

# The Walls Have Ears: Unveiling Cross-Chain Sandwich Attacks in DeFi

Chuanlei Li<sup>†</sup>, Zhicheng Sun<sup>†</sup>, Jing Xin Yuu<sup>†</sup>, Xuechao Wang<sup>†</sup>

<sup>†</sup>*The Hong Kong University of Science and Technology (Guangzhou)*

Email: {cli839,zsun277,jxyuu971}@connect.hkust-gz.edu.cn, xuechaowang@hkust-gz.edu.cn

**Abstract**—Cross-chain interoperability is a core component of modern blockchain infrastructure, enabling seamless asset transfers and composable applications across multiple blockchain ecosystems. However, the transparency of cross-chain messages can inadvertently expose sensitive transaction information, creating opportunities for adversaries to exploit value through manipulation or front-running strategies.

In this work, we investigate cross-chain sandwich attacks targeting liquidity pool-based cross-chain bridge protocols. We uncover a critical vulnerability where attackers can exploit events emitted on the source chain to learn transaction details on the destination chain before they appear in the destination chain mempool. This information advantage allows attackers to strategically place front-running and back-running transactions, ensuring that their front-running transactions always precede those of existing MEV bots monitoring the mempool of the destination chain. Moreover, current sandwich-attack defenses are ineffective against this new cross-chain variant. To quantify this threat, we conduct an empirical study using two months (August 10 to October 10, 2025) of cross-chain transaction data from the Symbiosis protocol and a tailored heuristic detection model. Our analysis identifies attacks that collectively garnered over 5.27 million USD in profit, equivalent to 1.28% of the total bridged volume.

## 1. Introduction

Decentralized Finance (DeFi) has emerged as one of the most transformative innovations in the blockchain ecosystem, fundamentally reshaping financial intermediation. The DeFi market has experienced exponential growth, with the total value locked (TVL) across protocols exceeding 145 billion USD as of November 3, 2025 [1]. At the heart of this ecosystem are decentralized exchanges (DEXs), which enable users to trade digital assets directly on-chain without intermediaries. Unlike traditional order-book exchanges, DEXs rely on Automated Market-Making (AMM) mechanisms, where smart contracts autonomously determine exchange rates based on liquidity pool reserves [2]. Every trade, price update, and liquidity adjustment is immutably recorded on the blockchain, ensuring transparency, verifiability, and trustless execution for all participants.

In traditional financial markets, access to transaction information has long been monetized. For example, insti-

tutional traders engage in high-frequency trading by minimizing latency to profit from microsecond-level price movements. A similar phenomenon has emerged in DeFi, where participants exploit informational advantages embedded in pending or on-chain transactions to extract profit, a practice broadly known as Maximum Extractable Value (MEV) [3]. Various MEV strategies have been proposed and observed. Common MEV strategies include liquidation, arbitrage, and sandwich attacks. In liquidation, an actor monitors DeFi lending protocols and profitably liquidates positions approaching their collateral thresholds [4]. Arbitrage opportunities arise when price discrepancies exist across exchanges or liquidity pools, allowing traders to buy low and sell high, restoring price equilibrium. Among these, sandwich attacks [5] are particularly harmful: an attacker observes a victim's pending transaction in the mempool and strategically places one transaction before it (front-running) and another after it (back-running). By manipulating transaction ordering via gas fees, the attacker inflates the asset price before the victim's swap and reverses the trade afterward, extracting a guaranteed profit at the victim's expense.

Several defense mechanisms have been proposed to mitigate sandwich attacks. Proposer-BUILDER Separation (PBS) mitigates the concentration of MEV by decoupling block construction from block proposal: competitive builders assemble candidate blocks, while validators merely propose them to the network [6], [7]. Another line of defense involves the encrypted mempool [8], [9], [10], where transactions remain encrypted during block construction. Miners or validators build blocks using ciphertexts, and a designated committee later decrypts the included transactions once the block is finalized, thereby preventing pre-execution visibility and front-running. Furthermore, several works introduce fair transaction ordering at the consensus layer [11], [12], ensuring that users' transactions are ordered according to predefined fairness principles rather than miners' discretion, thereby reducing opportunities for manipulation.

While front-running and sandwich attacks have been extensively examined in single-chain DEXs, the rise of cross-chain interoperability introduces a new dimension to these threats. As the DeFi ecosystem expands beyond individual blockchains, users increasingly engage in asset exchanges across multiple chains, relying on cross-chain bridges to facilitate these transfers [13], [14]. Such systems enable interoperability by transmitting messages or

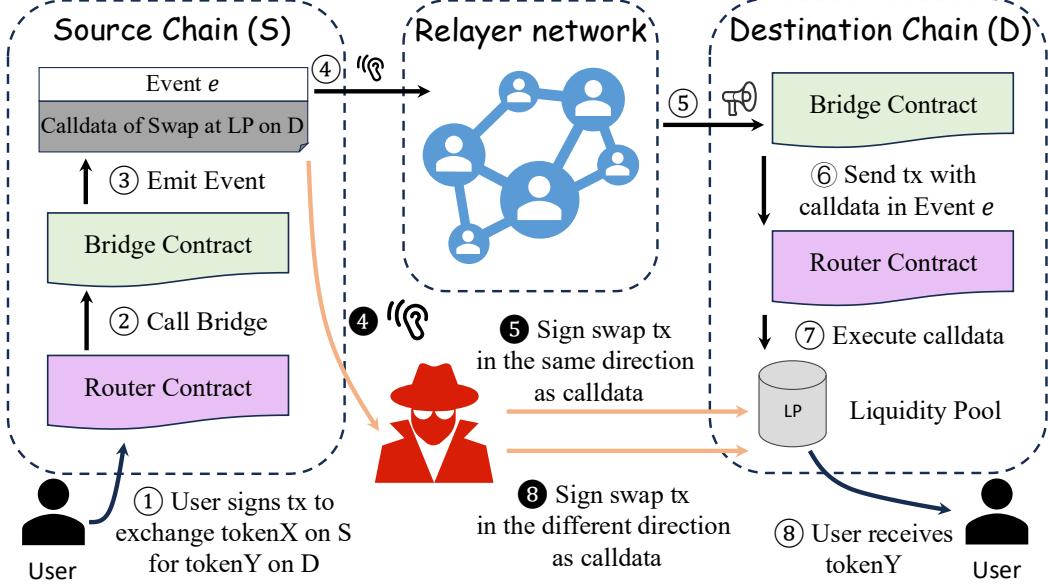


Figure 1. Overview of the sandwich attack on liquidity pool based cross-chain swaps. ①-⑧ represent normal cross-chain swap steps, while ④, ⑤, ⑥ are the attacker’s operations. Matching numbers indicate that the user and the attacker perform those concurrently.

tokens between heterogeneous blockchain networks, typically through trusted relayers or bridge operators responsible for observing and forwarding cross-chain events [15]. However, this process inherently exposes transaction metadata to the public: in EVM-based chains, relayers monitor events emitted by smart contracts, while in UTXO-based chains, they parse newly confirmed transaction outputs. Because these observations occur before the corresponding transactions appear in the destination chain’s mempool, adversaries monitoring the same on-chain events can gain early access to transaction details. This information advantage enables attackers to anticipate and exploit upcoming swaps on the destination chain, paving the way for cross-chain sandwich attacks that are invisible to existing MEV defenses

In this paper, we reveal how cross-chain sandwich attacks can be performed on liquidity pool-based bridge protocols. In these protocols, the source chain emits detailed cross-chain swap information, while the destination chain executes the corresponding token swap from a local liquidity pool and sends the output tokens to the user. As illustrated in Figure 1, an attacker continuously monitors specific events emitted by cross-chain protocols to learn about forthcoming cross-chain swap transactions on the destination chain. Once a profitable opportunity is detected, the attacker submits a front-running transaction to the corresponding liquidity pool on the destination chain and subsequently places a back-running transaction once the victim’s transaction appears on-chain or in the mempool, thereby completing the sandwich attack. Crucially, the attacker acquires the necessary information before the cross-chain transaction appears in any destination-chain context, giving them a temporal advantage over ordinary single-chain sandwich attackers. Moreover, this early access renders most previously proposed miti-

gations against single-chain sandwich attacks ineffective, as it does not need to wait for victim’s transaction to be visible in the destination chain’s mempool. We provide an example of a cross-chain sandwich attack that we identified on Ethereum Mainnet ([0x9d5f...d548](#), [0xef06...2d7a](#), [0xe16b...82c0](#), [0xc07d...2b8f](#), [0xe2a1...86de](#)), in which the cross-chain attacker achieved a significantly higher profit rate of 21.4%, compared to just 0.8% for existing MEV bots.

We begin by presenting a theoretical framework for analyzing cross-chain sandwich attacks. Since the attack dynamics depend on several stochastic parameters, we extract their empirical distributions from real-world data and verify that the observed values closely match our theoretical assumptions. Next, we examine historical cross-chain transaction data to identify concrete instances of such attacks. We collect two months of transaction records from the Symbiosis protocol [16] (August 10 to October 10, 2025) and apply our heuristic detection algorithm to systematically locate cross-chain sandwich behaviors. Our analysis reveals that these attacks generated a total profit of approximately 5.27 million USD, representing 1.28% of the total bridged volume during the observation period. By contrast, conventional single-chain sandwich attacks in the same dataset yielded only 6,109 USD in profit. Furthermore, we found that cross-chain sandwich attacks have the greatest impact on the BSC network. Among the transactions in our dataset, those targeting BSC as the destination chain generated the highest profits, with the BUSD↔WBAN pool being the most frequently attacked. Finally, we discuss potential countermeasures and outline design principles aimed at mitigating this emerging class of cross-chain threats.

**Contributions.** We summarize our contributions as follows.

- We identify a previously unrecognized cross-chain sandwich attack that exploits information advantage in cross-chain protocols. By accessing transaction details before they appear on the destination chain, the attacker can consistently outperform existing MEV bots in transaction-ordering competitions, effectively bypassing all known single-chain defenses.
- We conduct a large-scale empirical study using two months of cross-chain transaction data, revealing that attackers extracted approximately 5.27 million USD in profit through such attacks, equivalent to 1.28% of the total bridged volume.
- To foster further research, we construct and release the first large-scale dataset<sup>1</sup> of cross-chain sandwich attacks, providing an empirical foundation for future work on cross-chain MEV and security analysis.

