Cashmere Labs: Solving Cross-chain Aggregation

0xFust 0xReliq 0xTanx

Abstract

There is still no effective way for swapping native assets between chains, despite the demand for asset swaps between blockchains continually rising. Existing cross-chain aggregators have three fundamental problems: (1) they use synthetic assets, (2) they make swap processes centralized and cannot design a decentralized architecture, (3) these aggregators only support a few pathways and only a few amounts of specific assets, (4) they have high slippage due to low liquidity of assets and cannot be efficient, (5) they are vulnerable to MEV and front-run attacks and can cause users to lose funds by being attacked. In this study, we present the Cashmere architecture, which maintains completely decentralized, protects users from MEV front-run attacks, can swap any asset between any chain, and offers the lowest slippage thanks to its unique marginal slippage algorithm.

1 Introduction

We are just at the beginning of the road for DeFi. Just think about it: TVL in the De-Fi corresponds to 1/35 of JPMorgan's assets. The market has had an incredible growth rate for the last two years. A new player joins the market daily, and the competition grows more robust. As the number of conscious users increases, the technology and quality offered by the projects become more critical. Users no longer have to transact on a single network but instead, have various assets on various networks to make investments. It's thought that one of the essential processes for DeFi in the future is Layer2, and the cross-chain products will be the game-changer, which means it will be the mainstream itself. Cashmere Labs offer solutions to these problems users face.Cross-chain asset swap is a typical operation in users' daily blockchain life. Cross-chain aggregators handle this movement of assets, but unfortunately, current aggregators in the market compromise any of the following functions: decentralized architecture (section 2.1), protecting MEV attacks (section 2.2), efficient marginal slippage (section 2.3), the ability to support any asset (section 2.4).

2 Background

In this study, a network or chain network is defined as the collection of chains participating in the cross-chain asset exchange protocol. All chains within a network get connected via a pair of unidirectional "connections" in which native assets can be moved directly. Each connection is supported by liquidity on the receiving end to permit user withdrawals as part of the transfer protocol; the quantity of available liquidity on the receiving chain can be considered the "bandwidth" of the connection. Cashmere uses unified liquidity, decentralizes the assets and liabilities of each pool through contracts, and runs its algorithm to keep it in bandwidth.

2.1 Decentralized architecture

The major issue with current cross-chain aggregators is that they are centralized. User assets might be compromised when centralized backends and market makers fail, or other errors may occur. Most of the existing cross-chain aggregators, such as LI.FI, Hashflow uses centralized market-makers and backends; this causes users not to be able to find sufficient liquidity in the source or destination chain and not be able to reach a working pathway for their swap, so the user cannot access the correct pathway and swap their assets between chains, that is relatively inefficient for current cross-chain aggregators, that process pushes the user to perform inefficient operations such as: (1) manually revert, (2) waste of gas, (3) fail to make a profit of the application due to can't find a correct and working asset swap aggregation path. Cashmere's decentralized architecture involves completing the cross-chain aggregator process simply through completely permissionless contracts without being dependent on any person, institution, or another element.

In Cashmere transactions, the contract receives an on-chain signature from the user; the signature includes 'the token contract, token amount, minimum return amount, destination chain, etc. The signature will match the Cashmere relayers in the source and destination chain. If the match is incorrect, Cashmere relays are fraudulent, and the contract will not execute the transaction. However, if the signature matching is correct between relayers and on-chain signatures, Cashmere will complete the transaction, and the security problem gets solved in a completely decentralized way. In addition, cashmere relayers are distributed and decentralized, and PoS (proof of stake) is required to become a relayer.

2.2 Protecting MEV attacks

Existing cross-chain aggregators cannot provide decentralized MEV protection, creating inefficiency for users. Problems that may occur: (1) loss of funds by being a front-run attack, (2) price manipulations in cross-chain asset transitions. Cashmere has created an on-chain MEV protection solution that can track asset LP reserves real-time and provide the user with the best slippage value. The slippage value provided is not profitable for MEV bots; it calculates so that it does not get an MEV attack. The slippage value transparently informs the user of the minimum amount that the user will receive and records it on-chain; the user does not lose funds or gets an attack from MEV bots, which is very efficient for users. Even wallets that manage significant funds can use the system comfortably because the minimum amount they will receive on the destination chain gets recorded as decentralized and on-chain.

