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 cashmere's 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 cashmere's 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.

2.2.1 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.

2.2.2 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:	4174566336607022500000
Reserve Out:	4775084509488857400000000
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.

2.2.3 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.

2.2.4 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.

```
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

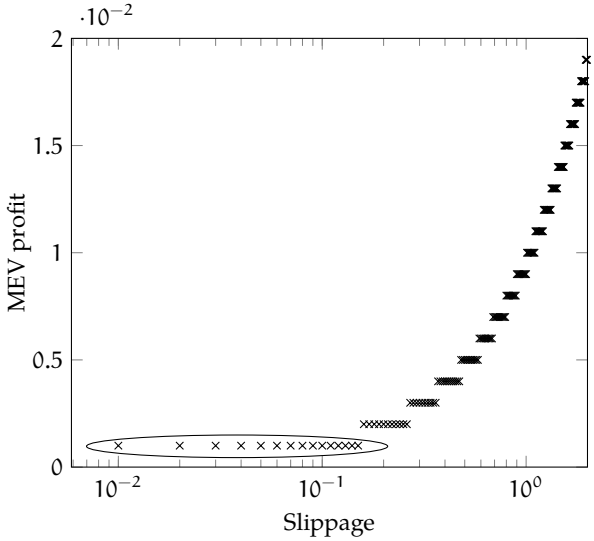

Figure 2: Existence of a safe zone against the MEV attacker

2.3 Efficient cashmere's 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 cashmere's 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, cashmere's 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

In the existing landscape of Decentralized Finance (DeFi), prevailing service protocols typically cater to a single type of asset class, such as stablecoins or native tokens. This approach falls short of fully addressing users' cross-chain interoperability needs. Distinctively diverging from these protocols, Cashmere offers support for an all-encompassing array of tokens prevalent in the industry. It leverages any decentralized liquidity available in the DeFi market, fortified by protection against MEV attacks. This unique capability significantly bolsters Cashmere's scalability, positioning it as a robust solution for users seeking to navigate cross-chain migrations.

3 Cashmere's AMM Algorithm

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. **Bandwidth** ($B_{s,d}$): Funds are allocated locally for transfers from S to D.
5. **Known bandwidth proof** (kbp): Last known amount of bandwidth that can swap from source to destination.
6. **Voucher** (v): Amount that will send to the destination chains in the next transfer.
7. **Optimal Bandwidth** (**OP**):

$$\mathrm{LP}(d) \times \frac{\mathrm{weight}(d)}{\mathrm{TotalWeight}(d)}$$

8. **Chain Route**: It is a defined liquidity path from any pool on any chain to any pool on any chain.
9. **Chain Route Weight** (w): Weight is defined from any pool on any chain to any pool on any chain. It determines the distribution percentages of liquidities.
10. **Outbound transaction** (O_t): The amount of transfer from the source chain route to the destination chain route.
11. **Inbound transaction** (I_t) : The amount of swap from the destination chain route to the source chain route.
12. **Stability Bandwidth** (s):

$$\frac{\text{Total Liquidity} \times \frac{\text{Chain Route Weight}}{\text{Total Weight}} + \frac{\text{Chain Route Weight}}{\text{Total Weight}} (\sum O_t - \sum I_t)}{\text{Total Liquidity} \times \frac{\text{Chain Route Weight}}{\text{Total Weight}}}$$

13. **User slippage** (U_s): It is the slippage value that the user is exposed to while swapping.

3.2 Innovation of Cashmere's AMM Algorithm

In this study, we formally demonstrate the existence of a tradeoff between convergence speed and the degree of slippage experienced by the user. Leveraging this understanding, we formulate an optimization problem aimed at deriving the most efficacious formula corresponding to a specified safety level, or alternatively, the inverse. We employ Bayesian Modeling to incorporate parameters of the system that are imbued with uncertainty, thereby integrating this uncertainty directly into the final slippage formula.

