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

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

1 Different Mixing Services on ETH Chain

Mixing Service	Information
Cyclone (www.cyclone.xyz)	Cyclone is a cross-chain, non-custodial protocol using zkSNARKs for transactional privacy in DeFi. It's governed decentralized, similar to TornadoCash.
Railgun (www.railgun.org)	Railgun serves as both a mixer and a cross-chain bridge, utilizing zkSNARKs cryptographic implementation for anonymous money transfers across different chains.
TC Nova (tornova.cash)	TC Nova is an upgraded version of Tornado Cash, allowing users to deposit and withdraw arbitrary amounts. It utilized the same zkSNARKs cryptographic mechanism for enhanced privacy and security.
Buccaneer V3 (BV3)	Only social media presence, no available public documents or accessible services.
White Ethereum (whiteethereum.com)	Only social media presence, no available public documents or accessible services.
0xTIP (0xmonero.com)	Only social media presence, no available public documents or accessible services. The only accessible thing is a coming soon landing page.
Messier 87 Black Hole (messier.app)	Only social media presence, no available public documents or accessible services.
eth mixer (ethereum-mixers.com)	Flagged as a scam in the community and no active transactions.
eth-mixing-eth.com, eth-mixers.com, eth-mixer-obfuscator.com	Flagged as scams in the community and no active transactions.

Table 1: Different mixing services on Ethereum blockchain

2 Different Mixing Services on Different Chains

We conducted a thorough investigation into various mixing services across different blockchains and found that the majority of them are scams and not legitimate mixing services. Additionally, we discovered that these fraudulent services have been widely flagged by the community. [1] [2] [4]

List of mixing services: Canonymix.cc, anonymix.io, anonymix.org, anonymixx.com, anonymousmixer.eth.link, anonymousmixers.com, bitcoin-laundry.online, bitcoin-laundry2.net, bitcoinlaundry.net, blender.cx, blender.io, blender.pw, blender.so, blenderbtc.com, blenderbtc.io, blenderbtc.pro, blenderio.com, blenderiocpxfema.onl, blenderio.to, blenderbitcoin.com, blenderbit.com, blenderbit.org, blendercoin-mixer.com, blendar.io, blennder.net, bitcoin-mixer1.com, bitcoin-mix.org, bitblender.in, bitmix.online, best-ethereum-mixer.com,

blog.tezro.com/best-tumbler-mixer-services, btcmixer.cc, chipbitmixer.com, coinmixer.shop, coinblender.org, coinmixer.online, crypto-mixer.cc, cryptomixer.net, cryptomixer0.com, cryptomixer2.com, cryptobank.co, cryptomixer-io.net, cryptomixers-reviews.com, criptomixer.io, kryptomixer.io, laundry-bitcoin.com, mixersinbad.io, silk-road.io, sinbady.com, sinbadmixer.com, sinbadmixer.net, sinbadiovpcdyohr3hg7i4hudbkxwnbdkewmsgsoiyjfrqhezdec7qad.onion, smartbitmix.com, smartcoinmix.com, smartmix-blender.com, smartmixer.me, slnbad.io, the-crypto-mixer.com, thebestbitcoinmixers.com, ethereum-mixer-eth-mixer.com, ethereum-mixers.com, eth-mixing-eth.com, eth-mixers.com, eth-mixer-obfuscator.com, litecoin-mixer-ltc-mixer.com, overtor.com, veio.io.

