Shedding Light on Shadows: Automatically Tracing Illicit Money Flows on EVM-Compatible Blockchains

YICHENG HUO*, Zhejiang University, China
YUFENG HU*, City University of Hong Kong, China
YAJIN ZHOU†, Zhejiang University, China
TING YU, Mohamed bin Zayed University of Artificial Intelligence, United Arab Emirates
LEI WU, Zhejiang University, China
CONG WANG, City University of Hong Kong, China

The pseudo-anonymity and rapidly expanding ecosystem of Decentralized Finance (DeFi) have brought about significant liquidity on EVM-compatible blockchains, making them lucrative targets for cybercriminals. In the modern financial landscape, the need for an automated, high-speed, and effective illicit money tracing system is more urgent than ever to support regulators, on-chain service providers and security practitioners in their efforts to combat the frequent and large-scale occurrences of cyber financial crimes.

In this paper, we propose MFTracer, an automated system for tracing illicit money flows on EVM-compatible blockchains. Against the backdrop of a domain where tracing remains labor-intensive and expert-driven, MFTracer is developed in response to two pressing real-world demands: operational *efficiency* and forensic *effectiveness*. In response to the sophisticated fund transfer mechanisms enabled by the EVM environment, we introduce a novel fine-grained technique that enables protocol-agnostic transaction-level fund flow analysis. We further propose MFA, a lightweight and purpose-built graph abstraction with a tailored storage backend, to support efficient data retrieval. We also present a simulation algorithm for downstream illicit flow discovery. We implemented MFTracer. Its infrastructure for data retrieval achieves 3.7× to 9.4× higher storage efficiency while being 14.1× to 300× faster than the leading graph database systems. Furthermore, applied to real-world cybercrime incidents, MFTracer achieved 94.09% coverage of illicit money flows. It also newly reported 686 blockchain addresses and 4183 related transactions involved in money laundering that were previously undiscovered. MFTracer was able to reconstruct complete fund flow trajectories and provide strong evidence to investigators for $120.9 million in stolen assets.

CCS Concepts: • **Security and privacy → Economics of security and privacy**.

Additional Key Words and Phrases: Illicit Money Flow Tracing; Anti-Money Laundering; Blockchain

ACM Reference Format:
Yicheng Huo, Yufeng Hu, Yajin Zhou, Ting Yu, Lei Wu, and Cong Wang. 2025. Shedding Light on Shadows: Automatically Tracing Illicit Money Flows on EVM-Compatible Blockchains. *Proc. ACM Meas. Anal. Comput. Syst.* 9, 3, Article 63 (December 2025), 35 pages. https://doi.org/10.1145/3771578

*Part of this work was conducted while the author was a research intern at BlockSec.
†Corresponding author: Yajin Zhou (yajin_zhou@zju.edu.cn).

Authors' Contact Information: Yicheng Huo, yicheng_huo@zju.edu.cn, Zhejiang University, Hangzhou, China; Yufeng Hu, yufenghu@zju.edu.cn, City University of Hong Kong, Hong Kong, China; Yajin Zhou, yajin_zhou@zju.edu.cn, Zhejiang University, Hangzhou, China; Ting Yu, ting.yu@mbzuai.ac.ae, Mohamed bin Zayed University of Artificial Intelligence, Abu Dhabi, United Arab Emirates; Lei Wu, lei_wu@zju.edu.cn, Zhejiang University, Hangzhou, China; Cong Wang, congwang@cityu.edu.hk, City University of Hong Kong, Hong Kong, China.

ACM 2476-1249/2025/12-ART63
https://doi.org/10.1145/3771578

1 Introduction

Fueled by the smart contract [50] functionality and DeFi landscape, EVM-compatible blockchains have emerged as a dominant force in the blockchain industry, attracting significant capital flows and maintaining high liquidity levels. Meanwhile, blockchain's nature of anonymity creates opportunities for malicious entities to commit fraud and attacks, such as phishing scams [65], contract exploits [71], DeFi protocol attacks [113], etc. Throughout the cybercrime landscape, to avoid investigation and regulatory enforcement, criminals commonly employ money laundering tactics to conceal illicit fund flows. Specifically, they strategically transfer illegally acquired cryptocurrencies (upstream money) into downstream money that appears clean and legitimate. (In this paper, we use *money* or *funds* to denote cryptocurrencies on the blockchain.) If no one can *trace* the connection between downstream money and upstream illicit money, the illicit funds evade regulatory scrutiny, signifying complete success of the crime.

As a result, illicit money flow tracing plays a crucial role in real-world anti-money laundering (AML) efforts. Tracing refers to identifying *how* illicit money moves and *where* it is eventually directed. The ability to trace the flow of illicit money invalidates efforts by criminals to sever links between the origin and destination of illicit assets, thus yielding actionable evidence and facilitates prosecution. However, existing research efforts in the field of AML for cryptocurrencies [49, 54, 67, 68, 79, 80, 98, 106] focus on detecting and predicting anomalous behavior on blockchains, rather than developing **automatic** illicit money flow tracing systems that meet real-world demands for speed and effectiveness. Several studies [78, 82, 86, 112] introduce heuristic methods to analyze (rather than trace) money flows on Bitcoin, but they fall short in two key aspects: ❶ These approaches are *not applicable* to EVM-based blockchains, which operate under a fundamentally different execution model that supports complex smart contract functionality. These methods are also constrained by their reliance on predefined money laundering patterns, limiting their scalability and generalizability to diverse and evolving cybercrime scenarios involving EVM-compatible blockchains. ❷ They do not offer approaches to meet the *efficiency demands* of practical investigations. Real-world forensic scenarios—be it regulators conducting rapid investigations and timely interventions in recurring cybercrimes, service providers performing large-scale crime risk assessments for their know-your-customer (KYC) obligations, or any tracing task that demands timeliness and scale—hinge on systems that can efficiently organize and retrieve massive volumes of data in live blockchain environments. Existing studies lack the performance solution needed for operational deployment.

In today's reality and practice, tracing illicit money flows is still a daunting challenge that relies heavily on the manual efforts of security experts. How to build an efficient and effective automated tracing system is still an open research question, and so far no substantial research has been devoted to solving it. Through a systematic study of real-world cases, we surface two critical yet unsolved obstacles to developing automated illicit-flow tracing systems on EVM-compatible chains.

Challenge-I: EVM-compatible blockchains offer criminals complex fund transfer mechanisms, allowing money laundering via intricate smart contracts and diverse fungible (ERC-20) tokens.

The EVM environment enables highly sophisticated and diverse mechanisms for transferring funds. In many cybercriminal cases with significant losses [4, 5, 35, 43], we have seen that criminals leverage DeFi protocols or self-deployed sophisticated smart contracts to create transfers of considerable types of ERC-20 tokens. Even *within a single transaction*, criminals can manipulate contract semantics to generate money flows that are difficult to trace. Fig. 1 presents an example. The criminal executed a Multi-call [41] transaction to invoke multiple functions in the Uniswap

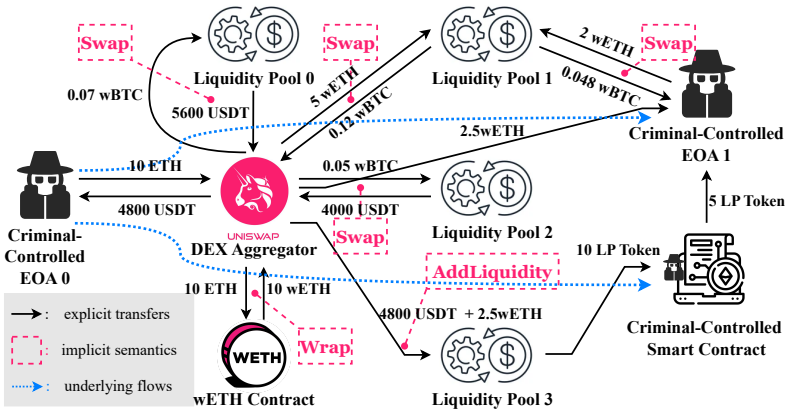

Fig. 1. A real-world example illustrates how a criminal executed a Multi-call transaction on a DEX aggregator to obscure illicit fund flows as part of their laundering strategy. The execution of this transaction can be divided into seven sub-processes (see Appendix A for details), including DeFi semantics of a `Wrap`, an `AddLiquidity` and four `Swaps`. This transaction caused $14,400 in illicit assets to flow out of EOA 0, with $9,600 directed to EOA 1 and $4,800 to a criminal-controlled smart contract.

(a decentralized-exchange) aggregator contract. By utilizing self-deployed contract and DeFi semantics like `Swap` and `AddLiquidity` [42], this transaction triggered transfers of multiple token types across multiple addresses. While ERC-20 transfers are visible in complex transactions, the functional roles of addresses, the semantic meaning of token interactions, and the underlying flow of funds are often implicit and opaque. Interwoven token transfers can form cyclic structures, making it difficult to even identify the true providers and recipients of the funds. Addressing this complexity necessitates the design of fine-grained, transaction-level fund flow analysis methods, which poses a non-trivial challenge for developing automated systems. (Interested readers may find more details of the example of Fig. 1 in Appendix A.)

Challenge-II: There is no efficient data retrieval infrastructure tailored to the heavy and pattern-specific data retrieval demands of tracing systems, posing a major obstacle to building practical solutions that achieve the efficiency required in real-world forensic investigations.

The low cost of account creation and rapid block production leads to a massive accumulation of fund transfers on the blockchain. This imposes a direct requirement for tracing systems to effectively organize large-scale data. In addition, criminals tend to employ long fund transfer paths to hinder tracing efforts. As an example, by limiting breadth-first search to a maximum of 15 hops (while the longest laundering path exceeds 18 hops) from the victim to identify potentially related addresses, the perpetrators of the incident [108] managed to create a downstream space containing over 56.4 million token transfers. Accurately identifying illicit fund flows within such a vast downstream space requires tracing systems to analyze massive volumes of transactions, resulting in a heavy data processing burden.

To meet the efficiency demands of real-world investigations and practical deployment, a high-performance data retrieval infrastructure is essential for tracing systems to handle queries, searches, and analyses on the enormous volume of data. Graph structure is a powerful approach to organize fund transfer data [69, 73, 75, 76]. However, the typical approach is to load raw transfer records into general-purpose graph databases, yet our benchmarks (Section 4.1) show that even leading systems

suffer up to an order-of-magnitude overhead in both storage and time under high-throughput blockchain workloads, impractical for real-world use. The absence of a dedicated data retrieval infrastructure poses a significant obstacle to building tracing systems that are fast and scalable enough to support timely intervention in massive and frequent cybercrime events.

Our Solution. This paper proposes a system MFTracer that provides the capabilities to automatically trace illicit money flows by solving the above challenges. To the best of our knowledge, MFTracer is the first automated system for tracing illicit money flows on EVM-compatible blockchains. In addition, it does not rely on specific traits or patterns of money laundering behavior. It traces illicit money flows through effective searches and simulations of money transfers rather than identifying or matching patterns of criminal behavior. This endows MFTracer with versatility and adaptability, enabling it to be applied to tracing tasks in various forms of cybercrime without being restricted by specific laundering traits. Specifically, to address **Challenge-I**, we propose an innovative fine-grained money flow analysis method with generalized protocol-agnostic parsing logic to decipher complex token transfers inside each transaction. For **Challenge-II**, we propose a purpose-built lightweight graph structure called Money Flow Abstract (MFA) along with a corresponding storage design to efficiently structure and store fund flow data.

The First Dataset. Moreover, due to the lack of relevant research, there is currently no publicly available real-world dataset for evaluating tracing systems on EVM-compatible blockchains. To address this gap, we curated LaunderNetEvm41. It is the *first* open and, to date, the *largest* dataset of ground-truth laundering flows on EVM-compatible blockchains. LaunderNetEvm41 traces $125 million in illicit transactions, documenting 1,939 blockchain addresses linked to criminal activities and 6,701 detailed records of illegal fund flows. Each entry was manually validated by professionals from a leading blockchain security firm[1] and domain experts from the broader security community to ensure its reliability. The dataset construction took over 350 hours of manual effort. In addition to enabling the evaluation of tracing systems, the dataset also offers address labels and annotations with visualizations of laundering behaviors, which can be used to support blockchain forensic and threat intelligence research. We release[2] it publicly to facilitate future advancements in this area.

Our Contribution. We summarize the contributions of this paper as follows:

- **Novel System.** We present an automated system for tracing illicit money flows on EVM-compatible blockchains.
- **Real-world Dataset.** We released a large-scale dataset from real-world cybercrime cases.
- **Efficient Solution and Implementation.** We implemented MFTracer and evaluated its data retrieval infrastructure, which delivers 3.7–9.4× better storage efficiency and 14.1–300× faster performance than leading graph databases. Our work removes a long-standing performance barrier to developing tracing systems by proposing a high-performance infrastructure design.
- **Effectiveness and Practical Impact.** Applied to real-world cybercrime incidents, MFTracer achieved 94.09% coverage of illicit flows and reported[3] for the first time 686 previously unknown laundering addresses plus 4,183 related transactions. The system was able to reconstruct fund flow trajectories and provide strong evidence to investigators for $120.9 million in stolen assets.

2 Background

2.1 EVM-Compatible Blockchain Basics

EVM. The Ethereum Virtual Machine (EVM) [18] is a decentralized virtual environment that executes code consistently across all nodes in a blockchain network. EVM-compatible blockchains

[1] BlockSec: https://blocksec.com/

[2] https://github.com/blocksecteam/MFTracer/tree/main/LaunderNetEvm41

[3] https://github.com/blocksecteam/MFTracer/tree/main/findings

[90], such as Ethereum, BSC, Avalanche, are blockchain networks that integrate EVM support, enabling seamless execution of EVM-based smart contracts without requiring modifications.

Address. Addresses are unique identifiers used to hold funds and send transactions on the blockchain. Each address is associated with one of two types of accounts: Externally Owned Account (EOA) or Contract Account (CA) [21]. EOAs are controlled by individuals to initiate transactions. CAs are controlled by smart contracts and can run predefined code.

Transaction. In EVM-compatible blockchains, external transactions are signed messages initiated by EOAs to transfer funds, interact with decentralized applications (dApps) [48], or deploy smart contracts. *Internal transactions* [45] are issued by smart contracts, instead of EOAs, to interact with other smart contracts, with similar functionalities as external transactions.

Smart Contract. Smart contracts are self-executing programs that are immutable once deployed. A smart contract is triggered to execute when it receives an external or internal transaction at its corresponding CA, and it can transfer tokens or further invoke other smart contracts by initiating internal transactions.

2.2 ERC-20 Tokens and Price Oracle Services

ERC-20 Standard. The ERC-20 [16] standard defines a set of interface rules that any fungible token must implement. An ERC-20 token is a fungible token with its smart contract implementing the ERC-20 interface. All operations involving an ERC-20 token, such as transfers, approving allowances [61], and checking balances, are executed by functions specified in the token's smart contract that complies with the ERC-20 standard. In addition, whenever an ERC-20 token is transferred, an event log is emitted by the corresponding token contract, known as the *Transfer Event Log* [16]. The Transfer Event Log records essential details, including the token type, transferred amount, source address, destination address, and timestamp of the transfer. This ensures transparency and serves as a basis for monitoring token activities on the blockchain.

Price Oracle Services. The price oracles [15] are tools used to view price information about ERC-20 tokens. There are many mature price oracle service providers, such as Chainlink [11], Euler [20], CoinMarketCap [10], etc. These services aggregate data from multiple sources, such as different decentralized-exchanges (DEXs) or centralized-exchanges (CEXs), and then apply various algorithms [55, 81, 83, 89] to figure out token prices, providing accurate USD valuations.

2.3 Illicit Money Flows on the Blockchain

Upstream & Downstream. Common forms of cybercrimes on the blockchain include phishing scam [65], contract attack [71], exploitation of DeFi protocols [113], etc. These criminal activities enable perpetrators to siphon cryptocurrencies from victims' blockchain addresses into their own. Addresses controlled by criminals that directly receive cryptocurrencies from victim addresses are referred to as *upstream addresses*. Due to their close association with victim addresses, upstream addresses are subject to regulatory scrutiny and enforcement, making it impossible for them to directly cash out illegally acquired funds through on-chain service providers (such as centralized exchanges). To evade detection, criminals systematically move funds from upstream addresses through layers of controlled addresses, using laundering techniques to obscure fund flows and hinder tracing. Eventually, these funds flow into *downstream addresses* that are strategically distanced from victim addresses and appear to be legally compliant and free of suspicion.

