Multidisciplinary Academic Grants in Cryptocurrencies
Final Report

# LORD OF THE RINGS: AN EMPIRICAL ANALYSIS OF MONERO'S RING SIGNATURE RESILIENCE TO ARTIFICIALLY INTELLIGENT ATTACKS

August 31, 2022

ACK-J
Monero Research Lab
LOTR_Monero_Paper@proton.me

Cryptocurrencies such as Bitcoin and Ethereum have seen a rapid increase in consumer adoption over the last decade. However, their lack of privacy guarantees has created a secondary market for more privacy-centric alternatives. Monero is a popular cryptocurrency with $2.9 billion in market capitalization and unique privacy properties which allow users to transact without a discernible history, similar to cash. In a transaction, the sender, receiver, and amount are hidden using well-established cryptographic primitives. The crux of Monero's strong privacy claims has historically surrounded *ring signatures*, used to obfuscate the transaction sender. A few previous works have analyzed the security of Monero's ring signature implementation, but none have assessed its updated on-chain resiliency to AI-based attacks. In this work, we develop a process to collect large-scale datasets composed of Monero transactions accompanied by ground truth labels. Using this process, we built two datasets from the Monero testing and staging networks and used them to explore feature engineering and model selection. These datasets are used to train various supervised-learning classi ers, simulating an adversary who aims to remove the anonymity set of a Monero ring signature. Our most e ective classi ers achieve a weighted F1-score of 34.60%, predicting an out-of-sample subset, and a macro F1-score of 13.30%, predicting against real mainnet Monero transactions. The model predictions show a marginal 4.30% increase in accuracy compared to the random guessing probability of 9%. Our research found that there to be minimal transaction risk posed by on-chain information leakage, correlated with adjacent Monero blockchains. We hope this work facilitates future multifaceted research into strengthening the Monero protocol against attacks correlating side-channel information.

## 1. Introduction

The cryptocurrency Monero has seen increased adoption and usage, largely attributed to the on-by-default privacy features preventing onlookers from learning any substantial information pertaining to users' transactions. The community of Monero developers abides by a traditional cypherpunk ethos, emphasizing end-user privacy and digital sovereignty over most other aspects. Rudimentary properties of a Monero transaction include inputs and outputs and can be thought of similarly to traditional cash. In a typical transaction using at currency, Alice pays Bob three $5 bills. If at a later point in time Bob needed to pay Alice a total of $10, Bob must combine two of his $5 bills as inputs into the transaction. Relating the analogy to Monero, each of Bob's $5 bills would be shu ed into a stack of 10 other identical counterfeit $5 bills, where Bob's $5 bill would be considered the *true spend* and the counterfeit bills are *decoys*. Hiding a transaction input within a group of other inputs is done within Monero using *ring sigantures* to create ambiguity within the blockchain.

Historically, most on-chain attacks have stemmed from imperfections surrounding Monero's use of ring signatures. These aws led to subsequent upgrades, such as in 2017, Monero switched to a more advanced ring signature construction, known as RingCT [1], shielding transaction amounts.

Large, well-resourced entities have recently been taking aim at Monero. While the cryptographic techniques used conceal transaction information from observers, companies such as CipherTrace claim to have some ability to track Monero transactions [2]. Ciphertrace has led for two US patents specically related to tracing Monero titled, *Techniques and Probabilistic Methods for Tracing Monero*, and *Systems and Methods for Investigating Monero*. The patent process is ongoing, and the applications have yet to be published publicly [3]. The techniques used are believed not to be an exploit within the Monero protocol but instead, a machine learning and graph theory analysis of the network utilizing on- and o -chain information to produce a probabilistic estimate of the true transaction graph [4]. Furthermore, CipherTrace has been reluctant to present evidence of its claim, creating an inherent skepticism about the e ectiveness of the company's technology. In September 2020, the United States Internal Revenue Service o ered two separate $625,000 bounties [5] for contractors to develop tools that could be used to trace Monero transactions. This bounty was awarded to Chainalysis and Integra-FEC to work closely with the IRS on Monero tracking technologies. From these e orts, we see the importance of understanding the privacy properties that Monero provides.

Previous works [6, 7] have assessed areas where Monero's resiliency to statistical analysis has fallen short and how the immutable nature of the blockchain poses a security risk for post hoc deanonymization. However, around a dozen network upgrades following these works have included signi cant security and privacy improvements. More recent publications [8, 9] have evaluated post-RingCT traceability. These works revealed the ine ectiveness of past heuristics such as guess-newest, zero mixin chain-reaction, and merging outputs, later de ned in Section 2.1. Even with the public nature of blockchain technologies, the private nature of Monero limits the features which could be extracted for analysis.

We propose the rst data collection pipeline to aggregate deanonymized Monero transactions and overcome these hurdles sti ing research. As only a single work [10] has brie y assessed the AI-resilience of Monero's ring signature implementation, we look to review the extent of the attack surface. More specically, we have the audacious goal to determine which features of the blockchain may be leveraged to predict the true spend of an arbitrary ring signature with accuracy greater than random guessing, absent external information. Unlike blockchain analytic companies, this research will intentionally disregard the abundant user-specic information collected and shared by various data brokers.

*The contributions of this work include:*

1. An open-source automated data collection system of Monero transactions to generate large-scale datasets. We used this system to collect two open-source datasets using

di erent spending patterns of users.

2. An empirical evaluation of various machine and deep-learning algorithms to predict the true spend of a ring signature on various Monero networks, including the main network.

## 2. Background

Monero emerged as an honest fork of the ByteCoin cryptocurrency, both based on the CryptoNote protocol [11]. When a user transacts with Monero, the transaction information is hidden on the blockchain through cryptographic assurances that achieve *unlinkability* and *untraceability* without a ecting the auditability of the ledger [12].

Unlinkability: the inability to link two addresses to the same entity.

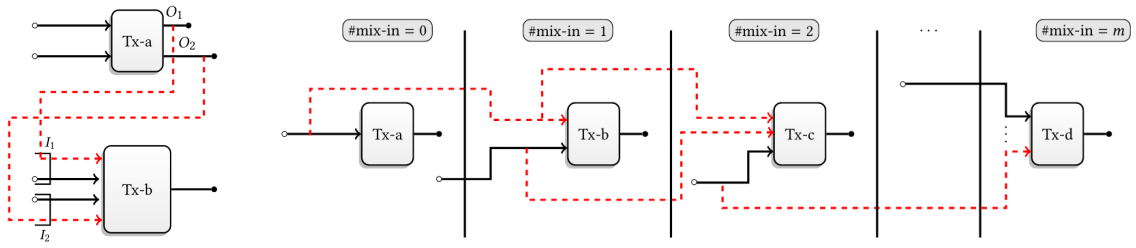
Untraceability: the inability to identify the redeemed input among a set of other equiprobable inputs.

Monero achieves these properties through a tripartite of cryptographic primitives, including Pedersen commitments [13], ring signatures [14], and stealth addresses [15]. This di erentiates Monero from most other cryptocurrencies using *transparent ledgers*, such as Bitcoin [16] or Ethereum [17] which are architected to record transactions and account balances pseudonymously, often revealed by quasi-identi ers. To the detriment of public blockchains, there is no limit to who can audit transactions, including friends, employers, corporations, or repressive governments. This is a large shift from the status quo, where traditionally, only banks, authorized third parties, and domestic governments have the same level of visibility into one's nancial records.

Unlike Bitcoin, a user's Monero wallet address never appears on the blockchain providing *unlinkability*. Instead, for every transaction, a new one-time *stealth address* is derived from the recipient's public key, along with some randomness. For each transaction input, ten other Monero users' inputs are added as *decoys* to prevent an adversary from identifying which coin was spent, providing *untraceability*. To shield the transaction amounts, Monero uses homomorphic properties of Pedersen commitments [18] to verify the number of coins input minus the coin's output, zero-sum, without having to reveal the amounts publicly. A zero-knowledge range proof, known as a *bulletproof* [19, 20], is added to prevent sending a value that is non-negative or could in ate supply. Combining these three techniques gives coins no clear history, achieving true *fungibility*, meaning that two monetarily equal units can be exchanged without a discrepancy.

