Lending Base Layer

team@multiplier.tech
October 2024

DRAFT

Abstract

Lending Base Layer is a decentralised protocol designed to serve as universal "back office" for collateral-based financial products. The protocol brings together mechanical elements common to a wide range of such designs, without imposing restrictions on their commercial logic, such as capital structure, risk management and pricing. It comprises a system of borrower and lender accounts connected by a mechanism for entering/exiting an abstract borrowing relationship paired with transfer of authority over collateral, and an intents-based mechanism for state transitions.

1 Introduction

Blockchain-based lending protocols have found strong product-market fit from the early days of decentralised finance (DeFi). One of their key success factors has arguably been relative simplicity – foregoing feature richness and capital efficiency in favour of lower technical risk. However, with increased maturity and understanding of such mechanisms, this trade-off is becoming less favourable. Traditional finance provides plentiful evidence that serving the full spectrum of demand requires a varied and sophisticated product range, which DeFi needs to deliver in order to compete on global scale.

A smart contract system designed for the Ethereum Virtual Machine, Lending Base Layer seeks to simplify the process of building more complex and performant solutions by providing a universal and un-opinionated lending "back office". The protocol comprises a system of smart wallet-like accounts combined with a lending-specific toolkit. The toolkit includes a specialised accounting system to express indebtedness, paired with a mechanism that implements the idea of collateral through transfer of authority (control) over borrower's account instead of the underlying assets. To enforce its rules and enable improvements in user experience, the protocol also implements a batch-based state transition mechanism that natively supports intents-based logic.

2 Protocol Overview

2.1 High-Level Principles and Architecture

Lending Base Layer is a collection of individually controlled accounts with a permanently assigned type of "borrower" or "lender", connected through a joint system of debt accounting and subject to a common set of rules. In broad terms, these rules comprise proper authority, lender uniqueness and mutual consent.

Discussed in more detail below, authority can be viewed as the right to exercise control over an account. It is designed to provide lenders with access to borrower-owned collateral without the need to take direct possession, while enabling borrowers to manage it subject to lender-defined rules (e.g. maximum loan-to-value ratio).

Lender uniqueness means that at no point can an account have outstanding loans from more than one lender. This ensures that authority over collateral can always be attributed to a single lender, and thus avoids double-counting (i.e. same assets securing several loans independently).

Mutual consent requires that a lender not be able to force a borrower into a loan, thus taking control of assets contained in borrower's account. In the language of the base layer, it means that entering into the borrower-lender relationship can only be done with express authority of both parties.

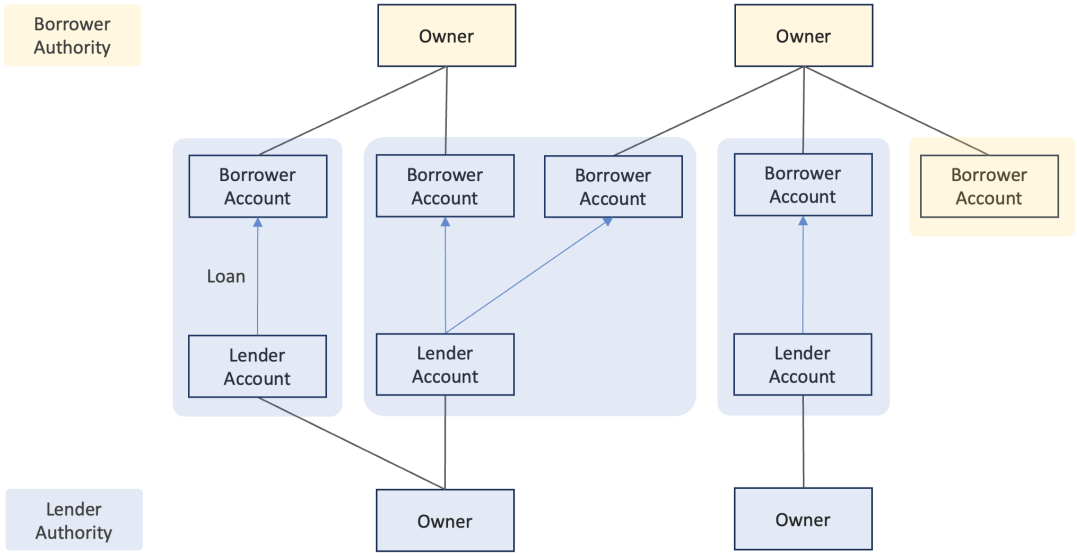

Figure 1: Lending Base Layer Architecture

The protocol's architecture enables independent parties to offer an unlimited number of risk-segregated products, to freely define their degree of interconnectedness and the rules for their individual and aggregated balance sheets.

