256 proposal_clarss,
uint256 proposal_nonce,
address _to,
address tokenA,
address tokenB
) public override returns(bool){
require(proposal_class<=1); require(checkProposal(proposal_class,
proposal_nonce) == true);
require(_proposalAddress[proposal_class][_proposalNonce[proposal_class]]
==msg.sender); require(ISigmoidBank(bank_contract).migratorLP(_to,
tokenA, tokenB));
```

```
return(true);
}
```

The calling of the governance function is strictly limited to an open sourced proposal contact. Here is an example of a proposal contract. In this example, the `updateBondContract()` function of the governance contract is called from the proposal contract. The new updated contract address and other parameters need to be written on the open-soured solidity code of the proposal contract.

```
contract proposal{
address public gov_address;
constructor (address dev_address) public {
gov_address=dev_address;
}

function updateBondContract() public returns(bool){
uint256 proposal_class,
uint256 proposal_nonce,
address new_bond_address,
require(ISigmoidGovernance(gov_address).updateBondContract(
poposal_class,proposal_nonce,new_bond_address)==true);
}
```

This system can be used ranges from salaries and investments to even more complex mechanisms such as an option, an algorithmic speculation bot. Decentralised governance platform can be used to apply any actions which can be expressed in mathematical expression. We believe that the idea of a decentralised governance is relying on the obligation to execute the collective will of the community. On this matter, most of the previous governance system has only the features of a vote counter.

6.2 DeCapital

The term DeCapital essentially replicates the legal trappings of a venture capital, an investment bank or a nonprofitable organisation. Through DeCapital, DeBond can invest or incubate any on-chain project. An Estonia-registered legal entity, called Sigmoid Labs [18] will be used as a gateway to the off-chain investment. The banking account of Sigmoid Labs will be audited by an accounting firm. All the investment decisions of the DeCapital, whether on-chain or off-chain, all need to be approved by voting on the DBG platform.

Along these lines, DeCapital can use the pledged asset of DeBond to build up LP for another DEFI project, invest in their seed phase, etc..

DeCapital will work like an investment bank, it borrows money from the investors and uses their investment to fund startups, buying stocks and speculating on the financial market. Using the cryptographic block-chain technology for the transparency enforcement, DeCapital will be accessible and trustworthy, like or even better than an traditional investment bank.

6.3 Voting

By pledging the governance token DGOV into the governance contract, the user will receive a specified DeBond called Vote. Vote can be used in the voting process of a proposal contract. Like any DeBond product, redemption conditions are applied on Vote. In order to redeem one's DGOV, the Vote bond holder must wait until a specified mature date. After this specified redemption date, Vote can be redeemed and the bondholders will receive the previous DGOV deposit and a small percent of DBIT as the reward of their participation.

7 TOKENS

DeBond issues two native tokens DBIT and DGOV. In DeBond system, tokens are minted by redeeming bond. User will fist buy or staking DBIT or DGOV bonds, then redeem the bond to a settlement currency DBIT or DGOV. The detailed description of this process can be found in section "4.4 DBIT and DGOV bond"

7.1 APM price influence

When adding liquidity using APM method, the liquidity of token t_1 is defined by Δl_1. And the increment of token t_2 is Δl_2

$$\Delta l_2 := l_{determined}(t_1, t_2, l_1)$$

The CDP of two tokens $\frac{\Delta l_2}{\Delta l_1}$ is not necessarily $= P(t_1, t_2)$. When this happens, the price of two tokens $P(t_1, t_2)$changes:

$$\Delta P(t_1, USD) = \frac{\Delta l_2 \cdot P(t_2, USD)}{s_{collateral}(t_1) + \Delta l_1}$$

When Alice redeem her DeBond, she will receive the paper value of the bond, repaid in the settlement currency associated to this bond class. In the case of interest payment in settlement token $t2$, after the payment:

$$k(l_1, l_2) := k_{new}(l_1, l_2)$$

In the case of the return of the principal using `stakingForBondFromToken()` method:

$$\begin{cases} l_2 := l_2 - P(l_1 - \Delta b_{erc20}, l_2)\Delta b_{erc20}, \\ l_1 := l_1 - \Delta b_{erc20}, \\ k(l1, l2) := k_{new}(l1, l2) \end{cases}$$

The APM method described in section 3.4 Automated Pair Maker (APM) changes the liquidity l_2,l_2 of the pair. However this change will not change the $P(t_1, t_2)$ of the pair in question, since the change in l_1, l_2 is proportional to the previous l_1, l_2:

$$\begin{cases} \dfrac{l_1}{l_2} = \dfrac{l_1 - \Delta l_1}{l_2 - \Delta l_2}, \\ P_{previous}(l_1, l_2) = P_{new}(l_1, l_2) \end{cases}$$

7.2 Average minting cost

The arbitrated price of DBIT $C_{DBIT}(s_{total})$ and DGOV $C_{DGOV}(s_{total}, s_{max})$ limit their lowest spot price in USD $P_{min}(t1, USD)$. This floor price of DBIT and DGOV depends on the average minting cost of the token in questions. Average minting cost in USD $P_{min}(t1, USD)$ is calculated by adding each bond nonce issued, Price in USD P of the time T multiplied by Quantity Q. Divided by the total amount of the bond issued.