In most of your transactions in a chain, with or without your knowledge, you are faced with MEV attackers which profit from your transaction. At the time of writing, various attacking techniques are present. Some of those techniques are fairly complicated and not widely studied/known. The most popular ones of those are **backrunning** and **frontrunning.** In this paper, we study frontrunners who manipulate the slippage of transactions, and we present two countermeasures involving the examination of an optimal slippage.

What Is Happening in MEV?

A typical attack consists of the attacker monitoring the mempool and examining the pending transactions, and continuously making a decision whether if it is possible to profit from the swap based on the minimum amount that the user is happy with. Firstly, the attacker searches for a maximum amount that can be placed in the according order, that does not overflow the user's slippage. Then, with this value, the attacker creates a sandwich attack simulation by taking the actual pool reserves into account. If the simulation suggests a profit, then the sandwich attack is performed. Meanwhile, the other attackers perform the same attack, resulting in a fee war which involves increasing `gweis` in response to themselves with the aim of being the first one manipulating the target reserves. Although there are various scenarios for the outcome of the war, generally, the losers of the war retreat by checking the reserve changes using their contracts.

The Slippage Issue

The decentralized exchanges offer predefined slippages which oversimplify the slippage decision. Even though the users can specify the slippage they want, they are still exposed to successful attacks which results vast amount of profit. The main reason of this is that "slippage" is dynamic in nature, and is strictly dependent to the pool reserves. Consider the following concrete case. An 25 ETH order was placed with 0.5% slippage that can be thought sufficiently small, but was attacked successfully.

Amount In:	24993115033964438000
Amount Out Min:	28192563480452695000000
Reserve In:	417456633660702250000
Reserve Out:	477508450948885740000000
Suggested Slippage:	0.004802227798474576%

The opposite also could have been the case: an order with 20% slippage that can be thought too big could have easily escaped from the attacks. Therefore, slippage must be determined carefully with respect to pool reserves.

How Does Frontrunner Think?

The essential part of the countermeasure is to take the path the attacker takes, preferably more quickly. To do that, we determine the maximum amount to be placed as if we will attack to our order. For this process, attackers can use Binary Search, although even this basic algorithm is not used most of the time because of performance issues. In some cases, Binary Search requires tens of thousands of iterations to give an accurate result. This is exactly why we use Golden Section Search for countermeasure. GSS gives accurate results with 23 iterations with 10^{15} tolerance.

Adjusting the Slippage

Response to an attacker is effectively done with adjusting the slippage accordingly. Start with a slippage, calculate the maximum amount to be placed and the expected profit of the attacker. Then, choose a direction for the slippage, and check if the expected profit is decreased. Repeat the process until you are happy with the attacker's expected profit. This is actually an equilibrium point between you and the attacker. We are sure that the attacker will not be happy about it. Thus, you will not be on this kind of attackers' radar.

The abovementioned, simplified version of the search can be done in a sophisticated way with GSS. The arguments are self-explanatory, but it should be noted that the `reserveIn` and `reserveOut` are the reserves before the block has yet been mined.

To sum up, attackers take advantage of poorly chosen slippage by users. It also seems that decentralized exchanges do not help users, either. This all roots from the dynamic nature of slippage. We dynamically specify it depending on the pool reserves.

In this article, we demonstrate a countermeasure to frontrun attacks by mimicking frontrunners and finding an equilibrium by adjusting slippage.

```go
func Slippage(
  amountIn, reserveIn, reserveOut float64,
) (float64, float64) {
  f := func(step float64) float64 {
    ou := GetAmountOut(
      amountIn,
      reserveIn,
      reserveOut,
    )
    ou -= (ou * step) / 100.
    frIn, _ := GetMaxAmountIn(
      amountIn,
      ou,
      reserveIn,
      reserveOut,
    )
    frProfit := ExpectedProfit(
      amountIn,
      ou,
      reserveIn,
      reserveOut,
      frIn,
    )
    return frProfit
  }
  return GssMin(f, 0.001, 20, 0.0001, nil)
}
```

