GuideEnricher: Protecting the Anonymity of Ethereum Mixing Service Users with Deep Reinforcement Learning

Ravindu De Silva, Wenbo Guo, Nicola Ruaro, Ilya Grishchenko, Christopher Kruegel, Giovanni Vigna
University of California, Santa Barbara
ldesilva@ucsb.edu, henrygwb@ucsb.edu, ruaronicola@ucsb.edu, grishchenko@ucsb.edu, chris@cs.ucsb.edu, vigna@cs.ucsb.edu

Abstract

Mixing services are widely employed to enhance anonymity on public blockchains. However, recent research has shown that user identities and transaction associations can be derived even with mixing services. This is mainly due to the lack of guidelines for properly using these services. In fact, mixing service developers often provide guidebooks with lists of actions that might break anonymity, and hence, should be avoided. However, such guidebooks remain incomplete, leaving users unaware of potential actions that might compromise their anonymity. This highlights the necessity for providing users with a more comprehensive guidebook. Unfortunately, existing methods for compiling anonymity-compromising patterns rely on postmortem analyses, and they cannot proactively discover patterns before the mixing service is deployed.

We introduce GuideEnricher, a proactive approach for extending user guidebooks with limited human intervention. Our key novelty is a deep reinforcement learning (DRL) agent, which automatically explores patterns for transferring tokens via a mixing service. We introduce two customized designs to better guide the agent in discovering yet-unknown anonymity-compromising patterns: design proper tasks for the agent that possibly lead to compromised anonymity, and include a rule-based detector to detect the known patterns. We train the agent to finish the task while evading the detector. Using a trained agent, we conduct a second analysis step, employing clustering methods and manual inspection, to extract yet-unknown patterns from the agent's actions. Through extensive evaluation, we demonstrate that GuideEnricher can train effective agents under multiple mixing services. We show that our agents facilitate the discovery of yet-unknown anonymity-compromising patterns. Furthermore, we demonstrate that GuideEnricher can continuously enrich the guidebook via an iterative update of the detector and our DRL agents.

1 Introduction

The emergence of the Ethereum blockchain has brought attention to user and transaction anonymity. Since all transactions are public, a malicious user can link transactions involving a particular wallet and group that wallet with others used in the linked transactions. Through such a chain of associations, the attacker could potentially collect a group of wallets belonging to the same user, along with the user's transaction history. This significantly increases the risk of identity exposure. A breach of a single public record linking the user's identity with any of the grouped wallets would reveal the user's identity, together with the wallets and transaction history they own.

To avoid transactions being linked and protect users' anonymity, blockchain developers design mixing services, such as Tornado Cash [7], TC Nova [6], Railgun [35], and Cyclone [9]. These services allow users to obfuscate their transactions by interacting with the mixing contracts rather than directly transferring tokens from one wallet (or user) to another. These mixing contracts use sophisticated cryptographic mechanisms that hide the link between a transaction's sender and recipient, increasing the complexity of chaining transactions.

The efficacy of preserving user anonymity through mixing services heavily depends on how users employ them. To foster proper usage, service developers commonly provide users with an original guidebook before deploying the service. This guidebook contains a list of action patterns that can lead to transactions being linked and jeopardize users' anonymity and, therefore, should be avoided. We denote these actions as *anonymity-compromising patterns*, i.e., a sequence of transactions that enable attackers to link transactions. For instance, an impatient user may initiate a deposit followed by a withdrawal with the same amount and without an adequate wait time. These transactions appear subsequently on the blockchain, and even though a mixing service might be used, they are easy to link and, thus, expose the actual transactions the user made to other users (or wallets). Given that linking transactions is an essential step for an attacker to unveil a user's wallets and transaction history, these patterns potentially compromise the user's anonymity.

In an ideal case, strict adherence to the original guidebook provided by the service developers can substantially reduce the risk of compromising users' anonymity. However, recent research [43, 46, 47] has discovered gaps in existing guidebooks and demonstrated that users have unintentionally compromised their anonymity through actions that were not part of the guidebook. These works highlight the importance of proactively identifying anonymity-compromising patterns and adding them to the guidebook *before deploying the corresponding mixing service*. Existing methods for discovering anonymity-compromising patterns operate postmortem, i.e., they analyze and link past transactions that have occurred on the blockchain to extract such patterns. The postmortem nature of these methods makes it impossible to discover yet-unknown anonymity-compromising patterns. As a result, these methods cannot construct and enrich the guidebooks *before a mixing service is deployed*.

In this paper, we propose GuideEnricher (mixing service **Guide**book **Enricher**), a novel method for proactively

discovering anonymity-compromising patterns to enrich guidebooks. Technically speaking, we first simulate the Ethereum blockchain together with a specific mixing service. Then, we design a DRL agent to automatically finish a certain task (i.e., token transferring) in our simulator. We include two specific designs to guide the agent in discovering yet-unknown anonymity-compromising patterns (not included in the original guidebook) while finishing its task. First, we carefully design the agent's task such that, to finish its task, the agent has a high chance of compromising its anonymity. Second, we integrate a rule-based detector into the agent's environment. The detector monitors the DRL agent's transactions and identifies those that adhere to the known anonymity-compromising patterns in the current guidebook. We design a reward system for the agent that incentivizes task completion and encourages evasion of the detector's scrutiny. This design encourages the agent to explore previously unseen anonymity-compromising patterns while accomplishing its task. We then train our agent with the state-of-the-art policy learning method: proximal policy optimization (PPO) [38]. Furthermore, we design a second analysis step that clusters the agent's transactions and extracts the most representative transactions. Finally, our human experts will analyze these transactions to find and summarize novel anonymity-compromising patterns.

Through extensive evaluation and case studies, we first show the realism of our simulation system and the efficacy of GuideEnricher in training agents that successfully evade the rule-based detector while accomplishing their token transfer tasks. Second, we verify the generalizability of our trained agents as well as the scalability and efficiency of our method. Furthermore, we demonstrate that using our trained agent, in conjunction with a second analysis step, we can efficiently extract yet-unknown anonymity-compromising patterns without requiring significant human effort. This allows for the construction and enhancement of guidebooks before service deployment or the occurrence of actual anonymity compromises that impact real users. To the best of our knowledge, this is the first work that enables proactively anonymity-compromising action discovery. This is also the first work that demonstrates the utility of DRL-driven systems in enhancing user anonymity protection.

In summary, our work makes the following contributions:

- We design and develop GuideEnricher[1], a DRL-driven method that simulates user interactions with mixing services to facilitate guidebook construction.
- We evaluate GuideEnricher across various task setups and different mixing services, confirming its realism and efficacy in training effective agents. Moreover, we demonstrate that GuideEnricher can facilitate extracting anonymity-compromising patterns of Tornado Cash without requiring significant human effort.
- We present the usage of GuideEnricher in continuously enriching the guidebook of Tornado Cash by iteratively updating the rule-based detector and our evaders.

Note that our proposed method is generalizable across different mixing services. We first study Tornado Cash in detail, as it is one of the most widely used mixing services on the Ethereum blockchain [21]. We then expand our analysis to TC Nova [6], Railgun [35], and Cyclone [9] to demonstrate the generalizability of GuideEnricher.

2 Background

2.1 Anonymity in Ethereum

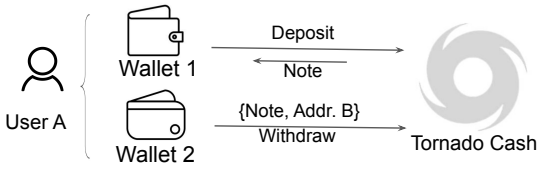
Anonymity Compromises in Ethereum. Ethereum is a decentralized, public blockchain that facilitates peer-to-peer transactions among users. One functionality is to transfer a certain amount of Ether[2] (or tokens) from one user's wallet to another user's wallet. Because of its inherent transparency, historical transaction records, including source and destination wallet addresses, are visible to all users on the blockchain. This accessibility, however, raises concerns regarding user anonymity. More specifically, by monitoring and aggregating transactions associated with the same wallet address, a malicious user can link transaction activities of a wallet and find other wallets that may belong to the same user. If the attacker can find one public record linking the user's identity with any of the grouped wallets, it would reveal the user's identity, together with their wallets and transaction history.

Mixing Services. Users of the Ethereum blockchain who want to enhance their anonymity rely on mixing services. At a high level, mixing services pool together multiple users' funds and then redistribute those funds in a way that breaks the linkage between source and destination addresses. Specifically, to initiate a transaction, the sender first deposits their desired amount of Ether into the mixing service. Then, in a later (and independent transaction), the user asks the service to withdraw the Ether to the recipient's wallet. This approach avoids direct transfers from sender to recipient on the public blockchain, making it challenging for attackers to link transactions and thus protecting users' anonymity.

The mixing process relies on Zero-Knowledge proof protocols, such as zk-SNARK [4], to obfuscate transactions while ensuring correctness. zk-SNARK stands for Zero-Knowledge Succinct Non-Interactive Argument of Knowledge, a cryptographic protocol that allows individuals to demonstrate possession of certain information without revealing the actual contents. zk-SNARKs are non-interactive zero-knowledge protocols that do not require back-and-forth communication between the information provider and the prover. zk-SNARKs offer fixed-size proofs, regardless of the complexity of the statements they validate. This property

[1] https://github.com/ucsb-seclab/GUIDE-ENRICHER

[2] Ether is the native currency on the Ethereum blockchain.



# Block	From	Func. call	Arguments	To
n	Addr. A1	Deposit	…	TC
n	Addr. A2	Withdraw	Note, Addr. B	TC
n	TC	Internal	…	Addr. B

Figure 1: Demonstration of money transfer via TC. Users A and B refer to the sender and recipient. "Addr. A1" and "Addr. A2" stands for the address of User A's two wallets. The last row in the table appears as an internal transaction in the withdrawal initiated by User A. Each transaction requires a transaction fee that is omitted in the figure.

makes zk-SNARKs well-suited for resource-constrained environments such as blockchains, significantly reducing computational overhead and enabling instant transactions.

Tornado Cash (TC) is a popular anonymity-preserving mixing service operating on the Ethereum blockchain [7]. It utilizes the zk-SNARK protocol to enable confidentiality and anonymity. Similar to other mixing services, TC breaks a transaction from a sender to a recipient into a two-step process, involving first a deposit into the TC service, followed by a subsequent withdrawal. As shown in Figure 1, the sender (User A) first initiates a deposit from their Wallet 1 to TC by invoking the deposit function of the TC smart contract with proper arguments. Then, TC's off-chain components generate a zk-SNARK proof – denoted as a "Note" – and send it back to the sender. The sender can initiate a withdrawal using a different wallet address from the one in the deposit, by passing the Note and the address of the recipient's wallet (Address B) to the TC's withdrawal function. The TC contract first verifies the Note's authenticity. Furthermore, TC ensures that the withdrawal amount does not exceed the deposited sum. After validating the withdrawal request, TC makes an internal function call to transfer the desired amount of Ether to the recipient's wallet. This process will appear as two individual (and independent) transactions on the blockchain (Figure 1). As a result, the direct link between the sender and the recipient is broken, enhancing the anonymity of the user's transactions.