Illicit Money Flow Topology. As illicit funds flow from upstream to downstream, they traverse multiple addresses, generating intricate transactional relationships between these addresses. This process can be described using graph topology, where nodes represent the addresses involved and edges denote fund transfers. Formally, we call this topology the *illicit money flow topology*, a directed graph structure $G = (V, \mathcal{F})$. Here, V denotes the set of addresses that criminals use to launder

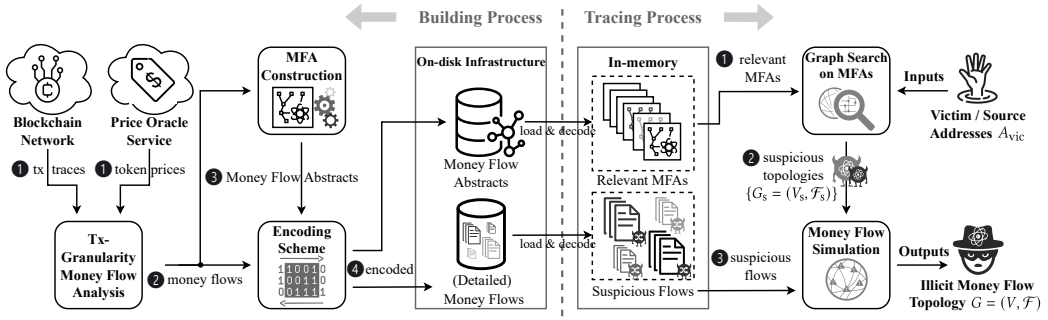

Fig. 2. An overview of the building process and the tracing process of MFTracer. Tx is the abbreviation of Transaction. During the (one-time) building process, MFTracer sets up the infrastructure required for handling tracing tasks. In tracing tasks, MFTracer takes the set of victim (source) addresses A_{vic} as input and derives the illicit money flow topology $G = (V, \mathcal{F})$.

illicit money, and $\mathcal{F} : V \times V \to \{0, 1\}$ denotes the directed edges indicating the existence of illicit fund flows between these laundering addresses. For $u, v \in V$, $\mathcal{F}(u, v) = 1$ means that there occur illicit fund transfers from u to v; otherwise, there is no such illicit fund movement. As discussed in **Challenge-I** and **Challenge-II**, criminals employ money laundering strategies to deliberately obscure and distort the true movement of illicit funds, essentially hiding the underlying illicit money flow topology. In judicial contexts, uncovering this topology invalidates criminals' attempts to obscure the connection between the downstream money they cash out and the upstream money they siphon from victims, thereby furnishing strong evidence for legal enforcement.

3 MFTracer Design

3.1 Overview

This paper presents MFTracer, an automated system for tracing illicit money flows on EVM-compatible blockchains. MFTracer builds a dedicated on-disk infrastructure that systematically organizes fund transfers to support high-performance data retrieval for tracing tasks. Our system adopts a "build-once, reuse-indefinitely" infrastructure model. The infrastructure encompasses all blockchain transaction data within a given time frame and is independent of specific criminal events. *Once* built, it supports tracing for *any* event within that time frame. This modeling approach enables MFTracer to be applicable to both historical and ongoing cybercrime incidents.

Figure 2 provides an overview of MFTracer, where the left part illustrates the phases of infrastructure construction, which we call the **Building Process**, and the right part depicts the phases of tracing task execution using the built infrastructure, which we call the **Tracing Process**. In the tracing process, MFTracer takes the victim addresses (since in most cases illicit cryptocurrencies are siphoned from victim addresses, in this paper we interchangeably refer to source addresses as victim addresses) as input, performs searches and simulations based on the infrastructure, and finally outputs the illicit money flow topology. As discussed in Section 2.3, this topology can serve as significant legal evidence that connects upstream and downstream funds. Our system is therefore designed to uncover it, which can be used by regulatory bodies and blockchain forensic investigators for further investigation.

Building Process. In the building process, MFTracer first retrieves execution traces of all transactions within a given time frame from the blockchain network, together with the necessary token price information from the token price oracle service provider (❶ in the left part of Figure 2). Next,

MFTracer applies transaction-level money flow analysis to decipher how funds are transferred between relevant addresses in each transaction. The transaction-granular analysis algorithm takes the trace of each transaction and the required token price as inputs, effectively detects and outputs all underlying fund flows triggered by the transaction (❷). Subsequently, MFTracer uses the extracted fund flows to construct a lightweight graph structure (❸), which we call Money Flow Abstract (MFA). MFA encodes all fund transfers occurring on the blockchain within the given time frame, supporting efficient graph search and pruning for large-scale fund flow analysis. It is used in the Graph Search phase of the tracing process. In the final step (❹), the lightweight MFA, which preserves only essential topology and timestamp information, and the complete details of money flows produced by the transaction-granularity analysis are separately encoded and stored on disk, making up the two core components of the infrastructure. The encoding scheme segments MFA into chronological subsets and optimizes the retrieval process of detailed money flows. This allows only data subsets relevant to the ongoing criminal investigation to be loaded into memory, further reducing data processing overhead.

Tracing Process. For the fund flow tracing task in a cybercrime incident, MFTracer takes the victim addresses (source addresses) as input. The relevant MFA subsets following the crime event's occurrence time are then retrieved from disk, loaded into memory and decoded (❶ in the right part of Figure 2). Following this, MFTracer executes a parallel graph search on the MFAs, using the victim addresses as the root nodes. This search quickly reconstructs a coarse-grained topology of the downstream fund flow network associated with the victim addresses (❷). This *suspicious topology*, a superset of the true illicit money flow topology, captures all potential addresses and paths traversed by illicit funds. Then, based on the address and transactional relationships indicated by the suspicious topology, MFTracer retrieves the relevant suspicious fund flow details from disk and loads them into memory. Finally, these suspicious flows are utilized in the money flow simulation phase (❸) to generate the output.

The following subsections explain the design goals and details of each phase. For clarity, the Graph Search phase is elaborated in Section 3.3 alongside the MFA data structure.

3.2 Tx-Granularity Money Flow Analysis

Motivation. As discussed in **Challenge-I**, the expressive nature of the EVM allows for sophisticated mechanisms for fund transfers. In addition to leveraging known DeFi protocols, criminals can freely deploy custom smart contracts deliberately designed to obscure fund movement. Even a single transaction can trigger transfers of multiple token types across multiple addresses and generate hard-to-trace fund flows. These realities lead to two key design insights for automated tracing systems: ① Tracing systems must support fine-grained analysis of fund transfers, capable of inferring fund flows from within transaction internals (see Section 2.1 and 2.2 for internal transactions and ERC-20 token transfers), rather than relying solely on raw transaction outputs involving native tokens as in prior work. ② Tracing systems must have generalized parsing logic capable of interpreting complex transactions involving arbitrary smart contracts and token operations beyond known DeFi protocols. To this end, we propose a fine-grained technique for MFTracer that enables protocol-agnostic transaction-level fund flow analysis.

In the Tx-Granularity Money Flow Analysis, MFTracer extracts all fungible token transfers from the transaction trace. It not only considers the native token explicitly transferred within the transaction but also the native token moved via internal transactions that are issued by smart contracts, along with all ERC-20 token transfers recorded in Transfer event logs. By incorporating these factors, MFTracer ensures a comprehensive coverage of token transfers inside each transaction.

MFTracer employs a three-step process to dissect the interwoven token transfers and determine the actual underlying fund flows caused by a transaction. First, it standardizes token values in USD

Algorithm 1 Tx-Granularity Money Flow Analysis

Input: tx: a transaction; O: the price oracle.
Output: out: the list of all underlying money flows caused by tx.

1: $G_{\mathrm{L}} \leftarrow$ newDirectedGraph() ▷ G_{L} is the local money transfer graph of tx.
2: $B \leftarrow$ newMap[Address $\rightarrow$ Balance]() ▷ B is the local balance change table of tx.
▷ For $a \in B$.keys, $B[a]$ denotes balance change of address a.
3: **for** t **in** tx.tokenTransfers **do** ▷ covering tx, its internal transactions and Transfer event logs
4: $m \leftarrow t$.value $* O(t$.token; tx.timestamp)
▷ $O(k; m)$ outputs the price per unit of token k at time m.
5: G_{L}.addEdge(t.from, t.to, m) ▷ add G_{L} an edge with weight m
6: $B[t.\text{from}] \leftarrow B[t.\text{from}] - m$; $B[t.\text{to}] \leftarrow B[t.\text{to}] + m$
7: **end for**
8: **for** $saddr$ **in** B.keys **where** $B[saddr] < 0$ **do** ▷ These are net sources that providing funds.
9: **for** $taddr$ **in** B.keys **where** $B[taddr] > 0$ **do** ▷ These are net targets that receiving funds.
10: $flowThru \leftarrow G_{\mathrm{L}}$.maxFlow($saddr, taddr$)
11: $ubound \leftarrow \min(-B[saddr], flowThru, B[taddr])$
12: out.append(newFlow($saddr \xrightarrow{ubound} taddr, tx$.Info))
▷ Append an underlying flow from $saddr$ to $taddr$ with value $ubound$ to the result list.
13: **end for**
14: **end for**

to unify the analysis of various tokens involved in a transaction. Then it constructs a balance change table for each address to quantify fund inflows and outflows in USD. This table helps identify the source addresses providing funds and the target addresses receiving them. Finally, it determines the maximum possible (upper bound) amount transferred from each source address to each target address to ensure that all underlying fund flows resulting from each transaction are fully captured. The identified fund flows are later used to build the MFA, which in turn ensures that the MFA can fully capture on-chain fund movements and support a coarse-grained graph search for building suspicious topologies. These flows are also used during the flow simulation. By imposing an upper limit on each simulated outgoing flow and treating victim addresses as the unique, finite source of illicit funds, MFTracer can precisely recover downstream illicit movements from the analytically derived upper bounds while avoiding simulated flows that exceed what is physically possible.

Details. Algorithm 1 presents the details. For each transaction tx, MFTracer iterates through the token transfers triggered by the transaction (line 3). Then MFTracer determines the real monetary value of each token transfer in USD based on the price oracle O (line 4). With the values, it builds a local money transfer graph G_{L} for tx and constructs the balance change table B for the addresses involved (line 5, 6). In the graph G_{L}, the nodes represent addresses and the directed edges are weighted by the transfer values in USD. B highlights the profits and deficits caused by tx for the involved addresses, thus revealing the sources and targets of the *net money flows*. That is, $B[a] < 0$ means that address a is a source providing money; $B[a] > 0$ means it's a target receiving money. Next, MFTracer applies the maximum flow algorithm [88] on G_{L} to compute the maximum amount of money that flows from each source through each target (line 10). The minimum value among this amount, the source's deficit, and the target's profit indicates the upper bound of the amount of funds that the source address can provide to the target address (i.e., the maximum possible amount). Finally, MFTracer appends the obtained fund flow along with necessary transaction information, such as the hash value and timestamp, to the result list and outputs the list (line 12).

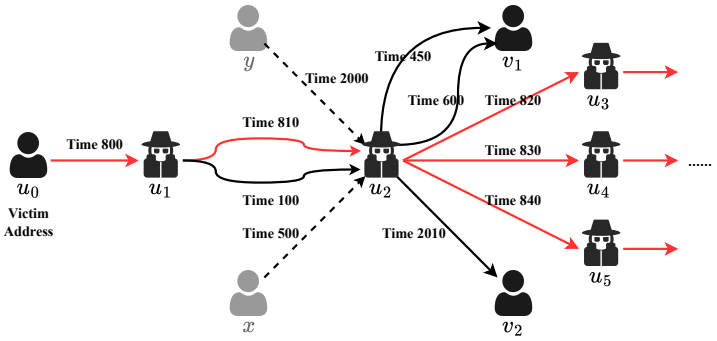

Fig. 3. A suspicious topology formed by addresses excluding x and y. Each u_i is a laundering address. x, y, v_1 and v_2 are innocent addresses, which are filtered out in the graph pruning and simulation phases. Red edges indicate the transfers of illicit funds. The clean funds transferred from u_2 to v_1 originate from x (time 600) and u_1 (time 450), and those transferred from u_2 to v_2 come from y.

In Appendix A, we include an example describing the walkthrough of Algorithm 1 applied to the transaction of Figure 1 to facilitate understanding.

3.3 Money Flow Abstract

To optimize the organization of fund flow data, we develop a novel lightweight graph structure termed Money Flow Abstract (MFA). Instead of relying on raw transaction records, MFTracer constructs the MFA using the accurate flows generated by the tx-level money flow analysis (❷ in the left part of Figure 2). This enables the MFA to provide a precise and reliable representation of fund movements. MFA's data structure is purposefully designed to support rapid graph searches and time-based pruning (applied in the Graph Search phase of the Tracing Process), enabling the efficient identification of the topology formed by the addresses reachable from the victim addresses, which we call the *suspicious topology*.

MFA Formulation. Formally, MFA is a graph $G_{\text{mfa}} = (V, E, T)$, where V is the set of addresses as vertices and E represents directed edges describing the existence of money flows. T is the set of attributes associated with edges, where MFTracer stores timestamps to enable pruning in the Graph Search phase. For $u, v \in V$, we use $\tau_{u,v} \in T$ to represent the collection of timestamps for all direct money transfers from u to v. To construct the MFA, for each flow from u to v derived from the Tx-Granularity Money Flow Analysis, MFTracer adds u and v into V, updates E by adding an edge from u to v, and updates T by adding the flow's timestamp to $\tau_{u,v}$.

MFA is a graph-based representation of fund transfers, where reachability indicates the presence of potential fund movements. For example, if there exists a reachable path from address u to v in the MFA, it implies that there is a potential fund movement from u to v. We define the suspicious topology as the topology formed by addresses reachable from the victim addresses and their fund flow relationships. As a superset of the illicit money flow topology, it includes all possible laundering addresses and paths. MFTracer constructs this topology in the Graph Search phase. And it is then further processed and refined in the Money Flow Simulation phase to extract the final output. We present an example of suspicious topology in Fig. 3 to enhance intuitive comprehension.

In the following, we start by introducing the data structure of MFA and then discuss how this structure enables efficient construction and pruning of the suspicious topology.

MFA Data Structure. We develop a data structure for MFA, designed with innovation to meet its specific needs. The data structure enables rapid graph searches and fast time-aware pruning. It

consists of only one hash map and four integer arrays. Its lightweight nature allows a large volume of graph data to reside in memory, eliminating the need for frequent disk access in the tracing process and significantly improving overall data retrieval efficiency.

$$\text{structure}(G_{\text{mfa}}) = (\mathcal{M}, P, N, T_{\min}, T_{\max}).$$

We use structure(G_{mfa}) to denote the data structure of a MFA G_{mfa}, where $\mathcal{M}$ is a hash map [56], and P, N, $T_{\min}$, and $T_{\max}$ are integer arrays. $\mathcal{M}$ maps each address in G_{mfa}'s address set $V = \{v_1, v_2..., v_{|V|}\}$ to an integer in the integer set $\{1, 2, ..., |V|\}$, which can be considered as the index of the address. P is an array of size $|V|$+1, with $P[0] = 0$, and for $0 < i \leq |V|$, $P[i] - P[i-1]$ gives the number of neighbors for v_i. Then, the indices of the neighbors of v_i are placed in the array N, starting at the $P[i-1]$-th element and ending at the $P[i]$-th element. In other words, assuming that v_i's neighbors are $\{n_1, n_2, ..., n_k\}$ and that $k = P[i] - P[i-1]$, then $N[P[i-1]{:}P[i]]$ will be $\{\mathcal{M}[n_1], \mathcal{M}[n_2], ..., \mathcal{M}[n_k]\}$, with $\mathcal{M}[n_j]$ denoting the index of address n_j. $T_{\min}$ and $T_{\max}$ are arrays used to store timestamp data, with each entry corresponding to an entry in array N. Continuing with the previous notation, then $T_{\min}[P[i-1]{:}P[i]]$ stores $\{\min \tau_{v_i,n_1}, ..., \min \tau_{v_i,n_k}\}$ and $T_{\max}[P[i-1]{:}P[i]]$ stores $\{\max \tau_{v_i,n_1}, ..., \max \tau_{v_i,n_k}\}$. Here $\min \tau_{v_i,n_j}$ and $\max \tau_{v_i,n_j}$ represent the minimum and maximum of elements in τ_{v_i,n_j}, respectively. Readers may recall the definition of $\tau_{u,v}$ in the MFA Formulation section. Only the maximum and minimum timestamps are retained to further streamline the MFA, which are used for the pruning.