Figure 1: Slippage function searching an optimal slippage

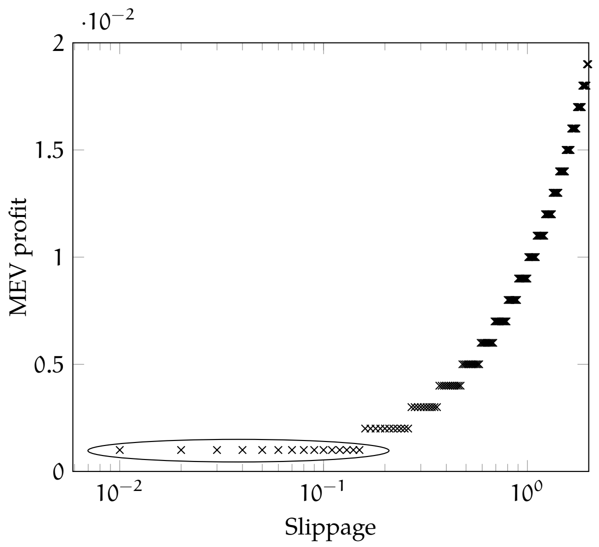

Figure 2: Existence of a safe zone against the MEV attacker

2.3 Efficient marginal slippage

Another problem with cross-chain aggregators is inefficiency and high slippage values. They cannot provide efficient slippage values because they do not have stable or non-stable pools due to their architecture and work only from external sources. If there is insufficient liquidity inexternal sources, they will offer inefficient slips, as can be foreseen. Although Cashmere is a cross-chain aggregator, it is a super-app with stablecoin liquidity and marginal slippage calculation. Since Cashmere has its stablecoin pools, it is not affected by insufficient liquidity and high slippages of external resources. In the source chain, Cashmere first swaps the assets via 1inch or other APIs to native stablecoins, transfers the native stablecoin between the chains through its pools, swaps the stablecoin accessed from its pool in the destination chain back to the desired asset with 1inch or other APIs and delivers to the user. Cashmere doesn't get high and inefficient slippages as the actual cross-chain transfer is with its stablecoin pools. On the other hand, marginal slippage is an efficient calculation applied to keep Cashmere pools in bandwidth between chains. Unlike other inefficient applications, Cashmere does not use a fixed slippage value; in contrast, it uses a dynamic and efficient value. Ifthe pools are in bandwidth, the user does not have to pay slippage; if the pools are lower than its bandwidth, the user has to pay slippage. It also provides positive and negative arbitrage opportunities.

2.4 Ability to support any asset

Another problem with cross-chain aggregators is inefficiency and high slippage values. They cannot provide efficient slippage values because they do not have stable or non-stable pools due to their architecture and work only from external sources. If there is insufficient liquidity inexternal sources, they will offer inefficient slips, as can be foreseen. Although Cashmere is a cross-chain aggregator, it is a super-app with stablecoin liquidity and marginal slippage calculation. Since Cashmere has its stablecoin pools, it is not affected by insufficient liquidity and high slippages of external resources. In the source chain, Cashmere first swaps the assets via 1inch or other APIs to native stablecoins, transfers the native stablecoin between the chains through its pools, swaps the stablecoin accessed from its pool in the destination chain back to the desired asset with 1inch or other APIs and delivers to the user. Cashmere doesn't get high and inefficient slippages as the actual cross-chain transfer is with its stablecoin pools. On the other hand, marginal slippage is an efficient calculation applied to keep Cashmere pools in bandwidth between chains. Unlike other inefficient applications, Cashmere does not use a fixed slippage value; in contrast, it uses a dynamic and efficient value. If the pools are in bandwidth, the user does not have to pay slippage; if the pools are not in bandwidth, the user has to pay slippage. It also provides positive and negative arbitrage opportunities.