Besides mixing services, the Ethereum blockchain integrates alternative mechanisms to preserve user anonymity. This includes zk-Rollups [12], and stealth address services [14]. These mechanisms are beyond the scope of this work.

2.2 Deep Reinforcement Learning

DRL trains deep neural network-based agents to solve sequential decision-making problems in complex environments. For example, DRL is widely used to train autonomous agents to play video games (e.g., Go [41] or StarCraft [10]) or solve networking challenges (e.g., [33, 49]).

Modeling an RL problem. An RL problem involves an agent continuously interacting with an environment to accomplish a pre-defined task. At each time step, the agent observes the current environment state and takes a corresponding action. The environment receives this action and transits to the next state. At the same time, the environment rewards the agent, indicating how much the action helps in accomplishing the task. The agent's goal is to learn an optimal policy, such that taking action following this policy maximizes the agent's total reward. In DRL, the policy is modeled as a deep neural network (a policy network), which takes as input the environment state and outputs the action. Solving a DRL problem is equivalent to learning the parameters of this policy network.

Formally, a DRL problem is modeled as a Markov Decision Process (MDP), represented as a 4-tuple $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R} >$. Here, $\mathcal{S}$ and $\mathcal{A}$ are the state and action sets. $s^{(t)} \in \mathcal{S}$ and action $a^{(t)} \in \mathcal{A}$ represent the state and action of the agent at time t. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the state transition function, which is typically unknown. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, where $r^{(t)}$ represents the reward of the agent at time t. The goal is to train a policy network $\pi(a|s)$ for the agent that maximizes the agent's total reward. The total reward can be modeled by the *state-value function* $V_\pi(s)$ defined as

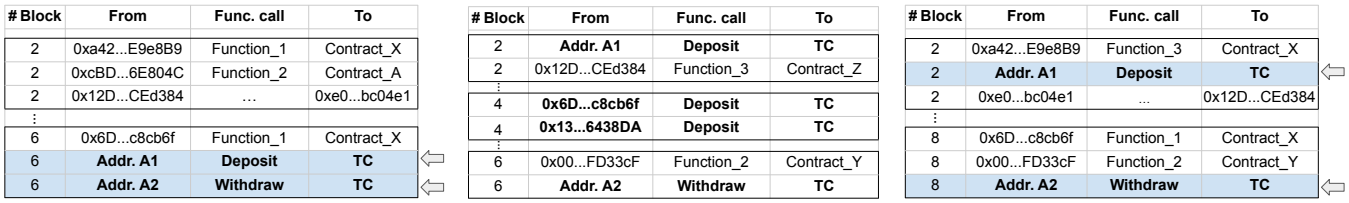
$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s)(r_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{T}_{ss'}^a V_\pi(s')), \tag{1}$$

where $\gamma \in [0,1]$ is a discount factor that controls the agent's emphasis on current or future rewards. $V_\pi(s)$ measures an agent's expected total reward starting from state s. An optimal policy is obtained by maximizing this function, enabling the agent to receive the maximum reward from the environment.

Proximal Policy Optimization. Instead of directly maximizing the value function ($\max_\theta V_\pi(s)$), PPO proposes the following surrogate objective function:

$$\begin{aligned} &\max_\theta \mathbb{E}_{(a^{(t)}, s^{(t)}) \sim \pi_{\theta_{old}}} [\min(\text{clip}(\rho^{(t)}, 1-\varepsilon, 1+\varepsilon) A^{(t)}, \rho^{(t)} A^{(t)})], \\ &\rho^{(t)} = \frac{\pi_\theta(a^{(t)}|s^{(t)})}{\pi_{\theta_{old}}(a^{(t)}|s^{(t)})}, \quad A^{(t)} = A_{\pi_{\theta_{old}}}(a^{(t)}, s^{(t)}). \end{aligned} \tag{2}$$

Here, $\text{clip}(\rho^{(t)}, 1-\varepsilon, 1+\varepsilon)$ denotes clipping $\rho^{(t)}$ to the range of $[1-\varepsilon, 1+\varepsilon]$ and ε is a hyper-parameter. As depicted in Eqn. (2), the objective function maximizes the advantage function $A^{(t)}$ (derived from the state-value function in Eqn. (1)) while constraining the disparity between an updated and prior policy. By solving this objective function, the PPO algorithm ensures a monotonic increase in the agent's total rewards during training. Thanks to this characteristic, PPO achieves faster convergence and better stability compared to other algorithms, e.g., A2C [31] and A3C [31]. During the training process, we can only collect the instant rewards without knowing the expected total reward of the agent (i.e., value function $V_\pi(s)$). As such, PPO trains another neural network to approximate the value function. In each interaction, it first updates the value



# Block	From	Func. call	To
2	0xa42...E9e8B9	Function_1	Contract_X
2	0xcBD...6E804C	Function_2	Contract_A
2	0x12D...CEd384	...	0xe0...bc04e1
⋮			
6	0x6D...c8cb6f	Function_1	Contract_X
6	**Addr. A1**	**Deposit**	**TC**
6	**Addr. A2**	**Withdraw**	**TC**

(a) TC transaction without waiting.

# Block	From	Func. call	To
2	**Addr. A1**	**Deposit**	**TC**
2	0x12D...CEd384	Function_3	Contract_Z
⋮			
4	**0x6D...c8cb6f**	**Deposit**	**TC**
4	**0x13...6438DA**	**Deposit**	**TC**
⋮			
6	0x00...FD33cF	Function_2	Contract_Y
6	**Addr. A2**	**Withdraw**	**TC**

(b) TC transaction with proper waiting.

# Block	From	Func. call	To
2	0xa42...E9e8B9	Function_3	Contract_X
2	**Addr. A1**	**Deposit**	**TC**
2	0xe0...bc04e1	...	0x12D...CEd384
⋮			
8	0x6D...c8cb6f	Function_1	Contract_X
8	0x00...FD33cF	Function_2	Contract_Y
8	**Addr. A2**	**Withdraw**	**TC**

(c) TC transaction with insufficient waiting.

Figure 2: Demonstration of a money transfer by User A (from Wallet A1 to Wallet A2) via TC, with different wait times between the deposit and withdrawal transaction (we show two or three transactions in each block). Figure 2a shows the user's actions without consulting the guidebook. Specifically, User A makes a deposit and immediately withdraws the funds, without any waiting (highlighted with blue canvas). Figure 2b demonstrates that, with a proper wait time, there are other transactions that involve TC, making it difficult to link a deposit with the corresponding withdrawal. Figure 2c further shows that although User A waits, no other TC transactions occur before User A makes the withdrawal. As such, the link between the deposit and withdrawal is still clear. "Contract_i" represents other contracts on the Ethereum blockchain. "Function_i" refers to the invoked function of the corresponding contract. We omit the internal transactions from TC to the recipient's wallet invoked in withdrawals.

function using the Temporal-Difference method [22]. Then, it computes the advantage A using the current value function and updates the policy according to Eqn. (2).

3 Problem Scope

3.1 Limitations of Mixing Services

Even when using mixing services, a user's anonymity can still be compromised. Specifically, users may perform incautious transactions, as shown in Figure 2a. With the mechanism of TC in mind, an attacker can search for (and link) a pair of deposit and withdrawal transactions that indicate a transfer of funds between two wallets of the same user. Note that we do not consider attacks against the cryptographic functionality of zk-SNARK and the implementations of the mixing service.

To prevent the linking of deposit and corresponding withdrawal transactions (and to protect user anonymity), it is critical for mixing service developers to provide a guidebook, specifying a list of actions that potentially break anonymity (denoted as anonymity-compromising patterns) and how to avoid them [46, 47]. By cautiously avoiding anonymity-compromising patterns, users can significantly reduce the risk that their deposit and withdrawal transactions are linked when leveraging mixing services. For example, TC recommends the sender to wait for a certain period of time before initiating a withdrawal. As shown in Figure 2b, this waiting time allows more deposits (from other users) to accumulate in the pool, making it challenging to link deposits with withdrawals.

In fact, mixing service developers typically provide a user guidebook with instructions to prevent anonymity compromises. For TC, for example, this guidebook suggests a certain wait period between deposit and withdrawal and recommends that users avoid using the same wallet address for deposit and withdrawal [7]. Unfortunately, anonymity compromises are still being reported [46], even with this reported, even with this original guidebook. As mentioned, the TC guidebook suggests that users wait a certain period (given as wall clock time) before they withdraw funds. However, as shown in Figure 2c, it is possible that the user indeed waits for the suggested time (or even longer). But no other transactions involving TC might be broadcast during that time. Hence, it might still be relatively easy for an attacker to link the two transactions and recover the original money transfer. Finding this issue will motivate TC developers to expand the guidebook by asking users to wait for a certain number of TC-related transactions instead of having a fixed waiting time.

Unfortunately, it is hard to compile a comprehensive guidebook just through manual analysis and testing [46, 47]. For example, another rule in the original guidebook recommends that users should not use the exact same gas price for a consecutive pair of deposits and withdrawals. Our method (detailed in Section 6) finds a pattern that makes multiple deposits with the same gas price before initiating the withdrawal. This is a rare usage of TC that mainly happens when a user wants to conduct multiple deposits and withdraw in a very short period of time. In other words, observing a large number of withdrawal and deposits transactions that use the same gas price in the same or adjacent blocks could allow one to link these transactions and contribute to de-anonymize the user.

The analysis above demonstrates that discovering yet-unknown anonymity-compromising patterns outside of the current guidebook helps make the guidebook more comprehensive and precise. The anonymity of users can be better protected if developers can identify as many anonymity-compromising patterns as possible and provide a guidebook on how to avoid them before deploying the service contract. However, existing research [46, 47] typically identifies anonymity-compromising patterns in a rearward-looking fashion, i.e., they include and summarize the anonymity-compromising patterns once they are observed and exploited on the blockchain. As such, it is extremely challenging for service developers to provide proactive guidance before deploying the service.

3.2 Problem Setup and Goals

We consider the setup where we are given a mixing service on the Ethereum blockchain, together with an original guidebook. This guidebook contains suggestions for avoiding the most obvious anonymity-compromising patterns, such as not using the same address for deposit and withdrawal. Starting from this guidebook, our goal is to discover more anonymity-compromising patterns before deploying the contract. Then, by adding these patterns to the guidebook, we provide users with a more complete guidebook that substantially protects their anonymity. Note that, in addition to discovering as many anonymity-compromising patterns as possible, we also aim to not compromise the anonymity of actual users. In other words, we aim to design an automated system that pinpoints anonymity-compromising patterns in a forward-looking fashion rather than conducting postmortem analysis of deposits and withdrawals linked to the same user.