Why is MFA Lightweight. A potential concern is whether the MFA data structure is truly lightweight and whether its storage overhead is tolerable. The answer is affirmative, owing to two key design choices. ① *Edge compression.* As shown above, the MFA structure compresses multiple transfers between the same pair of addresses into a single edge. This compression significantly reduces the total number of edges the system needs to store, thereby lowering storage costs. ② *Compact representation.* For each edge, the source address is stored in array P and the destination address in array N, each requiring only one integer cell. Moreover, multiple edges with the same source address share a single entry in P, avoiding duplication. Consequently, each edge requires on average less than or equal to two integer slots. Taken together, these two factors—① compressed edge representation and ② extremely compact per-edge storage—make the MFA structure highly space-efficient, as verified by the empirical evaluation in Section 4.1.

Suspicious Topology Construction. In the Graph Search phase, MFTracer constructs the suspicious topology G_{s} by performing graph search from each victim address A_{vic} on the MFA. The search traverses downstream reachable nodes and edges, adding them to G_{s}. It is worth noting that the search begins by mapping the victim address set to an index set using $\mathcal{M}$, and all subsequent operations are performed in the index space, eliminating further access to $\mathcal{M}$. Hence, accessing the hash map imposes almost no overhead. The final output is $G_{\text{s}} = (V_{\text{s}}, \mathcal{F}_{\text{s}})$, where V_{s} is the set of address indices and $\mathcal{F}_{\text{s}} : V_{\text{s}} \times V_{\text{s}} \to \{0, 1\}$ encodes edge existence.

During the graph search process, retrieving the neighbors of a given address is the most fundamental unit of operation. The specialized data structure of MFA is optimized for this operation. Specifically, for an address with index i, all its neighbors can be accessed directly through $N[P[i-1] : P[i]]$. *At the assembly instruction level, this access requires only two memory address lookups*: ① the first contiguously reads two integers at the address of the $(i-1)$-th element in the integer array P (calculated as the base address of P plus i times the integer size); ② the second reads a contiguous block of bytes at the address of the $P[i-1]$-th element in the integer array N (calculated as the base address of N plus $P[i-1]$ times the integer size). This assembly-level optimization significantly improves graph search performance on MFA.

Suspicious Topology Pruning. MFTracer employs an efficient pruning strategy by leveraging timestamp information in the MFA during graph search. For a money flow path $a_1 \to a_2 \to$

$\cdots \rightarrow a_n$, with t_i denoting the timestamp between address a_i and a_{i+1}, a valid flow requires that $\forall\ 1 < i < n,\ t_i \geq t_{i-1}$. This ensures temporal consistency along the path. We formally prove that a valid flow in G_{mfa} must satisfy the following inequality. The proof can be found in Appendix B.

$$\max \tau_{a_i,a_{i+1}} \geq \max_{0<j<i} \min \tau_{a_j,a_{j+1}}.$$

During graph search, MFTracer uses the above time-based inequality for pruning when visiting neighbors. Only those neighbors $\{a_{i+1}\}$ that satisfy the inequality are visited from address a_i. Similar to the neighbor access mechanism in graph search, the data structure of MFA optimizes timestamp access at the assembly level, ensuring efficiency in this pruning strategy. The left-hand value of the inequality can be accessed directly through $T_{\max}[P[i-1] : P[i]]$ with only two memory addressing operations. The right-hand represents the maximum of $\min \tau_{a_j,a_{j+1}}$, where the a_js are addresses in the current path between the current in-visit address a_i and the source address. After each visit, a_{i+1} is appended to the tail of the current path, and the right hand value can be immediately updated by accessing $T_{\min}[P[i-1] : P[i]]$.

For an intuitive understanding, we use Fig. 3 as an example. The search starts from u_0 to traverse downstream paths. After visiting u_2, the current path is $u_0 \rightarrow u_1 \rightarrow u_2$, and the right-hand value of the inequality is $\min \tau_{u_0,u_1} = 800$. Then, when it comes to visit v_1, the left-hand value is $\max \tau_{u_2,v_1} = 600$ and it does not satisfies the inequality. Thus v_1 is pruned. In contrast, when visiting u_3, the left-hand value is $\max \tau_{u_2,u_3} = 820$ that satisfies the inequality, and thereby u_3 is not pruned.

3.4 Encoding Scheme

Motivation. In the Encoding phase, MFTracer encodes and stores MFA and detailed money flows onto disk (❹ in the left part of Figure 2), establishing the infrastructure required for the Tracing Process. To optimize the organization of the infrastructure, our encoding scheme is designed to achieve two primary objectives: ① The MFA graph should be appropriately partitioned so that, during the Tracing Process, only the necessary portions related to the current fund flow tracing task are loaded into memory, reducing data processing and memory pressure. ② The system should be able to quickly retrieve the corresponding detailed money flows from disk into memory based on the addresses and transaction relationships indicated by the suspicious topology, ensuring the efficiency of the Tracing Process. To this end, we develop an encoding scheme based on Pebble[34]'s LSM-Tree-based [84] on-disk key-value storage. We opt for key-value storage in our infrastructure for two key reasons. First, the key-value structure allows data to be partitioned using different keys and stored as subsets on disk. Compared to other storage solutions such as relational databases, key-value storage offers more direct and efficient access to these subsets. Second, due to the lightweight nature of MFA, intensive computation and data processing are conducted in memory, eliminating the need for frequent disk access. The straightforward disk access methods provided by key-value storage (e.g., `get`, `scan`) are sufficient for our needs. This choice reduces system complexity and improves system stability. Next, we introduce the partitioning and encoding strategy.

Partitioning Strategy. MFTracer partitions fund flow data based on time, segments them into time-sequenced subsets. During MFA construction, the fund flows derived from the Tx-Granularity Analysis are sorted by the block numbers (heights) of their corresponding transactions. The money flows from every K consecutive blocks are grouped into the same subset, and MFTracer constructs an MFA for each subset. As a result, when analyzing a crime event, MFTracer can selectively load into memory, at granularity K, only the MFA subsets corresponding to blocks greater than or equal to the crime event's block number, while keeping older subsets on disk and excluding them from computation. This strategy significantly reduces computation and storage overhead.

Encoding MFAs. For an MFA slice $G_{\text{mfa}}^{(m)}$ constructed using money flows with block numbers between mK and $(m+1)K$, MFTracer uses the following key-value pairs to encode it.

$$\{\mathbf{Key}_{G_{\text{mfa}}^{(m)}} \rightarrow \mathbf{Value}_{G_{\text{mfa}}^{(m)}}\} = \{(k_g, m) \rightarrow \text{M}(\text{structure}(G_{\text{mfa}}^{(m)}))\}.$$

Here, k_g is a specific byte used to indicate the type of key. And structure$(G_{\text{mfa}}^{(m)})$ is the data structure of $G_{\text{mfa}}^{(m)}$ described in Section 3.3. $\text{M}(\cdot)$ refers to MessagePack [59], a highly efficient binary encoding method. Evaluation on large-scale data [64, 85] shows that it outperforms JSON and Protocol Buffers in both compacting and decoding efficiency for hash tables and arrays, making it well-suited for the data structure of MFA.

Encoding Money Flows. As explained in 3.1 (Fig.2 right ❸), after graph search, MFTracer loads into memory the *suspicious flows*—detailed fund transfers between address pairs (u, v) in the suspicious topology G_{s}, where $\mathcal{F}_{\text{s}}(u, v) = 1$. To improve the efficiency of this retrieval, we design the following key-value format to encode money flows.

$$\{\mathbf{Key}_{F_{u,v}^{(m)}} \rightarrow \mathbf{Value}_{F_{u,v}^{(m)}}\} = \{(k_f, m, \mathcal{M}_m[u], \mathcal{M}_m[v]) \rightarrow \text{M}(F_{u,v}^{(m)})\},$$

Here, k_f is a specific byte used to indicate the key type. $F_{u,v}^{(m)}$ denotes the set of all money flows from address u to address v within blocks numbered from mK to $(m+1)K$ (i.e., the m-th subset produced by the partitioning strategy). $\mathcal{M}_m$ is the hash map in the data structure of the MFA $G_{\text{mfa}}^{(m)}$. $\mathcal{M}_m[u]$ and $\mathcal{M}_m[v]$ denote the indices of address u and v in the index space of $G_{\text{mfa}}^{(m)}$, respectively. This encoding strategy enables efficient retrieval of *all* money flows between two addresses within a specific block range directly using the address indices in V_{s}, eliminating the dependency on the construction of the reverse hash table $\mathcal{M}_m^{-1}$. This helps accomplish the second objective.

Enabling Parallel Search. Moreover, the encoding scheme naturally partitions MFAs by time, enabling MFTracer to perform searches on relevant slices independently. During graph search, MFTracer runs *parallel* searches across multiple MFA slices to construct the corresponding suspicious topologies, further reducing search time and improving tracing efficiency. A complete description of the parallelization strategy is provided in Appendix C for interested readers.

3.5 Money Flow Simulation

The purpose-built MFA structure and encoding scheme enable MFTracer to efficiently retrieve large volumes of fund transfer data, laying the foundation for the Money Flow Simulation phase. In this phase, MFTracer executes an effective fund flow simulation algorithm that sequentially models how illicit funds propagate from source addresses to downstream over time. This enables the system to pinpoint downstream illicit flows and further remove irrelevant addresses from the suspicious topology to refine the output. The algorithm relies on detailed fund flow information (i.e., suspicious flows) to guarantee simulation accuracy. Specifically, MFTracer first retrieves all suspicious flows F_{s} into memory following the method described in 3.4. Then, the system simulates suspicious flows in temporal order to trace the movement of siphoned funds from source addresses. In the simulation, by filtering out irrelevant addresses uninvolved in the flow, MFTracer finally identifies the addresses used to launder illicit funds and reconstructs the illicit money flow topology.

Details. Algorithm 2 presents the details. The simulation takes the victim addresses A_{vic} and suspicious flows F_{s} as input and outputs the final illicit money flow topology G. During the simulation, MFTracer maintains a balance table B (line 3) to track the amount of illicit funds held by each downstream address. These illicit funds originate from upstream victim addresses (line 7). MFTracer iterates through the suspicious flows in time order and updates B to simulate the movements of illicit funds in the downstream (line 14, 15). The balance of each address determines

Algorithm 2 Pseudocode of Money Flow Simulation

Input: Suspicious Flows F_s, Victim (Source) Address Set A_{vic}.
Output: Illicit Money Flow Topology $G = (V, \mathcal{F})$.
Config. Parameter: Reserve Ratio ϵ.

```
1:  (V, F) ← newEmptyTopology()          ▷ first initial the output G as an empty topology
2:  V ← V ∪ A_vic                        ▷ initial the topology's address set V with source addresses
3:  B ← newEmptyMap[Address → Balance]()
      ▷ B[u] gives the balance of address u during the simulation.
4:  for flow f_i in F_s.SortedByTime() do   ▷ iterate the suspicious flows of F_s in time order
5:      if f_i.from ∉ V then continue   ▷ the from address haven't seen illicit funds, filter out f_i
6:      end if
7:      if f_i.from ∈ A_vic then
8:          outflow ← f_i.value                          ▷ fully release the illicit money
9:      else
10:         reserve ← B[f_i.from] * ε                    ▷ compute the reserve requirement
11:         outflow ← min(B[f_i.from] − reserve, f_i.value)
12:     end if
13:     if outflow ≥ Threshold then
14:         B[f_i.from] ← B[f_i.from] − outflow
15:         B[f_i.to] ← B[f_i.to] + outflow               ▷ handle the flow in the simulation
16:         V ← V ∪ {f_i.to}                             ▷ mark f_i.to as laundering address
17:         F(f_i.from, f_i.to) ← 1                      ▷ update the topology
18:     end if
19: end for
```

the upper limit for the amount of illicit funds it can provide (line 11). If the from address of a flow f_i is unable to provide an amount of illicit funds exceeding a small threshold, MFTracer filters f_i out (line 13). And consequently, f_i's to addresses, which do not handle the movements of illicit money, is not marked as a laundering address. Moreover, if a flow's from address has not seen illicit funds, MFTracer directly filters it out (line 5).

For intuitive understanding, we continue to use Fig.3 as an example. First, the irrelevant flow of time 100 from u_1 to u_2 occurs before u_1 is marked as a laundering address. It is filtered out by matching the condition in line 5. Second, y does not exist in the suspicious topology. Therefore, the flow from y to u_2 that provides the clean money transferred from u_2 to v_2 is not simulated and does not increase u_2's balance. Illicit funds are transferred to u_3, u_4 and u_5, and u_2 holds insignificant or no illicit funds at time 2010. Thus the $outflow$ value (line 11) for the flow from u_2 to v_2 is smaller than the threshold, and thereby v_2 is filtered out (line 13). In addition, the simulation algorithm is robust to aggregation of mixed (clean + illicit) balances. Concretely, suppose the transfer $u_1 \rightarrow u_2$ at time 100 carries clean funds, while a later transfer at time 810 carries illicit funds, and these amounts are aggregated and moved to u_2 at time 810 in a single operation. Algorithm 2 will still correctly simulate the correct illicit amount transferred from u_1 to u_2 and tags u_2 as laundering-related. This correctness follows from the constraint in line 11, which ensures that the simulated illicit outflow does not exceed the illicit balance associated with u_2 (i.e., the amount of illicit funds actually received by time 800). This constraint prevents the simulation from attributing more illicit value to transfers than is physically possible.

Moreover, inspired by reserve requirements [58, 62] in financial systems, we propose a reserve mechanism to control liquidity distribution in the downstream. By adjusting the reserve ratio ϵ

(line 10), MFTracer can achieve improved performance. Interested readers may consult Section 5.1, Appendices D and E for detailed theoretical analysis and empirical validation. And we also include a guide in Section 5.1 for practitioners to set a proper threshold (line 13) in practice.

4 Evaluation

We implement MFTracer with 12,381 lines of Golang code (source code can be found here) and evaluate our system along two key dimensions: **Efficiency** and **Effectiveness**. As discussed in Section 1, tracing systems must meet the efficiency demands of real-world forensic investigations, be practically deployable in high-throughput blockchain environments and capable of processing massive fund transfer data and frequent cybercrimes to enable timely intervention. Effectiveness measures the system's ability to comprehensively and accurately trace illicit flows. A tracing system must ensure that no critical link in illicit fund flow is overlooked, which is essential for supporting law enforcement and compliance efforts. Together, these metrics assess the system's real-world readiness. In the following, we answer six research questions **RQ1 - RQ6** to guide our evaluation.
Environment Setup. We conducted the evaluation on three identical machines. Multiple machines were used to enable distributed and multi-node deployments of Neo4j [29] and Elasticsearch [13], ensuring that both systems were tested under optimal conditions for a fair efficiency comparison (see **RQ3**). Each machine has two Intel Xeon Gold 5318Y (24-core, 2.1GHz, hyper-threading disabled), 128 GB DRAM, and three 1 TB SSDs, running Ubuntu 22.04.1 LTS (Kernel v5.15.0).