### 2.1. Heuristics

Notable on-chain heuristics include output merging [7, 9], chain reactions [6, 21], and temporal analysis [6, 7, 9, 22, 23]. Output merging, shown in Figure 1a, is the process of




(a) Output merging

(b) Zero mixin chain-reaction

Figure 1: Heuristics depicted by Kumar et al [7].

linking two or more outputs of a previous transaction as inputs to a subsequent transaction. These outputs are statistically more likely to be the *true spends* due to the improbable nature of the event. The *chain reaction* heuristic applies to pre-RingCT transactions shown in Figure 1b, where *Tx-a's* choice to not use decoys eliminated the anonymity set from *Tx-[b,c,d]*. A temporal heuristic that has become less e ective after decoy selection upgrades is the *guess newest* heuristic. This proposed that, given all members of a ring signature, the newest one is likely to be the true spend.

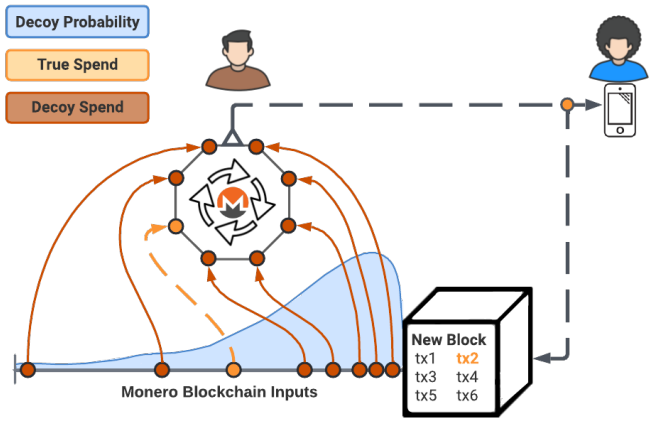



Figure 2: A simplistic visual of how ring signatures are constructed to obscure the true spend among decoys.

## 2.2. Ring Signatures

The use of membership proofs [24] within cryptocurrencies to conceal senders was rst proposed in 2013 by Saberhagen [11], and later improved and adapted into RingCT [1]. Alternatively to each input referencing a previous transactional output (TXO) stored on the blockchain, a ring signature allows one member to digitally sign a transaction on behalf

of multiple Monero inputs, obfuscating which input was truly spent. Monero transactions commonly use multiple inputs for a single transaction, each of these inputs must construct its own ring signature.

Ring signatures are composed of four distinct algorithms described by Saberhagen [11]:

1. GEN: The signer picks a random secret key $x$ to compute public key $P = xG$, and key image $I = xH_p(P)$, where $H_p$ is a deterministic hash function and $G$ is a base point on the Ed25519 elliptic curve [25].
2. SIG: The signer takes a message $m$, a set $S'$ of public keys $\{P_i\}_{i \in S}$ and outputs a signature $\sigma$ with a subset $S = S' \cup \{P_s\}$, where $P_s$ is the signer's own public key.
3. VER: The verifier checks the signature using message $m$, set $S$, and signature $\sigma$.
4. LNK: The verifier ensures $I$ is unique and has not been used by a past signature.

The method in which ring members are enforced must be carefully considered, as previous naive implementations allowed trivial decoy elimination using the methods shown in Figure 1b. In the past, ring signature decoys were chosen uniformly or with a triangular distribution leading to decoy elimination through heuristics such as *guess newest*. More recently, decoys have been selected using a shifted gamma distribution based on the spending patterns of deanonymized pre-RingCT transactions, as suggested by Möser et al. [6]. This distribution emphasizes newer outputs to mimic real user spending patterns, as shown in Figure 2. Monero transactions are constructed *autonomously* and *spontaneously* since a user mixes coins on their own without any latency. A constructed ring signature produces a verifiable zero-knowledge proof, where the true spend is theoretically equiprobable to be any of the participants. However, attackers can exploit side-channel information, the extent of which we seek to evaluate empirically. Ring signatures are the source of the Monero blockchain ledger obfuscation, such that tracing an output becomes exponentially difficult as the anonymity set compounds over time. Our work evaluates Concise Linkable Spontaneous Anonymous Groups (CLSAG) as they are the current variation of ring signatures enforced at the consensus level for the Monero network. We acknowledge that upon the competition of this work Monero increased the ring signature size from 11 to 16.

### 2.2.1. Future of Ring Signatures

There have been multiple proposals for new proofs to construct sub-linear sized linkable ring signatures. Triptych was among the first alternatives, proposed by Noether et al. [26] to replace CLSAG [14]. It includes a zero-knowledge proving structure with logarithmic growth built on top of the existing ring architecture. However, Triptych's inability to create multi-signature transactions easily resulted in its abandonment. The most promising upgrade is Seraphis [27], a complete redesign replacing linkable ring signatures with a combination of

membership and authorization proofs. Seraphis achieves logarithmic transaction size growth, similar to Triptych, and could increase the ring size from 16 to 128 while maintaining a similar transaction size; however, the implementation is not anticipated to be finished until 2024.

## 3. Related Works

The most notable prior work was written by Borggren et al. [10] who constructed multiple synthetic Monero blockchains including different simulated economies of miners and traders, training a random forest and neural network to predict RingCT true spends. The authors were able to achieve accuracies greater than random guessing and found *num_rings* to be the most important feature of the classifier, this is largely due to the experiment being conducted before Monero enforced a ring size of 11. Unfortunately, this work did not publicly release the data collection method, dataset, or trained models for future works.

An earlier publication by Möser et al. [6] in 2017 presented many novel tracing techniques and empirically analyzed *chain reaction*, a previously theoretical chain-analysis attack [21, 28]. The security posture of Monero has greatly matured since this work, and the fundamental privacy concern, *zero mixins*, has long been remediated. This particular heuristic emanated from users intentionally lowering or outright removing other ring members from their transactions, sacrificing personal privacy to reduce fees. Interestingly, out of the 12,158,814 transactions analyzed by the paper, 64% of them contained zero-mix-in decoys. This demonstrates that a majority of users will opt to sacrifice privacy to reduce fees if given a choice. In many cases, these actions linked which output on the blockchain was used, making Monero function more closely to Bitcoin. Consequently, many Monero users who chose an appropriate number of ring members unknowingly included previous *zero-mix-in* outputs. In Figure 1b, users unintentionally reduce or eliminate plausible deniability from future transactions. Immediately following the January 2017 network hard fork introducing RingCT [13], the accuracy of the chain-reaction attack plummeted from 90% to about 10%.

Several well-established attacks on the Monero protocol, still effective today [29], use *coercion* and *chain analysis*. Through social engineering, a Janus attack allows an adversary to link multiple addresses derived from the same seed. In a scenario where two addresses are suspected of belonging to a single entity, an attacker would send Monero to the wrong address and subsequently ask the entity to confirm the deposit of funds. A confirmation would prove a link between the two addresses. An EABE (Eve, Alice, Bob, Eve) attack occurs when a single entity, such as a cryptocurrency exchange, is positioned on either side of a single transaction and can make a deterministic link between the two individuals transacting with one another through *chain analysis*. Lastly, a key-image reuse attack is a form of *chain reaction*, which occurs after a network hard fork. Suppose a user spends a Monero output on fork A of the blockchain and the same output on fork B of the chain.

In this scenario, a passive attacker could match the *key images* and conclude that the unique member included in both ring signatures was the true spend for both transactions. If enough transactions are deanonymized in this manner, it could lead to a similar *chain-reaction* as demonstrated by Möser et al. [6]. All attacks mentioned, except for key-image reuse, assume intimate outside knowledge about the victim, not readily accessible on-chain. The pseudonymous nature of transactions can often be de-anonymized with tracing and real-world information about the identity of transaction owners and their outputs. Our study disregards this information and focuses solely on the protocol as it was implemented.

While no complete Monero transaction datasets have been distributed, there are multiple blackball lists of known spent transactions [30, 31], from pre-RingCT versions of Monero. While useful in past situations, the half-decade-old data has become less practical over time as the protocol advances. The lack of quality data and the difficulty of the collection process are the leading factors deterring researchers from this field of work. Borggren et al. [10] commented on this challenge, stating, "The difficulty in procuring labels for supervised learning remains a central challenge, despite the accessibility of blockchain activities themselves. In Monero, this is especially so." While the cryptography and obfuscation behind Monero are strong and well tested, it's uncertain whether an implementation error or erroneous side-channel information could allow an AI model to draw conclusions overlooked by security researchers. No works have analyzed the security of the more recent Monero ring signature implementation against AI attacks. Furthermore, no datasets have been published for researchers to verify privacy claims independently.