We use Tornado Cash as a concrete instance to introduce our proposed technique, given that it is one of the most popular anonymity providers [21]. As demonstrated in Section 5, our method is generic and can also be applied to other mixing services or other types of smart contracts.

4 Our Approach

We propose GuideEnricher, a DRL-driven system for enriching guidebooks of mixing services. At a high level, we first develop a simulator to replicate the money/token transfer process on the Ethereum blockchain, with Tornado Cash as the mixing service. This simulator enables the discovery of anonymity-compromising patterns without relying on historic blockchain data and exposing real users. Subsequently, we design a DRL agent that performs token transfers in the simulator. We include specific designs that guide the agent toward exploring previously yet-unknown anonymity-compromising patterns. In this section, we first provide an overview of our approach, together with our design rationales. Then, we discuss the technical details of the simulator and the DRL agent.

4.1 Technical Overview

Building the blockchain simulator. Our simulator replicates the functionality needed to capture money flows on the Ethereum blockchain, including token transferring, balance tracking, etc. We also emulate the functionality of TC as the mixing service. Furthermore, we implement an original user guidebook that lists basic anonymity-compromising patterns extracted from TC's documentation and existing studies. Finally, we implement regular users in the simulator who consistently avoid the known anonymity-compromising patterns in the original guidebook when making transactions.

Insights of using DRL. Existing methods focus on historical transactions to extract anonymity-compromising patterns. To discover yet-unknown patterns, our approach simulates potential users and collects their transaction traces for pattern extraction. Specifically, different from the regular users crafted above, which only take simple actions following a few fixed rules, we require a user who is "creative" and who keeps updating its strategy to explore a rich set of action patterns in interacting with the mixing service. We design this "creative" user as a DRL agent following the intuition that a DRL agent can effectively search for proper strategies to finish a sequential decision-making task. We give the agent the task of transferring Ether to other users using TC. While learning to accomplish this task, the DRL agent will automatically explore different and diverse actions (in the entire action space) as it is engaging with TC. By analyzing the transaction traces of the agent, human analysts can then discover previously unknown anonymity-compromising patterns.

Customized designs of our DRL system. To maximize the probability of finding interesting patterns, we include two designs that guide the agent to explore possible anonymity-compromising actions. First, we craft the task for the agent so it is easy for the agent to compromise its anonymity when trying to finish this task. Second, we include a rule-based detector in the environment that integrates (and looks for) the discouraged action (anonymity-compromising patterns) from the guidebook. This detector monitors the agent's actions and decides whether they violate the guidebook. The agent (denoted as the *evader*) needs to make a sequence of transactions to evade the detector and finish the given task. It will receive a positive reward if it finishes the task without violating the guidebook. Otherwise, it will receive a negative reward. The agent is pushed into avoiding known anonymity-compromising patterns. But its task will potentially introduce de-anonymity. As a result, the agent is forced to explore yet-unknown anonymity-compromising patterns to receive a high reward. When analyzing a well-trained evader's transaction traces, human analysts are much more likely to discover yet-unknown anonymity-compromising patterns. Figure 3 shows an overview of our proposed DRL system.

It is important to note that our ultimate objective is to discover yet-unknown anonymity-compromising patterns. However, we cannot directly set this as the agent's task because we lack prior knowledge of these patterns before agent training. Therefore, we take the approach of training the agent to interact with the mixing contract while encouraging it to explore actions that have the *potential* to compromise anonymity. Following the agent's training, human experts analyze the agent's actions to identify yet-unknown anonymity-compromising patterns. This process is similar to program fuzzing techniques, where a fuzzer explores program states and adjusts its strategy toward discovering vulnerabilities based on feedback signals. Human experts still play a crucial role post-fuzzing to confirm the discovery of yet-unknown vulnerabilities and assess their exploitability.

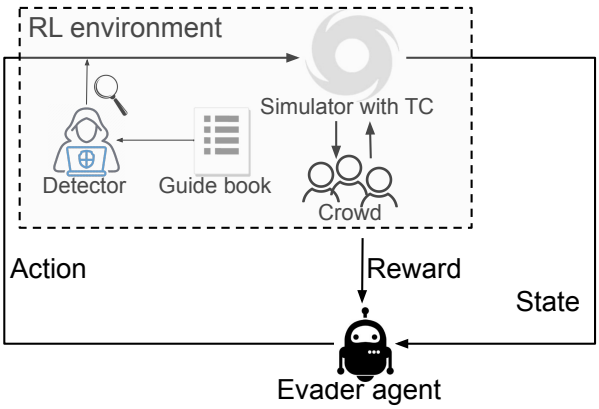


Figure 3: Overview of GuideEnricher. The DRL environment contains the simulator, the detector, and the crowd. The evader agent interacts with the environment through its actions.

4.2 Simulator and Environment Construction

Ethereum simulator with Tornado Cash. We built a simulator to replicate the basic functionality of the Ethereum blockchain, together with the mechanism of Tornado Cash. As shown in Figure 3, our simulator includes a crowd, where each user is allocated a random number of wallets (used to store tokens and invoke smart contracts), each containing a random amount of Ether (tokens). Users can perform various activities, such as transferring tokens among their own wallets or across different users. They can also engage with TC and other simulated contracts. An interaction with a simulated contract (other than TC) will be recorded as a transaction "from Addr. X to Other." These contracts do not implement any functionality nor do they change the state of the blockchain. We include these contracts to diversify the transaction types, making our simulator more realistic without introducing too much overhead.

When running a simulation, we also introduce a "crowd of users" who generate background transactions. The users of the crowd follow simple (hard-coded) rules when interacting with the (simulated) TC contract. The main goal is to create reasonably realistic, valid transactions. For instance, users cannot withdraw more funds than they have previously deposited. They also cannot access or withdraw from other users' wallets. We describe the rules that drive the behavior of the crowd in more detail in Appendix C. Despite the simplicity of our rules, as shown in Section 5, the behavior of crowd users and real TC users are aligned in key areas such as address usage frequency and reusability. Note that when training the evader, we do not update the rules for crowd users. Also, not all users perform transactions at each step. Instead, we let the crowd perform a number of actions that meets the minimum number of crowd transactions required between each action taken by the DRL agent. We simulate the entire process of exchanging tokens with TC (introduced in Section 2), including Note authentication and transaction validation. The only difference is that, to improve computational efficiency, we use plaintext Notes instead of cryptographic proofs.

Original guidebook and detector. We design our original guidebook based on the anonymity-compromising patterns reported in [46, 47]. A list of these anonymity-compromising patterns can be found in Appendix A.1. These behaviors are relatively straightforward ones, and they are summarized by [46, 47] through a postmortem analysis of historic transactions. Based on this original guidebook, we implement a rule-based detector that monitors our evader's actions and identifies those that violate the guidebook (that is, transactions that adhere to the known anonymity-compromising patterns).

DRL environment. As shown in Figure 3, our DRL agent's environment includes the simulator with TC, the crowd, and the detector. The crowd users follow the original guidebook when interacting with the TC server, fostering a more realistic blockchain simulation. Our DRL-driven evader operates in this environment to finish its given task (transferring certain tokens to other wallets). This is a single-agent environment in that all the users other than the evader are rule-based.

We build our simulator to be flexible, splitting it into modules for easy adaptation to various mixing services and blockchains later on. Specifically, we implement each module as a Python package with base classes containing the basic functionalities of that module. Users can simulate new contracts and blockchains by extending base classes and overriding methods. For example, the base class for the blockchain module has methods commit_transaction, get_current_block_id, get_gas_price, etc. Users can overwrite the base methods and change the logic based on the blockchain they want to simulate. Similarly, users can also overwrite the basic functions of the contract module, such as update_balance, get_balance, etc. See Appendix C for more details about the base classes and functions. Our framework also enables the users to customize the architecture of the evader agent's policy network (recall that in DRL, an agent's policy is a neural network called a policy network).

4.3 Agent Design and Policy Learning

Initialization and Task. We initialize our evader with several wallets with certain balances (see Section 5.2 for the actual configurations). We denote this initial asset of the evader as a "challenge table." The evader's task is to transfer all or part of the given Ether tokens from the challenge table to other wallets owned by the evader (see Section 5.2 for actual tasks). We design the task that may lead to anonymity compromises by crafting the challenge table, the number of transferred tokens, and the target wallets. In addition, we design the task to be complicated such that the agent needs to take a number of actions to finish it. This gives the agent the space to search for different action patterns. For example, the evader is assigned 250 wallets, in which five wallets have tokens, whereas the other 245 are empty. As the evader is only given five non-empty wallets, the agent is likely to use the same wallets for a subsequent deposit and withdrawal, which can be easily linked together. As such,

this task may lead to anonymity compromises. The task is also complicated in that the agent first needs to figure out which wallets have tokens. The TC contract only supports fixed deposit and withdrawal amounts that are predefined in the contract. We use one token for each transaction in this work. As such, the agent needs to take multiple transactions to transfer the tokens from these five wallets to the empty wallets. The agent needs to transfer all the tokens but can decide the distribution of targets. We do not specify the recipient wallets or the amount of tokens received by each recipient. If the agent accomplishes its tasks while bypassing the detector, it will likely trigger some yet-unknown anonymity-compromising patterns not covered by the original guidebook. Human analysts can then extract them by analyzing well-trained agents' transaction traces.

State and observation. State $\mathcal{S}$ represents the environment status. Designing an appropriate state space is crucial for policy training [16]. The key is to include only necessary information while eliminating potential noise. We also need to constrain the dimension of the state representation to avoid making the state space excessively large. In our problem, the necessary information mainly includes the evader's status and the current status of the TC contract. More specifically, the evader's own status is described by the overall balance of the challenge table, the balance of the wallets used in the previous time steps, and its last action. The status of TC mainly refers to the balance of the TC contract. This information is necessary for the agent to make decisions. For example, the agent needs to be aware of the current balance of a wallet before making a deposit to TC using this wallet. Otherwise, the agent may take an invalid action that deposits an amount that exceeds what the wallet has. In addition, the agent should also be aware of the status of the crowd, reflected in the status of TC, to better plan its action. Following this insight, we instantiate the state/observation s as a vector with 11 elements, including the agent's previous action, current balance, and the current balance in the TC contract (see Table 4 for a detailed description of the state vector).