2.2 Accounts

Lending Base Layer features two distinct types of accounts – for borrowers and lenders. Both have an external owner (EOA or smart contract), and are functionally similar to a regular smart wallet. The key distinction is that the former cannot lend, while the latter cannot borrow.

Beyond the typical, the accounts also feature *BorrowBalance* and *LendBalance*, quantifying borrower-lender relationships in which they are engaged. The former means that borrower account B owes lender account L a number of units of debt, denominated in some abstract asset A. In the same way, the latter means that lender account L is owed by borrower account B. These variables are effectively mirror-images of each other, split to ensure borrowing occurs with mutual consent.

For borrower account B and lender account L:

(1) $B = \{(TokenBalance), (BorrowBalance_{A,L,B}), Owner, Terms\}$

(2) $L = \{(TokenBalance), (LendBalance_{A,L,B}), Owner, Terms\}$

Terms can be viewed as an owner-controlled container for arbitrary logic that can be executed without the owner's signature. In other words, it is a mechanism for the owner to delegate, through explicit methods, some authority over account variables to third parties.

2.3 Transaction Batching

Account state can only be modified through a native transaction batching mechanism, which allows monolithic execution of an arbitrary sequence of methods in compliance with base layer rules.

The batch is designed to natively support intents, allowing for execution to be conditional on end state checks. This is achieved by dividing it into two phases – execution and verification. Execution phase can include logic that modifies base layer state variables, and contains a mechanism to request additional logic to be run in the verification phase (*requestCheck*), where state can no longer be modified but overall execution can be reverted if some condition is not met. The following configuration may offer a mental model of a typical batch:

(3) [**execution**: $Method_0, Method_1$ ... | **verification**: $Check_0, Check_1$ ...]

2.4 Authority

Lending Base Layer implements the notion of collateral through transfer of authority, or control, over borrower's account to the lender, removing the need to move underlying assets. In simple terms, authority stems from presence of an outstanding loan, and passes from owner of the borrower' account to owner of the lender's account upon the first borrow. Once the loan is repaid in full, authority reverts to the original state.

Proper functioning of authority requires lender uniqueness. This property is assumed in the definitions below, and enforced through methods defined in sections that follow.

2.4.1 Authority Check

To achieve the above functionality, we construct an authority check that must be passed in any attempt to modify account variables (with the usual exception of in-bound token transfers). First, we generalise the idea of account owner to that of *Comptroller*. For lenders, *Comptroller* and *Owner* are always the same. For borrowers, taking a loan means ceding *Comptroller* role to the lender, and with it the ability to control own account's contents. For borrower account B, lender account L and asset A we have:

(4) $$Comptroller_B = \begin{cases} Owner_B & if\ BorrowBalance_{A,L,B} = 0\ \forall\ A, L \\ Owner_L & otherwise \end{cases}$$

(5) $Comptroller_L = Owner_L$

For a given account, we further define *ActiveTerms* as *Terms* that have authority to change its state without *Comptroller*'s signature on the batch. In effect, they contain *Comptroller*'s delegated authority.

$$(6)\quad ActiveTerms_B = \begin{cases} Terms_B & if\ BorrowBalance_{A,L,B} = 0\ \forall\ A, L \\ Terms_L & otherwise \end{cases}$$

$$(7)\quad ActiveTerms_L = Terms_L$$

Finally, we have the following definition of *AuthorityCheck*, where *Executor* is the EOA or smart contract address that executes the batch.

$$(8)\quad AuthorityCheck = \begin{cases} Pass\ \ if\ Executor = Comptroller\ \vee\ Msg.Sender = ActiveTerms \\ Fail \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad otherwise \end{cases}$$

When acting directly and under own authority, the process closely resembles working with a regular smart wallet – owner's signature on the batch means account variables can be freely modified. Beyond the limits of own authority, one must make use of authority of others, delegated through explicit methods contained in *ActiveTerms*.

2.4.2 Delegation

While generality of the delegation mechanism could potentially see it applied for a wide variety of purposes (e.g. AMM-like rebalancing of account's contents or limit orders), we demonstrate its role through an example most relevant to lending.

Given a borrower account subject to lender's authority, suppose the lender wants to offer the borrower maximum freedom to manage their portfolio – to move assets in and out of the account, to borrow and repay loans – so long as at the end of each batch they can meet some loan-to-value threshold. To implement this functionality, what the lender needs to do is add explicit methods for each of these actions to their own *Terms*.