## 4. Data Collection Pipeline

The data collection involved setting up servers to construct, fund, and transact between wallet pairs over multiple months. Additional software was written [32] to extract all outgoing transactions from thousands of wallets into csv files. The exported files were parsed, filtered, and enriched before entering the cleaned and undersampled dataset. The scripts used to control the wallets were written using Bash, Python, and Expect. An Expect script allows dynamic interaction with programs that require a text-based terminal interface. This allowed more granular control over the wallet functionality and provided an easy way to manage and debug the process. Expect scripts were dynamically written to create, configure, and control each Monero wallet during the data collection. All the metadata from each transaction was later collected and enriched by feature engineering from adjacent blockchain data.

The data collection process was performed on the Monero staging and testing networks to achieve the most realistic results. Notably, the collection was not conducted on the Monero main network due to serious privacy implications through *chain reaction* caused by public disclosure of true ring members. User spending patterns on the staging/testing networks are assumed to largely differ from that of the main network due to the valueless nature

of coins. Another key distinction is that, while coinbase transactions are still present, no known mining pools are active on these networks, excluding a potentially useful heuristic. The bene ts of collecting a real public Monero blockchain dataset, as opposed to a private and locally simulated chain [33, 10], are that independent researchers can verify the results, and it allows una liated transactions with older histories to be included in ring signatures.

Each subsequent transaction sent by wallets, collecting the staging dataset, was delayed by a random number of seconds sampled from a shifted gamma distribution. It should be acknowledged that this distribution is used to simulate a perfectly ideal spending pattern as implemented and to distribute the true spending more evenly across each ring signature position.

A gamma distribution is described by:

$$p(x) = x^{k\ 1}\frac{e^{\frac{-x}{\theta}}}{k\quad(k)}$$

Where $k$ and are, respectively, the shape and scale values proposed by Moser et al. [6].

$$k = 19.28, \quad = \frac{1}{1:61}$$

Additional steps were taken to guarantee the dataset's quality and ensure it could accurately represent various transaction scenarios. To create variety, each transaction amount was chosen uniformly at random between the range [1 $10^{\ 4}$ - 1 $10^{\ 12}$]. Choosing di erent amounts for subsequent transactions had the added bene t of breaking up coins into unpredictable outputs within each wallet. Many smaller outputs could be selected, varying the number of inputs for each constructed transaction. Lastly, a random transaction priority was chosen uniformly within the range [1 - 4]. This altered the transaction fee advertised to the miners, thus changing how quickly the network would process it. The priority levels \unimportant", \normal", \elevated", and \priority" correspond to fee multipliers of x0.2, x1, x5, and x200, respectively. While priority levels are not enforced at the consensus level and are subjective to each wallet's implementation, these speci c multiples were chosen as they are implemented within the `monero-wallet-cli` maintained by the core developers.

To understand the distribution of transaction fees in Monero, we wrote a script [34] to scan the last 100,000 blocks on the main Monero network, starting at block 2566273. Figure 3 shows this distribution based on the ratio of transaction fees paid with respect to the transaction size (kB). While there were clear outliers outside of the normal priority levels, technical limitations of the `monero-wallet-cli` prevent arbitrary fee amounts from being selected. Note that a competitive fee market has yet to fully emerge within Monero.

### 4.1. Data Collection Scripts

An assortment of scripts rst automate transactions between wallets, then export and combine wallet data into a dataset. The extraction, cleaning, and ltering of transactions

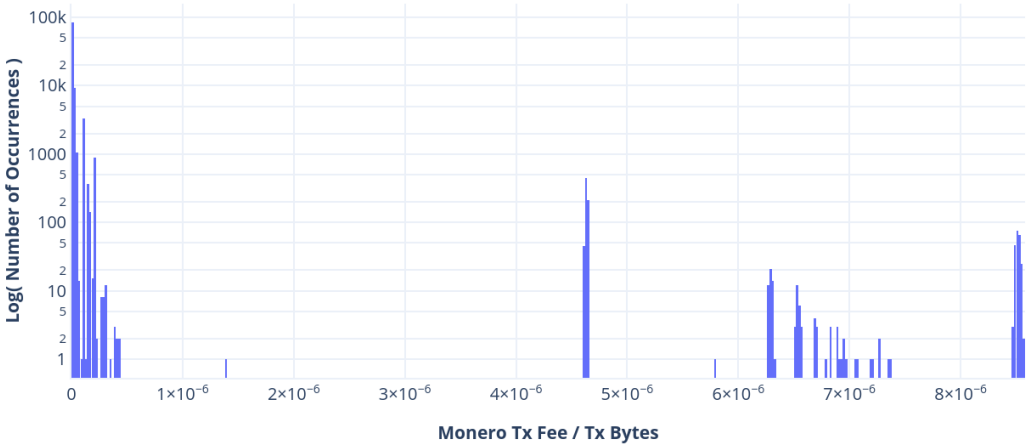



Figure 3: The number of mainnet Monero tx fees/byte over 100k blocks (log scale).

were non-trivial and required three processing stages. An Ubuntu 20.04 server con gured with an AMD Ryzen 64-core Threadripper 3970X and 256GB of RAM took roughly 54 hours to complete stages two and three of the pipeline for the stagenet dataset.

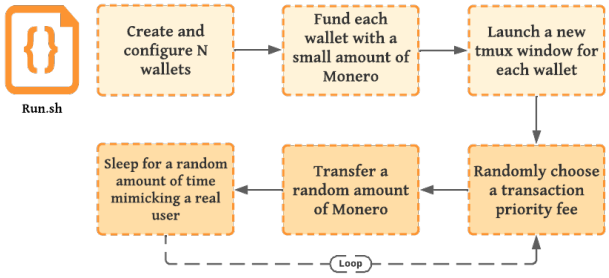



Figure 4: Run.sh execution ow chart.

The rst script in the pipeline, `run.sh`, referenced in Figure 4, is responsible for initiating and managing transactions performed on-chain. At the time of collection, we used the most recent release of the o cial Monero software, v0.17.3. Next, the user speci es the number of wallets to be created, funded, and automated to transact with an adjacent wallet until a certain deadline. Notably, the automation process used Expect scripts instead of an RPC wallet server. Each wallet independently samples a random Monero transaction fee between [1 - 4] and an amount of Monero to send from a uniform distribution between [1 $10^{4}$ - 1 $10^{12}$]. After successfully sending the transaction, the script will sleep for a random time sampled from a user's desired distribution. Once the sleep cycle has been completed, the process loops back to create a new transaction. A user will specify an end date for all transfers to halt, causing the wallets to shut down gracefully.

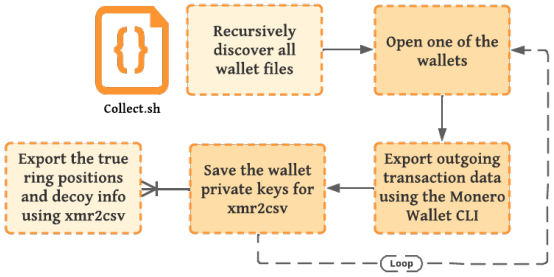



Figure 5: Collect.sh execution flow chart.

The second stage of the collection process is handled by the `collect.sh` script, shown in Figure 5. This script recursively locates wallet files within a given directory, opens each one using `monero-wallet-cli` to execute the `export_transfers` command, and outputs one csv file per wallet.

The csv file, `cli_export.csv`, is detailed in Appendix C. Next, a Monero wallet RPC server is started to programmatically extract the wallet's starting block height, private view key, and private spend key. This information is appended to a temporary file and will be used in a future step. The script cycles over the remaining wallets until each has successfully been exported. Finally, `collect.sh` opens the temporary file containing the wallets' private keys and passes them as command line arguments to `xmr2csv` [35], a program that scans the blockchain for outputs used as decoys and decrypts the true spend. This process procures the ground truth labels for the dataset.

Due to the single-threaded implementation and resource-intensive nature of `xmr2csv`, this process can be very slow for a large number of wallets. To improve performance, GNU Parallel was used for multiprocessing to distribute the computational load across multiple CPU cores. This dramatically increases efficiency and eliminates a bottleneck within the pipeline. Once complete, `xmr2csv` outputs five csv files per wallet containing unique features relevant to the final dataset, as shown in Appendix C.

## 4.2. Data Cleaning and Enrichment

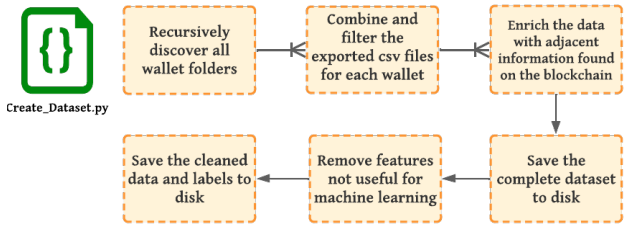



Figure 6: Create_dataset.py execution flow chart.