4.1 Efficiency

To evaluate the efficiency of MFTracer under realistic conditions, we selected Ethereum as the target blockchain due to its status as the most representative EVM-compatible platform with the richest DeFi ecosystem (efficiency analysis on other chains covered in Section 5.4). This choice ensures that the evaluation reflects the complexity and scale commonly encountered in real-world investigations. We built MFTracer's infrastructure using *over two years of full* Ethereum transaction data (August 8, 2022 to September 7, 2024; blocks 15,300,000 to 20,700,000), totaling 831.7 million transactions. This large-scale data serves as a rigorous benchmark for assessing infrastructure performance. We set the partition size K (as defined in 3.4) to 100,000, dividing the data into 54 subsets, each spanning 100,000 blocks (about 15.4 million transactions per subset) for MFA construction. On this foundation, we evaluate the efficiency of two core components: ① the infrastructure building process, and ② the data retrieval operations conducted on top of this infrastructure. This approach enables us to evaluate MFTracer's ability to operate in real-world high-throughput blockchain environments. In the following, we answer three research questions.
RQ1: How efficient is the Building Process of MFTracer? MFTracer's data retrieval functionality is supported by infrastructure constructed during the Building Process. The performance of this process is crucial for ensuring timely infrastructure updates. We assessed the infrastructure construction time for the 54 subsets. In Figure 4, the dark gray bars show the time required for the phases of Tx-Granularity Money Flow Analysis (Alg. 1) and MFA Construction, while the light gray bars depict the time taken for encoding and writing to disk (❹ in Figure 2). On average, each subset takes **394.8 seconds** to process, while it takes about 11.8 days to produce these blocks. This means MFTracer builds infrastructure about **2,600**× faster than the block generation rate. Such efficiency allows MFTracer to handle real-time updates with ease, making it suitable for both historical analysis and ongoing investigations.
RQ2: Is the MFA lightweight enough to be stored in memory? What is the storage efficiency of MFTracer? As discussed in 3.3, MFA is implemented as a lightweight data structure that supports in-memory graph search and pruning without relying on disk I/O. To verify whether the MFA can indeed be efficiently held in memory to improve the overall performance of data

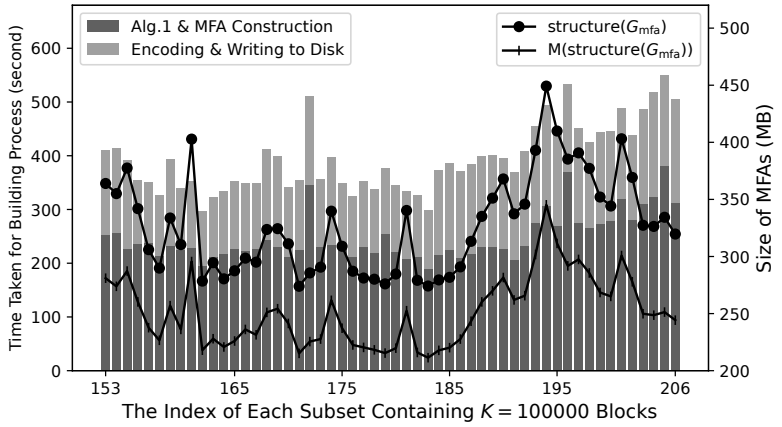

Fig. 4. Time cost for the Building Process and the corresponding MFA size of each subset. On average, each subset (~15.4 million transactions) takes 394.8 seconds to process. The MFA instances constructed from two years of transaction data occupy 17.2 GB of memory in total. The average size of each instance is 327.1 MB.

retrieval, we measured the footprint of all 54 MFA instances. As shown in Fig.4, the structure(G_{mfa}) line indicates raw size, while M(structure(G_{mfa})) shows the size after binary encoding. The full set of MFAs, built from two years of transaction data, occupies 17.2 GB in total, averaging 327.1 MB per subset. Even conservatively estimated at 400 MB each, 90 GB of RAM can hold more than 230 subsets—enough to cover all the transaction data on Ethereum. The evaluation results demonstrate MFA's lightweight nature and applicability for in-memory use.

Additionally, including full detailed money flow data (i.e. the output of Alg. 1) and all MFA instances, the entire infrastructure uses 107.1 GB on disk (about 1.98 GB per subset). In comparison, constructing a graph storage using Neo4j on the same data (i.e. the output of Alg. 1) results in a disk usage of 1010.9 GB. (To ensure a fair comparison, we used a single-machine deployment to minimize Neo4j's storage overhead.) Moreover, the graph constructed using two leading in-memory graph databases, Memgraph [25] and RedisGraph [38], each exceeded 400 GB in size, rendering them impractical for large-scale blockchain data. Even the combined memory of our three machines was insufficient to hold such graphs. In summary, MFA's lightweight design yields a **23.3**× to **58.7**× improvement in storage efficiency over the mainstream graph databases. Even with full data included, MFTracer is still **3.7**× to **9.4**× more efficient in terms of storage.

RQ3: How efficient is MFTracer in data retrieval? Retrieval speed directly impacts how long MFTracer takes to complete a tracing task, and is critical to enable timely intervention. We evaluated MFTracer's data retrieval performance by deploying it in realistic scenarios and executing real-world tracing workflows. We measured the time from providing MFTracer with source addresses to the completion of the graph search and retrieval of all suspicious flows into memory. Based on this, we compute its fund flow retrieval speed.

To ensure that the test data was **unbiased** and representative of **actual** blockchain data distributions, we adopted the following strategy: In each round, we randomly selected a source address that had not been visited in previous graph searches, and used it to initiate a new tracing task, repeating until all addresses across the two-year data were covered. As a baseline comparison, we repeated the test tasks with Neo4j [29] and Elasticsearch [13], replacing MFTracer's on-disk infrastructure with these leading disk-backed storage systems. To ensure a fair and rigorous comparison, both systems were deployed in their recommended *distributed* configurations across three identical physical

Table 1. Data retrieval speed comparison between MFTracer's data retrieval infrastructure and two leading storage backends that support graph data organization and query.

	MFTracer	Neo4j	Elasticsearch
Retrieve Flows (per ms)	**90.02**	6.37	0.30
Speedup	–	14.13×	300.0×

Table 2. Overview of the top-level cybercrime incidents. "Money" refers to the amount of money siphoned from victims. "Addresses" is the number of blockchain addresses involved in money laundering. "Records" refers to the number of illicit money flow records.

	Time	**Money**	**Addresses**	**Records**
TxPhish [35]	2024/08-2024/09	$54M	151	292
LIFI [3]	2024/07-2024/09	$11M	93	135
Atomic [1]	2023/06-2023/07	$10M	389	1080
HB [2]	2023/01-2023/02	$50M	469	3887
XScam [108]	2022/06-2022/11	$583k	837	1307

machines with purpose-built indexing strategy, allowing them to fully leverage their designed scalability and performance optimizations. Elasticsearch was set up as a three-node cluster with properly distributed shards and JVM heap capped under 32 GB to avoid GC issues [22], following official best practices [23]. Similarly, Neo4j used Causal Clustering [96] with one core server per machine. These setups ensured both systems operated under optimal conditions.

Table 1 shows the data retrieval speed (i.e., the average number of flows retrieved per millisecond). Compared to Neo4j and Elasticsearch, MFTracer achieves a **14.1**× and **300.0**× speedup, respectively. In practical cybercrime investigations (such as [1, 43, 108, 110]), especially with increasing laundering depth as discussed in **Challenge-II**, the number of fund flows to analyze often exceeds 10^7. Neo4j takes over four hours just to load the data, rendering it unsuitable as a backend for operational use. MFTracer's optimized infrastructure reduces this to less than 15 minutes. This level of performance underscores its practicality and suitability for real-world forensic use.

A potential concern is that, in the retrieval-efficiency evaluation, the chosen source addresses are not all known illicit addresses, and that illicit flows may exhibit different patterns from normal traffic. One might therefore ask whether the evaluation realistically reflects the system's performance in actual criminal investigations. We argue that it does. MFTracer's retrieval stage is designed to fetch *suspicious flows* that are covered by the suspicious topologies (❸ in the right part of Fig.2). These suspicious flows are a superset of illicit flows, and normal (benign) flows included in the suspicious flows are pruned during simulation (see Section 3.3 and 3.4). As mentioned above, in operational cases the volume of suspicious flows to retrieve and simulate commonly reaches 10^7. Because normal flows vastly outnumber illicit ones within the suspicious flow set—usually by several orders of magnitude—the retrieval workload is overwhelmingly dominated by normal flows. Consequently, the workload of retrieving suspicious flows is effectively identical to that of retrieving normal flows. Therefore, our evaluation strategy provides a realistic measure of retrieval cost. For completeness, we also repeated the experiments using a large set of confirmed laundering addresses (drawn from the dataset described in Section 4.2); these runs produced consistent results.

4.2 Effectiveness

Large-scale Ground Truth Dataset. Currently, there are no truly deployable baseline systems capable of operating on the massive-scale data typical of high-throughput blockchain environments and automatically handling real-world cybercrime investigations on EVM-compatible platforms, leaving us unable to demonstrate the effectiveness of MFTracer through comparisons with baseline methods. Furthermore, no publicly available dataset exists for such evaluation. To address this gap and enable rigorous evaluation of MFTracer, we collaborated with investigators from a leading security firm[1] to build LaunderNetEvm41. It is the *first* publicly available and, to date, the *largest* dataset of ground-truth laundering flows on EVM-compatible blockchains. In total, it contains 1,939 accounts involved in money laundering activities and 6,701 detailed illicit fund flow records—*an order of magnitude larger* than any previously disclosed (Bitcoin) illicit flow dataset, collectively covering over **US $125 million** in stolen assets. Categorized by victim addresses, the dataset includes 41 laundering cases drawn from some top-level cybercrime incidents between 2022 to 2024 (Table 2). Its cybercrime coverage spans phishing, smart contract exploits, private-key theft, social-engineering scams, etc., reflecting the modern threat surface. This diversity allows evaluating the generalizability and robustness of tracing systems under various real-world laundering scenarios.

To guarantee the reliability of each entry, seven experts from a leading blockchain security firm[1] and the broad security community annotated and cross-validated it using their domain-specific insights. The dataset was created in strict accordance with real-world forensic investigation workflows and a consensus-driven review process. Details of the dataset creation process can be found in the dataset repository or in Appendix F. On average, annotating and verifying a single record—including accurately extracting fund flows from massive transaction traces, understanding smart contract logic, interpreting DeFi protocol behavior and assessing laundering strategies—takes approximately 3-4 minutes. Cumulatively, the over six thousand records result in more than **350 hours** of specialized manual effort. To foster further research in this field, we have open-sourced the dataset along with address labels and case annotations with detailed visualizations. We hope this will serve as a stepping stone for reproducible evaluation and future benchmarking in this area.

Formally, the dataset can be modeled as a weighted graph $G_g = (V_g, \mathcal{M}_g)$, where V_g represents the set of addresses used for money laundering, and $\mathcal{M}_g : V_g \times V_g \rightarrow [0, +\infty)$ indicates the volume of illicit fund flows between them. The address set V_g is composed of three subsets: $V_g = A_{vic} \cup V_{inter} \cup V_{terminal}$. A_{vic} includes the victim addresses where illicit funds originate. V_{inter} contains addresses used for intermediate transfers. And $V_{terminal}$ represents endpoints where the funds are ultimately cashed out. For addresses $u, v \in V_g$, $\mathcal{M}_g(u, v)$ specifies the amount of illicit money transferred from address u to v. Then, we evaluate MFTracer by providing it with the source address set A_{vic} as input and analyzing the generated illicit money flow topology $G = (V, \mathcal{F})$. Readers may recall the formal definition of G provided in Section 2.3. Our evaluation of its real-world forensic effectiveness is guided by research questions **RQ4 - RQ6**.

Overall Performance. MFTracer achieves a flow coverage of 94.09% in the real-world laundering crimes. In terms of asset value, it successfully traces the final destinations of 96.27% of the stolen funds. This high coverage demonstrates its effectiveness in tracing illicit flows end-to-end and providing actionable forensic evidence that links upstream sources and downstream laundering endpoints. Furthermore, MFTracer newly identified 686 addresses involved in money laundering and 4183 previously undiscovered illicit fund movements. We have reported these new findings to the security community to support forensic and threat intelligence research. The complete list of addresses and transactions can be found here. In total, MFTracer reconstructed complete flow trajectories and provided strong evidence for illicit assets worth up to **$120.9 million**, highlighting its practical impact. We hope that this will contribute to the security of DeFi ecosystem.

Table 3. For multi-perspective presentation, evaluation results are disaggregated by metrics, parameters and top-level incidents. F.C. refers to FlowCoverage. T.C. refers to TerminalCoverage. P means Precision. ϵ is the reserve ratio discussed in Section 3.5 and 5.1. More values of ϵ are presented in Appendix E.

	TxPhish			**LIFI**			**Atomic**			**HB**			**XScam**		
Metric	F.C.	T.C.	P	F.C.	T.C.	P	F.C.	T.C.	P	F.C.	T.C.	P	F.C.	T.C.	P
$\epsilon = 0.00$	0.9534	0.9255	0.8455	0.9948	0.9953	0.6212	**0.9621**	0.9634	0.8118	**0.9713**	0.9995	1.0	0.8231	0.7382	0.7401
$\epsilon = 0.05$	0.9582	0.9424	0.8320	0.9930	0.9970	0.5222	0.9410	0.9273	0.8290	0.9699	0.9995	1.0	**0.8335**	0.7477	0.7185
$\epsilon = 0.15$	0.9601	0.9326	0.8319	**0.9954**	0.9974	0.4855	0.9187	0.8767	0.8207	0.9711	0.9995	1.0	0.7967	0.6912	0.7162
$\epsilon = 0.35$	**0.9618**	0.9517	0.8166	0.9929	0.9968	0.4555	0.8915	0.8042	0.8297	0.9692	0.9968	1.0	0.7754	0.6698	0.6958

RQ4: How effective is MFTracer in uncovering *how* illicit money moves? A core requirement for tracing tools is the ability to expose how stolen assets actually propagate on-chain. To answer this, we assess the *FlowCoverage* of MFTracer, defined as the proportion of illicit fund flows that are successfully captured in the output topology G generated by the system.

$$\text{FlowCoverage} = \frac{\sum_{u,v \in V \cap V_g} \mathcal{F}(u,v) * \mathcal{M}_g(u,v)}{\sum_{u,v \in V_g} \mathcal{M}_g(u,v)}.$$

A higher coverage rate means the tracer has recovered a larger share of the real laundering paths. Table 3 reports the results. With the reserve-ratio parameter left at 0%, MFTracer already attains an average flow coverage of 94.09%, convincingly revealing the trajectories of illicit funds—often regarded as the most critical challenge of laundering forensics. Tightening the reserve-ratio further nudges coverage upward, yet overall flow coverage stays above 90% under all tested settings. These numbers confirm that MFTracer consistently reconstructs almost the entire flow graph, giving investigators a near-complete picture of the money laundering scheme.

RQ5: Is MFTracer effective in tracing *where* illicit money is eventually directed? In addition to overall path coverage, another critical concern in real-world forensic investigations is how well a tracing system can identify the endpoints of illicit fund transfers—formally defined as

$$\text{TerminalCoverage} = \frac{\sum_{u \in V \cap V_g, v \in V \cap V_t} \mathcal{F}(u,v) * \mathcal{M}_g(u,v)}{\sum_{u \in V_g, v \in V_t} \mathcal{M}_g(u,v)},$$

where V_t refers to V_{terminal}. The denominator indicates the total volume of illicit money flowing to the terminal addresses and the numerator is the portion that MFTracer successfully traces. The terminal addresses correspond to centralized service providers where criminals attempt to cash out stolen assets. The ability to trace and identify these endpoints significantly strengthens the evidentiary basis for requesting cooperation from centralized entities during investigation and recovery efforts. The test results are shown in Table 3. MFTracer achieves an average address-level terminal coverage of 92.43%. When measured by asset value, it successfully identifies the endpoints of 96.27% illicit funds, meaning virtually all high-value outflows are captured. This high terminal fidelity is essential for issuing subpoenas, freezing accounts, and ultimately recovering funds. In short, MFTracer not only maps *how* dirty money moves but also *where* it stops, supplying investigators with actionable targets for downstream intervention.

RQ6: What is the precision of MFTracer? To answer this, we measure how many of the identified addresses are truly involved in laundering activities. The metric is Precision = $|V \cap V_g|/|V|$. Precision reflects the proportion of correctly identified laundering addresses among all addresses flagged by the system. It affects the manual effort required during legal verification, where human analysts must examine the output as potential evidence. A higher precision reduces the time

and cost of forensic review. MFTracer delivers an average precision of 80.37%—enough to be operationally useful while still preserving near-exhaustive coverage of illicit flows. In practice, that means analysts need to only filter a small fraction of the returned entities, shrinking the evidence-building effort from *hundreds of manual hours* to just *tens of minutes*.

Take some laundering cases in LIFI as a stress-test: the laundering path stretched over 21 hops and potentially entangled about 950,000 addresses via extensive interaction with popular on-chain services. Even under this extreme fan-out, MFTracer's fine-grained flow analysis, pruning and simulation logic retained > 99% flow coverage yet kept precision above 60%. Because every item submitted to court must ultimately be checked by a human, completeness of coverage (low false-negative rate) outweighs perfect purity for an automated tracer. Over-optimising for precision would risk *omitting* critical evidence links and undermine the investigation. MFTracer therefore aims for a forensics-friendly trade-off: broad coverage to ensure no key flow is lost, coupled with sufficient precision to keep the review queue manageable—an alignment that our field partners confirm meets real-world investigative needs. Beyond the quantitative results, we investigated the underlying reasons for false positives. Details and mitigation strategies are discussed in Section 5.3.