3 Algorithm Design

3.1 Terminology

1. S: source chain, D: destination chain.
2. **LP** (LP_s): Amount of assets executed in the `_deposit` function.
3. **Assets** (A_s): Current assets contained in the LP contract.
4. **Weight**: The percentage of liquidity that will be transferred to chain S to D.
5. **Bandwidth** ($B_{s,d}$): Funds are allocated locally for transfers from S to D.
6. **Known bandwidth proof** (kbp): Last known amount of bandwidth that can swap from source to destination.
7. **Voucher** (v): Amount that will send to the destination chains in the next transfer.
8. **Optimal Bandwidth** (**OP**): $\mathrm{LP}(d) \times \mathrm{weight}(d)$.
9. **Compensation Ratio** (c): The ratio showing the relationship between assets and liabilities in the related pool.

$$\mathrm{CompensationRatio}(s) = \frac{\mathrm{actualKBP}}{\mathrm{optimalDSTBandwidth}}$$

$$\mathrm{CompensationRatio}(d) = \frac{\mathrm{actualBandwidth}}{\mathrm{optimalSRCBandwidth}}$$

10. **ActualBandwidth** ($\alpha B_{s,d}$): Funds are allocated locally for transfers from S to D independent of their weights.
11. **actualKBP** (aKBP): Last known amount of actualBandwidth that can swap from source to destination.

3.2 Stableswap Algorithm

$$\mathrm{InitialBandwidth}_{s,d} = A_d W_{d,s}$$

The algorithm always tries to maintain this bandwidth

$$0 \le W_{s,d} \le 1$$

$$\sum sW_{s,x} = 1.$$

(The sum of the weights from any source network to all other unknown networks (x) must be 1.)

t: transaction amount
x: destination networks (can be more than one)
On the source chain:

$$\begin{array}{ll} \text{Reject transaction} & \text{if } B_{s,d} < t; \\ \text{Execute transaction} & \text{if } B_{s,d} > t. \end{array}$$

The transaction started, user swaps the amount t from the source chain to the destination chain

$$\text{execute\_update}\, A_s = A_s + t\_$$
$$\text{execute\_update}\, B_{s,d} = B_{s,d} - t\_$$

Check the bandwidth deficit or bandwidth surplus of all other destination chains (x) defined in the contract.
Diff check mechanism:

$$\text{diff}_{s,x} = \max\left(0, LP_s w_{s,x} - (kbp_x + v_{s,x})\right)$$

Check this diff function separately for all destination chain (x) pools defined in the Router contract. This check gives how far the destination networks are from the initial bandwidth value.

$$LP_s w_{s,x} = \text{Bandwidth}_d$$
$$kbp_x + v_{s,x} = \text{Bandwidth}'_d$$

After the transaction amount reaches the destination chain bandwidth, that is, $\text{Bandwidth}'_d$; if $\text{Bandwidth}'_d > \text{Bandwidth}_{d'}$, the $\text{diff}_{s,x}$ result will be negative whichmeans the destination chain bandwidth is in surplus. The "max" filter in the $\text{diff}_{s,x}$ function takes negative values as zero. If $\text{Bandwidth}'_d < \text{Bandwidth}_{d'}$, the result $\text{diff}_{s,x}$ will be positive, and the destination chain bandwidth is in **deficit**. The "max" filter in the $\text{diff}_{s,x}$ function takes the **positive value to be filled**.

Total is defined to be the sum of the distances between destination networks (x) and initial "bandwidth" value (except source chain):

$$\text{Total} := \sum \text{diff}_{s,x}$$

1. **if** $t > \text{Total}$,
   The transaction amount can fill all the required bandwidth deficits. In this case, firstly, **vouchers** are sent for chains equal to their bandwidth deficits.

   $\text{Total} - t$ is the remaining transaction amount after bandwidth deficits are filled with t' vouchers. t' distribution will be as follows: Regardless of whether their bandwidths are in deficit or surplus, the remaining amount will be distributed depending on weights as vouchers.

   $$\frac{t' W_{x_n}}{\text{TotalWeight}} = \text{voucher}_{x_n} \text{ to send}$$

   where x_n is the nth destination.

2. **if** $t \le \text{Total}$,
   In this case, the transaction amount is too small to fill the bandwidth deficits in the destination chains. Destination chains with a bandwidth deficit are the 1st priority to fill.