The final stage of the pipeline starts with the data cleaning and featurization process. The python script, `Create_Dataset.py`, discovers all csv files associated with each wallet, extracting only the unique characteristics of a transaction. The metadata goes through an enrichment phase where adjacent information on the blockchain is aggregated and used to create temporal and relational features.

The enrichment phase relies largely on a locally synced block explorer [36], as the efficiency of this process is improved using multiprocessing to send requests in parallel. The final dataset accumulates all metadata associated with each transaction, including features that should be irrelevant to machine learning, such as private keys and cryptographically random wallet addresses. These fields were intentionally not omitted from the dataset as it provides key contextual information for other applications. Lastly, the full dataset is randomized, undersampled, and flattened into a pandas dataframe, separating the associated labels.

# 5. Datasets

The three collected datasets are detailed below with the number of ground truth ring signatures plotted in Figure 7. Each of the 11 ring members is positioned in order of their relative age, where class 1 and class 10 are, respectively, the oldest and youngest ring members. The two training datasets can be downloaded using the links referenced on the Github repository [32].

- Testnet training dataset: The wallets sent subsequent transactions immediately after the minimum lockout period of twenty minutes.
- Stagenet training dataset: The wallets sent subsequent transactions after a delay sampled from the gamma distribution, proposed by Möser et al. [6].
- Mainnet validation dataset: The wallets sent real transactions to buy goods and services over a year period.

## 5.1. Testnet Dataset Using 20 Minute Delay

The testnet dataset was collected over 34 days between January 20, 2022, and February 23, 2022. Each of the 900 wallets transacted once every 20 minutes. This resulted in 763,314 transactions sent on the Monero testing network, including 1,333,756 ring signatures, where the true spends formed the distribution shown in Figure 7. The large imbalance of true ring positions is visibly left skewed, with over 250,000 transactions subject to the *guess newest* heuristic. The large uptick in the $11^{th}$ position is caused by the decoy selection algorithm being unable to properly obscure the true spend if immediately spent after the lockout period. The testing network transactions per day over the data collection period are charted in Figure 8. A random undersampling of 14,171 ring signatures was selected to

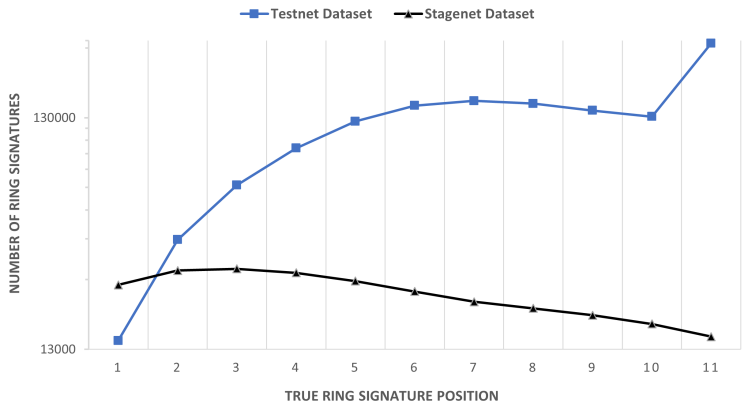



Figure 7: Dataset distributions of the true ring signature positions using di erent time delays, before undersampling.

eliminate the *guess newest* heuristic and balance model training. After this was performed on each of the 11 classes, the dataset totaled 155,881 samples, each possessing 81 features. The feature-set used for this dataset was similar to the stagenet dataset, except for input decoy-related metadata, shown in Appendix D.

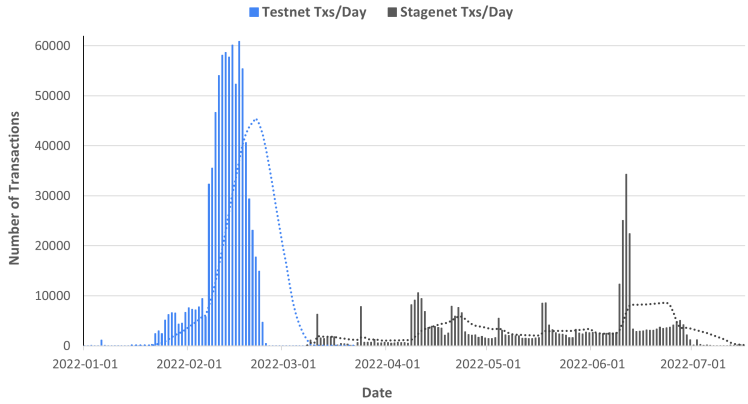



Figure 8: Transactions/day over di erent data collection periods.

## 5.2. Stagenet Dataset Using Gamma Distribution for Delay

The stagenet dataset was collected over 73 days between April 19, 2022, and July 1, 2022, with an aggregate of 9,342 Monero wallets, 165GB in size. This resulted in 184,980 transactions sent on the Monero staging network composed of 248,723 ring signatures, where the true spends formed the distribution shown in Figure 7. An undersampling of 14,750 ring signatures was randomly selected from the shu ed data for each of the 11 classes,

totaling 162,250 rings with 1,741 features. The staging network transactions per day over the data collection period are charted in Figure 8. Notably, on June 6, 2022, a power outage required all wallets to be restarted, creating a large in ux of transactions which allowed a developer to identify a bug when the mempool contains more than 100 new transactions and a user is connected to a restricted node [37]. This issue was subsequently xed in v18.0 of the Monero software.

### 5.3. Mainnet Validation Dataset

A dataset of 53 post-RingCT mainnet transactions was sourced from various wallets owned by the researchers. These transactions represent real usage on the Monero network to buy goods, send donations, and transfer between self-custodial wallets. This dataset included 76 total ring signatures and was used to validate the e ectiveness of the model's predictions. This validation dataset will not be released due to privacy concerns surrounding chain-reaction attacks but was processed in the same manner, using the same code, as the two previous datasets.

## 6. Feature Engineering and Models

### 6.1. Feature Engineering

Featurizing the data collected resulted in 1,741 distinct features, summarized in Appendix D. For e ciency purposes, all cryptographically random strings or columns that did not include more than one unique value were stripped from the feature set. For the neural network, all features were Z-normalized prior to training and predicting tasks. Features are di erentiated, in Figure 9, between 0-hop and 1-hop. A 0-hop feature pertains to information extracted from the transaction of interest, while a 1-hop feature incorporates data from previous transactions in the graph relative to the transaction of interest.

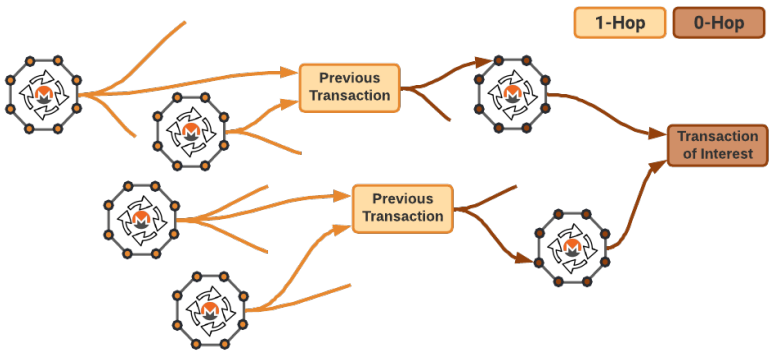



Figure 9: Distinction between 0-hop and 1-hop features.

Some 0-hop features covered using only metadata associated with the transaction of interest include `Tx_Fee`, `Tx_Size`, `Time_Deltas_Between_Ring_Members.1_2` and `Total_Ring_Time_Span`. Examples of 1-hop features include both temporal and relational information linking multiple transactions, such as `Previous_Tx_Num_Inputs.0`, `Previous_Tx_Block_Num_Delta`, and `Previous_Tx_TxExtra_Len`. Including both types of features should provide a classifier with more context to predict the true spend of a ring signature. The integer values within feature names represent the ring member position.

Due to the nature of ring signatures, users who send a transaction must include other outputs already on-chain as decoy members. This means that after you send a transaction and it is added to a block, your output will be included, as a decoy, in subsequent transactions. We believe the number of times an output was included in other ring signatures and the spacing between them is a novel heuristic that could identify anomalous activity. However, discovering all occurrences of an arbitrary output is quite computationally expensive. One must start at the output creation and exhaustively search each ring signature of every transaction until the transaction of interest. To drastically increase the efficiency of this process, we used a PostgreSQL database [38] to store the mapped relationships.