5 Discussion

5.1 Guidance: Setting Proper Parameters in Practice

Partition Size K. The partition size K controls the number of blocks included in each subset during infrastructure construction. Increasing K enlarges each MFA instance, covering more transactions and reducing the total number of instances. Since MFTracer is designed to execute parallel graph searches (detailed in Appendix C) over multiple MFAs, decreasing K (i.e., producing more instances) enables better utilization of available hardware threads and can accelerate tracing. More instances also provide finer-grained temporal slicing, preserving more precise maximum and minimum timestamp information (see Section 3.3, *MFA Data Structure*) and thereby improving time-based pruning, which reduces the number of suspicious flows that must be loaded into memory before the money flow simulation stage. However, there are trade-offs. Each MFA uses a hash map to map addresses to integers and store edges in integer space. Fewer MFA instances (i.e., larger K) reduce the number of times active addresses—often spanning long periods and generating many transactions—are redundantly stored across hash maps. Similarly, the edge-compression mechanism of MFA (see Section 3.3, *Why is MFA Lightweight*) benefits from larger K, since more transfers can share a single integer entry in array P, further improving storage efficiency. In practice, practitioners can tune K based on the deployment environment. With abundant hardware threads, using a smaller K helps to fully exploit parallelism; with limited threads, a larger K reduces redundancy and alleviates memory and storage pressure. Empirically, maintaining each MFA instance between 200 MB and 500 MB yields an effective balance.

Reserve Ratio ϵ. The reserve ratio is a parameter used in the fund-flow simulation algorithm (Section 3.5). It controls the breadth and depth of the topologies produced by the simulation: increasing the reserve ratio associates more outflows with the same inflow, concentrating liquidity closer to the source addresses; decreasing it shifts liquidity farther away, thereby increasing topology depth. A theoretical analysis of how reserve ratio affects topology width and depth is provided in Appendix D. Because this parameter operates during the tracing (simulation) stage rather than during infrastructure construction, practitioners can run the same forensic task with multiple reserve-ratio settings and select the most appropriate topology by comparing outcomes. In practice, an excessively large reserve ratio yields inflated intermediate balances and *traps* large fractions of illicit funds in intermediate laundering addresses that do not progress downstream; this can be detected by inspecting terminal nodes (out-degree 0) to see whether they correspond to plausible

cash-out services. Conversely, an overly small reserve ratio produces overly *sparse* topologies that contain too few addresses and have cash-out endpoints very far from the source, indicating that the reserve ratio should be increased. Empirical results (Appendix E) show that lowering the reserve ratio tends to increase precision at the cost of flow coverage, while raising it increases coverage but also raises the false-positive rate. Empirically, a reserve ratio in the 5%–10% range yields near-optimal overall performance.

Simulation Threshold. Line 13 of Algorithm 2 implements a per-flow threshold. A flow is discarded when its illicit amount (as computed in line 11) falls below this threshold. This parameter controls the *fineness (granularity)* of the resulting illicit-flow topology: lowering the threshold includes smaller-value illicit flows and yields a finer-grained topology, whereas raising it coarsens the topology by filtering out lower-value flows. The granularity is not unilaterally desirable to maximize: for example, in a case involving on the order of $1 billion in stolen funds, movements of only a few cents or a few dollars are typically immaterial to investigators. Practically, practitioners care more about how substantial sums are moved and cashed out—information that matters for asset recovery and prosecution. A properly chosen threshold thus reduces the manual review burden for court evidence: an excessively small threshold produces many low-value "noise" flows and increases review effort, whereas an overly large threshold may omit legally important transfers and weaken evidentiary value. Because this parameter, like the reserve ratio, operates during the tracing stage, it can be tuned at low cost for each investigation. Empirically, we find that setting the threshold 0.01%–0.1% of the total implicated amount offers a good balance between coverage and reviewer workload. *Adversarial note:* Some adversaries may deliberately split illicit funds into many micro-transactions at very low rates. In such cases a large threshold will cause the reconstructed topology to contain far fewer addresses/edges and fail to cover illicit flows adequately. If this behavior is detected, simply lowering the threshold and rerunning the tracing process recovers the missed micro-flows. Reasonable automated diagnostics include: ① monitoring the number of nodes/edges in the result and flagging unusually sparse topologies; ② checking the fraction of total asset value captured by terminal addresses in the topology; and ③ detecting heavy tail distributions of addresses' transaction counts that suggest micro-structuring.

5.2 Evasion Techniques and Countermeasures

MFTracer operates based on execution traces recorded by EVM and has a small trusted computing base, making it inherently more robust against evasion attempts. Here, we discuss potential evasion techniques that occur under normal network conditions. For threats that compromise the blockchain itself (e.g., consensus splits, large-scale DDoS), we refer readers to prior studies [51, 60, 63, 87, 102], which provide detailed discussions on attack vectors and defense mechanisms in such contexts.

① *Non-Fungible Tokens (NFTs).* Unlike fungible tokens, NFTs are unique and indivisible assets identified by globally unique public IDs. Criminals may attempt to exploit NFT transfers for laundering to evade detection by MFTracer. But each NFT transfer triggers a standardized Transfer event (ERC-721 [17] or ERC-1155 [14]) that includes the NFT's public identifier. This makes *each* NFT's on-chain movement easily traceable and thus less attractive for money laundering. In addition, mature NFT monitoring tools (e.g., OpenSea [32], NFTScan [30], Zapper [46]) are widely available. MFTracer can integrate with an ERC-721/1155 log decoder or existing monitoring tools to effectively address NFT-based laundering schemes.

② *Decentralized Cross-Chain Bridges.* Unlike centralized services subject to regulatory oversight, decentralized cross-chain bridges often lack legal entities, making them attractive for illicit fund transfers. Criminals may exploit them to move assets to new ledgers and evade tracing. However, MFTracer is designed with inherent multi-chain deployability that can efficiently model bridge interactions as atomic in/out edges in the MFA. When coupled with a specialized bridge transaction

parser (e.g., MetaSleuth [26], OKLink [31], Pulsar Finance [37]), MFTracer can prevent evasion through bridges and reconstruct the full path across networks without altering its core.

③ *Zero-knowledge Mixing Services.* Zero-knowledge mixers like Tornado Cash [40] cryptographically sever the link between input and output addresses, preventing any *on-chain* tracer like MFTracer from following assets through the mixing pool. Under AML compliance requirements, the widespread imposition of rigorous off-chain scrutiny on mixers across jurisdictions and the refusal of service providers to handle funds originating from them have served as a deterrent to their usage. There also exists a distinct line of off-chain research [57, 66, 92, 104] that focuses on recovering the linkability through social media activity, network traffic and other side-channel signals. Our work is orthogonal to these off-chain de-anonymization efforts and we address distinct layers of the problem space. These complementary methods can be used to reconnect flows around the pool, after which MFTracer seamlessly resumes downstream tracing.

5.3 Limitation: False Positives and Mitigation Strategies

While MFTracer demonstrates strong performance in flow coverage, a limited number of false positives were observed in certain real-world tasks. To better understand these, we conducted a detailed comparison between MFTracer's tracing outputs and the ground truth fund flows. Our analysis reveals two primary sources of false positives.

① *Parameter Sensitivity.* The reserve ratio parameter affects the depth and breadth of the generated fund flow topology (discussed in Section 5.1). An overly conservative or aggressive setting can inflate the scope of simulated flows, resulting in false positives. Due to the high running efficiency of MFTracer, this issue is readily identifiable: by running MFTracer under multiple reserve ratio settings and comparing the topological differences, investigators can isolate spurious branches and quickly discard them. This strategy offers a fast and low-effort diagnostic mechanism for tuning the system's sensitivity. Practitioners may also refer to Section 5.1 for details.

② *Token Volatility and Temporal Value Drift.* A more subtle source of false positives stems from *temporal valuation errors* due to fluctuations in token prices. MFTracer's fund flow simulation relies on USD-denominated token prices to infer monetary value at the moment of each transaction. The valuation performs well for short-term flows, but may exhibit slight deviations in the context of long-term laundering strategies. For example, suppose an attacker transfers $100,000 worth of ETH to address A in December 2024. These assets remain idle for over two months before being moved entirely to address B in February 2025. If ETH depreciates during that time, and is only worth $80,000 at the time of the outbound transaction, MFTracer's simulation will record a residual balance of $20,000 at address A. As a result, subsequent unrelated transactions from A may be falsely flagged as laundering activity—since the system assumes part of the illicit funds remain there. This phenomenon arises from valuation drift—a mismatch between the token's market price at time t_1 (deposit) and t_2 (withdrawal). Similar distortions occur in the presence of price slippage. For instance, during panic-induced sell-offs, DeFi swaps may execute at unfavorable rates, causing a discrepancy between the input and output values. From a balance-sheet perspective, part of the "missing" value is effectively transferred to public liquidity pools. Without address-level contextual information, MFTracer may incorrectly interpret such residual value flows as criminal transfers, leading to false labeling of common service addresses as laundering participants.

Manual Review and Mitigation Potential. As discussed in **RQ6**, the false positives can often be filtered out manually within a few dozen minutes by inspecting token types and public address tags. Here, to minimize human review time, we propose two practical improvements: ❶ *Token-Aware Asset Balances.* Currently, MFTracer maintains per-address USD-denominated balance sheets. We propose extending this to token-specific multi-asset ledgers. By appending token type metadata to the transaction-level fund flow outputs, MFTracer can construct *asset-based* rather

Table 4. Counts of external native token transfers, internal native token transfers, and ERC-20 token transfers across six mainstream EVM-compatible platforms, covering all transactions up to March 5, 2025. Following gas fee based rule [27], native token transfers with amounts below 0.001 were filtered out to accurately count *non-dust* [12, 91] transfers that cause effective fund movement.

Chain	External Native	Internal Native	ERC-20 Token	Total
Ethereum	1,298,145,992	2,203,265,568	2,041,316,016	5,542,727,576
BSC	1,362,908,846	1,856,106,341	9,762,413,707	12,981,428,894
Optimism	94,435,717	295,031,281	1,131,759,616	1,521,226,614
Arbitrum	229,462,448	710,453,173	1,650,238,189	2,590,153,810
Base	486,773,952	485,954,349	2,583,982,243	3,556,710,544
Avalanche	82,150,539	247,168,056	595,087,068	924,405,663

than value-based balance states. This approach would eliminate valuation drift caused by market fluctuations. Although this enhancement may marginally increase storage overhead, MFTracer already achieves a 3.7×—9.4× improvement in storage efficiency, making this trade-off acceptable. ❷ *Integration with Address Labeling Systems.* To address errors stemming from price slippage and transfers to public infrastructure, we suggest integrating MFTracer with blockchain address labeling services such as Etherscan [19], Moralis [95], MetaSleuth [26], or Chainlabs [24]. These platforms maintain curated labels for exchanges, bridges, liquidity pools, and other non-adversarial service addresses. Incorporating such data into MFTracer's simulation phase would allow the system to avoid flagging known public infrastructure as part of laundering trails—particularly in cases where residual value "leaks" to public endpoints during volatile market conditions.

5.4 Efficiency: Applying MFTracer to Other EVM-Compatible Platforms

Section 4.1 demonstrated that MFTracer can efficiently handle Ethereum's massive transaction scale. We next examine whether it can handle other common EVM-compatible blockchains, which differ in both data volume and transaction throughput.

Table 4 summarizes the total number of fund transfers across six mainstream EVM-compatible chains, covering all the three categories that MFTracer processes in its tx-level money flow analysis stage (❷ in the left part of Figure 2): external native token transfers, internal native token transfers, and all types of ERC-20 token transfers. Most of these chains (Optimism, Arbitrum, Base, Avalanche) exhibit total transfer volumes well below Ethereum (e.g., Optimism at 1.5B, Avalanche under 1B), suggesting that Ethereum already represents a worst-case baseline for our system.

The only exception is BSC, which recorded 12.98 billion transfers—2.34× the total observed on Ethereum—primarily due to its heavy reliance on ERC-20 token operations (9.76B alone). Even under this scale, MFTracer's MFA graph representation would require at most 2.34× the memory used for Ethereum, i.e., 40.2 GB for *two years* of data, and even less in practice due to MFA's edge compression and compact representation (see Section 3.3, *Why is MFA Lightweight*). Such memory usage is well within the capabilities of modern commodity servers.

Beyond data volume, throughput is another critical factor. Among the mainstream EVM-compatible platforms, BSC exhibits the highest transaction throughput. On July 1, 2025, BSC increased its block speed to sub-second levels (0.75s/block), boosting its transaction throughput to ~ 150 TPS—nearly 10× Ethereum's rate [9]. Even so, MFTracer maintains a processing rate of 39,013 TPS, which is more than 260× faster than BSC's post-upgrade generation speed. This ensures that MFTracer can not only keep pace with but also comfortably exceed the demands of real-time infrastructure updates and tracing tasks of ongoing crimes on BSC, even under its fastest production settings.

5.5 Extending MFTracer Beyond EVM-Compatible Blockchains

Although MFTracer is designed for EVM-compatible blockchains, its modular architecture enables broader applicability across heterogeneous ledger models. Components such as the Money Flow Abstract (MFA), time-partitioned on-disk encoding, and parallel graph search are largely blockchain-agnostic—they rely only on generic money-flow records with timestamps and values rather than on EVM semantics. These modules can thus be directly reused on UTXO- [97], DAG- [101], or resource-based [7, 39] chains, provided that their transactions can be normalized into address-to-address transfer tuples. Similarly, the money-flow simulation mechanisms, which operate on time-ordered fund flows, remain effective regardless of the underlying virtual machine or consensus design.

The main adaptation effort lies in the transaction-level parser and semantic interpreter. On EVM chains, MFTracer reconstructs internal transfers and ERC-20 token transfer events from execution traces and logs; for non-EVM blockchains, equivalent logic must be implemented to translate native transaction semantics into standardized flow records. In UTXO-based chains, this involves resolving multi-input–multi-output transactions, detecting change outputs, and handling coin-mixing patterns to express resulting flow records as equivalent `(from,to,amount,time)` tuples for MFA ingestion and flow simulation. Similarly, extended UTXO and Move-based resource models [52] would require chain-specific parsers to interpret datum fields or object transfers. DAG (Directed Acyclic Graph) based ledgers require time-order reconstruction from partial confirmations, while resource-oriented systems like Sui [39] or Aptos [7] necessitate extracting object transfers from VM effects. Beyond the parser, modules for price normalization, bridge handling, and timestamp synchronization may need moderate customization. Once these adapters are implemented, MFTracer's existing infrastructure—its lightweight on-disk MFA encoding, high-throughput parallel graph search, and simulation-based illicit-flow inference—can operate unmodified across heterogeneous blockchain paradigms, preserving both scalability and forensic utility beyond the EVM domain.

6 Related Work

Fund Flow Analysis. Existing work has primarily focused on fund flow analysis on the Bitcoin blockchain. Phetsouvanh et al. [86] developed EGRET, using path confluence to identify extortion-related flows. Zhao and Guan [82] clustered addresses and visualized money flows in cases like Mt. Gox [28]. Tovanich et al. [99] applied taint flow extraction and graph embeddings to analyze how coin flow patterns vary by actor, enabling source attribution and actor classification. Meiklejohn et al. [78] used co-spending heuristics and re-identification to uncover payment patterns for flow analysis. Möser et al. [112] proposed a risk-scoring model to assess the likelihood of future blacklisting for transaction flows.

MFTracer vs. Previous Work. Compared to these studies, MFTracer introduces several key advancements: ❶ *Full EVM-compatibility.* While existing approaches are not applicable to the sophisticated EVM environment, MFTracer supports complex smart contracts and diverse token types on EVM-compatible blockchains, enabling fine-grained, protocol-agnostic and robust tracing across both standard DeFi protocols and adversarially crafted smart contracts. ❷ *High-efficiency infrastructure.* Existing studies all need to process large-scale transaction data, but none of them provides designs for fast and scalable data organization and retrieval. Thus, they are not able to support timely intervention in real-world cybercrime events and do not meet the efficiency demands of real-world investigations and practical deployment. MFTracer fills this gap with a domain-specific infrastructure-level design optimized for both speed and storage. ❸ *Behavior-agnostic tracing.* Unlike existing studies that rely on predefined laundering patterns, MFTracer performs rapid search and simulation without assuming fixed behavioral signatures—ensuring generalization to diverse and evolving money laundering strategies observed in the wild.