Action. We design an action as a vector, $a = [a_1, a_2, a_3, a_4]$. This first dimension has two discrete values $a_1 = 0/1$, indicating either a deposit (0) or withdraw (1) action. The second dimension a_2 is an integer, ranging from zero to a maximum value (denoted as MAX_WAIT), that represents the number of transactions the agent will wait for from the crowd before initiating its next action.[3] A larger a_2 indicates a longer wait time, and $a_2 = 0$ means the agent takes the next action without waiting. The third (and fourth) dimensions store discrete values, indicating the deposit (withdrawal) wallet address if the agent chooses a deposit (withdrawal) action. The possible values for a_3 and a_4 are the total number of available wallets of the agent. Continuing the example above, where the agent is given 250 wallets (five of them have tokens). The possible values for a_3 and a_4 are 1-250. When the agent chooses a deposit action, the withdrawal address will be ignored. For example, an action $a = [0, 10, 20, 0]$ means the agent will wait for 10 transactions from the crowd before using the 20^{th} wallet to take the next deposit action. Since the TC contract only supports fixed deposit and withdrawal amounts that are predefined in the contract, there is no need for us to specify the amount in the agent's actions. Instead, we utilize the fixed deposit amount that TC uses, which is one token for each transaction. We acknowledge other choices are also available.

Reward. We design our reward R based on our agent learning goals: 1) picking up the rules of valid transactions and 2) finishing the task without being caught by the detector. R is a dense reward that is assigned after each action taken by the agent. Specifically, the agent will receive a negative reward $R = -10$ if the current action is invalid (e.g., the amount of tokens to be withdrawn exceeds the balance of the withdrawal wallet). If the current action is valid, but the detector identifies it as an anonymity-compromising pattern, the agent will receive a negative reward $R = -1$. Otherwise (the action is valid and it is not caught by the detector), the agent will receive a positive reward $R = 1$. In Section 5, we show that setting 1 (-1) as the reward (penalty) for bypassing (or being caught) by the detector is enough to guide the agent to evade the detector with a high probability. Maximizing this reward will guide the agent to learn a proper policy that takes valid and stealthy actions to accomplish its task. Given that it is difficult to add constraints to the RL process, we do not enforce rules in the DRL agent. The agents are randomly initialized and learn by trial and error, guided by rewards. In Appendix A.3, we demonstrate that our method is robust to the variations in negative reward. We do not design a final sparse reward awarding the agent for task completion. Given that our main goal is to guide the agent in learning to execute valid transactions and discover yet-unknown anonymity-compromising patterns. In our setting, each round concludes either when the agent accomplishes the task or when it reaches a maximum number of time steps (10,000 in our setup). We have observed that the agent typically completes the task within 500 steps, making it unnecessary to introduce an additional reward to incentivize task completion. Different action choices will affect the reward being received. For example, selecting different wallet addresses helps the agent bypass the rule-1 (Address Match) in the initial guidebook (Table 6) and thus will be assigned a positive reward.

Policy construction and learning. We model our agent (and its policy) as a deep neural network, denoted as π_θ. This network takes as input the state vector s_t and outputs the desired action a_t at the current time step. With this modeling, learning the policy is equivalent to solving for the optimal parameters of the policy network. As mentioned in Section 2, the objective function is to maximize the long-term reward for the agent, i.e., $\max_\theta \ V_{\pi_\theta}(s)$, where $V_{\pi_\theta}(s)$ is computed based on the reward defined above (i.e., sum over all the individual rewards). Here, we use the Proximal Policy Optimization

[3] In the DRL system, one time-step refers to the point when the agent takes a deposit or withdrawal action. In between two time-steps, the agent can wait for multiple transactions from the crowd. The actual (wall clock) time interval between two time-steps of the DRL system can vary.

(PPO) algorithm to train the policy. As discussed in Section 2, PPO solves the policy parameter θ by maximizing a surrogate objective function (Eqn. (2)). PPO guarantees the monotonic increase of the value function and thus improves the training efficacy and stability of the trained agents.

In each training iteration, before updating the agent's policy and value network, we need to run the current policy in the environment to collect a set of episodes (i.e., policy evaluations). An episode (round) of the game ends when the agent accomplishes the task, or the game reaches a maximum amount of (pre-specified) time steps. We collect the states, actions, and rewards in this round $(s_0, a_1, r_1, s_1, \ldots, a_{t-1}, r_{t-1}, s_t)$ as an episode. The collected episodes will then be used to update the policy. Our training process iteratively performs policy evaluation and update until it convergences, i.e., the agent's total reward no longer increases.

5 Implementation and Evaluation

We first evaluate whether our simulation is realistic and whether GuideEnricher can train effective evader agents to finish a certain task. Then, we test whether the trained evader agent is still effective (generalizability) when varying the number of transferred tokens and target wallets. Furthermore, we evaluate the effectiveness of our learning algorithm when increasing the size of the challenge tables (scalability) and verify our reward design via ablation studies. Finally, we test GuideEnricher on more mixing services and compare our selected learning algorithm PPO with other learning methods.

5.1 Implementation and Experiment Setup

Implementation. We use the `OpenAI-gym` [5] package to implement our reinforcement learning environment. `gym` is a Python library that supports customized designs of DRL environments. It provides standard APIs for communicating between the environment and the learning agent, i.e., for exchanging reward, action, and state information. We implement our DRL agent with `stable-baselines` [18], a Python package that supports constructing deep neural network-based agents and incorporates multiple learning algorithms, including PPO. To improve the training efficiency, we use `Ray` [27], a distributed reinforcement learning framework, to wrap our learning algorithm and, thus, enable parallel training. With this implementation, our training can be completed in a reasonable time (about 48 hours on average). Regarding the hyper-parameters (e.g., policy model architectures, training iteration, learning rate), we select the widely adopted choices suggested by the original PPO implementation. We keep them the same across all experiments (see Appendix A.2).

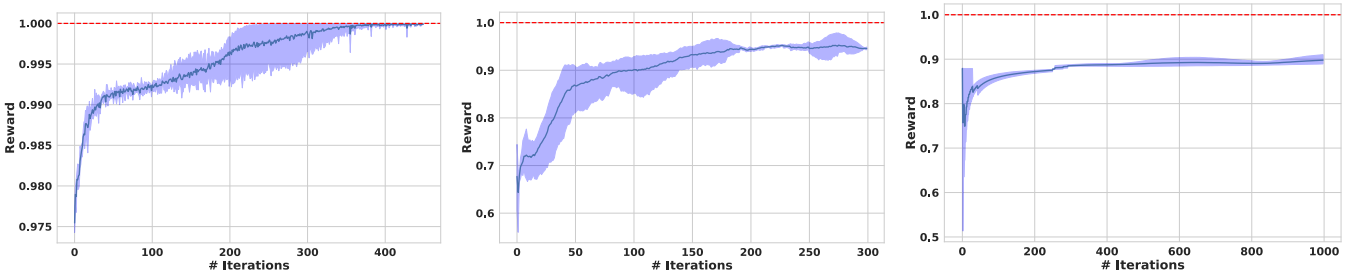
Evaluation metric. Different from supervised learning, online reinforcement learning does not have a pre-collected training set. Instead, it performs policy evaluation to collect episodes and update the agent in each iteration. In our experiment, we follow a commonly used approach and measure the reward of the evader agent at each iteration. That is, each time we update the agent during the training, we evaluate the agent for five rounds and calculate the mean reward. As a result, we will report the evader agent's reward changes across the training process. This helps to reveal the stability and convergence speed of the training process. We run each agent training process four times with different initial seeds for the RL algorithm. We report the mean and variance of the training results. This helps remove randomness and assess the stability of the training algorithm. After obtaining an agent from each run, we then run each of the final four agents against the detector for 1,000 episodes and record the corresponding rewards. We also report this testing result to assess the final effectiveness of the obtained agent. For better visualization, we normalize the agent's reward in each round to be between 0 and 1.

5.2 Experiment Design

Exp-0 Realism of our simulator. Before training the DRL agents, we conduct an experiment to validate the fidelity of our simulator and the realism of the crowd users. Specifically, we conduct a differential analysis between our simulator and the real (on-chain) Tornado Cash contract. To this end, we extract historical transactions from the real Tornado Cash 10 ETH contract. We then start with an empty, simulated TC contract and replay each historic transaction on the simulator, as a sequence based on their time stamps. After executing each transaction, we compare the balances in our simulator with the state of the real TC contract. We can assess whether the simulator is faithful by checking if its balances match the real TC contract.

In addition (and independent of the differential analysis discussed in the previous paragraph), we count the number of transactions generated by the crowd during our experiments, and we compare them with real historical transaction data. Moreover, we calculate the address reuse rate, a key factor in linking transactions, for both the real transactions and the simulated activity in our experiments. Here, the address reuse rate is the percentage of addresses that have been used for both deposits and withdrawals. A similar reuse rate between the experiment transactions and the real ones can demonstrate the realism of our crowd users.

Exp-1 Agent effectiveness. We first evaluate the effectiveness of our learning algorithm and the trained evader agent. Specifically, we set the challenge table for the evader as having three wallets where each wallet has three tokens. The agent is given 250 wallet addresses to interact with TC (without the knowledge of which wallets contain tokens). Its goal is to transfer all the tokens to any subset of the remaining 247 empty wallets. We do not constrain the way of distributing the tokens, i.e., they can be transferred to one or multiple wallets. Note that we provide the agent with more recipient (destination) wallets than is strictly needed to avoid the agent over-fitting to



(a) Exp-1. (b) Exp-3 with a single non-empty wallet. (c) Exp-3 with 10 non-empty wallets.

Figure 4: The agent's normalized reward in Exp-1 and Exp-3. The x-axis is the training iteration, and y-axis is the normalized reward. The solid lines are the average, and the lighter bands indicate the variations between the maximal and minimal reward.

a certain wallet address range. Meanwhile, we also initialize the crowd with 250 wallets with 100 tokens each. The users in the crowd can transfer tokens to each other, the TC contract, or other simulated (dummy) contracts. With this setup, we train the agent to accomplish the task and record the agent's reward during the training process. As mentioned above, we further vary the initial seeds of the RL algorithm and train the agent four times. We report the mean and variance of the agent's reward across these trials.

Exp-2 Agent generalizability. We evaluate the trained agents' generalizability against the variations in the number of target wallets and tokens needed to be transferred. First, we vary the given empty wallets from 247 to 497, 747, and 997 while keeping the same challenge table as Exp-1. The agent's task is still to transfer out all nine tokens. These variations result in three new setups (in addition to the setup in Exp-1). Second, we fix the challenge table and the empty number of wallets (247) and vary the task for the agent, i.e., transferring 6, 7, and 8 tokens to the empty wallets. These variations give another three new setups. In each of the six setups, we run the four agents trained in Exp-1 for 1,000 episodes and report the mean and standard deviation (STD) of the rewards. We also compute the percentages of the number of times that the agent takes invalid actions and evades the detector in each episode, respectively. We report the mean and STD of the invalid action rate and evading rate across the testing episodes.