   To bring their bandwidth deficits closer to initial bandwidth as much as possible, **vouchers will be sent according to their weight only for chains with bandwidth deficits**. In this case, no vouchers will be sent to chains that do not have bandwidth deficits.

   $$\frac{t' W_{x_1}}{\text{TotalWeight}} = \text{vouchers}_{x_1}$$
   $$\frac{t' W_{x_2}}{\text{TotalWeight}} = \text{vouchers}_{x_2}$$
   $$\frac{t' W_{x_n}}{\text{TotalWeight}} = \text{vouchers}_{x_n}$$

3.3 Marginal Slippage Algorithm

Cashmere Protocol uses a single-side AMM, unlike other AMMs. AMMs like Uniswap and Curve use invariant functions. But this causes some problems. One of them is impermanent loss. The impermanent loss leads to a liquidity shortage, mainly because the user's annual return cannot meet the impermanentloss. Cashmere Labs uses Single-Side AMM for stable swap. Single-side AMM allows stablecoins to be swapped with very low slippages. The system uses the Compensation Ratio system for this; LP logic is not the case. Additionally, with the assistance of the Cashmere Labs, stable swap transactions can be carried out with the least amount of slippage and without needing any bridge across networks. This process accelerates and simplifies the user's movement between Layer2 networks. Very high-volume transactions can be made with Single-Side AMM. You can receive positive slippage if you switch from a pool with a low compensation ratio to a pool with a high compensation ratio. In this way, financial players earn an arbitrage opportunity, which helps secure the equilibrium spot. We will now examine the swap mechanism in detail.

The compensation rate is one of the most considered variables in the system of Stableswap. The compensation ratio shows us the relationship between the system's asset value and liability. It functions like a kind of ledger. Liquidity provided to the protocol would become a liability. A higher compensation ratio indicates a lower default risk. The compensation ratio is an essential parameter in our protocol since it needs to be maintained above a certain level to avoid default. The protocol's liquidity would become a liability. A larger compensation ratio indicates a reduced default risk.

The token is under-compensated if the compensation ratio is less than one. And if the compensation ratio is more significant than one, the area is overcompensated. When a swap occurs in Cashmere, the swap-from token's liquidity (in the system pool) increases, while the swap-to-token's liquidity (in the system pool) falls. Cashmere supports convergence toward equilibrium while penalizing deviation from it. As a result, price slippage has been defined as a function of the compensation ratio.

We take the compensation ratio of USDT at 1.10 and the USDC at 0.90. Working this out, we get:

USDT:

$$g'(1.10) = -\frac{0.00002 \times 7}{1.10^8} = 0.006\%$$

USDC:

$$g'(0.90) = -\frac{0.00002 \times 7}{0.90^8} = 0.03\%$$

Hence we have

$$S_{\text{USDT}\to\text{USDC}} = 0.0006\% - 0.03\% = -0.024\%.$$

The liquidity provided by Cashmere Labs would become a liability to the protocol. The value is the L_i for the token i account. A_i for the token i account that the protocol currently holds. The compensation ratio c_i is defined to be $\frac{A_i}{L_i}$, which is a ratio of default risk of token i account. A higher compensation ratio means a lower risk of default. $c_i = 1$ is the optimal value for the protocol. The protocol uses several systems to bandwidth this compensation ratio, such as the positive arbitrage and interest rate model. If any kind of default risk occurs, it allows users to exit from different assets.

Cashmere does not use the slippage system. For this reason, it aims to penalize large amounts of transactions made in a single transaction. Cashmere does not use the slippage system. For this reason, it aims to penalize large amounts of transactions made in a single transaction. In these transactions, the exchange rate used in the transfer is rearranged. It can be defined as:

$$y^*_{i\to j} = y_{i\to j}(1 - S_{i\to j})(1 - \tau)$$

$y_{i\to j}$ is the external price oracles, $S_{i\to j}$ is the average slippage and τ is the trading fee. After a swap δ_i token i to token j, $A'_i = A_i + \delta_i$ and $A'_j = A_j - y^*_{i\to j}\delta_i$. Then we have two compensation ratios:

$$c'_i = \frac{A_i + \delta_i}{L_i} > \frac{A_i}{L_i} = c_i$$
$$c'_j = \frac{A_j - y^*_{i\to j}\delta_i}{L_j} < \frac{A_i}{L_j} = c_j$$

Cashmere uses the single variant slippage function instead of invariant curves. Define the account slippage function to be $g : \mathbb{R} \to [0, 1]$ that maps the compensation ratio of a token account to a slippage value between 0 and 1. The g function must have some properties.