For instance, a "borrow" method would contain logic along the lines that any borrower account whose owner is the batch executor can take tokens from lender's account while at the same time incrementing both *BorrowBalance* and *LendBalance*, and also passing a loan-to-value check in the verification phase of that batch.

2.5 State Transition

Lending Base Layer preserves general smart wallet functionality within the confines of its rules, and extends it with native methods that govern the internally defined *BorrowBalance* and *LendBalance* variables. The role of these methods is to ensure proper transition of authority, while enforcing lender uniqueness and mutual consent.

Lender uniqueness is implemented as an after-method check in (9). Mutual consent is enforced through equality check on *BorrowBalance* and *LendBalance* at the end of batch, thereby requiring authority of both parties to increment both variables on first borrow.

The methods are described in terms of access control (who can execute them), state transitions they affect, and checks to be carried out immediately after state transitions and at the end of batch (in verification phase).

We omit explicit specification of methods governing token balances, *Terms* updates and ownership transfer, which would likewise be subject to the authority check but otherwise depend on specific implementation details.

2.5.1 BorrowBalance Methods

For a given borrower account B, lender account L, asset A and $\Delta > 0$:

(9) $mintBorrowBalance$

$Access\ control$: $AuthorityCheck$
$State\ transition$: $BorrowBalance_{A,L,B} = BorrowBalance_{A,L,B} + \Delta$
$After\ method$: $BorrowBalance_{A,L',B} = 0\ \ \forall L' \neq L$
$End\ of\ batch$: $BorrowBalance_{A,L,B} = LendBalance_{A,L,B}$

(10) $burnBorrowBalance$

$Access\ control$: $AuthorityCheck$
$State\ transition$: $BorrowBalance_{A,L,B} = BorrowBalance_{A,L,B} - \Delta$
$After\ method$: na
$End\ of\ batch$: $BorrowBalance_{A,L,B} = LendBalance_{A,L,B}$

2.5.2 LendBalance Methods

For a given borrower account B, lender account L, asset A and $\Delta > 0$:

(11) $mintLendBalance$

$Access\ control$: $AuthorityCheck$
$State\ transition$: $LendBalance_{A,L,B} = LendBalance_{A,L,B} + \Delta$
$After\ method$: na
$End\ of\ batch$: $BorrowBalance_{A,L,B} = LendBalance_{A,L,B}$

(12) $burnLendBalance$

$Access\ control$: $AuthorityCheck$
$State\ transition$: $LendBalance_{A,L,B} = LendBalance_{A,L,B} - \Delta$
$After\ method$: na
$End\ of\ batch$: $BorrowBalance_{A,L,B} = LendBalance_{A,L,B}$

3 GOVERNANCE

Lending Base Layer is designed without any overarching governance, nor does it place any product-level requirements on lender accounts and their owners. Instead, it freely supports any exogenous governance mechanism that account owners of may want to implement. In other words, any governance mechanism seen in DeFi at the time of writing can be carried over to a lending product implemented on Lending Base Layer.

4 REFERENCES

[1] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, Dan Robinson. Uniswap v3 Core. 2021.

[2] Mathis Gontier Delaunay, Paul Frambot, Quentin Garchery, Mattieu Lesbre. Morpho Blue Whitepaper. 2023.

[3] Hayden Adams, Moody Salem, Noah Zinsmeister, Sara Reynolds, Austin Adams, Will Pote, Mark Toda, Alice Henshaw, Emily Williams, Dan Robinson. Uniswap v4 Core [Draft]. 2023.

[4] Emilio Frangella, Lasse Herskind. Aave v3 Technical Paper, 2022.

[5] Hayden Adams, Noah Zinsmeister, Mark Toda, Emily Williams, Xin Wan, Matteo Leibowitz, Will Pote, Allen Lin, Eric Zhong, Zhiyuan Yang, Riley Campbell, Alex Karys, Dan Robinson. UniswapX. 2023.

[6] Euler Team. Euler Vault Kit Whitepaper. 2024

[7] Euler Team. Ethereum Vault Connector Whitepaper. 2024

5 DISCLAIMER

This paper is for general information purposes only. It does not constitute investment advice or a recommendation or solicitation to buy or sell any investment and should not be used in the evaluation of the merits of making any investment decision. It should not be relied upon for accounting, legal or tax advice or investment recommendations. Opinions reflected in this paper are subject to change without being updated.