Exp-3 Algorithm scalability. We assess the scalability of our algorithm to large challenge tables. First, we create a challenge table with a single wallet holding 500 tokens, and the agent is given 999 empty wallets. Second, we also give the agent 1,000 wallets, with ten wallets containing 50 tokens each, while the remaining wallets are empty. In both setups, the agent's task is to transfer all tokens in the challenge table to the empty wallets. We train the agent four times and report its reward changes across the training process.

Exp-4 Ablation study. We conduct an ablation study to test two aspects of our reward design, utilizing the task setup from Exp-1. First, we modify the reward by reducing the penalty strength for invalid actions from -10 to -1 to examine whether this adjustment can still guide the agent to prioritize valid transactions. This variation tests whether our learning method is robust against the changes in relative penalty strength between taking invalid actions and being caught by the detector. Second, we introduce an additional reward of 10 at the final state as an incentive for the agent to complete the task and a negative reward of -10 for not completing the task. For each setup, we retrain the agent under the revised reward and record the agent's testing performance over 1,000 episodes. The performance measure includes task finishing rate, invalid action rate, and evading rate.

Exp-5 PPO vs. other learning algorithms. We repeat Exp-1 using the A3C [31] and A2C [31] algorithms and evaluate whether the PPO algorithm provides better results.

Exp-6 Generalizability to other mixing services. To demonstrate that our framework is general, we test it with more mixing services. To this end, we start with a comprehensive analysis of the ecosystem of Ethereum mixing services (Appendix C). Our analysis yielded three widely recognized implementations: TC Nova [6], Cyclone [9], and Railgun [35]. Cyclone is essentially a copy of TC and implements the same functionality, thus, no changes to our analysis were necessary. We add support for the other two services in our simulator. Then, we train an agent – using the same configurations as for Exp-1 — through three training iterations. Subsequently, we calculate and present the normalized mean reward attained across TC Nova and Railgun games. **TC Nova [6]** is similar to TC but allows users to deposit and withdraw arbitrary amounts of tokens. We modify the actions space facilitating TC nova features, and we expand the observation space of the agent, allowing it to learn more effectively. We utilize the same guidebook to train the agent. **Railgun [35]** uses an implementation that is similar to cross-chain bridges, letting users deposit an arbitrary amount of money and withdraw the same amount or less than the original deposit on another chain, using a zk-SNARK proof. In our simulation, we adjust the agent's action space to accommodate Railgun features and broaden the observation space, enhancing its learning capabilities. We also employ a simplified guidebook to train the agent, ensuring it does not withdraw the same amount as deposited on another chain (because withdrawing the same amount makes it easy to link transactions).

Task	Reward	Invalid rate (%)	Evading rate (%)
Exp-1	0.999(0.011)	0.129(1.119)	99.871(1.119)
6 tokens, 247 wallets	0.997(0.000)	0.066(0.662)	91.102(2.851)
7 tokens, 247 wallets	0.998(0.000)	0.062(0.622)	99.938(0.622)
8 tokens, 247 wallets	0.999(0.001)	0.103(0.758)	99.897(0.758)
9 tokens, 497 wallets	0.999(0.007)	0.121(0.817)	99.879(0.817)
9 tokens, 747 wallets	0.998(0.015)	0.213(1.411)	99.787(1.411)
9 tokens, 997 wallets	0.997(0.028)	0.240(2.003)	99.760(2.003)

Table 1: The performance of our agents trained in Exp-1 in different setups. The first number in each cell is the mean and the number in the parentheses is the standard deviation.

5.3 Experiment Results

Result of Exp-0. Our differential analysis shows that the balance in the simulator consistently matches the balance in the real TC contract after executing every historical transaction. This convergence strengthens our belief that the simulation accurately reflects the functioning of the TC contract, validating the fidelity of our simulator.

We then turn our attention to the activity of crowd users in our experiments. For the crowd users, we observe a 2.6% address reuse rate for the experiment transactions. This is only slightly higher than the address reuse rate observed for the real transactions, i.e., 2.54%. This result shows the similarity in behavior between the simulated crowd and real-world users in terms of the difficulty of linking transactions. In Appendix C, we conduct a more detailed analysis of address usage between experiment transactions and real-world ones. Also, note that users in our experiments (simulations) generally generate many more transactions than happened in the real world. For example, we repeat each game round 1,200 to 4,000 times and collect an average of 1 million to 1.5 million transactions. This far exceeds the real-world Tornado Cash transactions (around 45,000). This makes sense and is because we want our learners to find interesting evasion techniques by exploring more transaction patterns.

Result of Exp-1. Figure 4a shows the changes in the agent's rewards during the training phase. As shown in the figure, our agent exhibits rapid convergence to near-optimal rewards while exhibiting fairly low variance. This result demonstrates the effectiveness of our DRL agent in acquiring transaction rules and evading the rule-based detector.

Result of Exp-2. Table 1 shows the generalizability of the agents trained in Exp-1 across different setups. Specifically, Row 2 ("Exp-1") corresponds to the agents' performance in their training configurations (the setup in Exp-1). Rows 3-5 show the results for transferring a different number of tokens with the same challenging table and the same number of empty wallets. The agents' performance stays almost the same for transferring eight and seven tokens (Rows 4-5). However, we observe a relatively large drop (about 8%) in the evading rate when transferring only six tokens (Row 3). This is because the task becomes simpler, making it easier for the agent to choose anonymity-compromising actions in the guidebook. When the token number increases, the task becomes more complex and the agent needs actions to finish a task. This verifies our decision to choose more complicated tasks, as they are more useful for agents to discover anonymity-compromising actions outside of the guidebook. To improve the agent's evading performance for simpler tasks, we can fine-tune the agent with a stronger incentive (or penalty) for evading (or being caught by) the detector. The last three rows depict the results of transferring nine tokens from the same challenge table using a different number of empty wallets. The agents perform consistently well across all setups. Table 1 shows that our trained agent can generalize its learned strategies (including taking valid actions and evading the detector) with changes in the tasks and the number of empty wallets.

Reward design	Task finishing rate (%)	Invalid rate (%)	Evading rate (%)
Our design	100(0.000)	0.129(1.119)	99.871(1.119)
Variation-1	100(0.000)	0.136(1.461)	99.884(2.461)
Variation-2	100(0.000)	0.244(1.242)	94.214(1.173)

Table 2: The percentage of finished tasks and invalid actions of agents trained with different reward designs across 1,000 trials. Variation-1 represents the reward design that changes the penalty on invalid action from -10 to -1. Variation-2 represents the reward design that adds an additional final reward to encourage task completion.

Result of Exp-3. Figure 4b and 4c show the results of the two challenge table setups in Exp-3. Similar to Figure 4a, the agent's reward converges to a near-optimal point without too much variance. These results demonstrate that our method can learn effective agents for challenge tables with a relatively large number of tokens or wallets, confirming the scalability of our method. In Appendix B.1, we show that an agent trained with one challenge table may not exhibit enough effectiveness when applied to other challenge tables. Given that our primary goal is to discover yet-unknown anonymity-compromising patterns rather than training exceptional generalizable agents. We deem our method successful as long as it can train effective agents for distinct challenge tables.

Result of Exp-4. Table 2 shows the results of our ablation study. As we can observe from Row 2 ("Our design") and Row 3 ("Variation-1"), reducing the penalty strength only marginally increases the invalid rate. This result indicates that it is unnecessary to have a stronger penalty for making invalid actions than for being caught by the detector. In addition, the evading rate is still high, indicating -1 and 1 are enough for the agent to learn to bypass the detector. It also shows that GuideEnricher is robust against the variations in the relative penalty strength between taking invalid actions and being caught by the detector. This illustrates the practicality of our method, as our tool does not require one to meticulously tune the reward function to ensure the effectiveness of the trained agents. Row 2 ("Our design") and Row 4 ("Variation-2") in Table 2 further demonstrate that having an extra final reward does not introduce any extra advantage regarding the ability

to finish tasks. Instead, it actually harms the agent's capability to take valid actions and evade the detector. This is because the final reward is only awarded at the end of a round, which is less effective than our dense reward (assigned at each time step). As such, we remove this sparse reward to improve training efficiency. In Appendix B.2, we further show that varying key hyper-parameters (the maximum steps and policy architecture) has minor effects on agent training.

Result of Exp-5. Figure 5 shows that the PPO algorithm converges to the optimal reward, whereas the A3C [31] and A2C [31] algorithms do not reach the optimal reward. This demonstrates the advantage of PPO over other algorithms.

Results of Exp-6. Figures 6 and 7 display the results when performing Exp-1 (Section 5.2) with TC Nova and Railgun as the mixing service. In both experiments, the agent's reward converges to a high reward value (similar to what we get for TC), with minimal variation. This underlines that our system can be extended to other mixing services and allows agents to meaningfully learn.

6 Utility

We demonstrate the utility of GuideEnricher in two scenarios. 1) **Anonymity-compromising pattern discovery**: By analyzing the transaction traces of evader agents, the mixing server developers can identify yet-unknown anonymity-compromising patterns without relying on historic traces or instances that have occurred in the real world. We demonstrate that we can extract meaningful patterns from our DRL agents. 2) **Iterative discovery**: By updating the detector and the evader agent, we can pinpoint more anonymity-compromising patterns, which can be used to enrich the guidebook.

6.1 Pattern Discovery

Setup and method. We collect all the episodes of our evader agents trained in Section 5. To increase episode diversity, we include both the training and testing episodes of all the agents trained under different setups. With these episodes, we first filter out the ones with an evading rater lower than 90%, as they trigger the current detection rules and, hence, are less likely to contain yet-unknown anonymity-compromising patterns. Then, we group the remaining episodes using SOTA unsupervised clustering methods, DBSCAN [37] and K-means [1]. The run time of each clustering method is about 3-5 minutes. We vary the number of clusters and only select the results with a high silhouette coefficient, a popular unsupervised clustering evaluation metric [36]. We select six cluster results, each result has 3-5 clusters. For each set of these clustering results, we select 10 episodes near the centroid of each cluster as the most representative episodes for that group. Finally, we manually examine these representative episodes to find and extract yet-unknown anonymity-compromising patterns. To roughly quantify the amount of manual effort needed by our method, we report the average number of transaction traces analyzed by our methods to summarize anonymity-compromising patterns together with the average time needed to analyze one transaction trace. Note that this second analysis step requires experts who are familiar with Ethereum and mixing services (but do not need to be deep domain experts).

We deem an action pattern as an anonymity-compromising pattern only if human experts can provide a valid explanation for why it has the potential to compromise anonymity. In future work, we will design quantitative criteria for determining anonymity-compromising patterns. Note that we employ clustering methods in our second analysis step rather than the graph-based approaches used in existing research [26, 39]. This is because we already filter out the majority of non-related transactions, eliminating the need to construct a graph that captures the dependencies of all users on the blockchain.