Additional Work. Additional studies on AML in the cryptocurrency domain, which are not central to our focus, are covered in Appendix G.

7 Conclusion

In this paper, we propose an automated system MFTracer for tracing illicit money flows on EVM-compatible blockchains. We are the first to identify the critical real-world challenges and address them in a unified framework. Against the backdrop of a domain where tracing remains labor-intensive and expert-driven, MFTracer is developed in response to two pressing real-world demands: operational *efficiency* and forensic *effectiveness*. ① We propose a novel fine-grained technique that enables protocol-agnostic transaction-level fund flow analysis. ② We further propose a lightweight and purpose-built graph abstraction MFA with a tailored storage backend to support efficient data retrieval (**14**× to **300**× speedup). ③ We also present a simulation algorithm for downstream illicit flow discovery. Applied to real-world cybercrime incidents, MFTracer successfully traced **94.09%** of illicit fund flows on average. And we newly reported **686** blockchain addresses and **4183** related transactions involved in money laundering that were previously undiscovered. MFTracer was able to reconstruct complete fund flow trajectories and provide strong evidence to investigators for **US $120.9 million** in stolen assets. We have also released the first open and large-scale dataset, which we believe will support future forensic and threat intelligence research.

Acknowledgments

We would like to thank the anonymous reviewers for their comments that greatly helped improve the presentation of this paper. We also want to thank Prof. Lucianna Kiffer for shepherding our paper. In addition, the first author would like to thank Bowen He, Yuan Chen, and Zhuo Chen for their help. This work is partially supported by the National Key R&D Program of China (No. 2022YFE0113200), the Innovation and Technology Commission of Hong Kong (ITC) under Mainland-Hong Kong Joint Funding Scheme (MHKJFS) under Grant MHP/135/23, the InnoHK initiative, the Government of the HKSAR, and the Laboratory for AI-Powered Financial Technologies (AIFT). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of funding agencies.

References

[1] 2023. *Atomic Wallect Hack.* https://atomicwallet.io/blog/articles/june-3rd-event-statement Referenced July 2, 2025.
[2] 2023. *Harmony Bridge Attak.* https://x.com/zachxbt/status/1619489550233133056 Referenced July 2, 2025.
[3] 2024. *2024 LI.FI Hack.* https://li.fi/knowledge-hub/incident-report-16th-july/ Referenced July 2, 2025.
[4] 2024. *How Lazarus Group laundered $200M from 25+ crypto hacks to fiat from 2020–2023.* https://zachxbt.mirror.xyz/B0-UJtxN41cJhpPtKv0v2LZ8u-0PwZ4ecMPEdX4l8vE Referenced July 2, 2025.
[5] 2024. *Lazarus Group 305M DMM Bitcoin hack.* https://x.com/zachxbt/status/1812466959109521649 Referenced July 2, 2025.
[6] 2025. *Adding Liquidity vs. Taking Liquidity in Trading.* https://www.zeiierman.com/blog/adding-liquidity-vs-taking-liquidity-in-trading Referenced July 2, 2025.
[7] 2025. *Aptos: The World's Most Production-Ready Blockchain.* https://aptosfoundation.org/ Referenced October 2, 2025.
[8] 2025. *BlockSec Phalcon.* https://blocksec.com/phalcon Referenced July 2, 2025.
[9] 2025. *Chain Scalability.* https://chainspect.app/chain/bnb-chain Referenced July 2, 2025.
[10] 2025. *CoinMarketCap.* https://coinmarketcap.com/ Referenced July 2, 2025.
[11] 2025. *Decentralized oracle networks for any smart contract.* https://chain.link/use-cases Referenced July 2, 2025.
[12] 2025. *Dust Transactions.* https://coinmarketcap.com/academy/glossary/dust-transactions Referenced July 2, 2025.
[13] 2025. *Elasticsearch.* https://www.elastic.co/elasticsearch/graph Referenced July 2, 2025.
[14] 2025. *ERC-1155 Multi-Token Standard.* https://ethereum.org/en/developers/docs/standards/tokens/erc-1155/ Referenced July 2, 2025.
[15] 2025. *ERC-20 token price oracle.* https://docs.uniswap.org/contracts/v2/concepts/core-concepts/oracles Referenced July 2, 2025.

[16] 2025. *ERC-20: Token Standard.* https://eips.ethereum.org/EIPS/eip-20 Referenced July 2, 2025.
[17] 2025. *ERC-721: Non-Fungible Token Standard.* https://eips.ethereum.org/EIPS/eip-721 Referenced July 2, 2025.
[18] 2025. *Ethereum Virtual Machine (EVM).* https://ethereum.org/en/developers/docs/evm/ Referenced July 2, 2025.
[19] 2025. *Etherscan.* https://etherscan.io/ Referenced July 2, 2025.
[20] 2025. *Euler Price Oracles.* https://docs.euler.finance/euler-price-oracle/ Referenced July 2, 2025.
[21] 2025. *Event Logs.* https://blog.ambire.com/eoas-vs-smart-contract-accounts/ Referenced July 2, 2025.
[22] 2025. *High JVM memory pressure.* https://www.elastic.co/docs/troubleshoot/elasticsearch/high-jvm-memory-pressure Referenced July 2, 2025.
[23] 2025. *Important Elasticsearch Configuration.* https://www.elastic.co/guide/en/elasticsearch/reference/current/important-settings.html Referenced July 2, 2025.
[24] 2025. *Labelled Datasets for Your Own Analysis.* https://chainlabs.ai/wallet-labels/ Referenced July 2, 2025.
[25] 2025. *Memgraph: Fastest, Most Affordable Graph Database.* https://memgraph.com/ Referenced July 2, 2025.
[26] 2025. *MetaSleuth.* https://metasleuth.io/ Referenced July 2, 2025.
[27] 2025. *Minimum Withdrawal Amounts.* https://docs.zerohash.com/docs/are-there-minimum-withdrawal-amounts Referenced July 2, 2025.
[28] 2025. *Mt. Gox Event.* https://trustwallet.com/ja/blog/cryptocurrency/mt-gox-explained Referenced July 2, 2025.
[29] 2025. *Neo4j.* https://neo4j.com/ Referenced July 2, 2025.
[30] 2025. *NFTScan.* https://viction.nftscan.com/ Referenced July 2, 2025.
[31] 2025. *OKLink.* https://www.oklink.com/ Referenced July 2, 2025.
[32] 2025. *OpenSea.* https://opensea.io/ Referenced July 2, 2025.
[33] 2025. *Otterscan.* https://github.com/otterscan/otterscan Referenced July 2, 2025.
[34] 2025. *Pebble key-value store.* https://github.com/cockroachdb/pebble Referenced July 2, 2025.
[35] 2025. *A phishing transaction profited more than 54M Dai.* https://x.com/BlockSecTeam/status/1826200855827390652 Referenced July 2, 2025.
[36] 2025. *Providing Liquidity.* https://docs.uniswap.org/contracts/v2/guides/smart-contract-integration/providing-liquidity Referenced July 2, 2025.
[37] 2025. *Pulsar Finance.* https://app.pulsar.finance/ Referenced July 2, 2025.
[38] 2025. *RedisGraph is a queryable graph database built on Redis.* https://redis.io/docs/latest/operate/oss_and_stack/stack-with-enterprise/deprecated-features/graph/ Referenced July 2, 2025.
[39] 2025. *Sui delivers the benefits of Web3 with the ease of Web2.* https://sui.io/ Referenced October 2, 2025.
[40] 2025. *Tornado Cash.* https://tornado.ws/ Referenced July 2, 2025.
[41] 2025. *Uniswap Multicall.* https://docs.uniswap.org/contracts/v3/reference/periphery/base/Multicall Referenced July 2, 2025.
[42] 2025. *uniswap v2 protocol.* https://docs.uniswap.org/contracts/v2/reference/smart-contracts/router-02 Referenced July 2, 2025.
[43] 2025. *WazirX hacked for over $230m USD.* https://x.com/WazirXIndia/status/1814971015929409936 Referenced July 2, 2025.
[44] 2025. *Welcome to the Ethereum Signature Database.* https://www.4byte.directory/ Referenced July 2, 2025.
[45] 2025. *What are Internal Transactions?* https://docs.alchemy.com/docs/what-are-internal-transactions Referenced July 2, 2025.
[46] 2025. *Zapper.fi.* https://zapper.fi Referenced July 2, 2025.
[47] Ismail Alarab, Simant Prakoonwit, and Mohamed Ikbal Nacer. 2020. Competence of graph convolutional networks for anti-money laundering in bitcoin blockchain. In *Proceedings of the 2020 5th international conference on machine learning technologies*. 23–27.
[48] Andreas M Antonopoulos and Gavin Wood. 2018. *Mastering ethereum: building smart contracts and dapps.* O'reilly Media.
[49] Massimo Bartoletti, Barbara Pes, and Sergio Serusi. 2018. Data mining for detecting bitcoin ponzi schemes. In *2018 crypto valley conference on blockchain technology (CVCBT)*. IEEE, 75–84.
[50] Vitalik Buterin et al. 2014. A next-generation smart contract and decentralized application platform. *white paper* 3, 37 (2014), 2–1.
[51] Rajasekhar Chaganti, Bharat Bhushan, and Vinayakumar Ravi. 2023. A survey on Blockchain solutions in DDoS attacks mitigation: Techniques, open challenges and future directions. *Computer Communications* 197 (2023), 96–112.
[52] Manuel MT Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Michael Peyton Jones, and Philip Wadler. 2020. The extended UTXO model. In *International Conference on Financial Cryptography and Data Security*. Springer, 525–539.
[53] Longfei Chen, Hao Wang, Yuchen Zhou, Taiyu Wong, Jialai Wang, and Chao Zhang. 2025. SmartTrans: Advanced Similarity Analysis for Detecting Vulnerabilities in Ethereum Smart Contracts. *IEEE Transactions on Dependable and*

Secure Computing (2025).
[54] Weili Chen, Zibin Zheng, Edith C-H Ngai, Peilin Zheng, and Yuren Zhou. 2019. Exploiting blockchain data to detect smart ponzi schemes on ethereum. *IEEE Access* 7 (2019), 37575–37586.
[55] Yuzhou Chen and Hon Keung Tony Ng. 2019. Deep learning ethereum token price prediction with network motif analysis. In *2019 International Conference on Data Mining Workshops (ICDMW)*. IEEE, 232–237.
[56] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. 2022. *Introduction to algorithms*. MIT press.
[57] Hanbiao Du, Zheng Che, Meng Shen, Liehuang Zhu, and Jiankun Hu. 2023. Breaking the anonymity of ethereum mixing services using graph feature learning. *IEEE Transactions on Information Forensics and Security* (2023).
[58] Joshua N Feinman. 1993. Reserve requirements: history, current practice, and potential reform. *Fed. Res. Bull.* 79 (1993), 569.
[59] Sadayuki Furuhashi, Satoshi Tagomori, Yuichi Tanikawa, Stephen Colebourne, Stefan Friesel, René Kijewski, Michael Cooper, Uenishi Kota, and Gabe Appleton. 2013. MessagePack Specification. MessagePack.
[60] Giacomo Giuliari, Alberto Sonnino, Marc Frei, Fabio Streun, Lefteris Kokoris-Kogias, and Adrian Perrig. 2024. An empirical study of consensus protocols' dos resilience. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*. 1345–1360.
[61] Aveek Goyal. 2025. *What are ERC20 Approve and ERC20 Allowance Methods?* https://metaschool.so/articles/what-are-erc20-approve-erc20-allowance-methods/ Referenced July 2, 2025.
[62] Mr Simon Gray. 2011. *Central bank balances and reserve requirements*. International Monetary Fund.
[63] Abhishek Guru, Bhabendu Kumar Mohanta, Hitesh Mohapatra, Fadi Al-Turjman, Chadi Altrjman, and Arvind Yadav. 2023. A survey on consensus protocols and attacks on blockchain technology. *Applied sciences* 13, 4 (2023), 2604.
[64] Jean Carlo Hamerski, Anderson RP Domingues, Fernando G Moraes, and Alexandre Amory. 2018. Evaluating serialization for a publish-subscribe based middleware for MPSoCs. In *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*. IEEE, 773–776.
[65] Bowen He, Yuan Chen, Zhuo Chen, Xiaohui Hu, Yufeng Hu, Lei Wu, Rui Chang, Haoyu Wang, and Yajin Zhou. 2023. TxPhishScope: Towards Detecting and Understanding Transaction-based Phishing on Ethereum. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 120–134.
[66] Younggee Hong, Hyunsoo Kwon, Jihwan Lee, and Junbeom Hur. 2018. A practical de-mixing algorithm for bitcoin mixing services. In *Proceedings of the 2nd ACM Workshop on Blockchains, Cryptocurrencies, and Contracts*. 15–20.
[67] Huiwen Hu, Qianlan Bai, and Yuedong Xu. 2022. Scsguard: Deep scam detection for ethereum smart contracts. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 1–6.
[68] Yining Hu, Suranga Seneviratne, Kanchana Thilakarathna, Kensuke Fukuda, and Aruna Seneviratne. 2019. Characterizing and detecting money laundering activities on the bitcoin network. *arXiv preprint arXiv:1912.12060* (2019).
[69] Yufeng Hu, Yingshi Sun, Yuan Chen, Zhuo Chen, Bowen He, Lei Wu, Yajin Zhou, and Rui Chang. 2024. MFGSCOPE: A Lightweight Framework for Efficient Graph-based Analysis on Blockchain. *IEEE Transactions on Dependable and Secure Computing* (2024).
[70] Mingyuan Huang, Jiachi Chen, Zigui Jiang, and Zibin Zheng. 2024. Revealing hidden threats: An empirical study of library misuse in smart contracts. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–12.
[71] Johannes Krupp and Christian Rossow. 2018. {teEther}: Gnawing at ethereum to automatically exploit smart contracts. In *27th USENIX security symposium (USENIX Security 18)*. 1317–1333.
[72] Satpal Singh Kushwaha, Sandeep Joshi, Dilbag Singh, Manjit Kaur, and Heung-No Lee. 2022. Ethereum smart contract analysis tools: A systematic review. *Ieee Access* 10 (2022), 57037–57062.
[73] Xiangfeng Li, Shenghua Liu, Zifeng Li, Xiaotian Han, Chuan Shi, Bryan Hooi, He Huang, and Xueqi Cheng. 2020. Flowscope: Spotting money laundering based on graphs. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 4731–4738.
[74] Xigao Li, Anurag Yepuri, and Nick Nikiforakis. 2023. Double and nothing: Understanding and detecting cryptocurrency giveaway scams. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*.
[75] Dan Lin, Jiajing Wu, Yunmei Yu, Qishuang Fu, Zibin Zheng, and Changlin Yang. 2024. DenseFlow: Spotting Cryptocurrency Money Laundering in Ethereum Transaction Graphs. In *Proceedings of the ACM on Web Conference 2024*. 4429–4438.
[76] Shenghua Liu, Bryan Hooi, and Christos Faloutsos. 2017. Holoscope: Topology-and-spike aware fraud detection. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*. 1539–1548.
[77] Joana Lorenz, Maria Inês Silva, David Aparício, João Tiago Ascensão, and Pedro Bizarro. 2020. Machine learning methods to detect money laundering in the bitcoin blockchain in the presence of label scarcity. In *Proceedings of the first ACM international conference on AI in finance*. 1–8.