1. g is continuously differentiable.
2. $g' \leq 0$ if g' exists.
3. g is a weakly convex function.

g' will be referred to as "marginal slippage" in this context. g is the decreasing function to prevent a reduced compensation ratio. When any asset compensation ratio is recovered, the function can provide users with less slippage. g is the weakly convex function because if c_i is low, the cost of reducing the compensation ratio would increase.

We could generate an account slippage function by Riemann integrating the function g' according to Lebesgue's Theorem. The marginal slippage function is defined to be:

$$g'(c) = \begin{cases} -1 & \text{for } -\frac{xz}{c^{z+1}} < -1, \\ -\frac{xz}{c^{z+1}} & \text{for } -\frac{xz}{c^{z+1}} \in [-1, 0]. \end{cases}$$
$$g(c) = \int g'(c)\,dc = \begin{cases} -c + T_1 & \text{for } -\frac{xz}{c^{z+1}} < -1, \\ \frac{x}{c^z} + T_2 & \text{for } -\frac{xz}{c^{z+1}} \in [-1, 0]. \end{cases}$$

where x and z are fixed parameters to be specified. $x = 0.00002$ and $z = 7$ could be a competent choice of parameters.

The slippage function g is the reference point for slippage. In any swap, that account for token i involved, there will be an initial compensation ratio c_i and final compensation ratio c'_i. For token i, the account slippage is defined by a definite integral along the swap path S:

$$S_i := \frac{1}{c'_i - c_i}\int_S g'(c)\,dc = \frac{g(c'_i) - g(c_i)}{c'_i - c_i}$$

The account slippage is not the final slippage $S_{i\to j}$ that users would encounter when swapping token i to token j. The swap path should at least involve two tokens. Account slippage on token i is indemnity for the user, while account slippage on token j is a penalty for the user since $c'_i > c_i$ and $c'_j < c_j$. As a result, we define the swap slippage $S_{i\to j}$ to be