Results. We find eight action patterns that our agents learned for bypassing the detector. Among these, we identify three anonymity-compromising patterns where we can explain why they can lead to anonymity compromises. Figure 8 demonstrates these patterns. First, all three patterns attack the detector's rule that inspects the gas prices of a pair of deposit and withdrawal transactions. Specifically, this rule corresponds to the known anonymity-compromising pattern where a user tends to reuse the same gas price for a pair of deposits and withdrawals using TC. This makes it easy to link them. The reason why users' transactions might exhibit such an action pattern is that the EIP-1559 protocol [11] of Ethereum requires a user to specify a gas price when transferring tokens using smart contracts (including TC). Users are often not sophisticated enough to manage the gas price. Thus, given that these transactions happen within a short period of time, they commonly end up using the same gas price. Also, many users leave it to their wallet software to select a suitable gas price. This will also lead to the same gas price being used for multiple transactions (that happen in close temporal proximity).

As shown in Figure 8a, our agents first learn to perform multiple deposits before making a withdrawal, all using the same gas price. The agent bypasses the rule above by avoiding making a single deposit and withdrawal. Instead, the agent takes multiple deposits and then makes one withdrawal. This is acceptable according to the rule book since there is no 1-to-1 mapping between one deposit and withdrawal (that share the same gas price). Nonetheless, we count this pattern as anonymity-compromising because it is a very rare transaction pattern. For an attacker who is familiar with the guidebook, it is not difficult to determine that such a pattern is a sign of a user who needs to take frequent actions in a short period of time while trying to follow the guidebook. Of course, if the user has enough time to schedule and assign gas prices, they will avoid this pattern naturally. As such, it is relatively straightforward to link deposits and withdrawals in such a pattern and consider them as originating from the same user. Our agents find two other patterns to bypass the same rule: Pattern-2: make one deposit followed

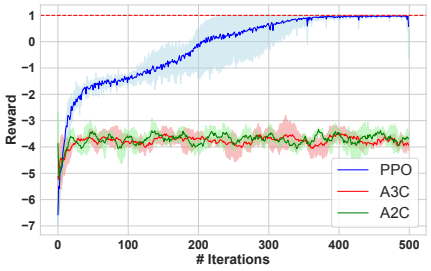


Figure 5: PPO vs. A3C and A2C.

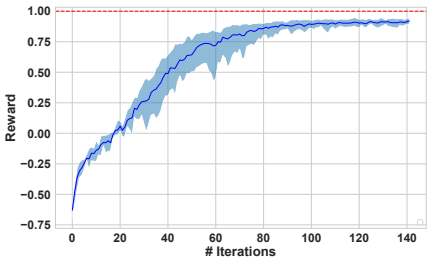


Figure 6: GuideEnricher on TC Nova.

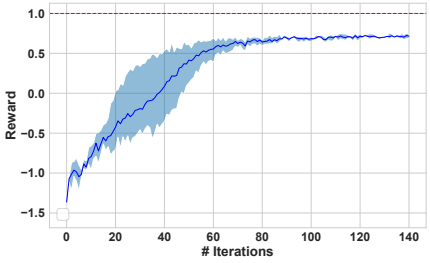


Figure 7: GuideEnricher on Railgun.

# Block	From	Gas Price	Method	To
	.			
	.			
n	0xa03...49e8B9	g2	Function1	Contract_Y
n	0xcBD...Fe6E80	g2	Function4	Contract_A
n	**Addr. A1**	**g2**	**Deposit**	**TC**
n	**Addr. A2**	**g2**	**Deposit**	**TC**
n	**Addr. A3**	**g2**	**Deposit**	**TC**
n	**Addr. A4**	**g2**	**Withdraw**	**TC**

(a) Pattern-1.

# Block	From	Gas Price	Method	To
	.			
	.			
n	0xa03...49e8B9	g2	Function1	Contract_Y
n	0xcBD...Fe6E80	g2	Function4	Contract_A
n	**Addr. A1**	**g2**	**Deposit**	**TC**
n	**Addr. A2**	**g2**	**Withdraw**	**TC**
n	**Addr. A3**	**g2**	**Withdraw**	**TC**
n	**Addr. A4**	**g2**	**Withdraw**	**TC**

(b) Pattern-2.

# Block	From	Gas Price	Method	To
	.			
	.			
n	0xa03...49e8B9	g2	Function1	Contract_Y
n	0xcBD...Fe6E80	g2	Function4	Contract_A
n	**Addr. A1**	**g2**	**Deposit**	**TC**
n	**Addr. A2**	**g2**	**Deposit**	**TC**
n	**Addr. A3**	**g2**	**Deposit**	**TC**
n	**Addr. A4**	**g2**	**Deposit**	**TC**

(c) Pattern-3.

Figure 8: Demonstration of the anonymity-compromising patterns discovered through our second analysis step.

by multiple withdrawals (Figure 8b); Pattern-3: finish all the deposits and then make withdrawals (Figure 8c). Note that, for Pattern-3, the withdrawal will appear in a later block (not shown in the figure). It is not difficult for the attacker to realize that these two patterns are performed by a user who tries to do high-frequency transactions while obeying the guidebook. As such, the attacker can link the transactions following these patterns.

Finally, we report the average number of transaction traces (episodes) analyzed by GuideEnricher to summarize anonymity-compromising patterns in Figure 8. We collect in total around 10,000 episodes. After our second analysis step, we only need to manually analyze 300 episodes to find these anonymity-compromising patterns. This process takes about 12 hours. This number demonstrates that GuideEnricher requires reasonable human effort when summarizing anonymity-compromising patterns. Furthermore, we calculate the percentage of episodes within each cluster, yielding the following results: Pattern-1 (Figure 8a) 0.15%, Pattern-2 (Figure 8b) 0.11%, and Pattern-3 (Figure 8c) 0.08%. Note that these patterns have yet to emerge in real historical transactions. This is because our system has a much larger transaction volume than the real TC history. We anticipate these patterns to emerge if TC continues to operate in the future.

6.2 Iterative Discovery

Setup and method. In the second experiment, we delve into the possibility of finding additional anonymity-compromising patterns through an iterative update of the detector and the evader. Specifically, we first update our rule-based detector as well as our crowd users by incorporating the three anonymity-compromising patterns identified in Section 6.1. As a result, our regular users will avoid these anonymity-compromising patterns, and the detector will catch any such transactions the evader makes. Then, under this updated setup, we give the agent six new challenge tables together with six tasks. Specifically, we start with a challenge table with one wallet containing three tokens. We give the evader 98 empty wallets and set the agent's task to transfer all the given tokens to the empty wallets (without specifying the token distribution). For simplicity, we represent this setting as [3] -> 98. We also configure the other five more complicated settings: [30] -> 249, [30,30] -> 248, [500] -> 999, [250, 250] -> 998. The agent's task in these five settings is to transfer all the tokens to given empty wallets without a specific distribution. We retrain the evader agents (trained on Exp-1 of Section 5) under these six settings and conduct a second analysis step to find yet-unknown (if any) anonymity-compromising patterns for the transaction traces of the updated agents. We updated the agents under six different challenge tables and tasks to increase the agent's probability of exploring diverse transaction patterns. While learning to finish these tasks, the evader could potentially learn to discover yet-unknown anonymity-compromising patterns. This is because the detector will identify all the known ones and thus result in a low reward for the evader. To achieve a high reward, the evader will learn to bypass the detector while learning to finish the task, which may explore yet-unknown anonymity-compromising patterns.

Challenge Table & Task	Task finishing rate (%)	Invalid rate (%)	Evading rate (%)
[3] -> 98	100(0.000)	0.056(0.297)	99.944(0.297)
[30]-> 249	100(0.000)	0.039(0.251)	99.961(0.251)
[30,30] -> 248	100(0.000)	0.047(0.284)	99.953(0.284)
[500] -> 999	100(0.000)	0.064(0.382)	99.358(0.382)
[250, 250] -> 998	100(0.000)	0.065(0.360)	99.348(0.359)

Table 3: The performance of the retrained evader agents trained with different challenge tables and tasks against the updated detector across 500 trials.

Result. Table 3 demonstrates the performance of the retrained evader against the updated detector (i.e., the detector incorporated with the anonymity-compromising patterns found in Section 6.1). As shown in Columns 2 and 3, the agent can always finish its task and seldom establishes invalid transactions. This demonstrates that the retrained agent still learns effective strategies to finish the task while maintaining its ability to take valid actions. This verifies the effectiveness of our learning algorithm in these retraining tasks. Moreover, we observe that the agent seldom takes the anonymity-compromising patterns in the updated guidebook (including the ones discovered in Section 6.1). This demonstrates that the evader agent indeed learns to avoid these patterns that the detector will catch. In other words, the high evading rates demonstrate that the retrained agents have a certain chance of exploring yet-unknown anonymity-compromising patterns. Note that we do not report the final reward in Table 3 in that it is proportional to the three values in the table (since all three values are high, the reward is also high). In addition, we also observe that the training converges faster than the first iteration. This is because we retrain the agents learned from Section 5, which already learn how to avoid invalid actions and bypass the original guidebook. The retrained agents do not need to pick up this knowledge again and, thus, can converge faster.

We find one yet-unknown anonymity-compromising pattern (Figure 9). This pattern tries to bypass the rule of "Linked Address" in the original guidebook. That rule detects if there are unique deposit-withdrawal or withdrawal-deposit pairs in TC with unique wallet pairs. Here, a unique wallet pair refers to two wallets only used in a deposit-withdrawal or withdrawal-deposit pair and that are not used by any other transactions in TC. If a unique wallet pair is used to transfer tokens outside TC, the corresponding deposit-withdrawal or withdrawal-deposit pair is linkable. All of the retrained agents bypass this rule by avoiding withdrawing tokens using the wallets that have been used to transfer tokens outside of TC. Instead, the agent first transfers the tokens to "fresh" wallets that have not been used before and then makes deposits and withdrawals using these fresh wallets. The attacker can examine the set of wallets that have been used to transfer tokens outside of TC. If the attacker can find a subset of wallets that always distribute the tokens to other "fresh" wallets rather than interacting with the TC contract, the attacker can then link the deposit-withdrawal associated with the "fresh" wallets. Our future work will explore guiding the agent to explore more specifically toward bypassing the newly added rules. For example, we can assign a stronger penalty when the agent takes the anonymity-compromising patterns corresponding to these rules than other patterns.