[78] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. 2013. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*. 127–140.
[79] Patrick Monamo, Vukosi Marivate, and Bheki Twala. 2016. Unsupervised learning for robust Bitcoin fraud detection. In *2016 Information Security for South Africa (ISSA)*. IEEE, 129–134.
[80] Patrick M Monamo, Vukosi Marivate, and Bhesipho Twala. 2016. A multifaceted approach to bitcoin fraud detection: Global and local outliers. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 188–194.
[81] S Monish, Mridul Mohta, and Shanta Rangaswamy. 2022. Ethereum Price Prediction Using Machine Learning Techniques–A Comparative Study. *International Journal of Engineering Applied Sciences and Technology* 7 (2022), 137–142.
[82] Malte Möser, Rainer Böhme, and Dominic Breuker. 2014. Towards risk scoring of Bitcoin transactions. In *Financial Cryptography and Data Security: FC 2014 Workshops, BITCOIN and WAHC 2014, Christ Church, Barbados, March 7, 2014, Revised Selected Papers 18*. Springer, 16–32.
[83] Philip Nadler and Yike Guo. 2020. The fair value of a token: How do markets price cryptocurrencies? *Research in International Business and Finance* 52 (2020), 101108.
[84] Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil. 1996. The log-structured merge-tree (LSM-tree). *Acta Informatica* 33 (1996), 351–385.
[85] Bo Petersen, Henrik Bindner, Shi You, and Bjarne Poulsen. 2017. Smart grid serialization comparison: Comparision of serialization for distributed control in the context of the internet of things. In *2017 Computing Conference*. IEEE, 1339–1346.
[86] Silivanxay Phetsouvanh, Frédérique Oggier, and Anwitaman Datta. 2018. Egret: Extortion graph exploration techniques in the bitcoin network. In *2018 IEEE International conference on data mining workshops (ICDMW)*. IEEE, 244–251.
[87] Mohammadreza Rasolroveicy and Marios Fokaefs. 2022. Impact of ddos attacks on the performance of blockchain consensus as an lot data registry: An empirical study. In *Proceedings of the 32nd Annual International Conference on Computer Science and Software Engineering*. 71–80.
[88] Alexander Schrijver. 2002. On the history of the transportation and maximum flow problems. *Mathematical programming* 91 (2002), 437–445.
[89] Preeti Sharma and RM Pramila. 2022. Price prediction of Ethereum using time series and deep learning techniques. In *Proceedings of Emerging Trends and Technologies on Intelligent Systems: ETTIS 2022*. Springer, 401–413.
[90] Mike Shin. 2025. *What are EVM Compatible Blockchains? A Guide to the Ethereum Virtual Machine.* https://blog.thirdweb.com/evm-compatible-blockchains-and-ethereum-virtual-machine/ Referenced July 2, 2025.
[91] Qunhong Sun, Chang Wang, Yifan Hu, Shen Su, and Ting Cui. 2025. LGTDA: Bandwidth exhaustion attack on Ethereum via dust transactions. *Future Generation Computer Systems* 163 (2025), 107549. doi:10.1016/j.future.2024.107549
[92] Yujia Tang, Chang Xu, Can Zhang, Yan Wu, and Liehuang Zhu. 2021. Analysis of address linkability in tornado cash on ethereum. In *China Cyber Security Annual Conference*. Springer, 39–50.
[93] tayvano. 2025. *Atomic Wallet Hack Report.* https://dune.com/tayvano/atomic-wallet-hack Referenced July 2, 2025.
[94] MetaSleuth Team. 2024. *54m phishing fund movements.* https://x.com/MetaSleuth/status/1826205736617279823 Referenced July 2, 2025.
[95] Moralis Team. 2025. *Check Wallet Activity and Get Crypto Address Labels.* https://developers.moralis.com/check-wallet-activity-and-get-crypto-address-labels/ Referenced July 2, 2025.
[96] Neo4j Team. 2025. *Causal Clustering in Neo4j.* https://neo4j.com/graphacademy/training-admin-35/04-neo4jadmin-3-5-causal-clustering-neo4j/ Referenced July 2, 2025.
[97] The Investopedia Team. 2025. *UTXO Model: Definition, How It Works, and Goals.* https://www.investopedia.com/terms/u/utxo.asp Referenced July 2, 2025.
[98] Christof Ferreira Torres, Mathis Steichen, et al. 2019. The art of the scam: Demystifying honeypots in ethereum smart contracts. In *28th USENIX Security Symposium (USENIX Security 19)*. 1591–1607.
[99] Natkamon Tovanich and Rémy Cazabet. 2022. Pattern analysis of money flows in the Bitcoin blockchain. In *International Conference on Complex Networks and Their Applications*. Springer, 443–455.
[100] Friedhelm Victor and Andrea Marie Weintraud. 2021. Detecting and quantifying wash trading on decentralized cryptocurrency exchanges. In *Proceedings of the Web Conference 2021*. 23–32.
[101] Qin Wang, Jiangshan Yu, Shiping Chen, and Yang Xiang. 2023. SoK: DAG-based blockchain systems. *Comput. Surveys* 55, 12 (2023), 1–38.
[102] Dinitha Wijewardhana, Sugandima Vidanagamachchi, and Nalin Arachchilage. 2024. Examining Attacks on Consensus and Incentive Systems in Proof-of-Work Blockchains: A Systematic Literature Review. *arXiv preprint arXiv:2411.00349* (2024).

[103] Jiajing Wu, Dan Lin, Qishuang Fu, Shuo Yang, Ting Chen, Zibin Zheng, and Bowen Song. 2023. Towards Understanding Asset Flows in Crypto Money Laundering Through the Lenses of Ethereum Heists. *IEEE Transactions on Information Forensics and Security* (2023).
[104] Mike Wu, Will McTighe, Kaili Wang, Istvan A Seres, Nick Bax, Manuel Puebla, Mariano Mendez, Federico Carrone, Tomás De Mattey, Herman O Demaestri, et al. 2022. Tutela: An open-source tool for assessing user-privacy on ethereum and tornado cash. *arXiv preprint arXiv:2201.06811* (2022).
[105] Shuohan Wu, Zihao Li, Luyi Yan, Weimin Chen, Muhui Jiang, Chenxu Wang, Xiapu Luo, and Hao Zhou. 2024. Are we there yet? unraveling the state-of-the-art smart contract fuzzers. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.
[106] Pengcheng Xia, Haoyu Wang, Bingyu Gao, Weihang Su, Zhou Yu, Xiapu Luo, Chao Zhang, Xusheng Xiao, and Guoai Xu. 2021. Trade or trick? detecting and characterizing scam tokens on uniswap decentralized exchange. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 5, 3 (2021), 1–26.
[107] Kailun Yan, Xiaokuan Zhang, and Wenrui Diao. 2024. Stealing Trust: Unraveling Blind Message Attacks in Web3 Authentication. *arXiv preprint arXiv:2406.00523* (2024).
[108] zachxbt. 2022. *Discord & Twitter Scamming.* https://x.com/zachxbt/status/1562473638992850947 Referenced July 2, 2025.
[109] zachxbt. 2022. *Harmony Bridge hack address report.* https://www.chainabuse.com/report/0a2e8e00-00e2-4749-9b00-ceb1c6202d33 Referenced July 2, 2025.
[110] zachxbt. 2022. *loyalist scamming.* https://x.com/zachxbt/status/1626200628308443139 Referenced July 2, 2025.
[111] zachxbt. 2022. *Tracking down Discord & Twitter phishing scammers.* https://zachxbt.mirror.xyz/svL1N4xPLX5nXHr6Cw4KLsjRtaYHxm4MAqmFy6zx3cw Referenced July 2, 2025.
[112] Chen Zhao and Yong Guan. 2015. A graph-based investigation of bitcoin transactions. In *Advances in Digital Forensics XI: 11th IFIP WG 11.9 International Conference, Orlando, FL, USA, January 26-28, 2015, Revised Selected Papers 11.* Springer, 79–95.
[113] Liyi Zhou, Xihan Xiong, Jens Ernstberger, Stefanos Chaliasos, Zhipeng Wang, Ye Wang, Kaihua Qin, Roger Wattenhofer, Dawn Song, and Arthur Gervais. 2023. Sok: Decentralized finance (defi) attacks. In *2023 IEEE Symposium on Security and Privacy (SP).* IEEE, 2444–2461.
[114] Yi Zhou, Deepak Kumar, Surya Bakshi, Joshua Mason, Andrew Miller, and Michael Bailey. 2018. Erays: reverse engineering ethereum's opaque smart contracts. In *27th USENIX security symposium (USENIX Security 18).* 1371–1385.

A Details of the Example of Figure 1

A.1 Example Overview

In this example, the criminal invokes multiple functions in a DEX aggregator contract within a single Multi-call transaction, triggering fund movements across multiple addresses. The execution of this transaction can be divided into seven sub-processes. Readers may find it helpful to consult Figure 5 while reading this part. ① The criminal-controlled EOA 0 sends 10 `ETH` (Ethereum's native token) to the DEX aggregator. ② The aggregator wraps the received 10 `ETH` into the ERC-20 standard token `wETH` using the wETH contract. ③ The aggregator transfers 2.5 `wETH` to the criminal-controlled EOA 1. ④ The aggregator calls the `Swap` protocol to exchange 5 `wETH` for 0.12 `wBTC` at Liquidity Pool 1. ⑤ The aggregator calls the `Swap` protocol to exchange 0.07 `wBTC` for 5,600 `USDT` at Liquidity Pool 0 and the remaining 0.05 `wBTC` for 4,000 `USDT` at Pool 2, totaling 9,600 `USDT`, where 4800 ones are transferred back to EOA 0. ⑥ The aggregator calls the `AddLiquidity` protocol to deposit 2.5 `wETH` and 4,800 `USDT` into Pool 2 and the received 10 `LP Tokens` are transferred to a criminal-controlled smart contract. ⑦ Since the smart contract had previously obtained an allowance from EOA 1, it executes predefined code to swap 2.5 `wETH` from EOA 1 for 0.048 `wBTC` at Pool 1, and transfer 5 `LP Tokens` to EOA 1.

A.2 Apply Algorithm 1 to this Example

As shown in Figure 6, MFTracer constructs the local money transfer graph G_L and the balance change table B for this transaction to decipher its intricate execution sub-processes. Upon examining B, MFTracer determines that the transaction does not result in any USD-denominated balance changes for the aggregator or liquidity pools. This is because the semantics of `Swap` and

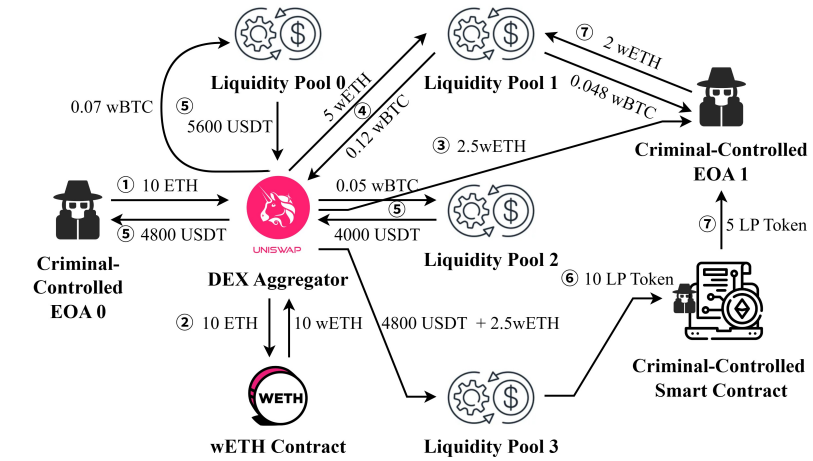

Fig. 5. An real-world example illustrates how a criminal executed a Multi-call transaction on a DEX aggregator to obscure illicit fund flows as part of their laundering strategy. The execution of this transaction can be divided into seven sub-processes, including DeFi semantics of a `Wrap` (②), an `AddLiquidity` (⑥) and four Swaps (④, ⑤ and ⑦).

`AddLiquidity` do not inherently transfer USD value but change the composition of tokens associated with each address [6, 36]. However, EOA 0 experiences a decrease in balance, whereas EOA 1 and the criminal-controlled contract see an increase. Using this information, MFTracer concludes that the actual fund provider is the criminal-controlled EOA 0, and that the fund recipients are EOA 1 and the criminal-controlled smart contract. Then it applies subsequent computational steps of Algorithm 1 on G_L to determine the fund flows from EOA 0 to EOA 1 and the smart contract and fully uncovers the underlying fund movements of this complex transaction.

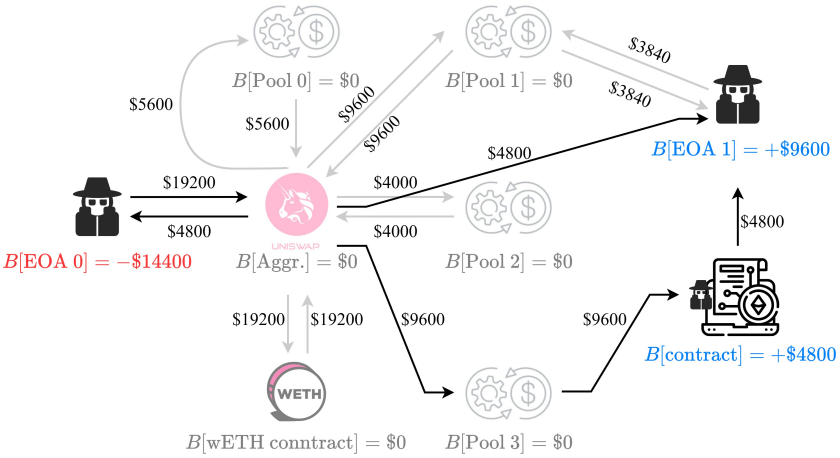

Fig. 6. The local money transfer graph G_L and balance change table B constructed by Algorithm 1 for the Multi-call transaction in Figure 5. We use $B[a]$ to refer to the local balance change of address a.

B Proof for the Pruning Inequality of III-C

For a valid money flow path $a_1 \rightarrow a_2 \rightarrow ... \rightarrow a_n$, let t_i represent the timestamp of the flow between address a_i and address a_{i+1}. We have $t_i \geq t_{i-1}$ holds for all $1 < i < n$. It follows that for all $0 < j < i < n$, we have $t_i \geq t_j$. Continuing with the notation of the paper, it follows straightforwardly that a valid path from a_1 to a_n exists in G only if

$$\forall\ 0 < j < i < n,\ \exists\ t_i \in \tau_{a_i,a_{i+1}},\ \exists\ t_j \in \tau_{a_j,a_{j+1}},\ t_i \geq t_j.$$

That is,

$$\forall\ 0 < j < i < n,\ \exists\ t_i \in \tau_{a_i,a_{i+1}},\ t_i \geq \min \tau_{a_j,a_{j+1}}.$$

Equivalent to

$$\forall\ 0 < j < i < n,\ \max \tau_{a_i,a_{i+1}} \geq \min \tau_{a_j,a_{j+1}}.$$

This is equivalent to the following inequality.

$$\forall\ 1 < i < n,\ \max \tau_{a_i,a_{i+1}} \geq \max_{0<j<i} \min \tau_{a_j,a_{j+1}}.$$

Therefore, a valid path from a_1 to a_n exists in G only if the inequality above holds, and the proof is complete.

C Graph Search Parallelization Strategy

Algorithm 3 Pseudocode of Parallel Search on MFAs

Input: MFA slices $\left\{G_{\text{mfa}}^{(m)}, ..., G_{\text{mfa}}^{(m+k)}\right\}$ and victim address set A_{vic}.
Output: Suspicious topologies $\left\{G_{\text{s}}^{(m)}, ..., G_{\text{s}}^{(m+k)}\right\}$.

```
G_s^(m), G_s^(m+1), ..., G_s^(m+k) ← new k + 1 empty topologies
for i such that 0 ≤ i ≤ k do
    A_s ← G_s^(m+i−1).AddressSet if i > 0 else A_vic
    parallel for j such that i ≤ j ≤ k do
        ^iG_s^(m+j) ← search on G_mfa^(m+j) using A_s as the source
        G_s^(m+j) ← G_s^(m+j) ⋃ ^iG_s^(m+j)
    end for
end for
```

The details of graph search operations on a single MFA have been discussed in the Suspicious Topology Construction part of Section 3.3. Here, we present the parallelization method for the search mentioned in Section 3.4. The parallelization strategy can be depicted using the above algorithm. Let the relevant MFA slices be temporally ordered as $\left\{G_{\text{mfa}}^{(m)}, ..., G_{\text{mfa}}^{(m+k)}\right\}$. The search begins with using A_{vic} as the source to perform parallel searches across all MFA slices, yielding the suspicious sub-topologies $\left\{{}^0G_{\text{s}}^{(m)}, ..., {}^0G_{\text{s}}^{(m+k)}\right\}$, where the ${}^0G_{\text{s}}^{(m+i)}$ denotes the search result on $G_{\text{mfa}}^{(m+i)}$. Then, for $0 \leq i < k$, the system repeats the process: it takes all addresses from the union $\bigcup_{0\leq j\leq i} {}^jG_{\text{s}}^{(m+k)}$ as the source and runs parallel searches on MFAs $\left\{G_{\text{mfa}}^{(m+i+1)}, ..., G_{\text{mfa}}^{(m+k)}\right\}$ to obtain suspicious sub-topologies $\left\{{}^{i+1}G_{\text{s}}^{(m+i+1)}, ..., {}^{i+1}G_{\text{s}}^{(m+i+k)}\right\}$. Finally, for $0 \leq i \leq k$, MFTracer merges suspicious sub-topologies $\left\{{}^0G_{\text{s}}^{(m+i)}, ..., {}^iG_{\text{s}}^{(m+i)}\right\}$ to obtain the suspicious topology $G_{\text{s}}^{(m+i)}$ as the search result on the MFA $G_{\text{mfa}}^{(m+i)}$.