$$P_{min}(t_1, USD) = \frac{\sum_{i=2}^{n} Q_i P_{determined}(t_i, USD, T_i)}{\sum_{i=2}^{n} Q_i}$$

7.3 Decentralized Bonds Index Token(DBIT)

DBIT is an ERC-20 token pegged to the bond index. The overall expectation of the DeBond market will be reflected in the form of a tokenized Index. The price of DBIT is highly associated with the general performance of the DeBond market [15].

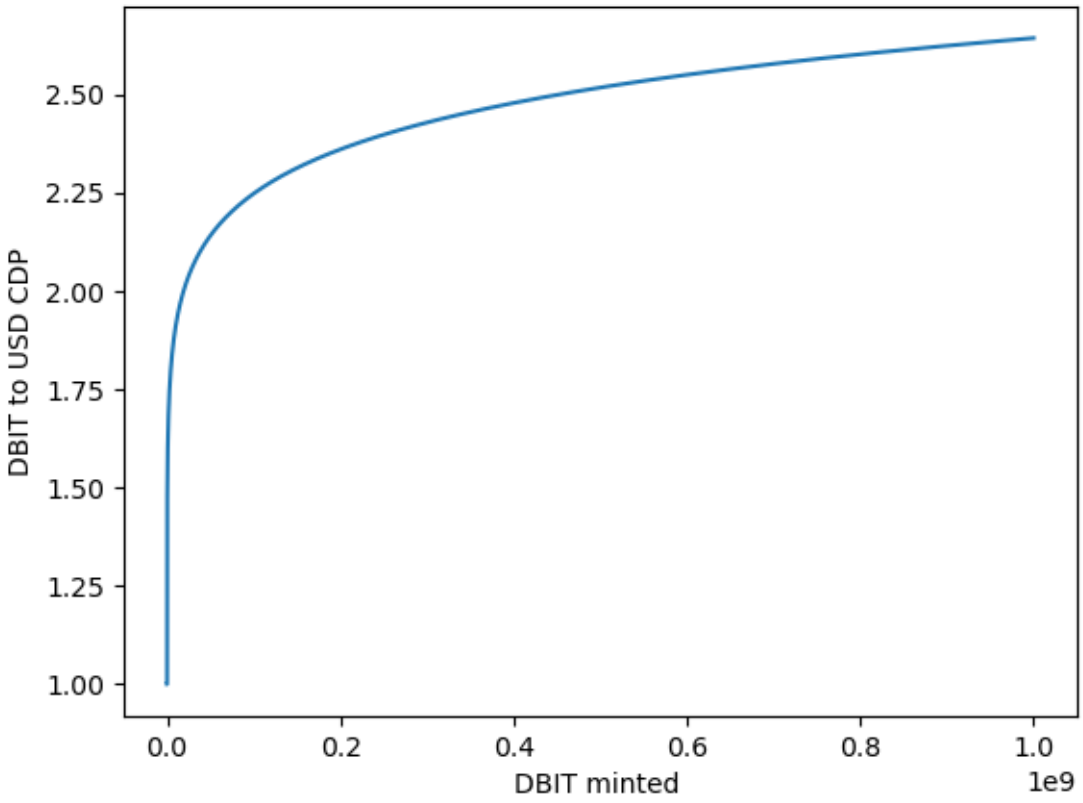
7.3.1 Settlement currency

DBIT will be used for bond interest payment and secondary bond market transaction. The final goal of DBIT is to become the market settlement currency. It's part of the index system that reflects the overall trend of the bond market.