To optimize the objective, we utilize numerical integration, specifically numerical quadrature, in conjunction with optimization techniques, in particular,

the Broyden–Fletcher–Goldfarb–Shanno algorithm. The resultant formula not only ensures expeditious computation and minimal storage overhead but also retains interpretability, making it optimally suited for practical implementation.

3.3 Convergence-Slippage Tradeoff

Within the system under consideration, we identify a fundamental tradeoff between the speed of convergence and the magnitude of slippage that a user experiences. This inherent tradeoff stipulates that it is unfeasible to derive a slippage function that simultaneously optimizes for maximum convergence speed and minimum user-incurred slippage.

Instead, we are limited to the development of a function that is capable of maximizing convergence speed, contingent upon a specified level of user-incurred slippage, or alternatively, one that minimizes user-incurred slippage in the context of a predetermined convergence speed. Consequently, it is this inescapable tradeoff which fundamentally shapes the potential to optimize within the system parameters.

3.3.1 Proof

To incentivize convergence of stability bandwidths, we need to have $U_s < 0$ if $s' \in (s)$. The magnitude of $U_s < 0$ determines the incentive level. Note that U_s also is the user-incurred slippage by definition. Any slippage function should be path-independent and symmetric. This is the only restriction that we impose on the slippage function in the following proof.

Imagine that we have an arbitrary slippage function which maximizes the incentive level for convergence and therefore has $U_s = -\alpha < 0$ for a given swap. Moreover, define $\tilde{U}_s = 0$ as the trivial slippage function with no slippage for any swaps. By applying the symmetry of the arbitrary slippage function (and the obvious symmetry of the trivial slippage function), we get:

$$U_s = -\alpha < 0 \iff U_s = \alpha > 0 \iff U_s > \tilde{U}_s$$

We have therefore shown that the user-incurred slippage of U_s is higher than the one of another valid slippage function, meaning U_s cannot minimize the user-incurred slippage.

3.4 Optimization Objective

Given the results from Section 3.3, our optimization objective is to find the slippage function that minimizes the user-incurred slippage, given a desired convergence incentive level (or maximize the convergence incentive level, given a user-incurred slippage, which is equivalent).

We first note that the convergence incentive level should depend on the current stability bandwidth. If the stability bandwidth is close to 1, our main objective should be minimizing slippage. On the other hand, if the stability bandwidth reaches a critical level, we want to incentivize convergence. Therefore, we introduce the parameter $\lambda \in (0, 1)$ to denote the maximum stability bandwidth deviation (from 1) the protocol is willing to accept. Given this parameter, we want to solve the following optimization problem:

$$\max_{y \in \mathcal{F}} \int_0^{1-\lambda} y(s)ds$$

$$\min_{y \in \mathcal{F}} \int_{1-\lambda}^{1} y(s)ds$$

Which can be simplified to one maximization problem:

$$\max_{y \in \mathcal{F}} \left(\int_0^{1-\lambda} y(s)ds - \beta \int_{1-\lambda}^{1} y(s)ds \right)$$

β represents the aggressiveness of the protocol. The higher we set it, the more importance is set on minimizing user-incurred slippage. Because of section 3.3, we therefore also know that the convergence will be slower.

We have considered an arbitrary function space $\mathcal{F}$ so far. However, in practice we want to have a function $y \in C^1$, i.e. a continously differentiable function. Moreover, $y'(s) \geq -1$ should hold. For $y'(s) < -1$, we can end up in situations where swapping more tokens reduces the tokens received for some target stability bandwidths. We want to avoid this, as no rational user would perform such swaps and the slippage function would therefore rule out some stability bandwidths completely.

3.4.1 λ

So far we assumed that there is one fixed, safe λ that has to be chosen. However, this is hardly the case in practice. By running simulations and with our domain knowledge, we can get an idea of what a reasonable range for the parameter looks like, but there is no single true value. We incorporate this fact with Bayesian modeling. Instead of assuming one single value, we assume that λ follows a certain distribution. A reasonable assumption is that it is normally distributed around some mid-point $\tilde{\lambda}$ and with some variance σ^2, i.e. we have $\lambda \sim \mathcal{N}(\tilde{\lambda}, \sigma^2)$.

