Detecting Ethereum Mixers

Amanda Akopova
College of Science and Engineering
American University of Armenia
Yerevan, Armenia
amanda_akopova@edu.aua.am

***Abstract*—This paper aims to investigate the role of non-custodial, trustless coin mixers, with the main focus on the notorious Tornado Cash mixer on the Ethereum blockchain. Through manual code review and experimental analysis, it was possible to identify patterns and characteristics unique to mixers. The proposed heuristic, based on deposit and withdrawal functions, aims to aid regulatory bodies in detecting potential illicit activities. Leveraging the available data analysis tools, it was possible to refine the detection model and integrate heuristics for improved accuracy. Overall, the research contributes to understanding privacy mechanisms on Ethereum and offers insights for regulatory compliance and financial crime prevention as well as brings to the playground a methodology for further refinement and research.**

I. Introduction

Ethereum is the largest blockchain in the world both by its market share and usage. Its cryptocurrency is the second most valuable after Bitcoin and is used widely in various fields. Ethereum transactions are all publicly stored on full nodes, and any node can have access to all the information from the chain. However, malicious actors can compromise this information and infer correlations between addresses or even de-anonymize users' privacy having some background knowledge. This concern brought to privacy-enhancing overlays being deployed on the Ethereum blockchain. One of these privacy preservation mechanisms are non-custodial, trustless coin mixers.

Unfortunately these "mixers" otherwise also known as "tumblers" also play a huge role in criminal activities, such as money laundering". In August 2022 one of the biggest tumblers Tornado Cash was sanctioned by the U.S. Department of Treasury for "laundering" over $7 billion in assets, $445 million of which were connected to the well-known cyber-criminal Lazarus Group. Tornado Cash was used to launder $96 million funds derived from the Harmony Bridge Heist (June 24, 2022) and at least $7.8 million from the Nomad Heist (August 2, 2022) performed by malicious cyber actors. Sanctions were placed on 45 addresses that were associated with Tornado Cash, as well as the asset pools where users deposit and withdraw from during the transactions [1]. The founders of the mixer were arrested and charged with money laundering [2], and the original code was taken down from the public sources, although it is being re-uploaded periodically. Despite all of these limitations, the decentralized structure of the blockchain allows Tornado Cash smart contracts to be deployed, and actively be in use till today.

Tornado cash is one of the biggest coin mixing tools on the Ethereum blockchain. To perform a coin mixing process, users are simply required to invoke the relevant smart contract. This is a very convenient way to enhance user privacy and protect sensitive data stemming from address linkability. On the other hand this provided anonymity can shield various illegal activities, which is a major issue in the digital currency world, complicating the efforts of tracing illicit funds.

With the described nature of the mixers and the blockchain, it is technically challenging to detect mixers and albeit crucial to be able to maintain the integrity of the blockchain ecosystem. Effective detection mechanisms that distinguish potentially unlawful actions can be very vital for regulatory bodies to enforce anti-money laundering (AML) restrictions as well as for the organizations that seek to maintain compliance with global financial regulations. Moreover, with the growth of blockchain technology and the transition to the mainstream financial platform, the use of mixers can cause a debate over privacy versus transparency. Thus it is important to develop methods for researchers as well as the regulators to gain insights into the transaction flows and enhance the ability to monitor and prevent financial crimes. This is also necessary for the public and institutions to gain assurance over blockchain being private and secure, yet transparent and regulated.

In summary, the contributions of this paper include:

- Analysis of Tornado Cash mixer, including summarizing the behavior patterns of transactions.
- Review of the Tornado Cash's smart contract's code, as well as comparison with other mixers' code.
- Identification of similarities and differences between the mixers.
- Performing an experimental analysis on the Kovan testnet and on real-life Ethereum Mainnet.

It is worth mentioning that there is a very limited, if any, number of tools for data analysis for the Ethereum blockchain. You can access all the historical data of the blockchain only if you are running an archive node, which will still require building a database manually and requires high computation power. The other way that is cost-effective compared to maintaining your own database is Ethereum ETL's public database, available on Google's Big Query, which is still costly since it charges for the amount of data processed by your queries. The other valid tool that gives information on smart contracts,

addresses, transactions, and their interactions is Etherscan, which is very limited in its functionality to perform extensive or automated data analysis and is used either for manual reviews or through its API. Lastly, there is the Glider tool, which is a novelty and is still being upgraded. With some upcoming features, it will be possible to verify all the findings of this research in a well-automated, comprehensive way.