Hypothetically, there may be anomalous instances where one's output does not get included as a decoy into other rings. A large drop in daily transactions on the blockchain could lower the number of decoy inclusions and thus push the new output into an area of lower probability of future inclusions, seen in Figure 2.

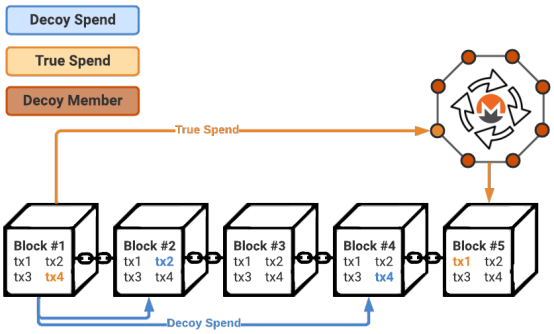



Figure 10: Occurrences of decoys used as features.

The occurrences between the output creation and the transaction of interest are depicted in Figure 10 with an example 5 block chain. First, the original occurrence of the output is located in block 1. Next, the ring signatures, which include the original output as a decoy between blocks [2 - 4], are queried using the PostgreSQL database [38]. Finally, the block deltas between all occurrences [1-2, 2-4, 4-5] and the total number of occurrences [2] are added as features to the dataset.

### 6.2. Models

We train our machine and deep learning models as multiclass predictors to forecast which of the 11 ring members is the true spend. An AdaBoost random forest [39], gradient-boosted classi er, and neural network was trained on 80% of the collected datasets. The random forest and gradient-boosted classi er were selected as traditional machine learning models due to their ensembled learning architecture, which e ciently generalizes high-dimensional data and provides some model interpretability. The neural network contained 26,555 trainable parameters, including a dropout and batch normalization layer, aiding the network in better generalizing the training subset. The loss of the neural network was calculated with cross-entropy and the adam optimizer. However, the neural network is more of a \black box," unable to provide insight into which features leak information but may correlate complex relationships easier. Understanding these tradeo s is critical to developing improved protections, so the interpretability of the traditional machine-learning models is valuable.

#### 6.2.1. Hyperparameter Tuning

The random forest and gradient-boosted classi er were hyperparameter-tuned using an exhaustive search over speci ed parameters taking an average 5-fold cross-validation. Models were trained in parallel across all available CPU cores [40] to reduce the time the search took. The Adaboost random forest's tunable parameters included the number of estimators and learning rate, while the gradient boosted classi er's tunable parameters included the learning rate, depth, and the number of estimators. The neural network layers were also subject to an exhaustive search using various amounts of neurons within the three hidden layers, dropout rate, and the number of epochs. The parameters which performed the highest were chosen to t the nal models.

## 7. Evaluation

### 7.1. Classi cation Performance

| Dataset | Transaction Delay | *Respective Dataset (Weighted F1-score)* | | | | | | *Mainnet Dataset (Macro F1-score)* | | | | | | Random Guessing |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | GBC | | NN | | RF | | GBC | | NN | | RF | | |
| Stagenet | Gamma Distribution | 25.27% | 0.05 | 18.95% | 0.50 | 22.53% | 0.00 | 13.24% | 2.47 | 5.74% | 2.64 | 12.90% | 0.03 | 9% |
| Testnet | Twenty Minutes | 34.60% | 0.02 | 21.17% | 1.38 | 28.52% | 0.00 | 8.00% | 1.70 | 5.60% | 2.21 | 13.30% | 0.04 | 9% |

Table 1: Performance: weighted and macro F1-scores over 10 runs with $\pm$ standard deviation. GBC = Gradient Boosted Classi er, NN = Neural Network, RF = Random Forest.

A random forest, neural network, and gradient-boosted classi er were t to each of the three datasets and, after hyperparameter tuning, achieved the accuracies displayed in Table 1. The model weights can be downloaded using the links referenced at the Github repository [32]. We evaluate our model in a multiclass setting and calculate the weighted precision, recall, and f1-score. For each class, a true positive (TP) is the correct identi cation of the true spend, a false positive (FP) is an incorrect identi cation of the true spend, and a false negative prediction (FN) is the incorrect identi cation of a decoy.

$$Precision = \frac{T_p}{T_p + F_p} \quad Recall = \frac{T_p}{T_p + F_n}$$

$$F_1\text{-}Score = 2 \quad \frac{Precision \quad Recall}{Precision + Recall}$$

Precision measures the number of times a ring member position was correctly classi ed compared to all true instances of the same position. Recall measures the number of times a ring member position was correctly classi ed compared to all predictions of the same position. Accuracies can often be misleading, especially with unbalanced data, thus we used variations of the F1-score metric. The macro F1-score was calculated to represent the macro-average of the harmonic mean between precision and recall, where $N$ is the number of classes and $i$ is the class index. This metric is insensitive to the imbalance of the classes and treats them all as equal, putting more value on the minority class, which is ideal for predicting on an unbalanced dataset such as our Mainnet dataset.

$$Weighted\ F_1\text{-}Score = \sum_{i=0}^{N} F_1\text{-}Score_i \quad W_i$$

$$Macro\ F_1\text{-}Score = \frac{1}{N} \sum_{i=0}^{N} F_1\text{-}Score_i$$

The out-of-sample accuracies in the respective dataset column use a weighted-f1 score, where the weight $W$ represents the number of true instances for each class. The stagenet and testnet datasets were previously undersampled before an 80%-20% train-test split, thus each class would likely have a similar number of samples. Notably, high scores for each model are emphasized in bold. The highest scores achieved for the respective datasets were both predicted by the gradient-boosted classi er. Due to the small number of mainnet samples collected, the dataset was not undersampled, and classes were imbalanced. The mainnet dataset, used for validation, saw a split of highest accuracies between the gradient-boosted classi er and the Adaboost random forest. The random forest achieved the most consistent accuracies throughout and over tted the least. Notably, the neural network performed slightly worse than the machine learning models when tested against the respective datasets and signi cantly below the random guessing threshold when tested against mainnet samples. We theorize that the neural network drastically over tted the training sets, leading to poor predictive performance.

## 7.2. Analysis

| AdaBoost Random Forest Features | Importance | Gradient Boosted Classi er Features | Importance |
|---|---|---|---|
| Tx_Size | 3.43% | Input.Previous_Tx_TxExtra_Len.0 | 9.65% |
| Tx_Fee | 3.00% | Input.Previous_Tx_TxExtra_Len.2 | 8.30% |
| Input.Previous_Tx_Time_Deltas.9 | 2.14% | Input.Previous_Tx_TxExtra_Len.1 | 8.27% |
| Input.Previous_Tx_Time_Deltas.10 | 2.14% | Tx_Size | 6.96% |
| Input.Previous_Tx_Num_Inputs.0 | 1.71% | Input.Previous_Tx_TxExtra_Len.3 | 4.71% |
| Input.Time_Deltas_Between_Ring_-Members.1_2 | 1.71% | Input.Previous_Tx_Block_Num_-Delta.10 | 3.96% |
| Block_Size | 1.71% | Input.Previous_Tx_Num_Outputs.1 | 3.77% |
| Input.Previous_Tx_Time_Deltas.2 | 1.43% | Input.Previous_Tx_Num_Outputs.2 | 2.23% |
| Total_Block_Tx_Fees | 1.43% | Input.Previous_Tx_TxExtra_Len.4 | 2.15% |
| Time_Since_Last_Block | 1.43% | Input.Total_Ring_Time_Span | 2.10% |

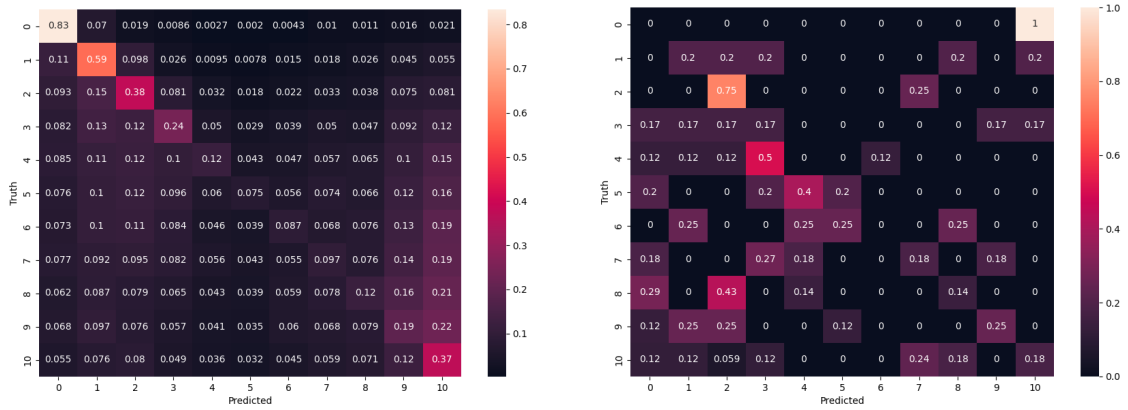
Table 2: Top ten important features extracted from machine learning classi ers trained on the stagenet dataset.