We now demonstrate that the parallel strategy yields identical results to the serial execution, proving its correctness. For ease of explanation, we use the notation $\mathcal{S}(G_{\text{mfa}}; G_{\text{s}})$ to refer to the suspicious sub-topology derived from searches on the MFA G using the addresses in the topology G_{mfa} as the source. Then for all $0 \le i \le k$, we have

$$\begin{aligned} G_{\text{s}}^{(m+i)} &= \bigcup_{0\le j\le i} {}^{j}G_{\text{s}}^{(m+i)} \\ &= \bigcup_{0< j\le i} {}^{j}G_{\text{s}}^{(m+i)} \cup {}^{0}G_{\text{s}}^{(m+i)} \\ &= \bigcup_{0< j\le i} \mathcal{S}(G_{\text{mfa}}^{(m+i)}; G_{\text{s}}^{(m+j-1)}) \cup \mathcal{S}(G_{\text{mfa}}^{(m+i)}; A_{\text{vic}}) \\ &= \mathcal{S}\left(G_{\text{mfa}}^{(m+i)}; A_{\text{vic}} \cup \bigcup_{0\le j< i} G_{\text{s}}^{(m+j)}\right). \end{aligned}$$

The first equality is based on line 6 of Algorithm 3. The third equality follows from line 3 and line 5. This equation demonstrates that $G_{\text{s}}^{(m+i)}$ is obtained by searching on $G_{\text{mfa}}^{(m+i)}$ using addresses from A_{vic} and all the suspicious topologies with earlier timestamps as the source, which completes the proof. We discuss the performance of this parallelization based on an empirical assumption: the subtasks in line 5 of Algorithm 3 have similar time costs of $O(C)$. Then we derive that the total time cost for a serial algorithm is $O(C) \cdot (k+1)(k+2)/2 = O(Ck^2)$, while the parallel time cost is $O(Ck)$ from the loop in line 2.

D Reserve Ratio ϵ Theoretical Analysis

The reserve requirement, also known as the cash reserve ratio (CRR) [58, 62], is a regulation set by central banks that mandates commercial banks to hold a certain percentage of their deposit liabilities in reserves. By adjusting the reserve requirement, central banks can influence the liquidity in the economy system. Drawing inspiration from this, we apply a cap on the amount of funds that can flow out from each address to control liquidity distribution in the downstream network. Here, we discuss how the reserve ratio $\epsilon \in [0, 1)$ affects the breadth and depth of the downstream topology. For this, we estimate the depth and breadth that a specific amount of money can travel through under a small threshold T by assuming that each outflow for each address reaches its upper limit controlled by ϵ.

First, let's assume an upstream flow of $M \gg T$ reaches a downstream address, which is the flow we're to analyze. We call this address as the starting address and mark its depth as 1. Then $M(1-e)^n$ indicates the maximum outflow from an address at depth n. The following can be determined, where d indicates the maximum depth.

$$M(1-\epsilon)^d \ge T, M(1-\epsilon)^{d+1} < T.$$

We can then derive that

$$d = \left\lfloor \frac{\log(T/M)}{\log(1-\epsilon)} \right\rfloor.$$

The equation above demonstrates that d decreases as ϵ increases.

Since the starting address is randomly selected from the downstream set, the number of addresses at depth 2 indicates the breadth b, which satisfies the inequality below.

$$M\epsilon^{b-1}(1-\epsilon) \ge T, M\epsilon^{b}(1-\epsilon) < T.$$

Then we obtain that

$$b = \left\lceil \frac{\log(T/M) - \log(1-\epsilon)}{\log \epsilon} \right\rceil.$$

Given that $M \gg T$, then $\log(T/M)$ is a negative number with a significantly large absolute value. Since that ϵ is usually chosen to be less than 0.5, then we have $|\log(1-\epsilon)| \ll |\log(T/M)|$. Thus, we can arrive at the following approximation.

$$b \approx \left\lceil \frac{\log(T/M)}{\log \epsilon} \right\rceil.$$

This demonstrates that the breadth increases as ϵ increases.

Altogether, a lower reserve ratio leads to more liquidity among farther[4] addresses, increasing the downstream topology's depth, while a higher distributes liquidity to closer addresses and thus increasing the downstream topology's breadth. This affects the final coverage rates and precision. For criminals using extremely deep laundering paths, a lower ϵ increasing both the coverage rates and precision. For criminals using wider laundering paths, a higher ϵ is a better choice. In real-world use, multiple values can be selected to ensure a better coverage of illicit flows.

E Evaluation Results on ϵ

The reserve ratio ϵ is a configuration parameter for the money flow simulation algorithm described in Section 3.4. Setting an appropriate ϵ value allows MFTracer to perform better in tracing tasks. Apart from theoretical analysis above, we also conduct experiments to examine the effect of different values of ϵ on the results. Figure 7 illustrates the curves of precision and two types of coverage rates as ϵ is set from 0% to 55%. For multi-perspective presentation, evaluation results are disaggregated by metrics and top-level incidents.

In all incidents, especially for Atomic and XScam, too large values of ϵ result in low coverage rates. Conversely, setting ϵ to 0% yields satisfactory coverage in all cases. Sometimes, such as the laundering cases of incidents TxPhish and XScam, reach their highest coverage rates when ϵ is set to a small positive value. Except for LIFI, the precision shows relatively small fluctuation. Overall, based on the empirical analysis, setting ϵ between 0% and 10% allows MFTracer to achieve the optimal effectiveness, balancing both two types of coverage rates and precision. For real-world tracing applications, multiple close values of ϵ can be selected to compare the differences in the respective resulting topologies.

F Dataset Construction Methodology

To rigorously evaluate the performance of illicit fund tracing systems, we constructed a ground-truth dataset LaunderNetEvm41 that reflects a diverse and realistic set of money laundering techniques. The dataset was collaboratively curated by a team of seven experts, including two members from our author group, three professionals from BlockSec[1], and two domain specialists from the broader security community. Each expert has at least one year of hands-on experience in tracking crypto-related cybercrime.

In total, LaunderNetEvm41 contains 1,939 accounts involved in money laundering activities and 6,701 detailed illicit fund flow records—an order of magnitude larger than any previously disclosed (Bitcoin) illicit flow dataset we are aware of, collectively covering over US $125 million in stolen assets. In curating cases from cybercrime incidents for our dataset, we applied three selection criteria to ensure soundness, persuasiveness, and cross-verifiability: ① *Verified sources.* The selected incidents must include verifiable evidence, such as official statements from the victim entity or

[4]These terms, closer and farther, are defined by the distance between addresses in the flow topology, i.e. the length of reachable path in the graph.

incident reports published by reputable security community experts, ensuring the authenticity and transparency. ② *Diverse coverage.* The cases must originate from multiple independent criminal

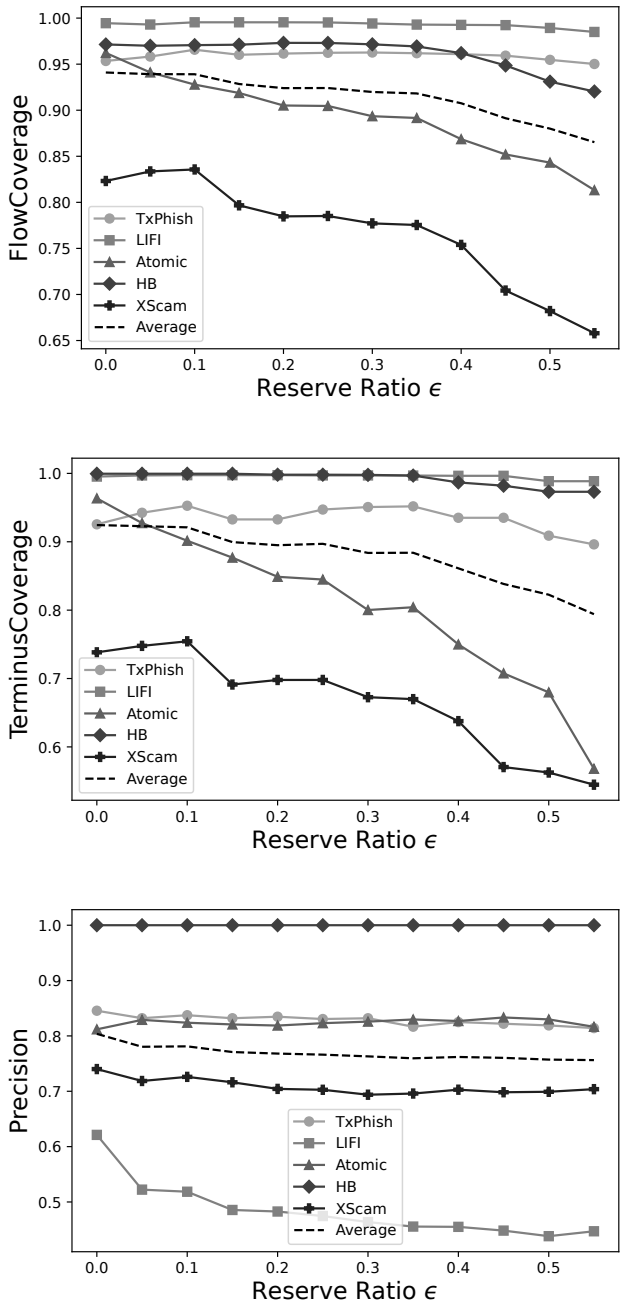

Fig. 7. FlowCoverage, TerminusCoverage and Precision as ϵ is set from 0% to 55%.

groups and span different types of cybercrimes, enabling us to test the generality, robustness, and practical applicability of tracing systems. ③ *Cross-validation support.* Each of the selected incidents must have at least one fund-tracing report published by other security community experts, allowing cross-checking of our dataset contents and enabling readers to validate the dataset themselves. Finally, we constructed the dataset based on a set of cybercrime incidents from 2022 to 2024 that meet the above criteria.

The construction process began with the collection of victim addresses and attack transaction hashes sourced from publicly available incident reports published by affected parties and security communities. These sources served as the origin points for tracing the downstream movement of stolen assets. Experts utilized a combination of tools and platforms—such as Etherscan [19], Phalcon [8], Otterscan [33], and self-hosted full nodes—to gather on-chain transactional data and token transfer events. The tracing process covered all types of fungible tokens, including native tokens and ERC-20 tokens, ensuring comprehensive coverage of the illicit asset flow. Experts analyzed transactions on a per-transaction basis. For each address identified as holding illicit funds, all subsequent outbound transactions were examined to determine how the assets were redistributed. These transactions were categorized into three general patterns: ① *Transactions to EOAs*: If the `to` address was an externally owned account (EOA), resulting in a native token transfer, the receiving address was marked as a potential downstream recipient and added to the pending analysis queue. ② *Transactions to ERC-20 Token Contracts*: In such cases, the expert team inspected the transaction's payload and analyzed the emitted token transfer events to determine fund movement: (i) If the transaction invoked the `transfer` function, the recipient of the token was added to the queue. (ii) If the transaction called `permit`, `approve`, or similar functions that delegated token spending rights to another address, the spender was recorded for further analysis. ③ *Transactions to Other Smart Contracts:* For these, a multi-tiered semantic analysis was conducted: (i) If the address matched a known public service address of a decentralized protocol, its function was analyzed using protocol documentation and public interfaces. (ii) If the contract was not recognized but its source code was available on platforms like Etherscan, experts manually audited the code using similarity detection [53], library usage analysis [70], and protocol-specific knowledge to interpret the contract's logic. (iii) If only bytecode was available (e.g., attacker-deployed contracts), experts used function selector databases [44] and dynamic analysis techniques such as reverse engineering [114], execution tracing [72], testnet deployment and black-box fuzzing [105] to infer contract behavior. Based on their domain-specific knowledge, experts used assistive tools to decipher smart contract semantics and identify the underlying fund movements resulting from the contract execution. The actual receivers of illicit money are marked as laundering addresses.

To ensure data accuracy and reduce bias, each tracing step was independently conducted by at least two experts. When interpretations differed, the findings were debated in peer-review sessions involving all seven members. Final decisions were made by majority vote. In cases of significant disagreement or ambiguity, deeper analysis was conducted using additional on-chain evidence, behavioral heuristics, and, where necessary, the recreation of suspicious transactions in controlled environments.

Once the full trace paths were established, the resulting dataset underwent a final round of verification. The two domain experts from the security community performed an extensive cross-validation process against publicly disclosed intelligence—including independent reports by other researchers, statements from victim entities, investigative reports from regulatory bodies, and analyses by external security firms. We list some of the references here [4, 93, 94, 109, 111]. Any contradictions or discrepancies between our dataset and these sources triggered a re-examination of the relevant tracing paths. For especially complex or conflicting cases, we proactively reached out to the authors of the referenced reports for clarification or technical discussion.

This rigorous, expert-driven process ensures that the dataset reflects a realistic, high-fidelity mapping of illicit fund flows across various money laundering strategies. It serves not only as a benchmark for evaluating tracing systems, but also as a resource for further research in blockchain forensics and threat intelligence. For more details about the dataset—such as visualizations of illicit fund flows, analyses of the criminal incidents, and guidelines for verifying the dataset contents—please refer to the dataset repository.

G Other Studies on AML in the Cryptocurrency Domain

Recent research on AML in the cryptocurrency domain focuses on both on-chain and off-chain anomalous behavior detection.

On-chain Anomalous Behavior Detection. On-chain methods often leverage graph mining or machine learning to uncover suspicious activities. Hu et al. [68] explore the structural differences in Bitcoin transaction graphs and apply node embedding methods like node2vec to identify laundering patterns with over 92% accuracy. Monamo et al. [79] investigate unsupervised fraud detection using trimmed k-means clustering and demonstrate superior performance in identifying anomalous Bitcoin transactions. Xia et al. [106] analyze scam tokens on the Uniswap DEX, identifying over 10,000 rug-pull tokens and exposing scammer collusion networks, highlighting the urgency for scam token monitoring in DeFi ecosystems. DenseFlow [75] proposes a method to identify money laundering activities by extracting dense subgraphs. Similarly, XBlockFlow [103] performs taint-based analysis to study the evolution and tactics of laundering behavior, including novel schemes like counterfeit token creation. Other graph-based approaches like the GCN-based method by Alarab et al. [47] enhance node embeddings by combining graph convolution with linear layers, improving illicit transaction classification on the Elliptic dataset. Lorenz et al. [77] address label scarcity by employing active learning, demonstrating that even limited supervision (5% labeled data) can achieve results comparable to fully supervised models.

Off-chain Anomalous Behavior Detection. On the off-chain side, Li et al. [74] introduce CryptoScamTracker, which uses Certificate Transparency logs and crawling infrastructure to detect large-scale cryptocurrency giveaway scams, discovering over 10,000 scam websites and tracing stolen funds across blockchains. He et al. [65] propose TxPhishScope, a system for detecting transaction-based phishing websites on Ethereum that trick users into signing malicious transactions, successfully identifying over 26,000 phishing domains and analyzing their laundering patterns. Yan et al. [107] explore vulnerabilities in Web3 authentication, revealing that a majority of dApps allow signature-based unauthorized access, posing a risk of laundering via impersonation. Victor and Weintraud [100] quantify wash trading on decentralized exchanges, detecting over $159M of fraudulent volume through identifiable trading patterns.

H Artifacts

Dataset (LaunderNetEvm41):
https://github.com/blocksecteam/MFTracer/tree/main/LaunderNetEvm41
Source code:
https://github.com/blocksecteam/MFTracer/tree/main/codes
Lists of addresses & txs that we newly reported:
https://github.com/blocksecteam/MFTracer/tree/main/findings

Received July 2025; revised September 2025; accepted October 2025