## 2. Background

In the following, we provide relevant background on DEX, sandwich attack, and cross-chain interoperability.

### 2.1. Decentralized Exchange

Unlike centralized exchange (CEX), which requires users to deposit funds and trust the centralized entity to manage order books and execute trades, Decentralized Exchange (DEX) allows users to trade cryptocurrencies directly from their personal wallets without the need for an intermediary.

DEX employs Automated Market Maker (AMM) instead of traditional order books. In this model, liquidity providers (LPs) supply assets to liquidity pools that contain at least two tokens. The price of a token is then determined algorithmically based on the asset ratio in the pool, according to rules defined in an immutable and transparent smart contract. A classic AMM model is the Constant Product Market Maker (CPMM), used in Uniswap V2 [17]. It is also the model leveraged in Uniswap V3 when the price does not cross a tick, which is the case for almost all trades. For the reserves  $x$  and  $y$  in the liquidity pool, their quantities always satisfy the equation

$$x * y = k. \quad (1)$$

We consider that a user aims to swap  $\Delta x$  of token X to token Y in a liquidity pool with transaction fee  $f \in (0, 1)$ , the reserves follow this equation:

$$(x + (1 - f)\Delta x)(y - \Delta y) = x * y, \quad (2)$$

the user receives

$$\Delta y = y - \frac{x * y}{x + (1 - f)\Delta x} = \frac{y(1 - f)\Delta x}{x + (1 - f)\Delta x} = \frac{k(1 - f)\Delta x}{x^2 + x(1 - f)\Delta x}, \quad (3)$$

and

$$\frac{\partial \Delta y}{\partial x} = -\frac{k(2x + (1 - f)\Delta x)(1 - f)\Delta x}{(x^2 + x(1 - f)\Delta x)^2} < 0. \quad (4)$$

1. <https://zenodo.org/records/17562882>

So an increase in reserve  $x$  leads to a decrease in  $\Delta y$ . This implies that large transactions can cause price deviations, a phenomenon known as slippage. A slippage tolerance  $s \in (0, 1)$  represents the difference between the expected out  $\Delta y$  and actual execution output  $\Delta y'$  of a transaction, where  $\Delta y' \geq (1 - s)\Delta y$ . If the actual execution amount falls below the minimum acceptable value, the transaction will be reverted.

### 2.2. Sandwich Attack

The slippage tolerance creates an opportunity for predatory traders to perform their strategies to extract value. The attack exploits the public nature of the mempool, which contains unconfirmed transactions broadcast to the network. Attackers monitor the mempool to identify profitable pending transactions targeting specific liquidity pools. They then bid higher gas fees to get miners to include a same-direction transaction (front-running) before the victim's transaction. This front-running raises the token price, so the victim receives fewer tokens than expected. Next, the attacker submits an opposite-direction transaction (back-running) with a slightly lower gas fee, ensuring it is placed immediately after the victim's transaction. Finally, the attacker trades the overpriced token back and captures the profit.

Consider a pool with reserves  $(x_0, y_0)$  for tokens X and Y, and a transaction fee rate  $f \in (0, 1)$ . At time  $t_0$ , the attacker identifies a victim transaction  $T_v$ .  $T_v$  aims to exchange  $\Delta x_v$  X for Y with slippage tolerance  $s \in (0, 1)$ . If  $T_v$  is not attacked, it will return

$$\Delta y = \frac{y_0(1 - f)\Delta x}{x_0 + (1 - f)\Delta x}. \quad (5)$$

If an attacker is trying to attack  $T_v$  at  $t_0$ . Attacker executes a front-running transaction  $T_{A1}$  to input  $\Delta x_{A1}$  X in the pool, and it will return

$$\Delta y_{A1} = \frac{y_0(1 - f)\Delta x_{A1}}{x_0 + (1 - f)\Delta x_{A1}}. \quad (6)$$

After  $T_{A1}$ , the pool reserves update to  $x_{A1} = x_0 + \Delta x_{A1}$  and  $y_{A1} = y_0 - \Delta y_{A1} = \frac{x_0 y_0}{x_0 + (1 - f)\Delta x_{A1}}$ . Then  $T_v$  is executed, the output of Y should be

$$\Delta y_v = \frac{y_{A1}(1 - f)\Delta x_v}{x_{A1} + (1 - f)\Delta x_v}. \quad (7)$$

If  $\Delta y_v \geq (1 - s)\Delta y$ , the execution will be successful, otherwise it will be reverted so that the sandwich attack fails. According to the theorem proved by Heimbach et al. [18], the attacker's profit-maximizing optimal input amount  $\Delta x_{A1}$  is the value that satisfies  $\Delta y_v = (1 - s)\Delta y$ , i.e.,

$$\frac{\frac{x_0 y_0}{x_0 + (1 - f)\Delta x_{A1}}(1 - f)\Delta x_v}{x_0 + \Delta x_{A1} + (1 - f)\Delta x_v} = (1 - s) \frac{y_0(1 - f)\Delta x}{x_0 + (1 - f)\Delta x}. \quad (8)$$

To retrieve the profit, the attacker places the back-running transaction  $T_{A2}$  closely following  $T_v$ , it exchanges  $\Delta y_{A1}$  Y with pool reserves  $x_v = x_{A1} + \Delta x_v$  and  $y_v =$

$y_{A_1} - \Delta y_v = \frac{x_{A_1} y_{A_1}}{x_{A_1} + (1-f)\Delta x_v}$ . In transaction  $T_{A2}$ , attacker will receive

$$\Delta x_{A_2} = \frac{x_v(1-f)\Delta y_{A_1}}{y_v + (1-f)\Delta y_{A_1}}. \quad (9)$$

Let  $G_c$  be the total gas cost for  $T_{A1}$  and  $T_{A2}$ , the profit  $P$  of this sandwich attack is

$$P = \Delta x_{A_2} - \Delta x_{A_1} - G_c. \quad (10)$$

### 2.3. Cross-Chain Interoperability

Cross-chain interoperability refers to the ability of distinct blockchain networks to reliably transfer or exchange data and assets, enabling seamless interaction among otherwise isolated systems. It has become a foundational component of modern multi-chain ecosystems, allowing users and applications to move liquidity, coordinate state, and compose protocols across heterogeneous chains.

One early interoperability approach is the Lock-and-Mint model [19]: assets are locked on the source chain, and an equivalent amount of wrapped tokens is minted on the destination chain, ensuring the original assets cannot be double-spent. The reverse process, Burn-and-Unlock [20], burns the wrapped tokens on the destination chain and unlocks the corresponding assets on the source chain, keeping the total supply consistent across chains. Another widely studied interoperability mechanism is the cross-chain atomic swap, which guarantees that a cross-chain token swap between two parties either executes fully on both chains or does not occur at all. Atomicity is enforced through Hashed Time-Locked Contracts (HTLCs) [21], [22], which coordinate conditional execution across chains with cryptographic timeouts.

While these mechanisms support basic asset transfers, they fall short when applications require sending arbitrary data across chains. As cross-chain use cases expand, general-purpose cross-chain messaging protocols have become essential, offering greater flexibility and a significantly improved user experience. As illustrated in Figure 2, such protocols allow applications to transmit structured messages rather than only wrapped tokens, enabling richer interactions across heterogeneous chains. Several notable designs exist. Chainlink's Cross-Chain Interoperability Protocol (CCIP) [23] provides a generalized messaging framework built atop decentralized oracle networks and a risk-management layer. Cosmos IBC [24] instead uses on-chain light clients so that two blockchains can mutually verify each other's state through Merkle commitments, avoiding trusted intermediaries altogether. We define a cross-chain messaging protocol as follows:

**Definition 1** (Cross-Chain Messaging Protocol). *Source chain  $S$ , destination chain  $D$ , relayers  $\mathcal{R}$ , users  $\mathcal{U}$  and contract  $\mathcal{C}$  engage in this protocol to transfer messages across different chains. The contracts  $\mathcal{C}$  are deployed on multiple chains. The cross-chain messaging protocol (CCMP) has the following sub-protocols:*

$$CCMP = (\text{COMMIT}, \text{MONITOR}, \text{CONSENSUS}, \text{EXECUTE})$$

- $\text{COMMIT}(\mathcal{S}, \mathcal{C}, \mathcal{U}, \mathcal{R}) \rightarrow m : \mathcal{U}$  send a request to the contract  $\mathcal{C}$  on  $\mathcal{S}$ , including the data they want to transfer.  $\mathcal{C}$  generates the message  $m$  after encoding the data and emits  $m$  to relayers  $\mathcal{R}$ .
- $\text{VERIFY}(m, \mathcal{R}) \rightarrow (0, 1) :$ This protocol is executed by every relayer in  $\mathcal{R}$ . After receiving  $m$ , they verify its validity. If  $m$  is valid, it outputs 1.
- $\text{CONSENSUS}(m, \mathcal{R}, \mathcal{D}) \rightarrow l :$ All relayers participate in the consensus to determine the validity of  $m$ . If the majority considers  $m$  valid, a leader  $l$  is selected to perform EXECUTE protocol.
- $\text{EXECUTE}(l, m, \mathcal{D}, \mathcal{C}, \mathcal{U}) \rightarrow \text{out} :$ The leader  $l$  parses the message  $m$  and gets the data from  $\mathcal{U}$ . Then  $l$  calls  $\mathcal{C}$  to sign the transaction corresponding to the data. At last, user can verify the cross-chain message by checking the transaction execution result.

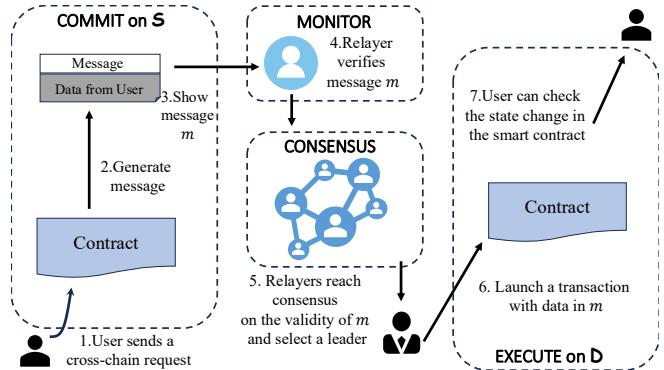


Figure 2. The workflow of the cross-chain messaging protocol.