A confusion matrix for each best-performing model is displayed using heatmaps below. Each box within the matrix shows the precision of the model, where the diagonal line indicates an accurate prediction. Class 0 denotes the oldest ring member, whereas class 10 is the youngest member relative to the Monero blockchain. Each machine learning model's top 10 most important features were extracted and shown in Table 2. Feature importances are computed as the mean and standard deviation of accumulation of the impurity decrease within each tree.

The AdaBoost random forest appears to distribute the correlation more evenly across features, indicated by the lower priorities, and greatly emphasized temporal features and standard metadata such as *Tx_Fee*, and *Block_Size*. Only a single overlapping feature between the two lists, *Tx_Size*, strongly indicates that no single feature leaks a large amount of information. Instead, the models likely relied on behavioral patterns. The gradient boosted classi er indicated that previous *Tx_Extra* lengths were largely informative during the prediction, with the number one feature importance being almost three times higher than that of the random forest.

The *Tx_Extra* eld contains a transaction public key and other arbitrary data. This eld's removal has been debated as the ability to append arbitrary data hurts transaction uniformity. The length of the *Tx_Extra* could identify patterns of users who modify this eld, compared to users who do not. With the data collection taking place on development versions of the Monero blockchain, the types of users are more likely to manipulate this eld value manually.

Each model performed exceptionally well on the out-of-sample testing split of the databases. However, the patterns extracted did not transfer well to mainnet transactions. This could be attributed to the spending patterns chosen not accurately representing Monero's mainnet.

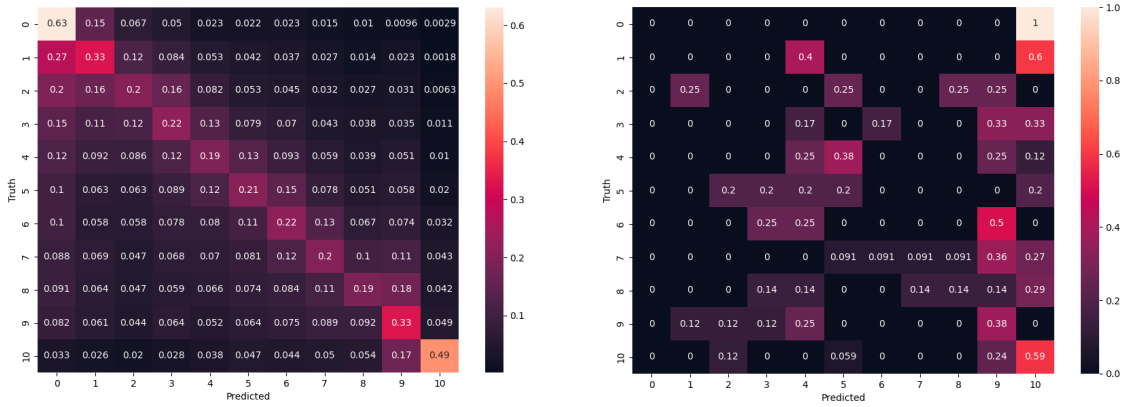



GBC prediction confusion matrix on the testing split of the training dataset

GBC prediction confusion matrix on the validation dataset

Figure 11: Precision heatmap of GBC models trained on the stagenet dataset.

The highest ranking gradient boosted classi er, trained on the stagenet dataset, could accurately predict the majority of classes greater than random guessing. The rst three classes were predicted with precision above 38% whereas class 0 was predicted correctly 83% of the time. This accuracy surpasses random guessing by over nine times. However, when tasked with predicting the mainnet validation dataset, the predictions appear much more sparse and random. Shockingly, using a more realistic spending pattern, compared to Figure 12, the GBC achieved 83% precision, 20% higher, when predicting the oldest ring member.

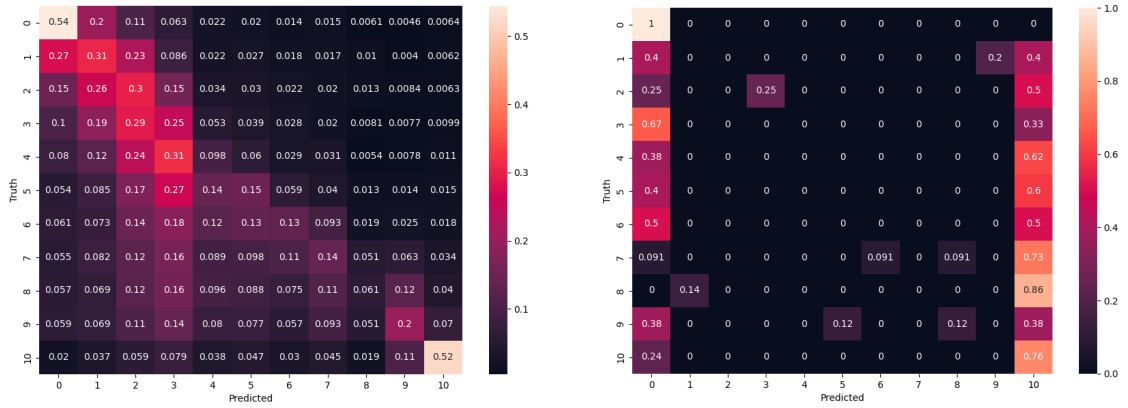



RF prediction confusion matrix on the testing split of the training dataset

RF prediction confusion matrix on the validation dataset

Figure 12: Precision heatmap of RF models trained on the testnet dataset.

The AdaBoost random forest, shown in Figure 11, predicted every class with precision greater than random guessing, shown in Figure 12 when trained on the testnet dataset. However, when the model predicted mainnet transactions, it frequently guessed the newest ring member. We believe the model did not learn the guess-newest heuristic. Instead, this is due to the bias of the training data. Subsequent transactions were sent with only a 20-minute delay, whereas mainnet transactions represent real user activity and could span weeks, months, or years. Since the classi er learned temporal information about a short timespan, it proved to be inadequate to predict on real transactions.




NN prediction confusion matrix on the testing split of the training dataset

NN prediction confusion matrix on the validation dataset

Figure 13: Precision heatmap of NN models trained on the testnet dataset.

The neural network also showed a strong precision when predicting the true spend of the testnet dataset testing-split. However, it is quite apparent that the predictions deviate from the true positive diagonal and are clustered around the surrounding true spend leading to more false positives. While the neural network performed adequately on the testing split, when tasked with predicting real mainnet transactions, it resorted to almost exclusively predicting the rst and last ring members. Due to the black-box nature of neural networks, there is no way to extract feature importance to understand the predictions further.

## 8. Conclusion

The key cryptographic primitive, ring signatures, of the privacy-centric cryptocurrency Monero, has been tested by limited previous works against AI attacks. This work contributed a rst-of-its-kind pipeline to produce de-anonymized datasets of Monero transactions. The two datasets published represent various user spending patterns. One mimics the expected distribution of the blockchain, while the other represents users who spend Monero as fast as possible. Additionally, we explored the e ectiveness of two machine learning models and

one deep learning model in the task to identify the true spend of an arbitrary Monero ring signature, absent external information. Our best performing models achieved accuracies upwards of 34.60% predicting on an out-of-sample subset of the dataset and 13.30% on real mainnet transactions. Compared to the 9% accuracy achieved by random guessing, the Monero protocol, as implemented, has shown signi cant resilience to on-chain information leakage. We hope our open-source datasets and collection pipeline enable future works to test the weaknesses of Monero against various adversarial scenarios.

# 9. Future Work

Train Models with Full Transaction Context. The models trained in this work were given access to information regarding a single ring signature within a transaction. As most transactions are composed of multiple ring signatures, the added context could aid a model in achieving higher accuracies or understanding heuristics such as merging outputs [7, 9]. This could be accomplished with the datasets released.