II. Related Work

Tang et al. [3] suggested three heuristic clustering rules to identify address correlations in Tornado Cash's mixing transactions. They formalized two transaction patterns and conducted an experimental analysis to reveal address linkability in the coin mixer. This methodology aimed to find the addresses that potentially belong to the mixer and partake in the mixing process. In [4], researchers proposed a taxonomy to classify Ethereum-based mixers. This taxonomy consists of five layers: General features, Terms of use, Mixer functionalities, Deposit functionalities and Withdrawal functionalities. This classification gives understanding and offers recommendations on several aspects:

- What code decisions define a mixer and what are the characteristics of the environment?
- What prerequisites must users meet to use the mixer?
- What are the main functionalities available in mixers?
- What are the restitutions and censorships managed during the deposit of assets?
- How execution fee coverage is handled and what censorships are applied to fund withdrawals?

During the deposit process if the mixer has a fixed denomination and the amount doesn't match the denominator two restitutions can be triggered: Revision or Refund. In the first case the deposit transaction won't be executed and the user will waste transaction fees. With the second case the depositor will get back the difference between the denomination and the deposited amount. If the smart contract doesn't have a denomination there are no restitutions applied. One possible solution to incorporate some regulatory compliance to the deposit transaction is Blocklisting and Allowlisting addresses that can use the mixer through deposit transactions [5]. This way mixer's smart contract will first check with authorities whether the address is associated with sanctioned or embargo lists before accepting the user's funds into its pools.

Mixer detection is yet a field that hasn't been extensively explored. Due to their decentralized and pseudonymous nature, mixers pose a myriad of unique challenges. This makes them particularly difficult to detect and analyze. Existing research has mainly focused on heuristic clustering rules or taxonomies, which helps to understand the characteristics and functionalities of mixers, but not on direct detection methods. In addition, there is a limited availability of tools and relying on manual verification processes add up to the challenge of effectively detecting mixers. Thus, there is a need for the development of specialized tools and methodologies dedicated to mixer detection, which will help to address the growing concerns surrounding their use in facilitating illicit activities.

III. Methodology

A. Selection of Mixers for Analysis

For this research, the main mixer that will be used is Tornado Cash as well as Miximus, Möbius, MixEth, MicroMix and Furrious Mixer. These mixers employ similar foundational mechanisms and architecture with some logical or structural differences like the usage of zk-SNARKs, other zero-knowledge proofs such as Chaum-Pedersen protocol, ring signatures etc. For instance MixEth [6] was developed as an enhancement to Möbius [7] and Miximus. The selection criteria for these mixers is based on several factors. The main one is the code availability of the mixer. With the regulatory processes stemming from the Tornado Cash incident there is very little public data available on mixers. Code accessibility is crucial for a thorough analysis to allow a detailed review on the main functionalities and algorithms present in mixers' logic.The second criteria is determining whether any of these mixers has been deployed on the Ethereum blockchain. This will help to identify the practical relevance and adoption of the mixer. A mixer's deployment on Ethereum indicates active usage and provides a real-world environment to evaluate its performance and security. Despite its legal issues, Tornado Cash still remains one of the most widely discussed and used mixers, with its original code being repeatedly reused in other smart contracts. This is why Tornado and its successors are used as the primary mixers for this analysis, with the goal to further expand to other mixer architectures and other chains.

B. Code Review Process

To further proceed with the analysis a manual detailed code review has been conducted. All the mentioned mixers' codes were written in Solidity, the highly utilized programming language for Ethereum smart contracts. Smart contracts are the computer programs that are stored on the blockchain. Once created, they can't be modified and all of them are guaranteed to execute according to the rules that are defined in their code [8]. Key functions of interest were deposit and withdrawal mechanisms which were closely examined. Special attention was given to the implementation of cryptographic techniques like zk-SNARKs, Chaum-Pedersen protocol, which are crucial for ensuring privacy while maintaining transparency. This analysis helped to map out the landscape of current mixer technologies.

C. Data Collection

1) Sources of Ethereum Transaction Data and Functions Data: Anyone can access the full Ethereum transactions' history via the archive node. Ethereum can be described as a transaction - based state machine. After each block the state of the blockchain is updated. Archive node is responsible for storing all the historical states starting from the very first block of the Ethereum blockchain [9]. Ethereum ETL provides a public relational database which is available on BigQuery [10]. For this research the BigQuery dataset has been utilized extensively, and specific data related to the mixers in question was extracted. The whole Ethereum transactions table consists

of 2,362,232,294 transactions and is 2 Terabytes in size. Additionally, the Ethereum Signature Database [11] was utilized to get insights into the functions and their signatures, which were used in identifying and differentiating the mixers based on their operational logic.