To support decentralization, many cross-chain messaging protocols rely on events emitted by smart contracts to serve as the message  $m$ , which permissionless relayers then observe and forward. Because these events are publicly visible and cross-chain execution introduces inherent delays, adversaries can access sensitive transaction information well before it is executed on the destination chain. This early visibility creates a predictable window for MEV extraction. Among all known MEV strategies, sandwich attacks are particularly harmful because they directly degrade user execution quality through price manipulation. As the pioneering work to study MEV in cross-chain settings, we focus specifically on cross-chain sandwich attacks, which arise in messaging protocols that integrate liquidity-pool swaps on the destination chain. In these systems, attackers can observe the emitted event on the source chain and predict the exact swap that will soon occur on the destination chain, well before the transaction reaches the local mempool. This advance knowledge allows attackers to reliably construct profitable sandwich transactions, a behavior that we have observed in the wild.

TABLE 1. SUMMARY OF NOTATIONS

Symbol	Description
$\mathcal{A}$	Attacker that performs cross-chain sandwich attack
$\mathcal{B}$	Attacker that performs single-chain sandwich attack
$f$	Swap fee rate of the pool which is in $(0, 1)$
$T_s$	Start transaction on the source chain
$T_v$	Victim's transaction on the destination chain
$\pi$	Representing symbols $A1, A2, v, B1, B2$ or numbers $1 \dots n$
$s_\pi$	Slippage tolerance of $T_\pi$ which is in $(0, 1)$
$N_\pi$	Block number of transaction $T_\pi$
$T_{A1}$	Front-running transaction from $\mathcal{A}$
$T_{A2}$	Back-running transaction from $\mathcal{A}$
$T_{B1}$	Front-running transaction from $\mathcal{B}$
$T_{B2}$	Back-running transaction from $\mathcal{B}$
$(x_0, y_0)$	Reserves of token X and Y before $T_{A1}$
$(x_\pi, y_\pi)$	Reserves of token X and Y after $T_\pi$
$(\Delta x_\pi, \Delta y_\pi)$	Amounts of token X and Y exchanged in $T_\pi$
$p$	Probability that $\mathcal{A}$ does not incur a loss after the execution of $T_v$ with noisy transactions.
$q$	Probability that there is no noisy transactions between $T_{A1}$ and $T_v$
$\theta$	Percentage that $\mathcal{A}$ expects to extract from $T_v$

Prominent cross-chain messaging protocols with integrated liquidity-pool swaps include ThorSwap [25], deBridge [26], and Symbiosis [16], which have processed total cross-chain volumes of 114.15 billion, 17.5 billion, and 6.4 billion USD respectively. Such scale underscores the importance of analyzing and mitigating these emerging cross-chain threats.

### 3. Cross-Chain Sandwich Attack

In this section, we present a realistic model of cross-chain sandwich attacks. Figure 3 illustrates all possible transaction orderings, including the cases of no attack, single-chain sandwich attacks only, and cross-chain sandwich attacks, both in isolation and when coexisting with a single-chain sandwich attack. Table 1 summarizes all notation used throughout the analysis.

The attacker  $\mathcal{A}$  monitors the event on source chain in *CCMP* protocols. For each observed transaction,  $\mathcal{A}$  locally executes the computations in Section 2.2 as those used in a single-chain sandwich attack. We assume that if transaction  $T_v$  is profitable,  $\mathcal{A}$  will immediately place  $T_{A1}$ , and then send  $T_{A2}$  right after  $T_v$ .

#### 3.1. Noisy Transactions

It is worth noting that between the event being emitted on the source chain and the swap being executed on the destination chain, the message must be relayed by at least one intermediary (⑤⑥⑦ in Figure 1). During this time interval, numerous swaps in unknown directions may occur in the corresponding liquidity pool, which is reflected as  $T_1 \dots T_n$  in the figure. We call these swaps noisy transactions and denote  $q$  as the probability that no noisy transactions exist between  $T_{A1}$  and  $T_v$ . Noisy transactions may change

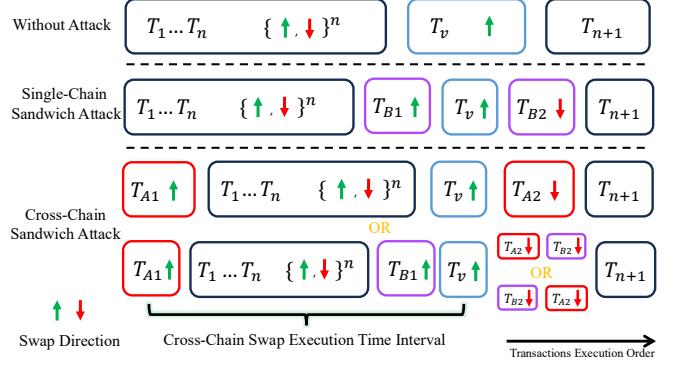


Figure 3. Visualization of possible transaction orderings.  $T_v$  is the victim transaction.  $T_{A1}$  and  $T_{A2}$  are cross-chain sandwich attack transactions.  $T_{B1}$  and  $T_{B2}$  are single-chain sandwich attack transactions.  $T_1 \dots T_n$  and  $T_{n+1}$  are other noisy transactions with unknown swap directions.

the pool's state to one that does not meet  $\mathcal{A}$ 's expectations, thereby directly affecting the potential profit of its cross-chain sandwich attack on  $T_v$ . As an extreme case, if a large-volume transaction in noisy transactions occurs in the direction opposite to  $T_{A1}$ , then even if  $T_v$  executes successfully,  $\mathcal{A}$  can only swap back fewer tokens than  $\Delta x_{A1}$ , leading to a loss in its sandwich attack.

As it is impossible to acquire or predict any information about noisy transactions at the time when the event is observed, we regard noisy transactions as a black-box transformation  $f : (x_{A1}, y_{A1}) \rightarrow (x_n, y_n)$ , where the input and output represent the pool states after  $T_{A1}$  and  $T_n$ . If  $T_v$  is exchanging X for Y, we denote  $p$  as the probability that  $(x_n + \Delta x_v, y_n - \Delta y_v)$  can ensure that  $\mathcal{A}$  does not incur a loss ( $\Delta x_{A2} \geq \Delta x_{A1}$ ) when  $T_1 \dots T_n$  are not empty, where  $\Delta x_{A2}$  is the output amount in the back-running trade  $T_{A2}$  and  $\Delta x_{A1}$  is the input amount in the front-running trade  $T_{A1}$ .

As  $p$  and  $q$  are derived from this empirical, black-box model, we estimate their values through large-scale historical data analysis rather than theoretical derivation. The resulting estimates are presented in the experiment Section 4.3.

In practical settings, leaving a small buffer for  $T_v$  is a reasonable strategy for  $\mathcal{A}$  to prevent noisy transactions from pushing  $T_v$  to its slippage limit and causing a transaction revert. Therefore, when  $T_v$  is about to be executed, there may still exist exploitable opportunities, allowing single-chain attackers  $\mathcal{B}$  to perform sandwich attacks on  $T_v$ . We denote by  $\theta \in (0, 1)$  the fraction extracted by  $\mathcal{A}$ . This leaves a buffer of  $(1 - \theta)s_v\Delta y$  in token Y, and the victim will expect to receive  $(1 - \theta)s_v\Delta y$  more than the baseline amount  $(1 - s_v)\Delta y$ . Now the condition in Equation 8 is tightened to

$$\Delta y_v = (1 - s_v)\Delta y + (1 - \theta)s_v\Delta y = (1 - s_v * \theta)\Delta y. \quad (11)$$

### 3.2. Profit of the Attack

A direct implication of such attack under our assumptions is that the maximum profit in token X of  $\mathcal{A}$  is  $s_v \Delta x_v$ . This value corresponds to the maximum tolerable loss of  $T_v$ , and it is independent of other transactions occurring between  $T_{A1}$  and  $T_v$ . This is because the minimum output amount of  $T_v$   $\text{minReturnAmount}$  is predefined in the emitted event, i.e., determined at the moment  $T_{A1}$  is placed. As long as  $T_v$  can be successfully executed, the corresponding token price in the pool at that time will not exceed the acceptable range of  $T_v$ , thus the profit in token X of  $T_{A1}$  can not exceed  $s_v \Delta x_v$ .

We now analyze the practical scenarios shown in Figure 3, assuming  $T_v$  is trying to exchange  $\Delta x_v$  X for Y. There are four possible scenarios for  $\mathcal{A}$ :

**No single-chain sandwich attack:** If there is no single-chain sandwich attack, the execution of a cross-chain sandwich attack is the same as that of a single-chain sandwich attack, where the transaction ordering is  $(T_{A1}, T_1 \dots T_n, T_v, T_{A2})$ . The maximum potential extracted value is  $s_v \Delta x_v$ .  $\mathcal{A}$  successfully attacks if  $s \Delta x_v > G_c + (1 - (1 - f)^2) \Delta x_{A1}$ . To show the expected profit, we assume that when the profit is positive, the average profit rate ( $\frac{\Delta x_{A2} - \Delta x_{A1}}{\Delta x_{A1}}$ ) is  $r^+$ , and when the profit is negative, the average profit rate is  $r^-$ . Considering the parameters  $p$  and  $q$  in noisy transactions, the expectation of profit is

$$\mathbb{E}(P) = \Delta x_{A1}((q + (1 - q)p)r^+ + (1 - q)(1 - p)r^-). \quad (12)$$

**$T_{A2}$  executed before  $T_{B2}$ :** Although  $\mathcal{A}$  is always able to place  $T_{A1}$  ahead of  $T_{B1}$ , it can not guarantee winning the race when inserting back-running transactions. If  $\mathcal{A}$  wins this race, the transaction ordering is  $(T_{A1}, T_1 \dots T_n, T_{B1}, T_v, T_{A2})$ . We can treat  $T_{B1}$  as one of  $T_1 \dots T_n$ , and the maximum potential extracted value and expectation of profit are the same as that in the situation with no single-chain sandwich attack.