3 Address Usage Analysis

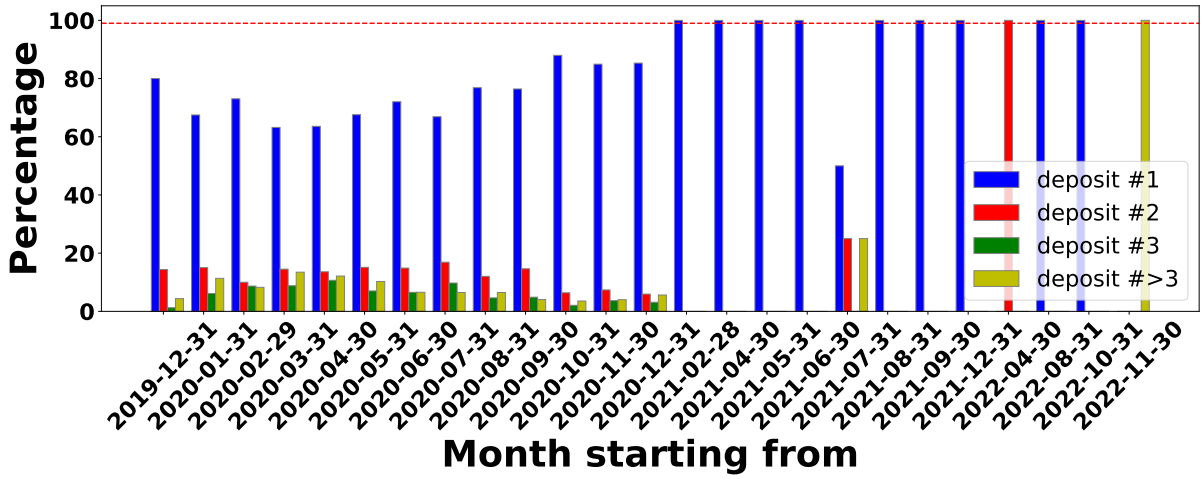


Figure 1: Address usage in TC 0.1 contract: Deposit.

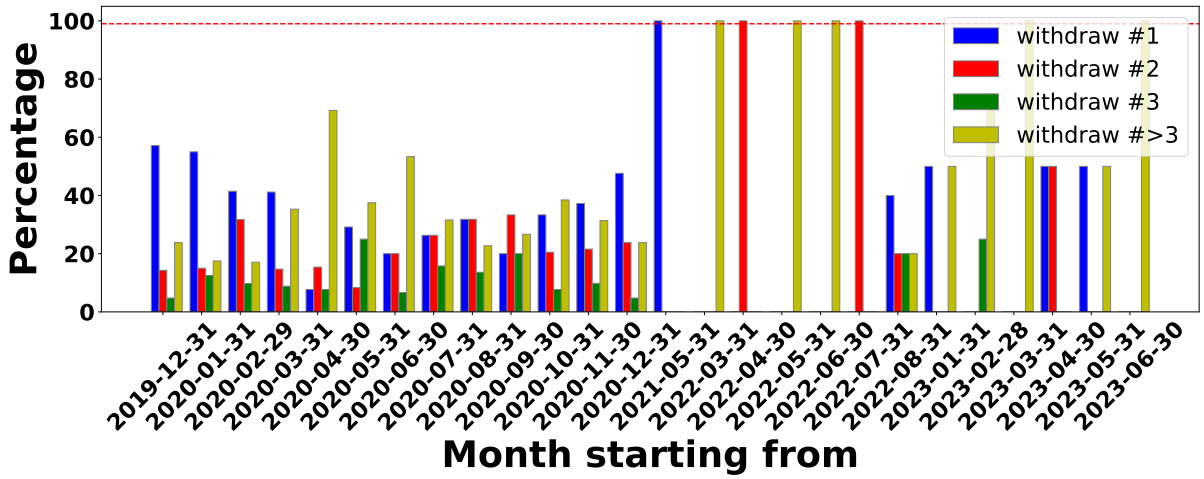


Figure 2: Address usage in TC 0.1 contract: Withdrawal.

Figure 1 and 2 show the percentage of addresses that were used once (#1), twice (#2), three times (#3), and more than three times (#> 3) for only deposits and only

withdrawals, respectively, in the real historical transactions. Specifically, Figure 1 displays the historical activity for the deposits into the TC 0.1 contract, aggregated over month-long windows. The Y-axis represents the percentage of addresses that are reused. Similarly, Figure 2 illustrates withdrawal actions for TC 0.1. The red dashed line in the graph captures the data from the simulation, indicating that around 99% of addresses (wallets) are only used once. The graphs show that multiple deposits (or withdrawals) from the same address are a bit more common in the real world. This is because our simulation discourages such behavior, as it makes it easier to link transactions in such cases.