Now, our new objective is to maximize the expected value over λ, i.e.:

$$\max_{y \in C^1} \left(\mathbb{E}_\lambda \left(\int_0^{1-\lambda} y(s)ds - \beta \int_{1-\lambda}^{1} y(s)ds \right) \right)$$

$$\max_{y \in C^1} \left(\int_0^1 \left(\left(\int_0^{1-\lambda} y(s)ds - \beta \int_{1-\lambda}^{1} y(s)ds \right) \frac{e^{-\frac{1}{2}\left(\frac{\lambda-\tilde{\lambda}}{\sigma}\right)^2}}{\sigma\sqrt{2\pi}} \right) d\lambda \right)$$

This is generally an untractable optimization problem, so we have to restrict the function space to make it tractable. To do so, we use the conjugate prior of the normal distribution for $y(s)$ and parametrize it as e^{-ar}. Because of the condition, $y'(s) \geq -1$ (which does not necessarily hold on $(0, 1)$ for e^{-ar}), we define the function piecewise:

$$y(s) = \begin{cases} e^{-a\frac{\log(a)-\log(1)}{a}} + \frac{\log(a)-\log(1)}{a} - r \\ = \frac{1}{a} + \frac{\log(a)-\log(1)}{a} - r & \text{if } r \leq \frac{\log(a)-\log(1)}{a} \\ e^{-ar} & \text{if } r > \frac{\log(a)-\log(1)}{a} \end{cases}$$

The parameter a (given $\tilde{\lambda}, \sigma, \beta$) that maximizes the optimization problem above can be found by numerical integration (for the inner integrals) and numerical optimization techniques such as the Broyden–Fletcher–Goldfarb–Shanno algorithm which was used in our simulations. For instance, we have $a = 21$ for $\tilde{\lambda} = 0.5$, $\sigma = 0.1$, and $\beta = 3$.

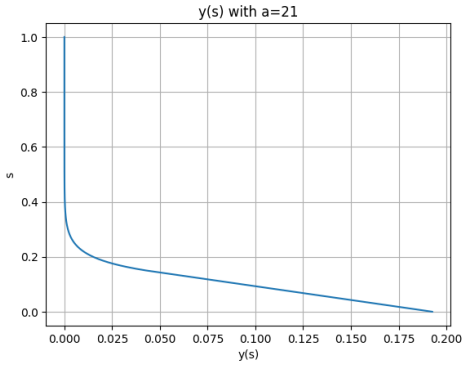

Figure 3: $y(s)$ with $a = 21$.

3.5 Cashmere's AMM Efficiency

In the decentralized finance (DeFi) sector, many Automated Market Maker (AMM) algorithms do not optimize liquidity utilization, nor do they provide cross-chain integration. These AMM models often incur substantial slippage at relatively early stages of stability bandwidth.

In contrast, CashmereLabs deploys an innovative AMM algorithm that delivers fluid and minimally invasive slippage trade-offs up to advanced stability bandwidth of 25%. This performance attribute can be attributed to CashmereLabs' well-structured liquidity architecture, which is bolstered by exclusively supporting native assets through cross-chain messaging protocols.

Furthermore, the team at CashmereLabs has identified a critical vulnerability within numerous Automated Market Maker (AMM) algorithms. In situations where market conditions lead to constrained liquidity, the majority of AMM algorithms become non-functional and are unable to fulfill their purpose due to the inefficiency of their curve functions as stability bandwidth approach zero.

CashmereLabs, however, has devised a strategy to circumvent this systemic issue. CashmereLabs has optimized its liquidity pools to ensure that they remain operational even when stability bandwidth trend toward zero. This optimization ensures a level of resilience and reliability in market conditions that can challenge the functionality of conventional AMM algorithms. Thus, CashmereLabs presents a notable advancement in the field, offering a robust solution that significantly improves upon the vulnerabilities detected in prevalent AMM models.