**$T_{A2}$  executed after  $T_{B2}$ :** If  $\mathcal{A}$  loses the race, the transaction ordering is  $(T_{A1}, T_1 \dots T_n, T_{B1}, T_v, T_{B2}, T_{A2})$ . Because  $\mathcal{B}$  captures the buffer's surplus,  $\mathcal{A}$ 's maximum attainable profit at this point is  $r s_v \Delta x_v$ , and the expectation is the same as Equation 12.

**$T_{A1}$  is attacked by  $\mathcal{B}$ :** If  $\mathcal{A}$  is attacked by  $\mathcal{B}$ , the transaction ordering is  $(T_{B1}, T_{A1}, T_{B2})$ . In each of the three scenarios described above,  $\mathcal{A}$ 's profit decreases by at most  $s_A \Delta x_{A1}$  respectively. However,  $\mathcal{A}$  can mitigate the risk of the sandwich attack from  $\mathcal{B}$  through various methods, such as decreasing  $s_A$  or leveraging a private mempool. Thus, this scenario is regarded as having no impact on  $\mathcal{A}$ 's profit.

### 3.3. Against Current Countermeasures

Existing defenses against sandwich attacks are designed for single-chain settings and therefore cannot address attacks that exploit information leakage across chains. In all cases, their protection applies only after a transaction reaches the destination chain, whereas cross-chain sandwich attacks

arise before the transaction ever appears in the destination-chain mempool or block-construction pipeline.

Proposer Builder Separation (PBS) [6] restricts the proposer's ability to reorder transactions after seeing their plain-text contents. It ensures that once builders submit sealed blocks, the proposer cannot manipulate the intra-block ordering for MEV extraction. Fair ordering consensus [27], [28], [11] enforce partial transaction-ordering guarantees at the consensus layer by ensuring that transactions are ordered roughly in the sequence they are observed by the network. Both mechanisms only regulate the ordering of transactions that are already visible to the destination-chain nodes. Since a cross-chain victim transaction is predicted from source-chain events before it ever enters the destination chain's network, these guarantees provide no protection.

Similarly, private or encrypted mempools [29], [8], [9] hide pending transactions after they reach the destination chain. However, in liquidity-pool-based cross-chain bridges, the essential details of the victim transaction are included in source-chain events that must be publicly emitted for relayers to function. Since the leak occurs before any mempool logic applies, hiding mempool contents does not stop the attack.

## 4. Empirical Analysis

In the following, we investigate the scale of such cross-chain sandwich attack by analyzing historical transaction data from a cross-chain protocol in order to assess its feasibility and potential impact on the DeFi market.

### 4.1. Detection Algorithm

Our cross-chain sandwich attack detection algorithm is inspired by the method to identify potential single-chain sandwich attacks provided by Qin et al. [3]. We employ the following heuristics to detect historical cross-chain sandwich attacks. The pseudocode is provided in Algorithm 1.

- **Direction:** Both  $T_{A1}$  and  $T_v$  exchange token X for token Y, while  $T_{A2}$  exchange token Y for token X.
- **Pair:** Each  $T_{A1}$  can only match one  $T_{A2}$ .
- **Amounts:** Consider a perfect sandwich attack, the amount of bought token in  $T_{A1}$  should be equal to the amount of sold token in  $T_{A2}$ . In our practice, we relax this constraint by requiring that the amount of sold token in  $T_{A2}$  is between 90% and 110% of the amount of bought token in  $T_{A1}$ .
- **Position:**  $T_{A1}$  and  $T_v$  should be included in different blocks, and  $T_{A2}$  has no such constraint. If  $T_{A1}$  and  $T_v$  are in the same block, we classify them as single-chain sandwich attacks. These heuristics above we adopt are exactly the same as those proposed by Qin et al. [3].
- **Timestamp:** The timestamp of transaction  $T_{A1}$  must fall between those of the start transaction on source chain  $T_s$  and end transaction on destination chain  $T_v$ . The timestamp of transaction  $T_{A2}$  should be

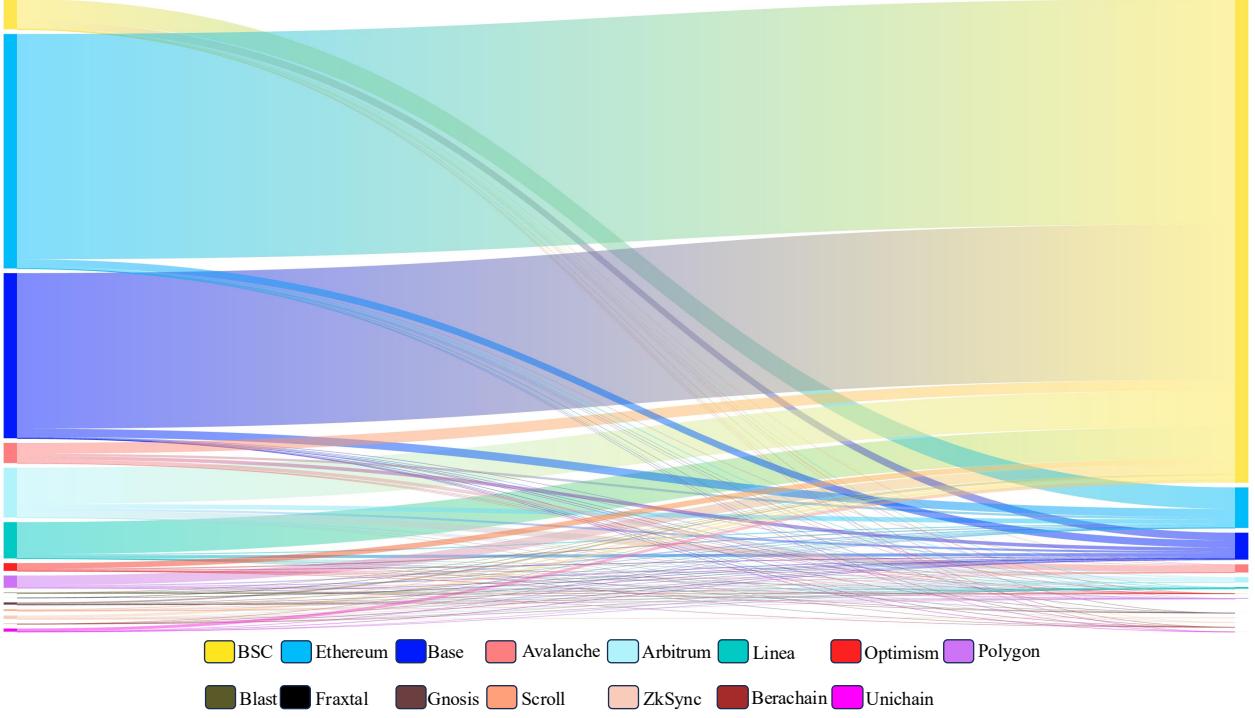


Figure 4. Visualization of profits across different blockchains. The left nodes represent the source chains and the right nodes are the destination chains. The width of a node represents the attacker's profit in USD realized on the destination chain.

TABLE 2. TOP 10 BLOCKCHAIN PAIRS BY PROFIT

Source chain	Ethereum	Base	Arbitrum	Avalanche	Ethereum	Optimism	Avalanche	Gnosis	Ethereum	Polygon
Destination chain	BSC	BSC	BSC	BSC	Base	BSC	Base	BSC	Arbitrum	Base
Profit(USD)	2096164	1447602	337532	100162	66914	51558	32647	14559	10225	8964
Transactions Value(USD)	247920634	90356394	34062217	6528034	5845820	2443226	1469088	495505	497432	434752
Percentage	0.85%	1.6%	0.99%	1.53%	1.14%	2.11%	2.22%	2.94%	2.06%	2.06%

controlled within a suitable range, where within  $num$  block times after  $T_v$ .

- **Address:** Either the same user address receives tokens in  $T_{A1}$  and  $T_{A2}$ , or two different addresses send  $T_{A1}$  and  $T_{A2}$  to the same pool contract address. In Algorithm 1, we first filter back transactions that satisfy the previous heuristics. Among these transactions, the transaction with the same token receiving address as  $T_{A1}$  has higher priority to pair with  $T_{A1}$ .

## 4.2. Detection Settings

We select a cross-chain protocol called Symbiosis [16] as an example. At the time of writing, it had processed over 4,578,130 cross-chain transactions, with a total cross-chain transaction volume reaching 6.44 billion USD [30] since November 2022. In this protocol, when a user signs a cross-chain transaction, the bridge contract<sup>2</sup> on the source chain emits an *OracleRequest* event that contains all calldata to be executed on the destination chain. An attacker can

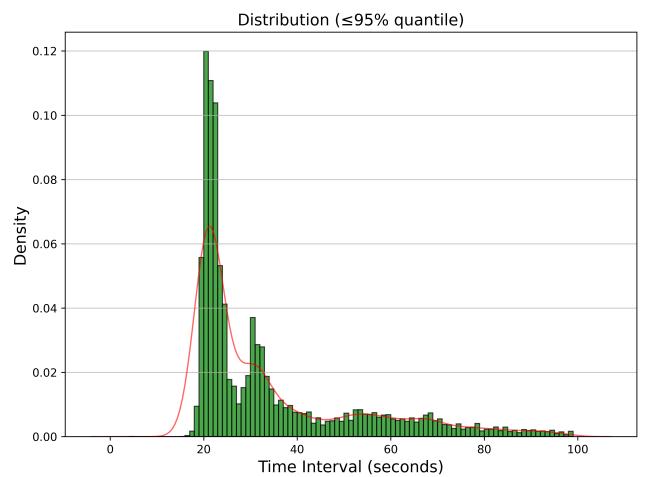


Figure 5. The distribution of the time interval between the event is emitted on the source chain and the swap is executed on the destination chain in the Symbiosis protocol.