7.3.2 CDP price of DBIT

The contract determined price of DBIT to USD is:

$$C_{\mathrm{DBIT}}(S_{collaterlised}) = 1.05^{\log_2(S_{\mathrm{collaterlised}}/1000)}$$



7.3.3 Index price

The price of DBIT to USD is guaranteed by the multi-collateral of the bonds in the market.

$$P(dbit, USD) = \frac{\sum_{i=1}^{n} l_{collateral}(i) P(t_i, USD)}{s_{collateral}(dbit)}$$

The reserve of mixed collateral and the fact that the token is pegged to the index can guarantee the floor price of DBIT.

$$P_{min,}(dbit, USD) = \frac{\sum_{i=1}^{n} l_{collateral}(i) P(t_i, USD)}{s_{total}(dbit)}$$

7.4 Decentralised bond governance token (DGOV)

DGOV is the governance token of a decentralised governance platform, a shareholding proof of their investment in DeCapital. By holding DGOV,The holder of DGOV indirectly holds the share of the On-chain and off-chain projects that DeCapital invested in. This allows the better use of the DeBond asset, and share profit with the DGOV holder.

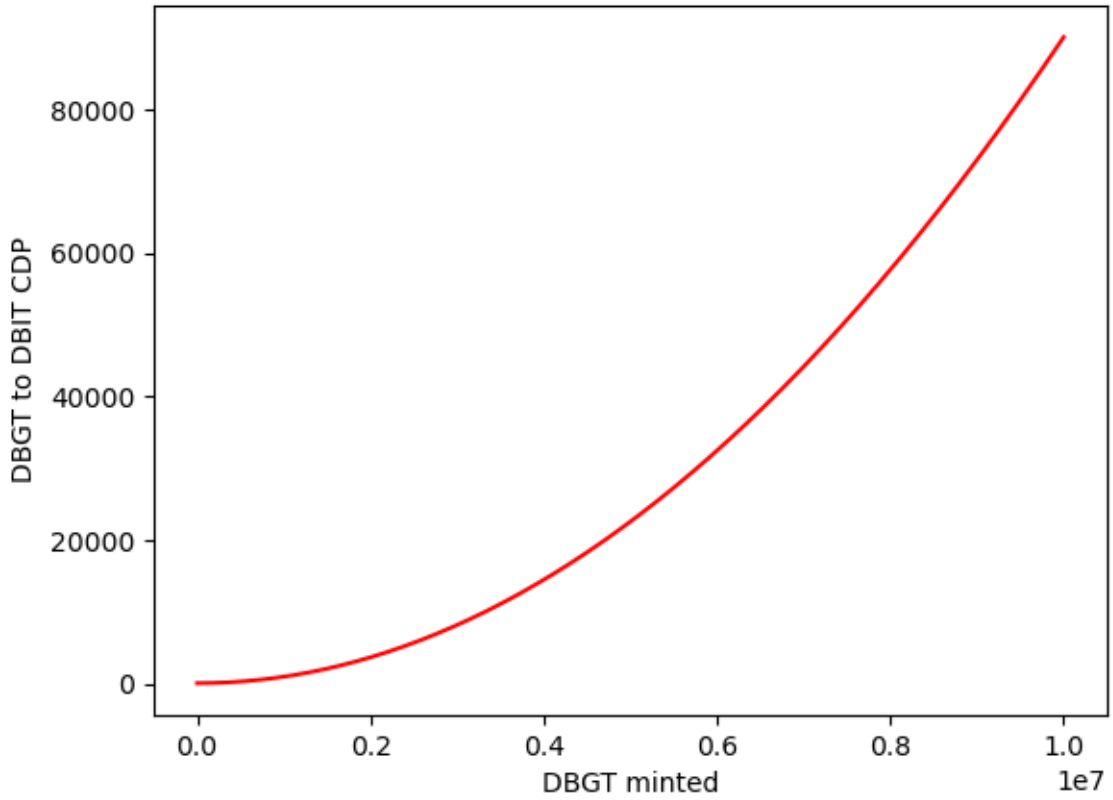
7.4.1 Bonus share

Apart from the governance reward when voting for a proposal, there will be bonus share from DeCapital and speculative actives. The profit will be pulled in to the DGOV swap pair, thus giving additional bonus reward to the DGOV holder.