2) Data privacy and security measures: Considering the sensitivity of the data, strict data privacy and security measures were followed. All data handling processes were done according to general data protection regulations (GDPR) to ensure that the privacy of individual users wasn't compromised. Moreover, transparency in data processing was maintained to allow for the reproducibility of the research.

IV. Analysis of Mixer Codes

A. Case Studies

During this research while looking through the smart contracts of the proposed mixers it was noted that only three out of six were ever used or even deployed on the Ethereum Mainnet. These are Tornado Cash and supposedly Furious Mixer and Miximus. Möbius, MixEth and MicroMix were all ruled out.

B. Overview of Common Functions in Mixers

Throughout the code analysis there were three functions that caught the main focus for mixer identification, those are `deposit()`, `withdraw()` and `denomination()`. During the deposit phase, the user initiates the mixing process by sending cryptocurrency to the smart contract's address. For example, in Fig. 1 we can see that the user sends Ethereum funds from *address a*. The transaction includes cryptographic elements to conceal the origin of the funds.The created deposit transaction is accompanied with a cryptographic note, which is a unique identifier that secures the details of the transaction without revealing the user's identity. In case there is a denomination the amount sent by the user must match to the denomination present in the smart contract. In case it doesn't, *Revision* or *Refund* will take place. If the smart contract doesn't have the denomination, we immediately proceed to the withdrawal. Before the withdrawal process begins, there might be a predefined delay, which helps to obfuscate the trail of funds. After that, the user uses the previously generated cryptographic note, and requests the smart contract to retrieve funds to defined *address b*. This can be any address that the user controls and is not linked to his/her identity. In case the withdrawal address doesn't have sufficient amount to cover the gas fee of this transaction, a *Relayer* can be called. The *Relayer* forwards all the necessary parameters, for a small fee *(f)*.The remaining balance *(N - f)* is then transferred to the user's address [3].

C. Tornado Cash Mechanisms

To better understand the role of the mentioned functions, it is necessary to analyze the exact mechanism behind the mixer's code. The reuploaded code of Tornado Cash mixer is available on GitHub and was referred to for this analysis [12]. Here are some highlights of how Tornado Cash functions which also represents the core mechanism of other mixers:

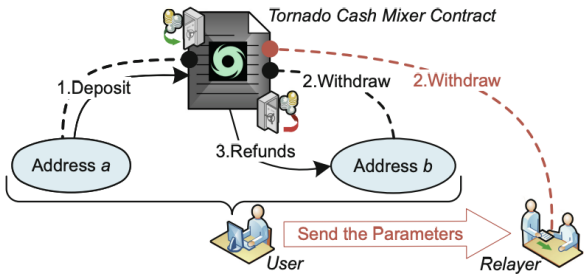

Fig. 1: The process of Tornado Cash coin mixing contract [3].

1) Cryptographic foundations: As a cryptographic foundation, Tornado Cash uses a pairing-based cryptography approach. It utilizes elliptic curve pairings to facilitate complex SNARK (Succinct Non-Interactive Argument of Knowledge) proofs, ensuring transactional privacy without revealing user identities. In addition there are two hashes being used:

- Pedersen Hash (H1): Used for creating commitments. A commitment to a secret (such as a transaction amount or user identity) allows a user to keep the secret hidden while proving that they know it.
- MiMC Hash (H2): Employed within the Merkle tree for efficient and secure aggregation of transaction data.

2) Merkle Tree: A binary Merkle tree of height 20 stores transactions in a privacy-preserving manner. Each leaf in the tree represents a transaction or a commitment, and the tree's structure allows for compact and secure proofs of inclusion.

3) Zero-Knowledge Proofs: These are zero-knowledge proofs that allow a prover to establish possession of certain information (e.g., a correct transaction or a valid state transition) without revealing the information itself.

4) Smart Contract Operations:

- Deposits: Users deposit into the smart contract by sending ETH along with a hash of their secret, which is added to the Merkle tree.
- Withdrawals: To withdraw, a user proves their right to a previous deposit without revealing which one, using a zero-knowledge proof that references the Merkle tree.
- Handling Fees: A fee mechanism compensates relayers who submit transactions on behalf of users, maintaining anonymity.

5) Security Measures:

- Unique Nullifiers: Ensure that each deposit can only be withdrawn once, preventing double-spending.
- History Tracking: The contract keeps a history of all root hashes of the Merkle tree to verify proofs against.