7 Discussion

Automatic detector update. As mentioned in Section 6.2, our approach requires manually updating the rule-based detector based on the yet-unknown anonymity-compromising patterns discovered by the evader. To automate this process, we can model the detector as another DRL agent and iteratively update the evader and detector without any human intervention. Human experts are only required to analyze the agents' transaction traces when they converge to an equilibrium. However, as discussed in Appendix C, automating the detector as another DRL agent introduces extra challenges that require non-trivial system design efforts. Given that our proposed design is already effective in identifying yet-unknown anonymity-compromising patterns, we keep our single-agent design and leave the two-agent model as future work.

# Block	From	Method	To
	.		
	.		
n-t	**Addr. A1(Origin)**	transfer	**Addr. A2**
n-t	**Addr. A1(Origin)**	transfer	**Addr. A3**
n-t	0x12D...CEd384	...	0xe0...bc04e1
	.		
	.		
n	0xa03...49e8B9	Function1	Contract_Y
n	0xcBD...Fe6E80	Function4	Contract_A
n	**Addr. A2**	**Deposit**	**TC**
n	**Addr. A3**	**Deposit**	**TC**
n	**Addr. A3**	**Deposit**	**TC**
n	**Addr. A4**	**Withdraw**	**TC**
n	**Addr. A5**	**Withdraw**	**TC**

Figure 9: A yet-unknown anonymity-compromising pattern discovered through the additional updates of the evader agent.

No guarantee of discovering yet-unknown patterns. It is extremely difficult to directly train the agent with the task of discovering yet-unknown anonymity-compromising patterns. As such, we take an alternative design with an idea similar to fuzzing. We are aware that there is no guarantee that the agent will always find yet-unknown patterns (explained in Section 4.1). This is similar in spirit to fuzzing, which also cannot guarantee to find all vulnerabilities or achieve very high coverage. We carefully design the agent's tasks and integrate a detector to guide the agent toward discovering yet-unknown patterns. In Section 6.1, we demonstrate these designs can help identify such patterns.

Potential risks. We design our method to facilitate the exploration and mitigating of anonymity compromises in mixing services. Our goal is to build better guidebooks for mixing services that protect users' anonymity. We also acknowledge that, like many security tools, our method could potentially be used by attackers to explore new attacks against mixing services. Nevertheless, we believe that this should not diminish the value of our method, nor should it hinder the development of DRL-based tools for security breach discovery in the blockchain. This scenario is similar to various other security techniques, such as software fuzzing [13, 24, 28, 34, 55] and exploit generation [23, 40, 48]. While hackers can wield these tools to find and exploit vulnerabilities, they have also significantly benefited the software industry by aiding in the discovery and remediation of software vulnerabilities (before

and after the software is released).

Limitations. Our approach has several limitations. First, human experts are necessary to analyze a few representative transactions (traces) to extract relevant patterns. An interesting future work could explore further reducing the required human involvement, potentially by leveraging some explanation methods for DRL agents [15, 53]. Second, while adding more rules to the guidebook decreases the risk of identity leakage, it also imposes more constraints on users when making regular transactions. This could potentially jeopardize the utility of the mixing service, as users may need to invest more time in learning and adhering to the guidebook. As part of our future research, we will quantify the impact on utility and explore methods to mitigate the impact (e.g., make the guidebook user-friendly, merge some rules in the guidebook, etc). Third, our future work will also explore improving the generalizability of our evader agent across challenge tables by training the agent with multiple challenge tables or via transfer learning techniques. Fourth, we use the Tornado Cash [7] to evaluate GuideEnricher. In our future work, we will extend GuideEnricher to more mixing services by incorporating them into our simulator and retraining the evader agents. We will also investigate customizing GuideEnricher to other blockchains beyond Ethereum.

8 Related Work

Anonymity analysis of mixing services. Existing research into anonymity compromises of mixing services of the Ethereum blockchain works in a post-mortem fashion [26, 39, 43, 46, 47], by extracting patterns from historical transactions. The key difference between existing works and GuideEnricher is that existing approaches cannot be used to find yet-unknown anonymity-compromising patterns proactively. In contrast, our method operates proactively and can be used to explore yet-unknown anonymity-compromising patterns proactively. Note that existing works [26, 39, 43, 46, 47] reduce human effort by leveraging certain automatic techniques (e.g., statistical analysis [46, 47], graph neural networks [26, 39]) for transaction grouping. In our method, we also utilize automatic statistical methods (i.e., clustering algorithms) when analyzing our agents' transaction traces.

DRL for security applications. Motivated by the great success of DRL across diverse domains such as robotics [20, 44], games [32, 41, 42, 45], and self-driving vehicles [25], security researchers also harness DRL for security applications. Specifically, there are four representative applications: malware mutation and detection [3, 8, 29, 50, 51, 54], network lateral movement attack and defense [2, 30], blockchain mining [19], and program analysis [17, 52]. Among these applications, the one most related to our research is SquirRL [19]. In this work, the authors design a DRL agent to explore novel selfish mining behaviors in blockchains. However, our research focuses on a very different security aspect of blockchains. Furthermore, our agent's design is more complex and, hence, requires a more advanced learning algorithm. Together with SquirRL [19], we demonstrate the utility of DRL in discovering security and privacy issues on the blockchain.

9 Conclusion

This paper introduces GuideEnricher, a DRL-driven method for proactively discovering anonymity-compromising patterns in mixing services, requiring limited human involvement. We extensively evaluate GuideEnricher from multiple aspects and demonstrate its utility. With these results, we conclude that DRL can facilitate automatic anonymity-compromising pattern discovery, greatly benefiting user anonymity.

Acknowledgement

This material is based upon work supported by the National Science Foundation under grant no. 2229876 and is supported in part by funds provided by the National Science Foundation, by the Department of Homeland Security, and by IBM. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or its federal agency and industry partners.

References

[1] Mohiuddin Ahmed, Raihan Seraj, and Syed Mohammed Shamsul Islam. The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics*, 2020.

[2] Iman Akbari, Ezzeldin Tahoun, Mohammad A Salahuddin, Noura Limam, and Raouf Boutaba. Atmos: Autonomous threat mitigation in sdn using reinforcement learning. In *NOMS*, 2020.

[3] Hyrum S Anderson, Anant Kharkar, Bobby Filar, David Evans, and Phil Roth. Learning to evade static pe machine learning malware models via reinforcement learning. *arXiv preprint arXiv:1801.08917*, 2018.

[4] Nir Bitansky and Alessandro Chiesa. Succinct arguments from multi-prover interactive proofs and their efficiency benefits. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, 2012.

[5] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[6] Tornado Cash. Nova. https://tornova.cash, 2023.

[7] Tornado Cash. Tornadocash. https://tornado.cash, 2023.

[8] Kai Chen, Peng Wang, Yeonjoon Lee, XiaoFeng Wang, Nan Zhang, Heqing Huang, Wei Zou, and Peng Liu. Finding unknown malice in 10 seconds: Mass vetting for new threats at the google-play scale. In *USENIX Security*, 2015.

[9] cyclone.xyz. Cyclone. https://cyclone.xyz/eth, 2023.

[10] DeepMind. Alphastar: Mastering the real-time strategy game starcraft ii. https://deepmind.com/blog/article/alphastar, 2017.

[11] ethereum.org. Eip-1559: Fee market change for eth 1.0 chain. https://eips.ethereum.org/EIPS/eip-1559, 2019.

[12] ethereum.org. zk-rollups. https://ethereum.org/en/developers/docs/scaling/zk-rollups, 2023.

[13] Andrea Fioraldi, Dominik Maier, Heiko Eißfeldt, and Marc Heuse. Afl++ combining incremental steps of fuzzing research. In *Proceedings of the 14th USENIX Conference on Offensive Technologies*, 2020.

[14] getmonero.org. Monero. https://www.getmonero.org, 2023.

[15] Wenbo Guo, Xian Wu, Usmann Khan, and Xinyu Xing. Edge: Explaining deep reinforcement learning policies. *NeurIPS*, 2021.

[16] Ameer Haj-Ali, Nesreen K Ahmed, Ted Willke, Joseph Gonzalez, Krste Asanovic, and Ion Stoica. A view on deep reinforcement learning in system optimization. *arXiv preprint arXiv:1908.01275*, 2019.

[17] Kihong Heo, Woosuk Lee, Pardis Pashakhanloo, and Mayur Naik. Effective program debloating via reinforcement learning. In *CCS*, 2018.

[18] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018.

[19] Charlie Hou, Mingxun Zhou, Yan Ji, Phil Daian, Florian Tramer, Giulia Fanti, and Ari Juels. Squirrl: Automating attack analysis on blockchain incentive mechanisms with deep reinforcement learning. In *NDSS*, 2019.

[20] Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 2021.

[21] George Kaloudis and Edward Oosterbaan . How Popular Are Crypto Mixers? Here's What the Data Tells Us. https://www.coindesk.com/layer2/2022/01/25/how-popular-are-crypto-mixers-heres-what-the-data-tells-us, 2022.

[22] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In *NeurIPS*, 1999.

[23] Johannes Krupp and Christian Rossow. teEther: Gnawing at ethereum to automatically exploit smart contracts. In *USENIX Security*, 2018.

[24] Jun Li, Bodong Zhao, and Chao Zhang. Fuzzing: a survey. In *ACM Computing Surveys*, 2018.

[25] Quanyi Li, Zhenghao Peng, Zhenghai Xue, Qihang Zhang, and Bolei Zhou. Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning. *arXiv preprint arXiv:2109.12674*, 2021.

[26] Sijia Li, Gaopeng Gou, Chang Liu, Chengshang Hou, Zhenzhen Li, and Gang Xiong. TTAGN: Temporal transaction aggregation graph network for ethereum phishing scams detection. In *WWW*, 2022.

[27] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *ICML*, 2018.

[28] Hongliang Liang, Xiaoxiao Pei, Xiaodong Jia, Wuwei Shen, and Jian Zhang. Fuzzing: State of the art. *IEEE Transactions on Reliability*, 2018.

[29] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon J. Ross, and Gianluca Stringhini. Mamadroid: Detecting android malware by building markov chains of behavioral models. In *NDSS*, 2017.

[30] Microsoft. Cyberbattlesim: Gamifying machine learning for stronger security and ai models. https://github.com/microsoft/CyberBattleSim, 2021.

[31] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

[32] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[33] Thanh Thi Nguyen and Vijay Janapa Reddi. Deep reinforcement learning for cyber security. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[34] Hui Peng, Yan Shoshitaishvili, and Mathias Payer. T-fuzz: fuzzing by program transformation. In *S&P*, 2018.

[35] railgun.org. Railgun. https://www.railgun.org, 2023.

[36] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 1987.

[37] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 2017.

[38] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[39] Jie Shen, Jiajun Zhou, Yunyi Xie, Shanqing Yu, and Qi Xuan. Identity inference on blockchain using graph neural network. *Communications in Computer and Information Science*, 2021.