Wallet Fee Fingerprinting. One of the notable features not enforced at the consensus level is transaction fee amounts per priority level. It is believed [41] that wallets using non-uniform fees create *anonymity puddles*. It is unknown if an unsupervised model could cluster di erent wallets based on the fee-to-kB ratio paid. Di erent wallet implementations of the decoy selection algorithm have been cataloged by the Monero Research Lab [42].

Replay Mainnet Transaction. It is possible to replay real transactions from the main Monero network, including all relevant metadata such as `tx_extra` and similar decoys onto a testing network. This method would replicate real instances of users incorporating plain text messages into their transactions. However, it is unclear what privacy implications this could have for users and thus should rst be evaluated by an ethics board.

# 10. Acknowledgment

I sincerely thank the MAGIC Monero Fund committee members, as this work would not be possible without their funding. I would also like to thank Gingeropolous from the Monero Research Lab for generously providing the computing resources needed, along with Matthew Wright, Isthmus, Rucknium, and Neptune, for their help and guidance.

## 10.1. MAGIC Monero Fund Donations

If you found value in this research and would like to subsidize similar projects surrounding Monero, consider donating to the non-pro t MAGIC Monero Fund, using their advertised methods [43].

## References


[1] S.-F. Sun, M. H. Au, J. K. Liu, and T. H. Yuen, \Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero," in *Computer Security { ESORICS 2017*, S. N. Foley, D. Gollmann, and E. Snekkenes, Eds. Cham: Springer International Publishing, 2017, pp. 456{474.

[2] D. Jevans, \Ciphertrace les two monero cryptocurrency tracing patents," Nov 2020. [Online]. Available: https://ciphertrace.com/ciphertrace- les-two-monero-cryptocurrency-tracing-patents/

[3] \Ciphertrace patents." [Online]. Available: https://patents.google.com/?assignee=Ciphertrace2BInc

[4] \Algorithm battle ares between ciphertrace and monero." [Online]. Available: https://www.monerooutreach.org/news/ciphertrace-monero.html

[5] K. P. Erb, \Irs will pay up to $625,000 if you can crack monero," Jun 2021. [Online]. Available: https://www.forbes.com/sites/kellyphillipserb/2020/09/14/irs-will-pay-up-to-625000-if-you-can-crack-monero-other-privacy-coins/?sh=613592ee85cc

[6] M. Moser, K. Soska, E. Heilman, K. Lee, H. He an, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan *et al.*, \An empirical analysis of traceability in the monero blockchain," *arXiv preprint arXiv:1704.04299*, 2017.

[7] A. Kumar, C. Fischer, S. Tople, and P. Saxena, \A traceability analysis of monero's blockchain," in *European Symposium on Research in Computer Security*. Springer, 2017, pp. 153{173.

[8] S. Vijayakumaran, \Analysis of cryptonote transaction graphs using the dulmage-mendelsohn decomposition," *Cryptology ePrint Archive*, 2021.

[9] C. Ye, C. Ojukwu, A. Hsu, and R. Hu, \Alt-coin traceability," *Cryptology ePrint Archive*, 2020.

[10] N. Borggren, H.-y. Kim, L. Yao, and G. Koplik, \Simulated blockchains for machine learning traceability and transaction values in the monero network," *arXiv preprint arXiv:2001.03937*, 2020.

[11] N. Van Saberhagen, \Cryptonote v 2.0," 2013.

[12] P. J. E. Sarang Noether, \Blog: About supply auditability." [Online]. Available: https://web.getmonero.org/2020/01/17/auditability.html

[13] S. Noether, A. Mackenzie *et al.*, \Ring con dential transactions," *Ledger*, vol. 1, pp. 1{18, 2016.

[14] B. Goodell, S. Noether, and A. Blue, \Concise linkable ring signatures and forgery against adversarial keys," 2019, https://ia.cr/2019/654.

[15] K. M. Alonso *et al.*, \Zero to monero: Second edition," 2020.

[16] S. Nakamoto, \Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[17] V. Buterin *et al.*, \A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, 2014.

[18] T. Kanstren, \Zero knowledge proofs: Example with pedersen commitments in monero," Jun 2021. [Online]. Available: https://medium.com/coinmonks/zero-knowledge-proofs-um-what-a092f0ee9f28

[19] B. Bunz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, \Bulletproofs: Short proofs for con dential transactions and more," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 315{334.

[20] H. Chung, K. Han, C. Ju, M. Kim, and J. H. Seo, \Bulletproofs+: shorter proofs for privacy-enhanced distributed ledger," *Cryptology ePrint Archive*, 2020.

[21] S. Noether, S. Noether, and A. Mackenzie, \A note on chain reactions in traceability in cryptonote 2.0," *Research Bulletin MRL-0001. Monero Research Lab*, vol. 1, pp. 1{8, 2014.

[22] N. Borggren and L. Yao, \Correlations of multi-input monero transactions," *arXiv preprint arXiv:2001.04827*, 2020.

[23] A. Hinteregger and B. Haslhofer, \An empirical analysis of monero cross-chain traceability," *arXiv preprint arXiv:1812.02808*, 2018.

[24] D. Chaum and E. v. Heyst, \Group signatures," in *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 1991, pp. 257{265.

[25] D. J. Bernstein, \Curve25519: new di e-hellman speed records," in *International Workshop on Public Key Cryptography*. Springer, 2006, pp. 207{228.

[26] S. Noether and B. Goodell, \Triptych: logarithmic-sized linkable ring signatures with applications." Springer, 2020, pp. 337{354. [Online]. Available: https://eprint.iacr.org/2020/018.pdf

[27] Koe, \Seraphis: A privacy-preserving transaction protocol abstraction (wip)*," Oct 2021. [Online]. Available: https://github.com/UkoeHB/Seraphis/releases/tag/DRAFT-v0.0.13

[28] A. Mackenzie, S. Noether, and M. C. Team, \Improving obfuscation in the cryptonote protocol," *Monero research lab report MRL-0004*, 2015.

[29] \Breaking monero." [Online]. Available: https://www.monerooutreach.org/breaking-monero/

[30] Monero-Blackball, \Monero-blackball/monero-blackball-site: Site for sharing monero blackball information and resources." [Online]. Available: https://github.com/monero-blackball/monero-blackball-site

[31] A. Hinteregger, \Monero: Public keys of spent txos," Jul 2018. [Online]. Available: https://zenodo.org/record/1304033

[32] Ack-J, \Ack-j/monero-dataset-pipeline: A pipeline that automates the creation and transaction of monero wallets used to collect a dataset suitable for supervised learning applications." [Online]. Available: https://github.com/ACK-J/Monero-Dataset-Pipeline

[33] Moneroexamples, \Moneroexamples/private-testnet: Setting private monero testnet network." [Online]. Available: https://github.com/moneroexamples/private-testnet

[34] Ack-J, \Ack-j/monero-transaction-fee-dataset: This script queries the monero block explorer api to retrieve transaction fee information for a speci ed amount of txs and saves the data to disk." [Online]. Available: https://github.com/ACK-J/Monero-Transaction-Fee-Dataset

[35] Moneroexamples, \Moneroexamples/transactions-export: Searches blockchain for your outputs and ring members using given address and view key." [Online]. Available: https://github.com/moneroexamples/transactions-export

[36] | |, \Onion monero blockchain explorer." [Online]. Available: https://github.com/moneroexamples/onion-monero-blockchain-explorer

[37] Monero-Project, \Chunk /gettransactions to avoid hitting restricted rpc limit by tobtoht · pull request 8388 · monero-project/monero." [Online]. Available: https://github.com/monero-project/monero/pull/8388

[38] Neptuneresearch, \Neptuneresearch/ring-membership-sql: Ring membership sql is a set of materialized views for postgresql that can be used to show ring member relationships between monero transactions." [Online]. Available: https://github.com/neptuneresearch/ring-membership-sql

[39] Y. Freund, R. Schapire, and N. Abe, \A short introduction to boosting," *Journal-Japanese Society For Arti cial Intelligence*, vol. 14, no. 771-780, p. 1612, 1999.