6) External Interactions: Relayers: Facilitate the process of broadcasting transactions to the network, helping users maintain privacy by masking the origin of Ethereum transactions [16].

D. Function Selector Calculator

When a user creates a deposit transaction he/she proceeds with an input field. This Input Data field allows to include additional information related to the transaction. For this research the main focus was on one type of data that can be passed to the input field: function call data. Function call data shows the function signature (Method ID) that the transaction calls. This signature is represented in the first four bytes of the input field followed after 0x [13]. In Fig. 2 is an example of a function call data for a deposit transaction to the Tornado Cash's 10 ETH denomination address. The *Method ID* 0xb214faa5 is the function signature corresponding to the deposit(bytes32 _id) function. And as it was already established, _id is the _commitment in Tornado Cash's code.

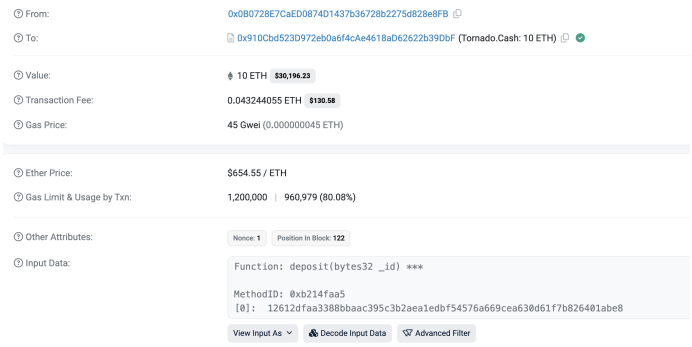

Fig. 2: Deposit Transaction to Tornado.Cash: 10 ETH Address (Etherscan) [14].

With the help of function signatures it was possible to establish whether the mixers were deployed on Ethereum Mainnet or not. A Function Selector calculator tool was used [15] to determine the function signatures of the known mixers. In Table 1. are illustrated the functions of the mixers and their respective signatures. Afterwards these signatures are being tested through the BigQuery Ethereum public dataset as the first four bytes of the input field. If no data is found it explains that no such smart contract has ever been deployed on Mainnet or no transaction with that function has been sent(Fig. 3). This approach helped to rule out the patterns in known mixers that don't need to be considered in the heuristic proposal.

```
SELECT *
FROM `bigquery-public-data.crypto_ethereum.
transactions`
WHERE input like '0xc515627e%'
LIMIT 10
```

Fig. 3: SQL Query on Ethereum Transactions

V. Heuristics

A. First Heuristic Proposal

The proposed heuristic is based on the patterns discovered during the code review of the mixers' smart contracts. The key focus was on deposit() and withdraw() functions that are persistent throughout the architecture of the mixers. The important distinctive features specific to mixers are also the availability of MerkleTrees, verifier() function, as well as the use of nullifier, denomination and relayer. However the primary heuristic chosen for detecting the mixers is the pattern recognition of the deposit() function, which is the most consistent mechanism out of all the proposed patterns. The verifier() function can be easily omitted from the pattern recognition heuristic, since it is present in all smart contracts that utilize prover algorithms such as zero-knowledge proofs, Chaum-Pedersen proof and other cryptographic proofs. The denomination is excluded since we discussed that not all of the mixers might necessarily have denomination, however we still can consider it as a distinctive pattern. Merkle Trees are a fundamental component in many blockchain applications, since they provide an efficient and secure way to summarize and verify the integrity of large datasets. Relayers are also a broad utility since they facilitate transactions in a decentralized environment by helping users to interact with various smart contracts without directly paying gas fees. Nullifiers, playing a huge role in mixers, are used to ensure that a particular action is not repeated, thus preventing double-spending. They are used in different protocols where the uniqueness of an operation is required. Finally, the deposit function is also present in various smart contracts, however the code analysis results showcased one distinctive pattern. In Table 1 it is indicated that three mixers are using specifically deposit(bytes32) function, the two other functions were never used in any transactions and deposit(uint256) was present in the input field, however it is a more common function among smart contracts and as an initial heuristic will give more false positives. The withdraw() function is also distinctive throughout the mixer codes since it contains checking the proofs, verification and nullifier(Fig. 4). However, it was not chosen as the primary heuristic pattern due to the logic of the mixers' cycle, as deposit() logically precedes withdrawal and the latter can't be performed without the first one. Despite the main heuristic focusing on the deposit(bytes32) function, all mentioned patterns must be considered for validating the proposed heuristic.