2. 0x5523985926Aa12BA58DC5Ad00DDca99678D7227E on Ethereum

---

**Algorithm 1:** Detect sandwich attack in cross-chain swaps

---

**Input:** Start transaction on the source chain:  $T_s$ ;  
 Victim transaction on the destination chain:  
 $T_v$ ; Detection range of block numbers after  
 $T_v$ : num

**Output:** Detected sandwich attack pairs  $pairs$  that include all  $\{T_{A1}, T_v, T_{A2}\}$

```

1 Query blockNumber of  $T_s$  and  $T_v$ :  $N_s, N_v$ ;
2 Get pools used in  $T_v$ ;
3 for pool in pools do
4   Query log  $log_v$  in pool which belongs to  $T_v$  and read direction  $d$ ;
5   Query all swap logs  $FrontLogs$  with direction  $d$  in pool between  $N_s$  and  $N_v$  on the destination chain, query all swap logs  $BackLogs$  with direction  $-d$  in pool between  $N_v$  and  $N_v + num$  on the destination chain;
6   for frontlog in  $FrontLogs$  do
7     Read receive address  $addr$ ,  $tokenInAmount$  and  $tokenOutAmount$  from  $frontlog$ ;
8   for backlog in  $BackLogs$  do
9     Read  $tokenInAmount'$  and  $tokenOutAmount'$  from  $backlog$ ;
10    if  $tokenInAmount'$  in  $[0.9, 1.1]$  then
11      similarAmountTx.append(backlog)
12    for backlog in similarAmountTx do
13      Read receive address  $addr'$ ;
14      if  $addr == addr'$  then
15        pairs.append( $\{frontlog, log_v, backlog\}$ );
16        BackLogs.remove(backlog);
17        goto line 6;
18    targetbacklog = similarAmountTx[0];
19    pairs.append( $\{frontlog, log_v, targetbacklog\}$ );
20    BackLogs.remove(targetbacklog);

```

---

locally reconstruct the bridge contract’s function logic on the destination chain and iteratively parse that calldata to extract the information they need, for example, which destination chain and liquidity pool is targeted, how much token X will be swapped, and the minimum acceptable amount of token Y. With this information, the attacker can then perform a cross-chain sandwich attack.

The Symbiosis protocol relies on external DEX aggregators to compute the optimal swap path, so the calldata emitted in the event contains the aggregator’s swap execution logic. Specifically, taking 1inch [31] as an example, Symbiosis ultimately calls the *swap* function in *AggregationRouterV5* to perform the token exchange in the corresponding pools. In practice, however, we found that the *executor* contract<sup>3</sup> responsible for parsing the calldata in

3. 0x5141B82f5fFDa4c6fE1E372978F1C5427640a190 on Ethereum

*swap* is not open source, so the exact format of the calldata is unknown to us. To address this, we use Tenderly [32] to replay the transactions corresponding to the calldata on specific historical blocks and record the swap details along the contract execution path.

We collect two months of historical cross-chain transaction data from this protocol (August 10 to October 10, 2025)<sup>4</sup>. We first perform an initial filtering of the dataset based on two criteria. The first criterion is that the swap on the destination chain must occur within a liquidity pool and must not involve stablecoin-to-stablecoin exchanges. The second criterion requires that the time interval between  $T_s$  and  $T_v$  falls within a reasonable range. In practice, we observed cases where this interval exceeded 10 minutes, such cross-chain transactions provide little analytical value for our study. As shown in Figure 5, the distribution of these time intervals indicates that 95% of them are within 100 seconds. Therefore, we set 100 seconds as the maximum allowable interval. We then apply the detection heuristics described above to find instances of such attacks within the filtered transactions.

### 4.3. Empirical Results

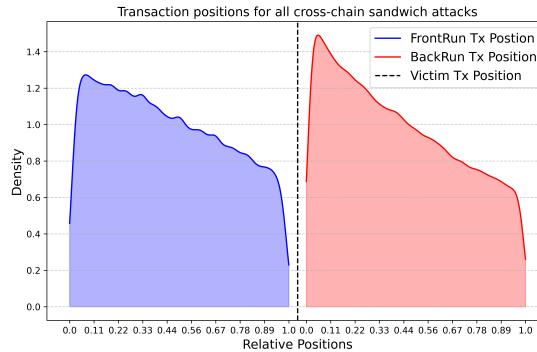
**Transactions.** In total, we collect 60,130 cross-chain transactions from the protocol, of which 37,649 remain as valid transactions after applying the filtering criteria. By applying Algorithm 1 to each transaction, we identify 316,809 potential cross-chain sandwich attack pairs. Among them, only 269 pairs occur within the same block, which represents single-chain sandwich attacks, they account for merely 0.085% of the total. Table 3 provides an overview of pools that were attacked more than 500 times in the dataset. We observe that the BUSD↔WBON pools are the most frequently targeted, accounting for 57.65% of all identified attacks.

We provide a perfect case of a cross-chain sandwich attack in which both single-chain and cross-chain sandwich attacks coexist on Ethereum Mainnet: transactions (0x0aaf...b44f, 0x9d5f...d548, 0xef06...2d7a, 0xe16b...82c0, 0xc07d...2b8f, 0xe2a1...86de) correspond to components ( $T_s$ ,  $T_{A1}$ ,  $T_{B1}$ ,  $T_v$ ,  $T_{B2}$ ,  $T_{A2}$ ) described above, and this example is shown in Figure 7.  $T_v$  attempted to swap WETH for MXY.  $\mathcal{A}$  placed  $T_{A1}$  at block 23286093, then  $T_v$  was successfully executed at block 23286100. Concurrently,  $T_v$  was also targeted by  $\mathcal{B}$ , and  $\mathcal{B}$  is labeled as a MEV Bot in this case. However, existing MEV monitoring platforms, such as EigenPhi<sup>5</sup>, only detect  $T_{B1}$  and  $T_{B2}$  for  $T_v$ . Subsequently,  $\mathcal{A}$  placed  $T_{A2}$  at block 23286105 to extract the value. In the end, excluding gas costs,  $\mathcal{A}$  gained an additional 0.034 WETH and achieved a 21.4% profit rate  $((\Delta x_{A2} - \Delta x_{A1})/\Delta x_{A1})$ , whereas the profit rate of  $\mathcal{B}$  is only 0.8%.

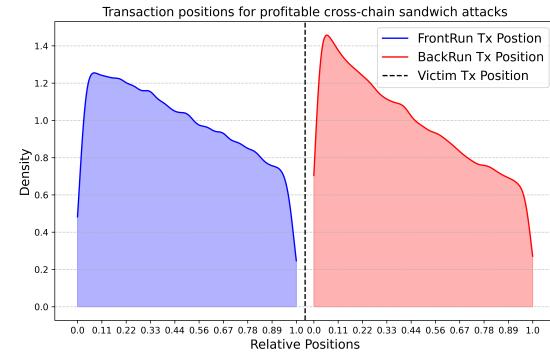
**Extracted profit.** The profit is calculated based on the tokens’ price in USD as of October 31, 2025 [33], and we

4. <https://zenodo.org/records/17562882>

5. <https://eigenphi.io/mev/ethereum/tx/0x4c0d2d55daef8fce10b340445a14789532e47a9c0d45bf3265489abf87b05731/>



(a) Transaction relative positions for all sandwich attacks



(b) Transaction relative positions for profitable sandwich attacks

Figure 6. Relative positions of cross-chain sandwich attacks

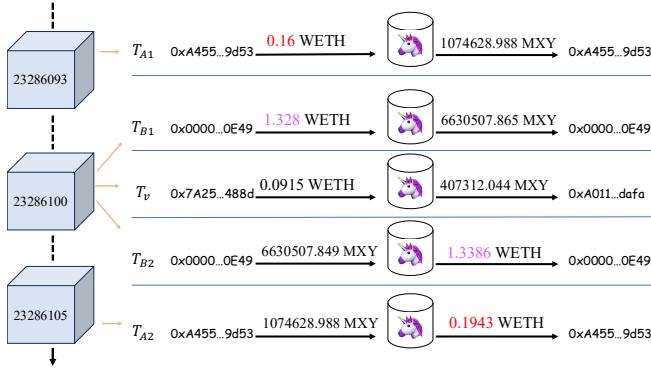
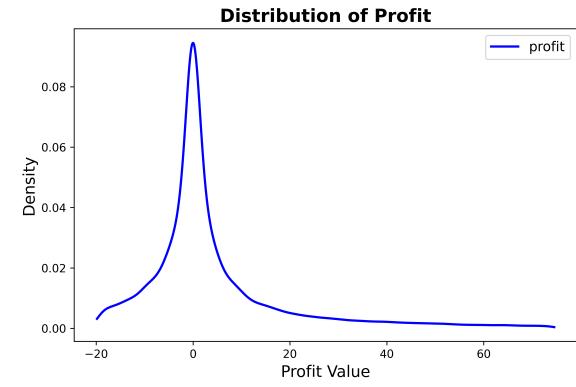


Figure 7. Case study of one cross-chain sandwich attack on the Ethereum Mainnet

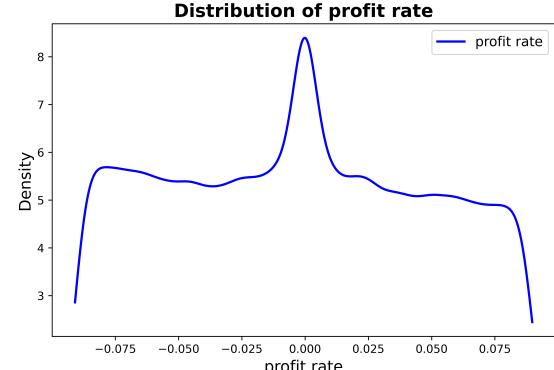
do not consider the gas cost here. Overall, the total profit we calculated from our dataset is 5,273,857 USD out of a total trading volume of 412,632,065 USD, and the largest profit from a single sandwich attack is 20,284 USD. Additionally, there remains 1,425,500 USD worth of unexploited potential profit.

Figure 8 presents the distributions of absolute profit and profit rate. The profit rate appears roughly symmetric around zero, while the absolute profit is predominantly positive, indicating that large-volume opportunities overwhelmingly produce positive returns.

We count the profits for each cross-chain pair and present the top 10 pairs with the highest total profits in Table 2. Among them, the chain pair Ethereum→BSC yields the highest attacker profit, with a total of 2,096,164 USD extracted. Figure 4 illustrates the proportions of profits generated on the destination chain across different cross-chain pairs. We also observed that BUSD↔WBONB is the most sandwich attack-prone token pair on the destination chain with a profit of 3,170,661 USD, this token pair's pools are also the most frequently attacked, with a total of 182,620 sandwich transaction pairs (cf. Table 3). The profit accounts for 60.1% of the total attacked volume. One possible reason



(a) Distribution of the profit in USD ( $\Delta x_{A2} - \Delta x_{A1}$ )



(b) Distribution of the profit rate ( $(\Delta x_{A2} - \Delta x_{A1})/\Delta x_{A1}$ )

Figure 8. The distribution of the absolute profit and the profit rate

for such a high concentration is that this token pair itself has a large cross-chain trading volume, reaching 330,121,414 USD.