Undeniably, CashmereLabs stands at the forefront of applications that facilitate both cross-chain and intra-chain swaps. It offers the most favorable slippage rates in the industry, as evidenced by our comprehensive study. The discovery of the slippage curve further substantiates this claim, marking a pivotal breakthrough in our understanding of digital asset exchange dynamics.

CashmereLabs operates as a chain-agnostic entity and engages in cross-chain activities. Due to the inherent logic of smart contracts, not all pools across various chains are consistently cognizant. In response to this, CashmereLabs has innovated a unique slippage formula. This formula is exclusively dependent on the pool in the originating chain, thereby eliminating the necessity of a second pool parameter.

This specially developed slippage formula provides robust protection against potential security threats arising from on-chain disruptions. It bolsters security by only requiring parameters from the source chain pool with which interaction has taken place in order to compute slippage. This computational model negates the need for information from additional pools, effectively decreasing potential vulnerabilities.

Owing to its unique architectural design, CashmereLabs has positioned itself as a paragon of reliability within the Automated Market Maker (AMM) sector of the omnichain industry. Its level of slippage, a key metric of performance, is notably lower when juxtaposed with its industry counterparts. As such, CashmereLabs represents a compelling advancement in the realm of omnichain technologies.

This unique approach allows CashmereLabs to heighten liquidity efficiency dramatically, achieving near-zero slippage during cross-chain transactions. The principal objective of CashmereLabs is to orchestrate the most efficient resource allocation within the omnichain ecosystem, showcasing a model that is both dynamic and highly adaptable.

4 Liquidity Distribution Algorithm

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

The algorithm always tries to maintain this bandwidth

$$0 \leq W_{s,d} \leq 1$$
$$\sum sW_{s,x} = 1.$$

(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:

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

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.

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

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):

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

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** $t \leq \text{Total}$,
   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.

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

5 Cross-chain Asset Aggregation Algorithm

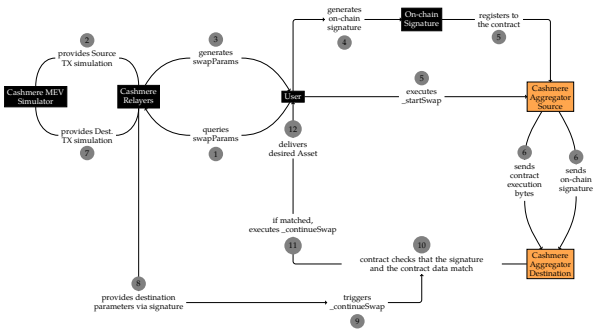

Figure 4: Workflow of the aggregation algorithm with MEV protection on step 2 and 7

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

1. The user initiates the process by selecting the SRC and TRG tokens for a swap between the originating and destination blockchains. In response, Cashmere relayers query the swap parameters for the transaction.

2. MEV simulator subsequently executes a frontrunning mitigation simulation for the originating blockchain.

3. The queried swap parameters are relayed back to the user for verification.

4. As a safeguard for user security, an on-chain digital signature is produced, embedding information such as the originating token, destination token, minimum acceptable return, slippage, and the amount intended for swap. The user is thereby empowered with the ability to transparently scrutinize the embedded data within the signature, ensuring their protection.

5. This generated digital signature is then registered within the source Cashmere Aggregator Source contract, and the commencement of the swap transaction is triggered. The SRC token is swapped to LWS and prepared for cross-chain transmission.

6. The source aggregator contract data, along with the on-chain signature, are transmitted to the destination blockchain utilizing the Layerzero cross-chain messaging infrastructure. The LWS token will be obtained as HGS in the destination chain for users to achieve higher output and positive slippage.

7. On the destination blockchain, the swap parameters and the necessary MEV simulation are queried once more via Cashmere relayers to ensure that the swap parameters remain current, thereby minimizing and circumventing potential swap reversions.