B. Refined Heuristic

After testing the first heuristic, in total 737 addresses were found with approximately 72% false positive results. For cleaning up the data sentimental filtering was performed narrowing it down to 211 addresses, which still yielded nearly 10% false positives. This required a lot of manual testing and code reviews, especially with the lack of tools. The results of this testing revealed that the pattern recognition of deposit(bites32) function must be paired with the withdraw() function. As showcased in Fig. 4, withdraw() function contains various parameters such as the relayer, recipient, nullifier, etc. Most importantly, it calls the verifyProof() function, which plays a crucial role for users to be able to withdraw their mixed funds. Thus the improved heuristics suggests that paired with the deposit(bytes32) function, it is crucial to check that the

TABLE I: Overview of Ethereum Mixers and Their Deposit Functions

Mixer	Deposit Function	Function Signature	Availability
Tornado Cash	deposit(bytes32 _commitment)	0xb214faa5	Yes
Miximus	deposit (bytes32 leaf)	0xb214faa5	Yes
Möbius	depositEther(address token, uint256 denomination, uint256 pub_x, uint256 pub_y)	0xc515627e	No
MixEth	depositEther(uint256 initPubKeyX, uint256 initPubKeyY)	0xb1265107	No
MicroMix	deposit(uint256 _identityCommitment)	0xb6b55f25	Maybe
Furrious Mixer	deposit(bytes32 note)	0xb214faa5	Yes

```
function withdraw(
    bytes calldata _proof,
    bytes32 _root,
    bytes32 _nullifierHash,
    address payable _recipient,
    address payable _relayer,
    uint256 _fee,
    uint256 _refund
  ) external payable nonReentrant {
    require(_fee <= denomination,
    "Fee exceeds transfer value");
    require(!nullifierHashes[_nullifierHash],
    "The note has been already spent");
    require(isKnownRoot(_root),
    "Cannot find your merkle root");
    require(
      verifier.verifyProof(
        _proof,
        [uint256(_root), uint256(_nullifierHash),
        uint256(_recipient),
        uint256(_relayer), _fee, _refund]
      ),
      "Invalid withdraw proof"
    );

    nullifierHashes[_nullifierHash] = true;
    _processWithdraw(_recipient, _relayer,
    _fee, _refund);
    emit Withdrawal(_recipient,
    _nullifierHash, _relayer, _fee);
  }
```

Fig. 4: Code Snippet of Tornado Cash Mixer's Withdraw Function

smart contract contains a withdraw() function which includes any type of "verification".

VI. Implementation

A. Glider

Glider [16] is a code query engine which can perform variant and data analysis on smart contracts. It is the first and only tool on blockchain that gives the ability to query through the source code of all the verified smart contracts deployed on integrated EVM blockchains. The language used for querying is Python. Users can query freely through testnet (Kovan) contracts, but for mainnet queries they are required to get access. In the scope of this research Glider was used to perform the pattern recognition and get the smart contracts that contain the specified patterns. The most important feature that makes Glider an amazing tool for blockchain analysis is that it treats smart contracts as data objects.

1) Pattern Matching using Glider: The refined query designed to perform pattern recognition based on second heuristic (Fig. 5) firstly filters all the contracts where there is a deposit() function that takes an argument of bytes32 and also has the withdraw() function. After iterating through the resulting contracts it takes withdraw functions where "require" function is used. If the "require"-'s operand has "verif" (to match all the variations) name, the query returns the matched results. The result includes the address, contract name and the code snippet of the resulting smart contracts.

```
from glider import *

def query():

    contracts = (
        Functions().
        with_signature("deposit(bytes32)").
        contracts().
        with_all_function_names(["withdraw"]).
        exec()
    )

    output = set()

    for contract in contracts:
        instrs = (
            contract.
            functions().
            with_name("withdraw").
            instructions().
            with_called_function_name("require").
            exec()
        )

        for ins in instrs:
            calls = ins.get_callee_values()
            for call in calls:
                if call.get_name() == "require":
                    if "verif"
                    in call.get_args()[0].expression:
                        output.add(contract)
                        break

    return list(output)
```

Fig. 5: Python Query in Glider For Matching Smart Contracts

The smart contracts in Glider are grouped by their bytecodes. This means if two contracts have the same bytecode Glider will return only one address. For example, if contract "ERC20Tornado" and "Tornado" have exactly the same bytecode, since their source codes are the same, as a result it will be possible to get only one address instead of two. Sadly, this will decrease the number of extracted addresses.