In our dataset, single-chain sandwich attacks produced only 6,109 USD in profit, approximately 0.116% of the total profits. Taken together with the single-chain sandwich

TABLE 3. POOLS SORTED BY NUMBERS OF ATTACKS

Address	Tokens	Chain	Protocol	Number of attacks
0x172fcd41e0913e95784454622d1c3724f546f849	BUSD-WBNB	BSC	PancakeSwap V3	118731
0x47a90a2d92a8367a91efa1906bfc8c1e05bf10c4	BUSD-WBNB	BSC	Uniswap V3	63889
0xf2688fb5b81049dfb7703ada5e770543770612c4	USDC-WBNB	BSC	PancakeSwap V3	63049
0x62fc3c1794fb95bd8b1a97f6ad5d8a7e4943a1e	ETH-WBNB	BSC	PancakeSwap V3	7458
0x72ab388e2e2f6facef59e3c3fa2c4e29011c2d38	WETH-USDC	Base	PancakeSwap V3	6344
0x9f599f3d64a9d99ea21e68127bb6ce99f893da61	ETH-BUSD	BSC	PancakeSwap V3	4509
0xe0554a476a092703abdb3ef35c80e0d76d32939f	USDC-WETH	Ethereum	Uniswap V3	3224
0x62cf00528cb7af872c1f9dd426e655c903f16770	ETH-USDC	BSC	PancakeSwap V3	2822
0x6f38e884725a116c9c7fbf208e79fe8828a2595f	WETH-USDC	Arbitrum	Uniswap V3	2439
0x7e58f160b5b77b8b24cd9900c09a3e730215ac47	ASTER-BUSD	BSC	PancakeSwap V3	2352
0xa20c959b19f114e9c2d81547734cdc1110bd773d	WAVAL-USDC	Avalanche	Aerodrome	2346
0xe30d5bf485f7476ac15884a28ffb3c9cea635dcb	AVNT-USDC	Base	Aerodrome	2303
0xdbc6998296caa1652a810dc8d3baf4a8294330f1	WETH-USDC	Base	Aerodrome	1865
0xb4cb800910b228ed3d0834cf79d697127bbb00e5	WETH-USDC	Base	Uniswap V3	1443
0x247f51881d1e3ae0f759afb801413a6c948ef442	BUSD-BTCB	BSC	PancakeSwap V3	1364
0x62edad2a56c9fb55be5f9b1399ac067f6a37013b	BTCB-WBNB	BSC	PancakeSwap V3	1271
0x8bf0fb037b30562fdb7be3f71440575664ab74e	ASTER-BUSD	BSC	Uniswap V3	1264
0x2325e3f261cadb1c30cef66c9f95f6fb016c0d4	WETH-PUPPIES	Ethereum	Uniswap V2	1197
0x50203df8efcddba9755c886f086b9b2d537a15f9	XPL-BUSD	BSC	PancakeSwap V3	1107
0x36696169c63e42cd08ce11f5deebbbebae652050	BUSD-WBNB	BSC	PancakeSwap V3	1088
0x90166b5795250fe7f0831e844121cc91799787e9	USDC-STBL	BSC	PancakeSwap V3	1077
0x70f536b375296b60078a6e1bb0790919a13efe77	USDC-LINEA	LINEA	NA	1066
0x16b9a82891338f9ba80e2d6970fdda79d1eb0dae	BUSD-WBNB	BSC	PancakeSwap V2	1015
0xfae3f424a0a47706811521e3ee268f00cfb5c45e	WAVAL-USDC	Avalanche	Uniswap V3	929
0xc7bbec68d12a0d1830360f8ec58fa599ba1b0e9b	WETH-USDT	Ethereum	Uniswap V3	877
0xe1799b52c010ad415325d19af139e20b8aa8aab0	BTCB-USDC	BSC	PancakeSwap V3	848
0x6ec31af1bb9a72aacec12e4ded508861b05f4503	WBNB-MYX	BSC	PancakeSwap V3	767
0x4141325bac36aff9db165e854982230a14e6d48	USDC-WBNB	BSC	Uniswap V3	737
0x1b54cd932b6b751803c996cbef36280b53795f81	BUSD-HEMI	BSC	PancakeSwap V3	716
0xb1026b8e7276e7ac75410f1fcbbe21796e8f7526	WETH-USDC	Arbitrum	Algebra	612
0x7fdc35463e3770c2fb992716cd070b63540b947	WETH-USDC	Arbitrum	PancakeSwap V3	570
0x85faac652b707fdf6907ef726751087f9e0b6687	WBNB-BUSD	BSC	PancakeSwap V3	561
0x882df4b0fb50a229c3b4124eb18c759911485bfb	DAI-LGNS	Polygon	Uniswap V2	517

transaction counts reported above, this suggests that when cross-chain sandwich attacks are already in play, little to no exploitable volume remains when the victim transaction appears in the mempool.

**Sandwich transaction positions.** Unlike single-chain sandwich attacks that seek to place transactions immediately adjacent to the victim transaction, cross-chain sandwich attacks prioritize maximizing use of early-obtained information while avoiding direct competition with existing MEV bots. We analyze the relative block positions of transactions in each cross-chain sandwich pair, specifically, the position of  $N_{A1}$  relative to  $N_s$  and  $N_v$ , and the position of  $N_{A2}$  relative to  $N_v$  and  $N_v + num$ . In our experiments we set the parameter  $num = 100$  in Algorithm 1. The resulting position distribution is plotted in Figure 6, where the left subfigure depicts the distribution for all detected attacks, while the right subfigure shows the distribution restricted to profitable attacks. Empirically,  $\mathcal{A}$  tends to sign  $T_{A1}$  immediately upon observing a profitable event and to submit  $T_{A2}$  immediately after  $T_v$  is executed in order to realize

profit. This behavior aligns with the assumption in Section 3. We observe that the distribution of relative positions for all sandwich attacks closely resembles that of profitable sandwich attacks, indicating that the attackers follow similar strategies. However, their specific profits are influenced by the impact of intermediate noisy transactions.

**Sandwich transaction gas price.** For each sandwiching transaction pair, we examine the relationship between the gas price of  $T_{A1}$  and that of  $T_v$ , as well as between the gas price of  $T_{A2}$  and  $T_v$ . For  $T_{A1}$ , we found that in 99.56% of cross-chain transaction pairs, the ratio  $\frac{GasPrice_{T_{A1}}}{GasPrice_{T_v}} < 1$ , meaning that almost all gas prices of  $T_{A1}$  is lower than that of  $T_v$ . This result is intuitive, as the attacker does not need to participate in a gas bidding competition to obtain the right to place the front-running transaction.

The difference between  $T_v$  and  $T_{A2}$  is shown in Table 4. We observe that it exhibits a certain degree of randomness. We infer that this is because there are other transactions in the same direction following  $T_v$  in some cases, meaning that  $\mathcal{A}$  does not exclusively attack  $T_v$ . As a result, the

corresponding gas prices deviate from our expected pattern.

In addition, we found that 31,311 front-running transactions had a gas price of 0. This indicates that some attackers submit their front-running transactions to private miners to avoid attack from single-chain sandwich attackers. In contrast, we did not observe any victim transactions with a gas price of 0, since victim swaps are automatically initiated by smart contracts and the protocol does not provide a mechanism to submit these transactions to private pools.

TABLE 4. THE DIFFERENCE OF GAS PRICE

$\delta = \text{GasPrice}_{T_v} - \text{GasPrice}_{T_{A2}}$	Number	Percentage
$\delta < 0 \text{ GWei}$	175496	55.39%
$0 \text{ GWei} \leq \delta < 1 \text{ GWei}$	140015	44.2%
$1 \text{ GWei} \leq \delta < 10 \text{ GWei}$	1271	0.4%
$10 \text{ GWei} \leq \delta$	27	0.009%
Total	316809	100%

**Empirical parameters.** These 37,649 valid cross-chain transactions were executed through a total of 73,391 liquidity pools (some pools are duplicated among them), as a swap may traverse multiple pools. For each pool, we extract  $T_1 \dots T_n$  that occurred between the timestamps of  $T_s$  and  $T_v$ , and examine their impact on the pool’s state. The results show that in 41,811 pools, no transactions occurred between these timestamps, yielding an empirical value of  $q = 0.57$ . Among the remaining 31,580 pools where  $T_1 \dots T_n$  existed, 21,579 pools exhibit final states in which  $T_{A1}$  did not incur loss after the execution of  $T_v$ , resulting in an empirical value of  $p = 0.68$ . Overall, the noisy transactions  $T_1 \dots T_n$  and  $T_v$  ultimately give  $T_{A1}$  about a  $q + (1 - q)p = 86.2\%$  chance of remaining profitable. This suggests that  $T_1 \dots T_n$  tend to follow the same directional trend as  $T_v$  in such cross-chain scenarios.

We computed the positive and negative return rates,  $r^+$  and  $r^-$ , for all detected cross-chain sandwich attacks. We observed several extreme cases in which  $r^+$  approached or exceeded 1 and  $r^-$  approached -1. For example, cross-chain attack transaction pairs ( [0xaf59...9607](#), [0x7872...6598](#), [0xe143...c752](#), ) and ( [0x6590...12c1](#), [0x57d7...31b2](#), [0xbbf8...156a](#), ) had profit rates of 0.96 and -0.82 respectively. Both involved UXLINK $\leftrightarrow$ USDC swaps that coincided with a black swan event on September 23, 2025<sup>6</sup>, when UXLINK’s price collapsed and then surged. After excluding these outliers and using the 95th-percentile statistics from the remaining data, we obtain empirical values  $r^+ = 0.045$  and  $r^- = -0.047$ , so the attacker’s expected profit rate under these empirical values can be written as

$$\mathbb{E}(r) = (q + (1 - q)p)r^+ + (1 - q)(1 - p)r^- = 3.23\%.$$

**Limitations.** Currently, we have only collected cross-chain transaction data between EVM-based chains and parsed them with event and function shown in Table 5. However, in the Symbiosis protocol, there also exist destination chains

of the UTXO type, such as Bitcoin. Since these chains lack smart contract support, we are currently unable to access the exact data format used by the protocol in such cases, and therefore can not parse the corresponding transactions. As a result, the potential profits from these cross-chain interactions remain unknown and the real profit is larger than that we calculated.