$$S_{i\to j} = S_i + (-S_j) = S_i - S_j.$$

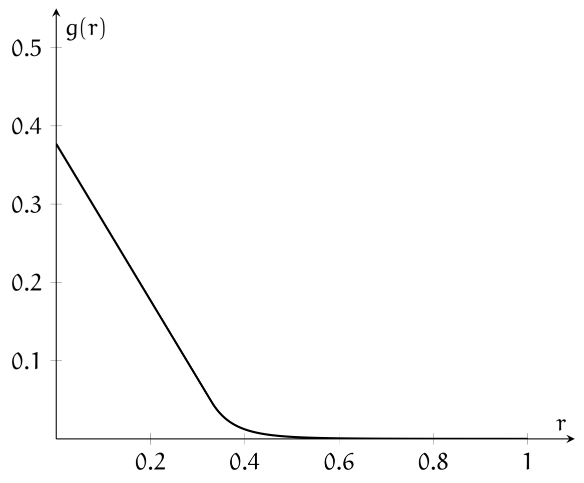

Figure 3: Slippage function for $x = 0.00002$ and $c = 7$

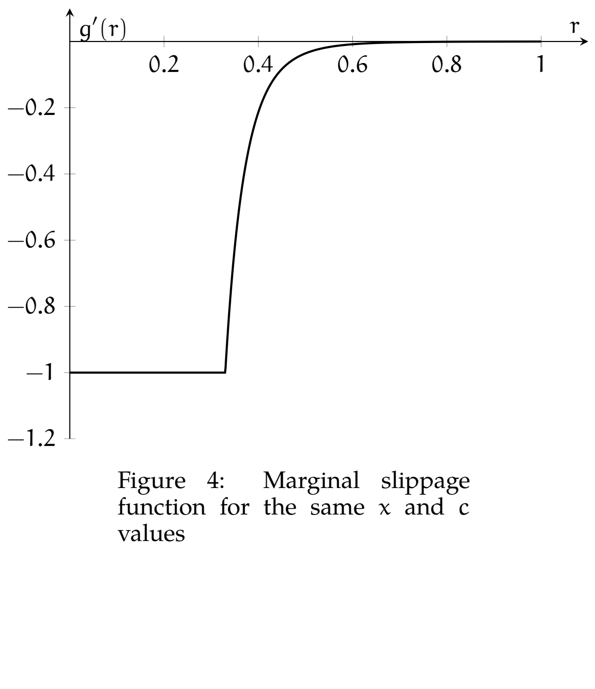

Figure 4: Marginal slippage function for the same x and c values

3.4 Cross-chain Asset Aggregation Algorithm

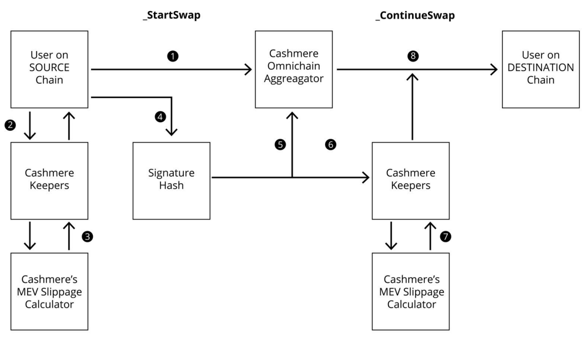
_StartSwap
_ContinueSwap
User on SOURCE Chain
Cashmere Omnichain Aggreagator
User on DESTINATION Chain
Cashmere Keepers
Signature Hash
Cashmere Keepers
Cashmere's MEV Slippage Calculator
Cashmere's MEV Slippage Calculator
1
2
3
4
5
6
7
8

Figure 5: Workflow of the aggregation algorithm with MEV protection on step 3 and 7

Source chain	Polygon
Destination chain	Arbitrum
Source Token	SRC
Destination token	TRG
Lowest compensation ratio stablecoin in Cashmere Pools	LWS
Highest compensation ratio stablecoin in Cashmere Pools	HGS

1. The user determines which asset he wants to trade from which network to which network (SRC is on the source chain, TRG on the destination chain).
2. Cashmere Keepers get swap parameters from 1inch, ODOS API to swap SRC to LWS on the source chain.
3. According to the incoming swap parameters, the optimal slippage value that will prevent the MEV attack is again queried by the Cashmere Keepers on the source chain.
4. The usersigns an on-chain signature containing "SRC, TRG, source chain, destination chain, minimum received amount, LWS, HGS".
5. `CashmereAggregatorRouter` executes `_startSwap` according to the swap parameters contained in the on-chain signature.
   5.1. `CashmereAggregatorRouter` swaps SRC to LWS on Source Chain via 1inch, ODOS swap parameters.
   5.2. `CashmereAggregatorRouter` gets LWS.
   5.3. LWS swaps to HGS from SourceChain to DestinationChain via CashmerePools.
   5.4. The interoperability messaging protocol starts here, LWS locked on the CashmerePool on the Source Chain by `CashmereAggregatorRouter` the cross-chain message sent to the destination chain.
   5.5. Marginal slippage is applied when moving from LWS on the Source Chain to HGS in the Destination Chain. It can be positive or negative.
6. `CashmereAggregatorRouter` gets equal HGS on the Destination Chain.
   6.1. Cashmere Keepers get swap parameters from 1inch, ODOS API for HGS to TRG on the destination chain.
7. According to the incoming swap parameters, the optimal slippage value that will prevent the MEV attack is again queried by the Cashmere Keepers on the destination chain.
8. Cashmere Keepers on the Destination Chain compare the signature signed by the user on the source chain with the transaction from the contract. If it matches correctly, Cashmere Keepers allow the continuation of the transaction.
   8.1. Cashmere Keepers execute _swap HGS to TRG transaction via `CashmereAggregatorRouter` on the destination chain.
   8.2. `CashmereAggregatorRouter` delivers TRG assets to the user.