Thus data preprocessing was conducted, by joining those smart contracts' names under the same address. As a result we get a list of unique addresses, and smart contract names. The total number of extracted addresses is 110. Lastly, with the confirmed list of the mixers, which included their names, it was possible to match the contract names with the initial result of the first heuristic and get a list of 152 addresses. The additional 42 addresses are the addresses associated with our confirmed mixers, that don't have the specified withdrawal functionalities, rather are the proxies, routers or wrappers.

B. BigQuery

The resulting addresses were joined with BigQuery's Ethereum transactions table, to filter out the addresses that have received transactions, in other words "have been used". With the help of the transactions database, it was also possible to retrieve the number of transactions sent to these mixers, as well as the last block timestamp. The block timestamp shows when the block was created. It takes from 15 seconds up to a minute for a new block creation in Ethereum, thus the offset between the actual transaction time and the block timestamp is very small. With the last block timestamp it is possible to identify approximately when was the last transaction sent to the specified address, therefore detecting when was the last time the mixer was used. Out of 152 addresses, only 84 were matched through the transactions table and confirmed to participate in transactions. This implies that the other 68 contracts were never used after their creation.

1) Additional research: As an additional research it was decided to test unverified contracts. Glider contains the database of all the verified contracts, however there can be mixers that didn't get their addresses verified to protect their anonymity. With the use of BigQuery it was possible to extract all the addresses that have received transactions with the input starting with 0xb214faa5, which falls in the pattern recognition technique of the first proposed heuristic. As mentioned above 0xb214faa5 is the signature of deposit(bytes32) function, and this method will give us the list of addresses that have received this specified transaction. It yielded a list of 160 addresses. These addresses were run through Etherscan's API to differentiate the ones that don't have verified contracts. As a result 44 addresses were found to be unverified. Unfortunately from this point it is possible to check whether these addresses are mixers by manually decompiling each source code, which is not necessarily accurate. As an initial result at least three addresses were mixers.

VII. Results

A. Summary of Identified Patterns

The analysis conducted on Tornado Cash and other Ethereum-based mixers has uncovered several distinguishing patterns and behaviors which are pivotal in identifying mixer use within the blockchain. The summary of identified patterns revolves around key smart contract functions, usage characteristics, and the application of cryptographic principles.

- The review identified deposit(bytes32) as a distinctive function signature commonly used across Tornado Cash and similar mixers. This function involves depositing funds using a unique identifier, which preserves anonymity but is traceable to a specific transaction pattern.

- Mixers with differing deposit functionalities were tested through their function signatures. The presence of these signatures on the blockchain helped to pinpoint active and inactive mixers.

- Examination of block timestamps indicated that despite various sanctions and legal actions, Tornado Cash and several other mixers remain actively used on the Ethereum Mainnet.

- The use of zero-knowledge proofs, specifically zk-SNARKs, and other cryptographic methods like the Pedersen and MiMC hashes were notable. These are employed to validate transactions without compromising the privacy of the involved parties.

- A binary Merkle tree structure is commonly used to store transaction data, ensuring both privacy and verifiability of the mixer's operations. Additionally, nullifiers are utilized to prevent double-spending by marking each spent note as "nullified," ensuring that once a note is withdrawn, it cannot be reused.

- Mixers implement denominations as a practical mechanism, which helps users deposit and withdraw fixed amounts (e.g., 1, 10, or 100 ETH).

- The use of relayers and the implementation of transaction fee mechanisms within these mixers also stood out. Relayers help mask the original source of a transaction, further complicating the traceability.

- The initial heuristic based on the 'deposit(bytes32)' function pattern yielded a large number of potential mixer addresses. However, it provided a significant amount of false positives.

- Refined heuristics incorporating both deposit and withdrawal functions, and their associated cryptographic checks (proof verification), proved more effective in identifying mixers with higher accuracy.

These patterns form the foundation of a heuristic approach that can be further refined and used by regulatory bodies and researchers to monitor and analyze mixer usage on the Ethereum blockchain. The study has shown that while mixers can be used for illicit activities, they also play a critical role in protecting user privacy. This dual nature presents ongoing challenges and opportunities in the pursuit of blockchain transparency and privacy.

B. Findings

n total 152 mixer addresses were identified, 84 of them being used for funds deposit or withdrawal. Fig. 6 illustrates the number of addresses associated with each mixer, comparing their total occurrences versus how many were actually used for transactions.