In addition to Symbiosis, other cross-chain protocols, such as Thorswap [25], also meet the prerequisites for the cross-chain sandwich attack. Thorswap reached a peak monthly swap volume of 11.18 billion USD [34]. It maintains its own liquidity pools and executes swaps via its relay chain. However, it does not provide an interface to query pool swap information on the relay chain, preventing us from analyzing its historical data. By examining more protocols of this type in the future, we will be able to obtain a more accurate assessment of cross-chain sandwich attack potential.

#### 4.4. Discussion of Future Strategy

The foregoing analysis is based on historical data. However, from the attacker’s perspective there exist multiple viable strategies in the cross-chain setting. The first strategy corresponds to the one described in Section 3: upon observing a profitable cross-chain event, the attacker immediately submits a front-running transaction. After the victim transaction is executed on the destination chain, it immediately submits the back-running transaction and has no other intermediate actions. This strategy is based on the attacker’s belief in the empirical value of probability  $q$ .

A second class of strategies involves dynamic decision-making: the attacker adaptively adjusts behavior in response to noisy transactions. For example, if the reverse-direction transaction volume exceeds the attacker’s tolerance, the attacker may cut losses early. Conversely, when same-direction noisy transactions appear, some attackers may adopt a greedy policy by submitting a small back-running transaction to capture a portion of the profit. In extreme cases, a large and same-direction transaction pushes the victim transaction’s execution beyond its slippage tolerance and causes the transaction revert, the attacker may place the back-running transaction immediately after that large transaction to realize even greater profit. However, because the Symbiosis protocol does not expose information about cross-chain transactions that are reverted, we can only discuss this possibility qualitatively and can not identify empirical instances to validate the existence of such strategies.

It is worth noting that some cross-chain protocols do not publish their event data formats (i.e., the corresponding smart contract code is not open source). In such cases, an attacker must locally simulate the calldata execution on the most recent block to infer the swap path on the destination chain and then place  $T_{A1}$  in the relevant pool. This approach introduces uncertainty because the actual execution time of  $T_v$  is unknown. If execution is delayed, the state of blockchain may change and the calldata’s execution result may different. That can cause the attacker to misjudge the

6. <https://dex.coinmarketcap.com/token/arbitrum/>  
0x1a6b3a62391ecaaa992ade44cd4afe6bec8cff1/

details of  $T_v$ , and thus increase the risk that the attacker will incur a loss.

## 5. Mitigation

The information necessary for cross-chain sandwich attacks is obtained on the source chain. Consequently, future changes to mempool opacity [35], [8], [9] or to transaction-ordering mechanisms [27], [28] on the destination chain would not prevent this attack.

To eliminate cross-chain sandwich attacks in such protocols, one must either prevent information emitted on the source chain from being publicly exposed, or ensure that any publicly available information is useless to an attacker (cut ❸ or ❹ in Figure 1). This argument relies on the assumption that the protocol itself is honest and reliable. If the protocol designers or operators are malicious, the attack can not be stopped. In the worst case, a malicious designer can read cross-chain information directly from the frontend or other privileged channels and execute such sandwich attacks. They observe the information (❶ in Figure 1) strictly earlier than any external attacker (❷ in Figure 1). Based on this assumption, we present several potential countermeasures against cross-chain sandwich attacks.

**Private relayers.** A cross-chain protocol could choose not to publish events on-chain, and instead send the corresponding cross-chain information privately to its own relay nodes or a set of trusted relays. While this design can completely prevent cross-chain sandwich attacks, it also means the protocol must give up the market of decentralized relay network and take on the cost of maintaining its own relay network. Moreover, this increases the level of centralization in the protocol and reduces users' trust in it.

**Encrypted message.** A protocol can encrypt the cross-chain swap information on the source chain and only reveal the plaintext on the destination chain after the relayer forwards the encrypted message. However, decrypting data directly on the bridge contract is impractical. First, all key variables and decryption logic within a smart contract are publicly visible. Second, employing cryptographic techniques such as DKG [36] to achieve decentralized key management would incur extremely high gas costs and introduce significant time delays, making it unsuitable in practice. Therefore, some entity must have access to the plaintext of the event in order to execute it. The protocol can introduce a trusted off-chain decryption committee: after receiving the encrypted event from the relayer, the committee selects a member to decrypt the event and execute the calldata logic on the destination chain. In this design, even if a relayer or an attacker observes the event emitted on the source chain, they can not extract useful information and therefore cannot perform a cross-chain sandwich attack.

**Use real-time swap calldata.** The root cause of cross-chain sandwich attacks is that such cross-chain protocols compute the exact swap calldata on the source chain (via DEX aggregators like 1inch [31] and OpenOcean [37]) so the frontend can show the user a precise expected output. That calldata which contains the chosen pools and execution

path is then sent to the destination chain and becomes observable to relayers or attackers.

A simple and effective fix is to move the swap calldata-generation step to the destination chain. On the source chain, the relayer should only receive the minimal necessary information, such as that the user wants to swap  $\Delta x$  token X for token Y, without including any details about the pool, path, or routing. Because aggregators' optimal paths change over time, an attacker who only sees this rough intent can not predict the exact future calldata or chosen route. Guessing the specific pool and submitting a front-running transaction carries risk, as a rational attacker, they would not attempt such an attack.

## 6. Related Works

**Maximum extractable value.** Eskandir et al. [38] are the first to introduce the concept of front-running from traditional financial markets into DeFi. Subsequently, Flash Boys 2.0 [5] provide a study on priority gas auctions, where the winner gains priority in transaction ordering. This work introduced the concept of MEV and empirically demonstrated that MEV poses a significant threat to Ethereum. Torres et al. [39] and Qin et al. [3] conducted a systematic analysis based on historical Ethereum transaction data and showed that many miners have already been extracting MEV through gas bidding. In addition, Qin et al. [3] evaluate the profits miners obtained from arbitrage, liquidation, and sandwich attacks, respectively. More specifically, Zhou [40] formally define the sandwich attack problem in AMM exchanges. They analyze it both theoretically and empirically from the attacker's perspective, identifying the conditions under which such attacks can yield profit.

To detect the MEV behaviors, some works explore learning-based detection of malicious transactions and MEV bots at scale [41], [42], [43]. Li et al. [44] conduct a comprehensive study of Flashbots bundles and discovered 17 previously unknown DeFi MEV strategies by leveraging their presented detection tool.

There are currently several effective countermeasures against sandwich attacks. Proposer-Builder Separation (PBS) mechanism [6] in Ethereum splits block production into two roles. Builders construct blocks and include transactions, while proposers only select which block to propose. This separation improves fairness in transaction ordering. Aequitas provided by Kelkar et al. [27] is a protocol designed to achieve order fairness at the consensus layer, and it is later extended to a permissionless setting [28]. Furthermore, there are some works that leverage the encrypted mempool [8], [9], [10] to hide transaction content until ordering is finalized.

We can summarize that all existing mitigations aim to ensure fair transaction ordering within the block containing the victim transaction. However, cross-chain sandwich attacks do not rely on transaction ordering within that block, they place front-running transaction in the block preceding the target block. Therefore, all of the aforementioned methods are ineffective against this type of attack.

**Cross-domain MEV.** Cross-domain MEV arises when extractable value spans different domains, such as Layer 1 to Layer 2 or cross-chain environments. Its essence lies in exploiting manipulation opportunities caused by network communication delays, state differences, or information asymmetry between different domains.

Cross-chain arbitrage is a common form of cross-domain MEV, arbitrageurs observe price differences of the same token across different chains and execute rapid buy and sell trades to capture profits. Oz et al. [45] analyze one year of data across nine blockchains and identified 868.64 million USD worth of cross-chain arbitrage transactions. Additionally, they observe a centralization trend in cross-chain arbitrage, with five addresses generate more than half of these transactions. Gogol et al. [46] investigate non-atomic arbitrage in cross-rollup scenarios, where arbitrage is not executed within a single atomic transaction and there are delays between buy and sell operations introduce some risk. They found that price differences on rollups often last for 10–20 blocks, and discovered over 0.5 million unexploited arbitrage opportunities on these rollups.

In the rollup setting, Ferreira et al. [47] exploit the centralized sequencing mechanism of rollups to manipulate transaction ordering and mount cross-layer sandwich attacks. Their analysis estimates that attackers could have earned roughly 2 million USD through such sequencer-level manipulation. However, this line of work is fundamentally different from ours. Cross-layer attacks rely on privileged control or structural properties of a specific rollup’s sequencer, whereas we uncover an attack vector that arises purely from cross-chain message flow. Prior studies do not examine the possibility that cross-chain infrastructure itself can leak actionable transaction information and enable sandwich attacks across chains. To the best of our knowledge, our work is the first to identify, formalize, and empirically analyze cross-chain sandwich attacks, supported by real-world historical data.

## 7. Conclusion and Future Works

In this paper, we identify a previously unrecognized cross-chain sandwich attack in which an attacker exploits information leaked by cross-chain protocols to execute profitable front-running and back-running transactions. In this setting, existing defenses against sandwich attacks fail, allowing attackers to consistently outperform current MEV bots. We provide a theoretical analysis of the cross-chain sandwich attack and conduct an empirical study on historical data from a major cross-chain protocol. Our results show that this class of attacks produced at least 5.27 million USD in profit over a two-month period. To the best of our knowledge, we are the first to construct and publicly release a large-scale dataset of cross-chain sandwich attacks, providing a foundation for future research on cross-chain security. This work highlights a critical security risk introduced by cross-chain interoperability and aims to inspire more robust DeFi defense mechanisms.

**Future works.** We plan to extend this line of reasoning to examine whether other MEV activities, such as arbitrage or liquidation, can gain additional advantage in cross-chain settings. We also intend to investigate the existence and prevalence of these strategies using historical data. More interestingly, cross-chain environments may enable entirely new forms of MEV that do not arise in single-chain settings.