In this way,

- Cashmere Keepers cannot act against the user.
- Cashmere Keepers cannot harm the user's assets.
- Cashmere Keepers cannot make a query other than the amounts requested by the user.
- Cashmere Keepers cannot query other than the entities requested by the user.
- Protects from front-run as the user also signs `_minimumTokenReturn` on-chain.
- Cashmere makes the swap process completely decentralized & permissionless.
- Cashmere provides a high-quality user experience, and the smart contract will execute the transaction quickly, in 30-40 seconds and with one click.

3.5 Discussion

As we proved in the previous section, Cashmere stable swap and cross-chain aggregator algorithms using weighted liquidity, marginal slippage rate, MEV-protected slippage tool, and on-chain signatures. Cashmere only processes native assets, which maximizes security. Based on our experience and knowledge, no existing cross-chain aggregator can provide this functionality, as all existing cross-chain aggregators fundamentally use centralized backends & market makers. Unlike other cross-chain aggregators, Cashmere provides a complete solution to all the problems described in the previous sections.

4 Example

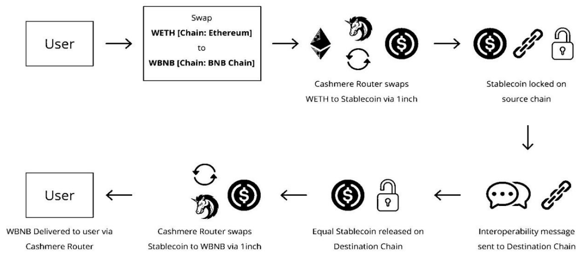
User
Swap
WETH [Chain: Ethereum]
to
WBNB [Chain: BNB Chain]
Cashmere Router swaps WETH to Stablecoin via 1inch
Stablecoin locked on source chain
Interoperability message sent to Destination Chain
Equal Stablecoin released on Destination Chain
Cashmere Router swaps Stablecoin to WBNB via 1inch
User
WBNB Delivered to user via Cashmere Router

Figure 6: Example showing a swap operation from Ethereum to BSC

5 Conclusion

In this study, we present the Cashmere architecture, which maintains complete decentralization, protects users from MEV front-run attacks, can swap any asset between any chain, and offers the lowest slippage thanks to its unique marginal slippage algorithm. The algorithm is presented with a formalization. Cashmere architecture represents a new evolution in the cross-chain aggregator's ecosystem, delivering three advantages that no other solution offers: no existing cross-chain aggregators use 100% native assets, no cross-chain aggregator can provide completely decentralized MEV protection, there are no cross-chain aggregator supports all tokens in the whole crypto-market. The Cashmere algorithm enables cross-chain aggregators to address these deficiencies, delivering unmatched adaptability and convenience. The efficiency provided by the cashmere algorithm increases the scalability of De-Fi interest and volume in cross-chain asset swaps, attracting more users with efficiency and creating opportunities to integrate more with other De-Fiapplications. We envisage the Cashmere algorithm in a new level of cross-chain aggregator that uses fast, completely permissionless interoperability.

References

[1] Angeris, Guillermo and Chitra, Tarun, "Improved price oracles: Constant function market makers," *SSRN Electronic Journal*, 2020. DOI: http://dx.doi.org/10.2139/ssrn.3636514.

[2] C. Team, "Curve dao whitepaper," Tech. Rep., 2020.

[3] M. Egorov, "Stableswap - efficient mechanism for stablecoin liquidity," Tech. Rep., 2019.

[4] H. Adams, *Uniswap birthday blog – v0*, 2019.

[5] MStable, *Mstable gitbook*, 2021.

[6] A. Othman, D. M. Pennock, D. M. Reeves, and T. W. Sandholm, "A practical liquidity-sensitive automated market maker," *ACM Transactions on Economics and Computation*, vol. 1, no. 3, pp. 1–25, 2013. DOI: https://doi.org/10.1145/2509413.2509414.

[7] R. Zarick, B. Pellegrino, and C. Banister, "Layerzero: Trustless omnichain interoperability protocol," 2021. DOI: https://doi.org/10.48550/arXiv.2110.13871. arXiv: 2110.13871 [cs].

[8] P. Team, *Platypus amm technical specification*, 2022.