7.4.2 CDP price of DGOV

The contract determined price of DGOV to DBIT is:

$$C_{dbit}(S_{collaterlised}) = 100 + (S_{collaterlised}/33333)^2$$



With the CDP of DIBT C_{dbit} in mind, the CDP of DGOV to USD is:

$$C_{DGOV} \cdot C_{dbit}$$

7.4.3 Market price

The price of DGOV to USD is guaranteed by the multi-collateral of the bonds in the market.

$$P(DGOV, USD) = \frac{\sum_{i=1}^{n} l_{collateral}(i) P(t_i, USD)}{s_{collateral}(DGOV)}$$

The reserve of mixed collateral and the fact that the token is pegged to the index can guarantee the floor price of DBIT.

$$P_{min,}(DGOV, USD) = \frac{\sum_{i=1}^{n} l_{collateral}(i) P(t_i, dbit)}{s_{total}(DGOV)} P(dbit, USD)$$

8 Pair Contract

A pair contract is a contract that can safeguard any number of tokens as token-pairs.

8.1 Price

When some liquidity of one of the tokens or of both of them are added to one of the pool pairs, the price of assets in the pool changes, therefore, each pair must be updated in order to update the price. Let's consider the liquidity pool ($\mathcal{L}$) of tokens $t_1, t_2, t_3, ...$, the liquidity of the token t_1 in the pair (t_1, t_2) is given by

$$l_{t_1(t_2)} \ = \ r_{t_1(t_2)} \times L_1 \tag{1}$$

where L_1 is the the total liquidity of tokens t_1 in the pool ($\mathcal{L}$)

$$L_1 \ = \ \sum_{k=1}^{n} l_{t_1(t_k)} \tag{2}$$

The price P_{t_2,t_1} of the token t_1 in the pair (t_1, t_2) is the ratio of the liquidity of tokens t_1 and t_2 in that pool

$$P_{t_2,t_1} \ = \ \frac{l_{t_1(t_2)}}{l_{t_2(t_1)}} \tag{3}$$

Now let's consider adding Δl_1 liquidity in the pool (t_1, t_2) for token t_1 and Δl_2 for token t_2, the liquidity of each token in the pair and the total liquidity of the two assets in the pool then change to

$$\begin{aligned} l_{t_1(t_2)} \ &:= \ l_{t_1(t_2)} + \Delta l_1 \, , \quad L_1 \ := \ L_1 + \Delta l_1 \\ l_{t_2(t_1)} \ &:= \ l_{t_2(t_1)} + \Delta l_2 \, , \quad L_2 \ := \ L_2 + \Delta l_2 \end{aligned} \tag{4}$$

The ratio factor $r_{t_1(t_2)}$ in therms on the total liquidity L_1 can be deduced from equation (1)

$$r_{t_1(t_2)} \ = \ \frac{l_{t_1(t_2)}}{L_1} \tag{5}$$

When adding Δl_1 and Δl_2 to the pair, the change in the ratio $r_{t_1(t_2)}$ and the price P_{t_2,t_1} becomes

$$\begin{aligned} r'_{t_1(t_2)} &= \frac{l_{t_1(t_2)} + \Delta l_1}{L_1 + \Delta l_1} \\ P'_{t_2,t_1} &= \frac{l_{t_1(t_2)} + \Delta l_1}{l_{t_2(t_1)} + \Delta l_2} \end{aligned} \tag{6}$$

From equations (6) it is simple to deduce the equation that can be used to update the price P_{t_2,t_1} through the ratio factor $r_{t1(t_2)}$ when some liquidity is added to the pool

$$\boxed{P'_{t_2,t_1} = r'_{t_1(t_2)} \frac{L_1 + \Delta l_1}{l_{t2(t_1)} + \Delta l_2}} \tag{7}$$

8.1.1 Case: $\Delta l_1 \ll l_1$ and $\Delta l_2 \ll l_2$

If the liquidity added to the pool is much smaller than the corresponding amount of tokens already in the pool ($\Delta l_1 \ll l_1$ and $\Delta l_2 \ll l_2$), the new ratio factor $r'_{t_1(t_2)}$ from (5) becomes

$$r'_{t_1(t_2)} \rightarrow \frac{l_{t_1(t_2)}}{L_1} = r_{t_1(t_2)} \tag{8}$$

The new price after adding liquidity in the pool can then be deduced from (7) and (8)

$$\boxed{P'_{t_2,t_1} \rightarrow \frac{l_{t_1(t_2)}}{l_{t_2(t_1)}} = P_{t_2,t_1}} \tag{9}$$

Relation (9) shows that the price remains unchanged in a pool with sufficiently high enough reserves.