[40] Yan Shoshitaishvili, Ruoyu Wang, Christopher Salls, Nick Stephens, Mario Polino, Andrew Dutcher, John Grosen, Siji Feng, Christophe Hauser, Christopher Kruegel, and Giovanni Vigna. Sok:(state of) the art of war: Offensive techniques in binary analysis. In *S&P*, 2016.

[41] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

[42] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[43] Xiaobing Sun, Wenjie Feng, Shenghua Liu, Yuyang Xie, Siddharth Bhatia, Bryan Hooi, Wenhan Wang, and Xueqi Cheng. MonLAD. In *WSDM*, 2022.

[44] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[45] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 2019.

[46] Zhipeng Wang, Stefanos Chaliasos, Kaihua Qin, Liyi Zhou, Lifeng Gao, Pascal Berrang, Ben Livshits, and Arthur Gervais. On how zero-knowledge proof blockchain mixers improve, and worsen user privacy. *arXiv*, 2023.

[47] Mike Wu, Will McTighe, Kaili Wang, Istvan A. Seres, Nick Bax, Manuel Puebla, Mariano Mendez, Federico Carrone, Tomás De Mattey, Herman O. Demaestri, Mariano Nicolini, and Pedro Fontana. Tutela: An open-source tool for assessing user-privacy on ethereum and tornado cash. *arXiv*, 2022.

[48] Wei Wu, Yueqi Chen, Jun Xu, Xinyu Xing, Xiaorui Gong, and Wei Zou. FUZE: Towards facilitating exploit generation for kernel Use-After-Free vulnerabilities. In *USENIX Security*, 2018.

[49] Liang Xiao, Yuzhen Ding, Donghua Jiang, Jinhao Huang, Dongming Wang, Jie Li, and H. Vincent Poor. A reinforcement learning and blockchain-based trust mechanism for edge networks. In *IEEE Transactions on Communications*, 2020.

[50] Jiayun Xu, Yingjiu Li, and Robert H. Deng. Differential training: A generic framework to reduce label noises for android malware detection. In *NDSS*, 2021.

[51] Ke Xu, Yingjiu Li, Robert H. Deng, Kai Chen, and Jiayun Xu. Droidevolver: Self-evolving android malware detection system. In *EuroS&P*, 2019.

[52] Yichen Yang, Jeevana Priya Inala, Osbert Bastani, Yewen Pu, Armando Solar-Lezama, and Martin Rinard. Program synthesis guided reinforcement learning for partially observed environments. In *NeurIPS*, 2021.

[53] Herman Yau, Chris Russell, and Simon Hadfield. What did you think would happen? explaining agent behaviour through intended outcomes. *NeurIPS*, 2020.

[54] Xiaohan Zhang, Yuan Zhang, Ming Zhong, Daizong Ding, Yinzhi Cao, Yukun Zhang, Mi Zhang, and Min Yang. Enhancing state-of-the-art classifiers with API semantics to detect evolved android malware. In *CCS*, 2020.

[55] Xiaogang Zhu, Sheng Wen, Seyit Camtepe, and Yang Xiang. Fuzzing: A survey for roadmap. In *ACM Computing Surveys*, 2022.

Name	Description	Value range
action	Agent's previous action (Deposit or Withdraw)	0/1
wait time	Minimum number of transactions that the agent has waited previously	0-200
deposit call address	The address used in a prior transaction if the action was a deposit.	0-250
withdraw call address	The address used in a prior transaction if the action was a deposit.	0-250
number of deposits	Total number of deposits	0-9
balance of deposit call address	Last balance of the "deposit call action"	0-3
balance of withdrawing call address	Last balance of the "withdraw call address"	0-9
Balance current TC contract	Latest balance of the tornado cash contract	0-Inf
The current balance of the evader	Latest total balance of the evader (sum of current challenge table)	0-9
Is deposit call address from the challenge table	A boolean variable that retains whether the previous deposit address is distinct from the address initially containing funds for task.	0/1
Is withdraw call address from challenge table	A boolean variable that retains whether the previous withdraw address is distinct from the address initially given to withdrawal task.	0/1

Table 4: Explanation of each dimension/feature in the evader agent's observation. The last column specifies this dimension is about the TC server or the evader.

A Additional Technical Details and Experiment Setups

A.1 Original Guidebook

Table 6 shows the original guidebook of the Tornado Cash we used in this paper. It lists a set of known anonymity-compromising actions discovered by existing works.

A.2 Hyper-parameters

Here we specify the hyper-parameters for our DRL agent. We use a multi-layer perceptron with the architecture of 32 by 32 dense layer layers and the numbers indicate the number of neurons in each layer. We set the maximum time step in each episode as 10,000 as the horizon and train the agent for 500 to 1,000 iterations. We use the `ADAM` optimizer with a learning rate of 0.001.

A.3 Impact of Negative Reward

We use the same configurations of Exp-1 in Section 5, Figure 10 shows the different penalty configurations. Penalizing the same as the reward makes the game more stable in the training process, resulting in better and faster convergence for the maximum reward, and low fluctuation in the training process (resulting in a better model).

B Additional Experiments

B.1 Agent Generalizability across Challenge Tables

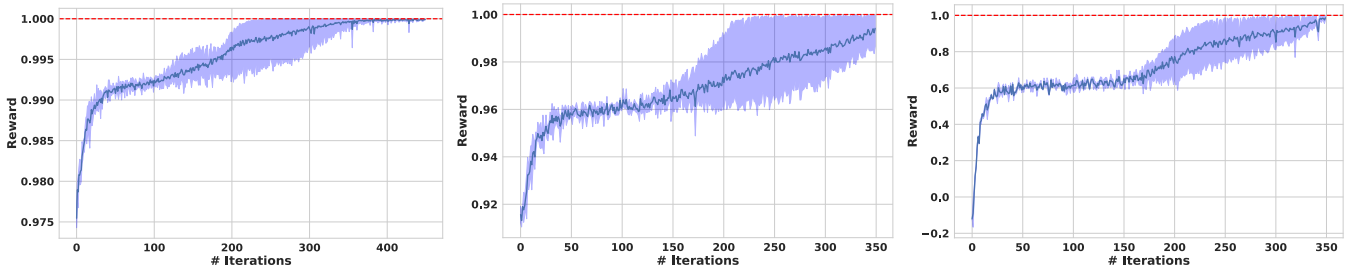
Note that this pattern corresponds to a rule in the original guidebook rather than the three new ones (added based on anonymity-compromising patterns found in Section 6.1). In this experiment, we use the agents trained from Exp-1 5 and test their effectiveness when we change the challenge tables. In the first scenario (V1), we asked the agent to transfer 3 tokens using only 1 wallet. In the second scenario (V2), the agent was tasked with transferring 6 tokens using 2 wallets, with 3 tokens in each wallet. In the third scenario (V3), the agent had to transfer 17 tokens that were distributed across three addresses, with 10 tokens in one address, 5 tokens in another, and 2 tokens in the third address. For each setup, we run the agent for 1,000 episodes and record the testing performance. The result in 5 demonstrates that an agent trained with one challenge table cannot exhibit enough effectiveness when applied to other challenge tables.

Challenge table	Task finishing rate (%)	Invalid rate (%)	Evading rate (%)
Exp-1	100(0.000)	00.129(1.119)	99.871(1.119)
V-1	100(0.000)	47.717(18.272)	52.283(18.272)
V-2	100(0.000)	25.073(19.572)	74.927(19.572)
V-3	100(0.000)	87.905(10.216)	12.095(10.216)

Table 5: The percentage of finished tasks and invalid actions of agents trained with different challenge tables across 1000 trials.

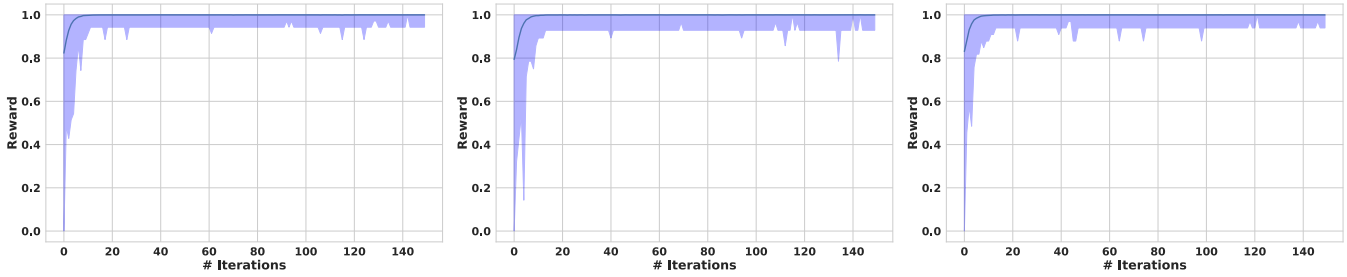
B.2 Hyper-parameter Sensitivity

In this experiment, we use an agent with one single wallet with 3 tokens and tasked to transfer tokens to any given 100 empty



(a) Detection penalty reward equal -1 (b) Detection penalty reward equal -5 (c) Detection penalty reward equal -10

Figure 10: Different Detection penalties.



(a) Policy network with 2, 32 hidden perceptron layers. (b) Policy network with 3, 32 hidden perceptron layers. (c) Policy network with 2, 64 hidden perceptron layers.

Figure 11: The agent's normalized reward when varying its policy network and maximum time steps in each episode.

wallets. We change the parameters of the policy network and measure the sensitivity of key hyper-parameters of the policy network architecture. We varied the policy network architecture length from 2 to 3 and changed the number of neurons in each layer from 32 to 64. Our goal was to make the task relatively small and evaluate the effectiveness of the policy network. In conclusion, both networks have similar performance (see Figure 11). Hence we can use two hidden layers with 32 perceptrons in each layer as our base policy network.

C Additional Experiments

We conducted a few additional experiments. First, we conducted an inclusion study on different mixing services on the ETH chain and other chains. Second, we conducted experiments on the simulation realism to real-world scenarios such as address usage analysis. Third, we explained the modular implementation of simulation, automatic detector updates, and crowd users' hard-coded rules. Finally, we encapsulated the agents' general evasion strategies against the initial guidebook rules. Our code, model, scaled plot diagrams, and experiment explanations are publicly available [4].

# Rule	Tutela: An Open-Source Tool for Assessing User-Privacy on Ethereum and Tornado Cash [47]	On How Zero-Knowledge Proof Blockchain Mixers Improve, and Worsen User Privacy [46]
1	Address Match	• Deposit Address Reuse • Improper Withdrawal Sender
2	Unique Gas Prices	
3	Linked ETH Addresses	Related Deposit-Withdrawal Address Pair
4	TORN Mining	

Table 6: Initial guidebook.

[4] https://github.com/ucsb-seclab/GUIDE-ENRICHER