As a result Tornado Cash has 29 addresses out of 63 that have been used for mixing. A total of 19 mixers were identified. After filtering through BigQuery's dataset, 14 of these

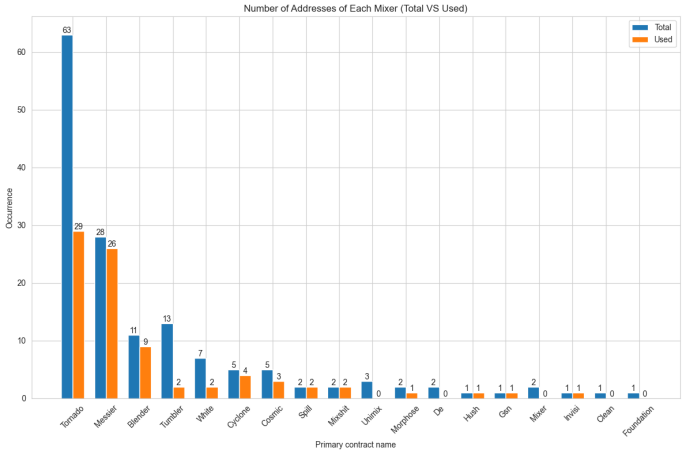

Fig. 6: Number of addresses associated with mixers.

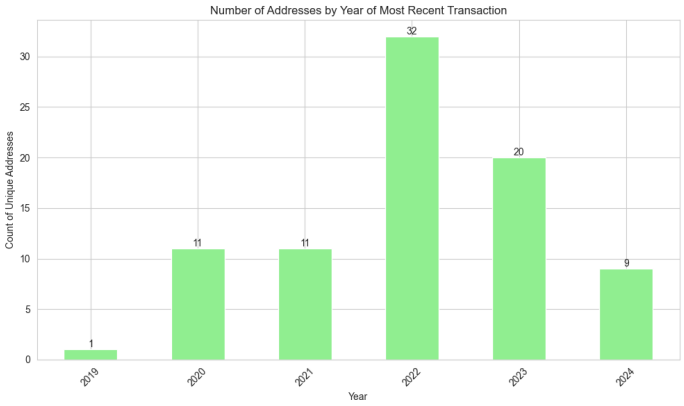

Fig. 8: The number of addresses and their last transaction's year

mixers were confirmed to have participated in transactions. In Fig.7 is shown the number of total transactions of each mixer, the count of the addresses and the year of the most recent transaction. Tornado Cash's 29 addresses have received 356217 transactions, last one being in 2024. It also highlights the mixers that are still in use: Tornado Cash, Blender, Cosmic Mixer, Mixshit, and Hush Mixer.

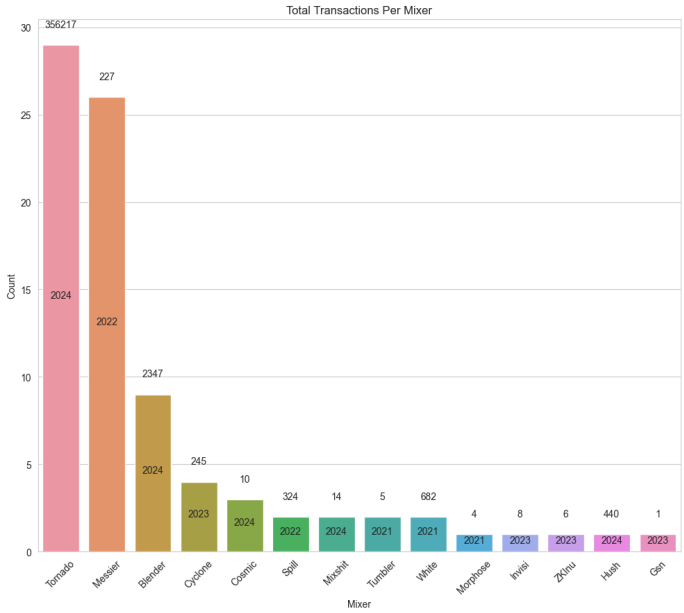

Fig. 7: Total amount of transactions of all the mixer's addresses and the year of the last transaction

VIII. Future Work

Of the 84 addresses analyzed, 32 were last used in 2022, 20 in 2023, and 9 in 2024. This pattern may be attributed to legal issues arising from the Tornado Cash incident and subsequent sanctions(Fig. 8).

IX. Future Work

In future work, the research can expand on improving detection mechanisms by incorporating machine learning models that evolve with changing mixer patterns. With the future updates on blockchain's analytical tools it will be possible to conduct more extensive and automated research to unravel illicit activities and anomalies.

Moreover, developing advanced privacy-preserving analytics will enable more effective monitoring of mixer operations without compromising the privacy of innocent users.