8. To guarantee user security and maintain decentralized operation of relayers, the parameters embedded within the on-chain signature and those provided by the destination blockchain's relayers must align. The on-chain signature additionally ensures the infallibility of relayers.

9. The relayers are then responsible for triggering the continueSwap transaction, which effects the destination token swap.

10. The Cashmere Aggregator Destination contract cross-validates the on-chain signature with the contract data. Upon a match, it validates the correctness of the data furnished by the Cashmere relayers, thereby permitting the transaction to proceed. In this case HGS will be swapped to TRG token.

11-12. The user finally receives the TRG token on the destination chain, which is safeguarded with MEV protection. The desired destination chain token is consequently delivered to the user, completing the process.

In this way,

- Cashmere Relayers cannot act against the user.
- Cashmere Relayers cannot harm the user's assets.
- Cashmere Relayers cannot make a query other than the amounts requested by the user.
- Cashmere Relayers 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 with a very low rate of swap reversion.
- Cashmere provides a high-quality user experience, and the smart contract will execute the transaction quickly, in 30–40 seconds and with one click.

5.1 Discussion

As we proved in the previous section, Cashmere stable swap and cross-chain aggregator algorithms using weighted liquidity, cashmere's 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.

6 Example

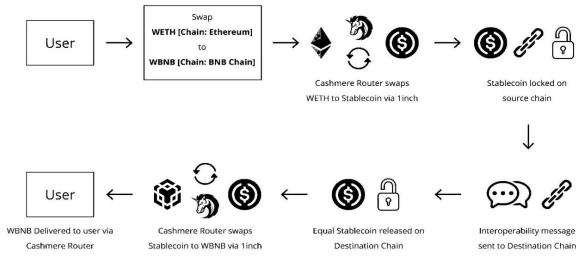

Figure 5: Example showing a swap operation from Ethereum to BSC

7 Conclusion

In this research, we introduce the revolutionary Cashmere architecture——an innovative system that guarantees absolute decentralization, safeguards users from MEV front-run attacks, and facilitates the swapping of any asset across any blockchain. Uniquely, Cashmere ensures minimal slippage through the application of a proprietary cashmere's slippage algorithm, which we explicate in detail within this paper.

The Cashmere architecture signifies a pivotal advancement in the domain of cross-chain aggregators, delivering six unparalleled benefits that are absent in existing solutions; MEV Protection: Cashmere outperforms competing cross-chain swap applications by providing superior swap output due to its robust protection against MEV. Real-Time Parameter Query: With the capacity to instantaneously and continuously query swap parameters in both the source and destination chains, Cashmere dramatically reduces the swap revert rate. This optimization results in significant gas savings and an improved user experience. Optimal Swap Routes: In stark contrast to other industry players who struggle to identify optimal swap routes for a majority of user requests, Cashmere can efficiently pinpoint swap routes, even for high-value transactions, by leveraging its ability to query swap parameters in real-time. On-Chain Signature: Cashmere has pioneered a fully decentralized and permissionless cross-chain swap process via on-chain signatures. This mechanism safeguards users from potential attacks and guarantees that the swap process can always be executed by relayers. Native Asset Utilization: Unlike existing cross-chain aggregators, Cashmere uniquely employs 100% trustworthy native assets. Universal Token Support: Cashmere stands alone in its capacity to support all tokens across the entire cryptocurrency market.

The Cashmere algorithm presents a solution to the prevalent deficiencies in existing cross-chain aggregators, offering unmatched adaptability and convenience. It enhances the scalability of Decentralized Finance (DeFi) interest and volume in cross-chain asset swaps, thereby attracting a larger user base through increased efficiency. Furthermore, it opens up avenues for further integration with other DeFi applications. With its rapid, entirely permissionless interoperability, the Cashmere algorithm is envisaged to elevate the cross-chain aggregator to an unprecedented level.

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.