9 CONCLUSION

DeBond protocol was originally conceived as an upgraded version of DEFI yield farming system, providing extended settings such as LP fragmentation and derivatives creation, etc.. We believe that the native use case of smart contract should be the financial derivatives. Since ERC-20 token can't fulfil this task, we propose a possible solution.

What we think is exciting about this project is that the update of the current yield farming system will bring more changes, possibilities and believers to the DEFI system, pushing DEFI to the next stage, where a more complex economic system can be designed using the ERC-3475 bond standard. We are expecting it to go beyond the boundary of our imagination. New protocols around decentralized options, derivatives and any other form of financial product, among many existing or non-existing such concepts. It has all the potentials needed to substantially increase the efficiency and computational complexity of DEFI. DeBond is an open-ended platform by design. We are looking forward to seeing a series of applications and use-cases built on top of the ERC-3475 infrastructure.

Hayek once said in an interview [19], "Freezing in its most primitive form, money was never allowed to be involved since its invention." We firmly believe that the Decentralized autonomous organisation or DAO is a possible answer to many social problems that we are facing in our current economic system. The Crypto-anarchism would allow the DeBond system to grow organically as a decentralized community. By assimilating developers and investors alike, the DeBond Eco-system will be evolving to a point that the collective intelligence will find an optimal solution to any problems that we will be facing.

References

[1] Ethreum Whitepaper, https://ethereum.org/en/whitepaper/

[2] EIP-20:Token Standard, https://eips.ethereum.org/EIPS/eip-20/

[3] EIP-3475:Multiple Callable Bonds, https://eips.ethereum.org/EIPS/eip-3475

[4] Volckart, Oliver (1997). "Early beginnings of the quantity theory of money and their context in Polish and Prussian monetary policies, c. 1520–1550". The Economic History Review. Wiley-Blackwell. 50 (3): 430–49.

[5] O'Sullivan, Arthur; Sheffrin, Steven M. (2004). Economics: Principles in action. Upper Saddle River, New Jersey 07458: Prentice Hall. pp. 197, 507. ISBN 0-13-063085-3

[6] Fisher Irving, The Purchasing Power of Money, http://public.econ.duke.edu/ kdh9/Courses/Graduate

[7] Bitcoin: A Peer-to-Peer Electronic Cash System, https://bitcoin.org/bitcoin.pdf

[8] Metamask Documentation, https://docs.metamask.io/guide/

[9] Documentation of Uniswap v1, https://docs.uniswap.org/protocol/V1/introduction

[10] Nash, John (1951) "Non-Cooperative Games" The Annals of Mathematics 54(2):286-295.

[11] Christos Papadimitriou. Computational Complexity. Addison-Wesley, 1994. Section 14.3: Oracles, pp. 339–343.

[12] Liebowitz, Martin L. and Homer, Sidney (29 April 2013). Inside the Yield Book (third ed.). Hoboken NJ: John Wiley & Sons, Inc. p. 117. ISBN 978-1-118-39013-9.

[13] The Maker Protocol: MakerDAO's Multi-Collateral Dai (MCD) System, https://makerdao.com/en/whitepaper/#notes

[14] AAVE protocol whitepaper, https://github.com/aave/aave-protocol/blob/master/docs/Aave_Protocol_Whitepaper_v1_0.pdf

[15] Kelly, Frank K. and Weight, David J. (1997). The Handbook of Fixed Income Securities (Frank J. Fabozzi, Editor). New York: McGraw-Hill. p. 129-130. ISBN 0-7863-1095-2.

[16] Pearson, Karl (1894). "On the dissection of asymmetrical frequency curves". Philosophical Transactions of the Royal Society A. 185: 71–110. Bibcode:1894RSPTA.185...71P. doi:10.1098/rsta.1894.0003

[17] SNAPSHOT documentation https://github.com/snapshot-labs/snapshot

[18] Company registration information https://www.infoproff.com/en/companies/search/

[19] An Interview with F. A. Hayek (1984) https://www.youtube.com/watch?v=s-k_Fc63tZI