4 Automatic Detector Update

Recall that we hard-code the detector with rules rather than modeling it as another DRL agent. The reasons are as follows. First, the detector in our game is designed to make decisions based on the current guidebook. Recall that the guidebook specifies a list of anonymity-compromising patterns. The detection process, thus, adheres to a straightforward procedure that directly checks whether a transaction or a sequence of transactions matches a rule in that list. Given the probabilistic nature, there is no guarantee that the detector can consistently align with the current guidebook if we model it as another DRL agent. Second, in the current setup, we can treat the fixed detector as a part of the game environment. This simplifies the original two-player game into a one-player DRL task, where the PPO algorithm can efficiently resolve a (local) optimal policy for the agent. Instead, the state-of-art training algorithm for two-player games, i.e., self-play, is ineffective in searching for the optimal solutions [3]. As such, hard-coding the detector enables us to train effective evaders to uncover anonymity-compromising patterns.

We tried the two-agent solution, and our results verify the two challenges discussed above. More specifically, our detector cannot always follow the current guidebook. To make matters worse, the training process for both agents is difficult to converge. Given that our proposed design is already effective in identifying novel anonymity-compromising patterns, leading to a substantial enhancement in anonymity protection in the Ethereum blockchain. As such, we stick to our single-agent design and leave the two-agent model as part of our future work.

5 Modular Implementation of Simulation

The implementation primarily comprises six modules: `Contract`, `Environment`, `Heuristics`, `Models`, `Mainnet` and `Wallet`. In summary, the `Contract` module encapsulates basic functionality for creating a new contract for the simulation, and the `Environment` module encapsulates basic functionality for creating a simulation such as provisioning crowd users and letting the DRL agent interact with the contract and the crowd. The `Mainnet` module encapsulates basic functionality for creating a new blockchain (inspired by the functionality of the Ethereum blockchain). The `Wallet` module comprises the functionality of a cryptographic wallet. Likewise, `Heuristics` contains the basic functionality for creating the initial guidebook. The `Models` provides the basic functionality for creating and configuring the DRL agent's policy network. Table 2 outlines the base class and

some important functions implemented within the base class, which users can extend and overwrite for their specific implementations.

Module	**Basic Functionalities**
Contract	`deposit, withdraw, transfer`
Environment	`step, reset, crowd_steps, init_agent, init_crowd`
Heuristics	`heuristic`
Models	`forward, value_function`
Mainnet	`commit, get_transaction_hash, get_current_block_id, get_current_block, get_gas_price,get_balance_of_addr`
Wallet	`get_wallet_id, get_balance, send_transaction`

Table 2: Modular implementation of simulation

6 Hard-coded Crowd Rules

All hard-coded crowd rules are implemented in the `Environment` module of Appendix 5. This section explains the hard-coded rules that determine how the crowd functions during each step of the simulation.

In each step, the DRL agent requests the crowd to generate a certain maximum number of transactions. To this end, Algorithm 1 is employed to generate valid traffic. First, the algorithm selects a random user from the crowd user pool, and then randomly chooses a wallet belonging to that user. This process ensures that each wallets (correctly) belong to the particular selected user. Subsequently, the crowd user randomly chooses between a deposit and withdrawal action. If it is a deposit, the algorithm checks whether the selected address has a sufficient balance. If true, the algorithm proceeds to commit the transaction; otherwise, it rolls back the transaction. Similarly, if it is a withdrawal, the algorithm retrieves a note (a previous deposit in the TC contract belonging to that particular user), verifies the note, and if verified, commits the transaction; otherwise, it rolls back the transaction. These hard-coded functions are not learnable during the simulation, but they ensure that the crowd generates valid (and random) transactions.