Regulatory frameworks also present an important area for further exploration. Future studies could delve into assessing the impacts of global and regional regulations on mixer adoption, seeking ways to implement regulatory measures that balance security with blockchain's decentralized nature.

Cross-chain mixer detection would offer a broader and more comprehensive view of laundering networks, as criminals may increasingly utilize multiple blockchains. Developing tools that work across various networks would yield more accurate insights into global mixing practices.

Lastly, research should focus on building more sophisticated network analysis models to map out relationships between mixer contracts, wallets, and decentralized applications. This would illuminate the structure of illicit operations, helping to anticipate and disrupt their strategies before they evolve further.

In the scope of this research, plans include further improving the pattern recognition model to detect mixers that differ in architecture from Tornado Cash and other mixers identified in this study.

References

[1] "Treasury Sanctions Virtual Currency Exchange for Laundering Ransomware Proceeds," U.S. Department of the Treasury, September 16, 2022. [Online]. Available: https://home.treasury.gov/news/press-releases/jy0916. [Accessed: March 3, 2024].

[2] "Tornado Cash Founders Charged with Money Laundering and Sanctions Violations," U.S. Department of Justice, Southern District of New York, [Online]. Available: https://www.justice.gov/usao-sdny/pr/tornado-cash-founders-charged-money-laundering-and-sanctions-violations. [Accessed: March 3, 2024].

[3] Y. Tang, C. Xu, C. Zhang, Y. Wu, and L. Zhu, "Blockchain Transaction Analysis for Privacy Preservation in Ethereum Mixer," in Communications in Computer and Information Science (CCIS), vol. 1506, China Cyber Security Annual Conference, Springer, 2021, pp. 23-35.
[4] T. Barbereau, E. Ermolaev, M. Brennecke, E. Hartwich, and J. Sedlmeir, "Beyond a Fistful of Tumblers: Toward a Taxonomy of Ethereum-based Mixers," presented at the 44th International Conference on Information Systems, Hyderabad, India, Dec. 2023.
[5] Burleson, J., Korver, M., and Boneh, D. (2022). Privacy-protecting regulatory solutions using zeroknowledge proofs. a16z crypto.
[6] Seres, I.A., Nagy, D.A., Buckland, C., Burcsi, P.: Mixeth: efficient, trustless coin mixing service for ethereum. In: International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2019). Schloss Dagstuhl-LeibnizZentrum fuer Informatik (2019)
[7] Meiklejohn, S., Mercer, R.: Möbius: trustless tumbling for transaction privacy. Proc. Priv. Enhanc. Technol. 2018(2), 105–121 (2018)
[8] Ethereum, "Smart Contracts," Ethereum.org, 2024. Online. Available: hhttps://ethereum.org/en/smart-contracts/. [Accessed: 1-May-2024]
[9] Ethereum, "Nodes and Clients - Archive Nodes," Ethereum.org. [Online]. Available: https://ethereum.org/en/developers/docs/nodes-and-clients/archive-nodes/. [Accessed: 1-May-2024]
[10] Ethereum ETL, "Google BigQuery," in Ethereum ETL Documentation. [Online]. Available: https://ethereum-etl.readthedocs.io/en/latest/google-bigquery/. [Accessed: 3-May-2024]
[11] 4byte Directory, "Home," 4byte Directory. [Online]. Available: https://www.4byte.directory [Accessed: 19-April-2024].
[12] Tornado Cash, GitHub repository, [Online]. Available: https://github.com/tornadocash [Accessed: 12-April-2024].
[13] Etherscan, "Understanding Transaction Input Data," Etherscan Informational Articles, [Online]. Available: https://info.etherscan.com/understanding-transaction-input-data/ [Accessed: 23-April-2024].
[14] Etherscan, "Ethereum Transaction Hash (Txhash) Details," [Online]. Available:https://etherscan.io/tx/0x4af2f444e8067556884df01cd6a3894bac108119a122df390d7fd4420b5a5dd5 . [Accessed: 23-April-2024].
[15] ."EVM Function Selector," [Online]. Available: https://www.evm-function-selector.click. [Accessed: 1-May-2024].
[16] A. Pertsev, R. Semenov, and R. Storm, "Tornado Cash Privacy Solution Version 1.4," December 17, 2019. Available: https://berkeley-defi.github.io/assets/material/Tornado
[17] Glider Documentation:https://glide.gitbook.io/main
BigQuery's Ethereum ETL: ttps://cloud.google.com/blog/products/data-analytics/ethereum-bigquery-public-dataset-smart-contract-analytics