[40] Ray-Project, \A drop-in replacement for scikit-learn's grdsearchcv." [Online]. Available: https://github.com/ray-project/tune-sklearn

[41] \Monerokon 2019 - visualizing monero: A gure is worth a thousand logs," Jun 2019. [Online]. Available: https://www.youtube.com/watch?v=XIrqyxU3k5Q

[42] Monero-Project, \Catalogue of monero decoy selection algorithms · issue 99 · monero-project/research-lab." [Online]. Available: https://github.com/monero-project/research-lab/issues/99

[43] M. G. Board, \Magic monero fund." [Online]. Available: https://magicgrants.org/funds/monero/

# Appendix

## A. Exhaustive Dataset Fields of Transaction Metadata

| | | |
|---|---|---|
| Transaction Metadata | Block_Number | The block position within the chain containing the transaction. |
| | Direction | The direction with respect to the sender's crypto wallet. (Ex. \in", \out") |
| | Block_Timestamp | The timestamp reported by the miner when the block was created. |
| | Block_Timestamp_Epoch | The block timestamp converted into epoch (Unix) time. |
| | *Amount*[1] | The amount of Monero sent in the transaction. |
| | *Wallet_Balance*[1] | The amount of Monero held by the wallet at the time of sending the transaction. |
| | Tx_Fee | The amount of Monero paid to the miners for the transaction to be processed. |
| | *Destination_Address*[1] | The wallet address which the Monero is being sent to. |
| | *Sender_Address*[1] | The wallet address where the Monero is being sent from. |
| | Network | The broadcasted network. (Ex. \mainnet",\testnet",\stagenet") |
| | Tx_Version | The Monero transaction version. |
| | Tx_Public_Key | The public key of the transaction, commonly stored in the tx_extra eld. |
| | *Output_Pub_Key*[2] | The public key of the output is returned to the sender's wallet as \change." |
| | *Output_Key_Img*[2] | The key image of the output returned to the sender's wallet as \change." |
| | *Out_idx*[2] | The index identifying the output returned to the sender's wallet as \change." |
| | *Wallet_Output_Number_Spent*[1] | The index of the TXO within the sender's wallet before being sent. |
| | Tx_Size | The size of the transaction in bytes. |
| | Tx_Fee_Per_Byte | The transaction fee divided by the transaction size. |
| | Num_Con rmations | The number of blocks mined after the transaction was added to the chain. |
| | Time_Of_Enrichment | The epoch time when create_dataset.py was run. |
| | Tx_Extra | A eld which contains public keys and arbitrary data. |
| | Tx_Extra_Length | The number of characters used in the Tx_Extra eld. |
| | Ring_CT_Type | The RingCT signature version. |
| | Payment_ID | A (Deprecated) 32 byte ID assigned by merchants or exchanges. |
| | Payment_ID8 | An encrypted version of the payment ID. |
| | Total_Block_Tx_Fees | The sum of all transaction fees within the same block. |
| | Block_Size | The cumulative size of the block in bytes. |
| | Time_Since_Last_Block | The epoch time di erence between the current block and the prior block. |
| | Num_Inputs | The number of inputs used for the transaction. |
| | Num_Outputs | The number of outputs used for the transaction. |

Table 3: Generic transaction metadata dataset elds.

[1] Private data only known by the sender, not by a passive blockchain observer.
[2] Public data viewable by everyone, except the knowledge of which output was returned to the sender.

## B. Exhaustive Dataset Fields of Input and Output Metadata

| | | |
|---|---|---|
| Input Metadata | Amount | Input amount is a legacy field for pre-RingCT transactions. (Always set to 0) |
| | Key_Image | A one-way function of the private key to prevent double-spends. |
| | Ring_Member ! block_no | The block number from where the ring member originated. |
| | Ring_Member ! public_key | The stealth address of the ring member. |
| | Ring_Member ! tx_hash | The transaction hash from where the ring member originated. |
| | Previous_Tx_Num_Output | A dictionary containing the number of outputs in the previous transaction for each ring member. |
| | Previous_Tx_Num_Input | A dictionary containing the number of inputs in the previous transaction for each ring member. |
| | Previous_Tx_Time_Delta | A dictionary of time deltas of the current block time and each ring member's block time. |
| | Previous_Tx_Block_Num_Delta | A dictionary of block deltas of the current block and each ring member's block number. |
| | Previous_Tx_TxExtra_Len | A dictionary of the TxExtra length for each ring member's previous transaction. |
| | Time_Deltas_Between_Ring_Members | A dictionary of the time deltas between each subsequent ring member. |
| | Total_Ring_Time_Span | The difference in time between the newest and oldest ring members. |
| | Time_Delta_From_Newest_Ring_To_Block | The time between the current transaction and the newest ring member. |
| | Time_Delta_From_Oldest_Ring_To_Block | The time between the current transaction and the oldest ring member. |
| | Mean_Ring_Time | The average time of the ring members. |
| | Median_Ring_Time | The median time of the ring members. |
| | *Ring_no/Ring_size*[1] | The position of the true spend within the ring signature. |
| Input Decoys | On_Chain_Decoy_Block_Deltas | A dictionary of block deltas between each occurrence of the ring member used on-chain from its creation until the block containing the transaction in question. |
| | Number_Of_On_Chain_Decoys | The number of times the ring member's stealth address was included in other transactions between its creation and the block containing the transaction in question. |
| Output Metadata | Amount | A legacy field used in pre-ringCT transactions. (Always set to 0) |
| | Stealth_Address | A one-time address derived from the recipients public key. |

Table 4: Transaction input and output metadata field descriptions.

## C. Unique Fields of Exported CSV Files

| xmr_report.csv | xmr2csv_start_time.csv | xmr_report_ring_members.csv | cli_export.csv |
|---|---|---|---|
| *Tx_Version* | *xmr2csv_Data_Collection_Time* | *Stealth_Address* | *Direction* |
| Tx_Public_Key | xmr_report_ring_members_freq.csv | *Output_Pub_Key* | *Amount* |
| *Output_Pub_Key* | *Ring_Member_Freq* | *Block_Number* | *Wallet_Balance* |
| *Output_Key_Img* | xmr_report_outgoing_txs.csv | *Block_Timestamp* | *Tx_Fee* |
| *Out_idx* | *Ring_no/Ring_size* | *Key_image* | *Destination_Address* |
| *Wallet_Output_Number_Spent* | | *Tx_Hash* | *Block_Timestamp_Epoch* |
| | | *Ring_no/Ring_size* | |

Table 5: The unique fields for each of the exported CSV files.

# D. Dataset Features for Machine and Deep Learning

| | | |
|---|---|---|
| Transaction Metadata | Tx_Fee | The amount of Monero paid to the miners for the transaction to be processed. |
| | Tx_Size | The size of the transaction in bytes. |
| | Tx_Fee_Per_Byte | The transaction fee divided by the transaction size. |
| | Tx_Extra_Length | The number of characters used in the Tx_Extra eld. |
| | Total_Block_Tx_Fees | The sum of all transaction fees within the same block. |
| | Block_Size | The cumulative size of the block in bytes. |
| | Time_Since_Last_Block | The epoch time di erence between the current block and the prior block. |
| | Num_Inputs | The number of inputs used for the transaction. |
| | Num_Outputs | The number of outputs used for the transaction. |
| Input Metadata | Previous_Tx_Num_Output | A dictionary containing the number of outputs in the previous transaction for each ring member. |
| | Previous_Tx_Num_Input | A dictionary containing the number of inputs in the previous transaction for each ring member. |
| | Previous_Tx_Time_Delta | A dictionary of time deltas of the current block time and each ring member's block time. |
| | Previous_Tx_Block_Num_Delta | A dictionary of block deltas of the current block and each ring member's block number. |
| | Previous_Tx_TxExtra_Len | A dictionary of the TxExtra length for each ring member's previous transaction. |
| | Time_Deltas_Between_Ring_Members | A dictionary of the time deltas between each subsequent ring member. |
| | Total_Ring_Time_Span | The di erence in time between the newest and oldest ring members. |
| | Time_Delta_From_Newest_Ring_To_Block | The time between the current transaction and the newest ring member. |
| | Time_Delta_From_Oldest_Ring_To_Block | The time between the current transaction and the oldest ring member. |
| | Mean_Ring_Time | The average time of the ring members. |
| | Median_Ring_Time | The median time of the ring members. |
| Input Decoys | On_Chain_Decoy_Block_Deltas | A dictionary of block deltas between each occurrence of the ring member used on-chain from its creation until the block containing the transaction in question. |
| | Number_Of_On_Chain_Decoys | The number of times the ring member's stealth address was included in other transactions between its creation and the block containing the transaction in question. |

Table 6: Dataset features used for machine and deep learning.