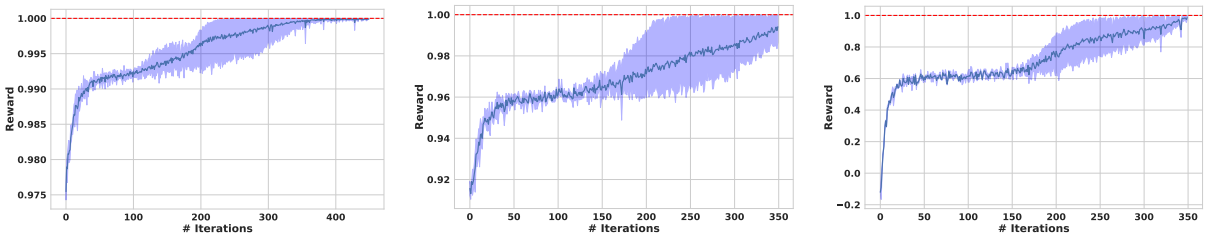
Algorithm 1 Generate Crowd Step

```
Function GenerateCrowdTransactions(numTx):
    for each in MAX(numTx) do
        user ← pickRandomUser()
        wallet ← randomWallet(user)
        action ← randomAction(user)
        if action == "DEPOSIT" then
            balance ← getBalance(wallet)
            if balance ≥ deposit_amount then
                commitTx()
            else
                rollbackTx()
            end
        else if action == "WITHDRAW" then
            note ← getDepositInfo(user)
            if verify(note) then
                commitTx()
            else
                rollbackTx()
            end
    end
```

7 Evasion Strategies

# **Rule**	**Evasion Strategy**
1	• Use a unique address in the 'TO' section of the note creation for each 'deposit()' function invoke (withdraw money to unique addresses as much as possible - no use in any other note or no previous use of invoking deposit()/withdraw() function). • Unique address to invoke the 'withdraw()' function in TC (no use in any other note or no previous use of invoking deposit()/withdraw() function).
2	• Execute withdrawal actions in blocks with multiple deposit actions from either the agent itself or the crowd. • Insert multiple deposits to one block and perform one or more withdrawals (need more examples). • Perform a withdrawal on a block without any deposits or with more withdrawals.
3	• Avoid selecting such addresses for invoking deposit() or withdraw() functions and creating notes (withdrawing money).
4	• Choose not to participate in TORN mining and wait until such mining scenarios are concluded in real-world scenarios. • Execute deposit actions in blocks with multiple deposit actions from either the agent itself or the crowd.

Table 3: Evasion Strategies



(a) Detection penalty reward equal -1

(b) Detection penalty reward equal -5

(c) Detection penalty reward equal -10

Figure 3: Differnt Detection penalty

8 Impact of Negative Reward

We use the same configurations of Exp-1 in Section 5 (Implementation and Evaluation in the paper), Figure 3 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).

9 Hyper-parameter Sensitivity

(a) Policy network with 2, 32 hidden perceptron layers.

(b) pPolicy network with 3, 32 hidden perceptron layers..

(c) Policy network with 2, 64 hidden perceptron layers.

Figure 4: The agent's normalized reward when varying its policy network and maximum time steps in each episode.

In this experiment, we use an agent with one single wallet with 3 tokens and tasked to transfer tokens to any given 100 empty 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 4). Hence we can use two hidden layers with 32 perceptrons in each layer as our base policy network.

References

[1] bitcointalk.org. bitcointalk.org. https://bitcointalk.org/, 2023.

[2] chainabuse.com. chainabuse.com. https://chainabuse.com/, 2023.

[3] Wenbo Guo, Xian Wu, Lun Wang, Xinyu Xing, and Dawn Song. Patrol: Provable defense against adversarial policy in two-player games. In *USENIX Security*, 2023.

[4] scam alert.io. scam-alert.io. https://scam-alert.io/, 2023.