## 8. Acknowledgment

This work is supported by the Guangzhou-HKUST(GZ) Joint Funding Program (No. 2024A03J0630 and No. 2025A03J3882), the Guangzhou Municipal Science and Technology Project (No. 2025A04J4168), and the Guangdong Provincial Key Lab of Integrated Communication, Sensing and Computation for Ubiquitous Internet of Things (No.2023B1212010007).

## References

- [1] “Defi llama,” <https://defillama.com/>, 2025.
- [2] L. Zhou, X. Xiong, J. Ernstberger, S. Chaliasos, Z. Wang, Y. Wang, K. Qin, R. Wattenhofer, D. Song, and A. Gervais, “Sok: Decentralized finance (defi) attacks,” in *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2023, pp. 2444–2461.
- [3] K. Qin, L. Zhou, and A. Gervais, “Quantifying blockchain extractable value: How dark is the forest?” in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 198–214.
- [4] K. Qin, L. Zhou, P. Gamito, P. Jovanovic, and A. Gervais, “An empirical study of defi liquidations: Incentives, risks, and instabilities,” in *Proceedings of the 21st ACM internet measurement conference*, 2021, pp. 336–350.
- [5] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, “Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability,” in *2020 IEEE symposium on security and privacy (SP)*. IEEE, 2020, pp. 910–927.
- [6] S. Yang, K. Nayak, and F. Zhang, “Decentralization of ethereum’s builder market,” in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2025, pp. 1512–1530.
- [7] S. Wang, Y. Huang, W. Zhang, Y. Huang, X. Wang, and J. Tang, “Private order flows and builder bidding dynamics: The road to monopoly in ethereum’s block building market,” in *Proceedings of the ACM on Web Conference 2025*, 2025, pp. 2144–2157.
- [8] A. R. Choudhuri, S. Garg, J. Piet, and G.-V. Policharla, “Mempool privacy via batched threshold encryption: Attacks and defenses,” in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 3513–3529.
- [9] A. R. Choudhuri, S. Garg, G. V. Policharla, and M. Wang, “Practical mempool privacy via one-time setup batched threshold encryption,” in *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 3477–3495.
- [10] J. Bormet, S. Faust, H. Othman, and Z. Qu, “{BEAT-MEV}: Epochless approach to batched threshold encryption for {MEV} prevention,” in *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 3457–3476.
- [11] M. Kelkar, S. Deb, S. Long, A. Juels, and S. Kannan, “Themis: Fast, strong order-fairness in byzantine consensus,” in *Proceedings of the 2023 acm sigsac conference on computer and communications security*, 2023, pp. 475–489.
- [12] Z. Li and E. Pournaras, “Sok: Consensus for fair message ordering,” *arXiv preprint arXiv:2411.09981*, 2024.

- [13] P. Sheng, X. Wang, S. Kannan, K. Nayak, and P. Viswanath, “Trust-boost: Boosting trust among interoperable blockchains,” in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 1571–1584.
- [14] A. Augusto, R. Belchior, M. Correia, A. Vasconcelos, L. Zhang, and T. Hardjono, “Sok: Security and privacy of blockchain interoperability,” in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3840–3865.
- [15] Y. Guo, M. Xu, X. Cheng, D. Yu, W. Qiu, G. Qu, W. Wang, and M. Song, “{zkCross}: A novel architecture for {Cross-Chain}{Privacy-Preserving} auditing,” in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 6219–6235.
- [16] “Symbiosis documentation,” <https://docs.symbiosis.finance/>, 2025.
- [17] “Uniswap,” <https://app.uniswap.org/>, 2025.
- [18] L. Heimbach and R. Wattenhofer, “Eliminating sandwich attacks with the help of game theory,” in *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, 2022, pp. 153–167.
- [19] E. Tairi, P. Moreno-Sánchez, and C. Schneidewind, “Ledgerlocks: A security framework for blockchain protocols based on adaptor signatures,” in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 859–873.
- [20] ChainLink, “Understanding cross-chain token transfers,” <https://chainlink/education-hub/cross-chain-token-transfers>, 2025.
- [21] M. Herlihy, “Atomic cross-chain swaps,” in *Proceedings of the 2018 ACM symposium on principles of distributed computing*, 2018, pp. 245–254.
- [22] Y. Xue, D. Jin, and M. Herlihy, “Fault-tolerant and expressive cross-chain swaps,” in *Proceedings of the 24th International Conference on Distributed Computing and Networking*, 2023, pp. 28–37.
- [23] ChainLink, “Cross chain interoperability protocol,” <https://docs.chain-link.ccip>, 2025.
- [24] Cosmos, “Cosmos inter-blockchain communication protocol,” <https://docs.cosmos.network/ibc/v10.1.x/intro>, 2025.
- [25] “Thorswap,” <https://runescan.io/txs>, 2025.
- [26] deBridge, “debridge,” <https://app.debridge.com/orders>, 2025.
- [27] M. Kelkar, F. Zhang, S. Goldfeder, and A. Juels, “Order-fairness for byzantine consensus,” in *Annual International Cryptology Conference*. Springer, 2020, pp. 451–480.
- [28] M. Kelkar, S. Deb, and S. Kannan, “Order-fair consensus in the permissionless setting,” in *Proceedings of the 9th ACM on ASIA Public-Key Cryptography Workshop*, 2022, pp. 3–14.
- [29] W. Ding, Y. Tang, and Y. Wang, “Asymmetric mempool dos security: Formal definitions and provable secure designs,” in *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2025, pp. 1584–1602.
- [30] “Symbiosis explorer,” <https://explorer.symbiosis.finance/transactions>, 2025.
- [31] “linch,” <https://linch.com/>, 2025.
- [32] Tenderly, “Tenderly,” <https://docs.tenderly.co/>, 2025.
- [33] “Coinmarketcap,” <https://coinmarketcap.com/>, 2025.
- [34] “Thorstatistics,” <https://runescan.io/stat/swapVolumeUsd?time=month>, 2025.
- [35] B. Weintraub, C. F. Torres, C. Nita-Rotaru, and R. State, “A flash (bot) in the pan: measuring maximal extractable value in private pools,” in *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022, pp. 458–471.
- [36] A. Kate, E. V. Mangipudi, P. Mukherjee, H. Saleem, and S. A. K. Thyagarajan, “Non-interactive vss using class groups and application to dkg,” in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 4286–4300.
- [37] “Openocean,” <https://openocean.finance/>, 2025.
- [38] S. Eskandari, S. Moosavi, and J. Clark, “Sok: Transparent dishonesty: front-running attacks on blockchain,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 170–189.
- [39] C. F. Torres, R. Camino *et al.*, “Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain,” in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1343–1359.
- [40] L. Zhou, K. Qin, C. F. Torres, D. V. Le, and A. Gervais, “High-frequency trading on decentralized on-chain exchanges,” in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 428–445.
- [41] Z. Wu, J. Wu, H. Zhang, Z. Zheng, and W. Wang, “Hunting in the dark forest: A pre-trained model for on-chain attack transaction detection in web3,” in *Proceedings of the ACM on Web Conference 2025*, 2025, pp. 4519–4530.
- [42] Z. Ma, M. Jiang, F. Luo, X. Luo, and Y. Zhou, “Surviving in dark forest: Towards evading the attacks from front-running bots in application layer,” in *Proceedings of the 34th USENIX Security Symposium (USENIX SEC)*, 2025.
- [43] T. Niedermayer, P. Saggese, and B. Haslhofer, “Detecting financial bots on the ethereum blockchain,” in *Companion Proceedings of the ACM Web Conference 2024*, 2024, pp. 1742–1751.
- [44] Z. Li, J. Li, Z. He, X. Luo, T. Wang, X. Ni, W. Yang, X. Chen, and T. Chen, “Demystifying defi mev activities in flashbots bundle,” in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 165–179.
- [45] B. Öz, C. F. Torres, C. Schlegel, B. Mazorra, J. Gebele, F. Rezabek, and F. Matthes, “Cross-chain arbitrage: The next frontier of mev in decentralized finance,” *arXiv preprint arXiv:2501.17335*, 2025.
- [46] K. Gogol, J. Messias, D. Miori, C. Tessone, and B. Livshits, “Cross-rollup mev: Non-atomic arbitrage across l2 blockchains,” *arXiv preprint arXiv:2406.02172*, 2024.
- [47] C. Ferreira Torres, A. Mamuti, B. Weintraub, C. Nita-Rotaru, and S. Shinde, “Rolling in the shadows: Analyzing the extraction of mev across layer-2 rollups,” in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 2591–2605.

## Appendix A. Other Data

Table 5 presents an overview of the events and functions we leveraged in our detection.

TABLE 5. OVERVIEW OF EVENTS AND FUNCTION WE USED IN THE DETECTION

<b>Protocol</b>	<b>Event Name</b>	<b>Event Topic Hash</b>
Uniswap V2	Swap	0xd78ad95fa46c994b6551d0da85fc275fe613ce37657fb8d5e3d130840159d822
PancakeSwap V2	Swap	0xd78ad95fa46c994b6551d0da85fc275fe613ce37657fb8d5e3d130840159d822
Uniswap V3	Swap	0xc42079f94a6350d7e6235f29174924f928cc2ac818eb64fed8004e115fbcca67
PancakeSwap V3	Swap	0x19b47279256b2a23a1665c810c8d55a1758940ee09377d4f8d26497a3577dc83
Symbiosis	OracleRequest	0x532dbb6d061eee97ab4370060f60ede10b3dc361cc1214c07ae5e34dd86e6aaaf
<b>Protocol</b>	<b>Function Name</b>	<b>Function Signature</b>
Symbiosis	metaMintSyntheticToken	0xc29a91bc
Symbiosis	metaBurnSyntheticToken	0xe66bb550
Symbiosis	receiveRequestV2Signed	0x84d61c97
Symbiosis	metaUnsynthesize	0xc23a4c88
Symbiosis	externalCall	0xf5b697a5
1inch	swap	0x12aa3caf
1inch	uniswapV3SwapTo	0xbc80f1a8
OpenOcean	swap	0x90411a32
Uniswap V2	swapExactTokensForTokens	0x38ed1739
Uniswap V2	swapExactTokensForETH	0x18cbafe5