[Draft v0.2] Heuristic framework for generalized transaction tree analysis

August 17, 2022

Contents

to-do: intro, references, SQL implementation, figures, esploristo warnings

1 Notation

Let x refer to an output on the blockchain. Let t refer to a transaction on the blockchain. Define $T(x)$ to indicate the transaction that created the output x. Let z refer to the ring size (currently 11) and let $|I(t)|$ be the number of inputs (i.e. number of ring signatures) for transaction t, and $|O(t)|$ be the number of outputs created by transaction t. For convenience, in this draft we will discard structure for which ring members belong to which ring signatures, and throw them all into one set $R(t)$ with $k = z * |I(t)|$ elements indexed by any arbitrary deterministic order (e.g. sorted ascending by one time "stealth" address value). Defining x_t^i to reference the ith output used as a ring member in transaction t, we see $R(t) = \{x_t^1, x_t^2, x_t^3, ..., x_t^k\} = \{r_1, r_2, r_3, ..., r_k\}$. Whereas $R(t)$ refers to a set of outputs, we define $P(t)$ as the set of transactions that generated those outputs, i.e. $P(t) = \{T(x_t^1), T(x_t^2), T(x_t^3), ..., T(x_t^k)\} = \{p_1, p_2, p_3, ..., p_k\}$

2 Formalizing fungibility defects

Let f refer to some type of fungibility defect, such as an incorrect decoy selection algorithm, juvenile ring members, an unusual unlock time, etc. $B_f(t)$ is a Boolean where the logic for each heuristic is described based on characteristics of the transaction.

> Example: The core wallet always waits 10 blocks for outputs to unlock before including them in transactions. Wallets that ignored this rule and spent outputs too quickly left on-chain evidence evidence of 'juvenile' transactions, which allowed them to be fingerprinted as originating from non-core software.
>
> First, we define $H(t)$ as the height of the block containing the transaction t. To analyze juvenile spending within this framework, define B_f as follows:

$$B_{juv}(t) = \begin{cases} \text{True} & \text{if } \exists x \in R(t) \text{ such that } H(t) - H(T(x)) < 10 \\ \text{False} & \text{otherwise} \end{cases} \tag{1}$$

2.1 Framework for Boolean traits

Now we define a way to keep track of chains in the transaction tree, so we'll use an integer counter for number of transactions in a chain that all have the same defect.

$$C_f^{max}(t) = \begin{cases} 0 & \text{if } B_f(t) \text{ is False} \\ 1 + max(C_f^{max}(P(t))) & \text{otherwise} \end{cases} \tag{2}$$

Where $max(C_f^{max}(P(t)) = max(\{C_f^{max}(p_1), C_f^{max}(p_2), C_f^{max}(p_3))), ...\})$. This is interpreted as follows: Given a particular transaction t', if $C_f(t') = 5$, this indicates that defect f was observed in this transaction t, and was observed in one or more of the (parent) transactions $P(t)$, and observed in *their* parent transactions, and so forth, in an unbroken chain of 5 transactions (4 hops). The selection of max chooses to track the longest defective chain if there are more than 1 ring members satisfying $C_f^{max}(p_i) > 0$. This type of situation ($B_f(t)$ Boolean and $C_f^{max}(t)$ counter) is shown in figure 1.

An analogously constructed C_f^{min} will be useful later

$$C_f^{min}(t) = \begin{cases} 0 & \text{if } B_f(t) \text{ is False} \\ 1 + min(C_f^{min}(P(t))) & \text{otherwise} \end{cases} \tag{3}$$

Naturally, these chains will not *always* represent the actual spend pattern, as there will be some false positives, which can be modeled from the empirical distribution of outputs with $B_f(T(x)) = \text{True}$ and the decoy selection algorithm (at that time). Longer chains are (statistically) less likely to represent false positives so, while a chain with $C_{juv}^{max} = 2$ may not be remarkable, a chain with $C_{juv}^{max} = 15$ is likely to represent the actual spend path (perhaps due to impatient churn, or change address linking).

2.2 Extension for valued traits

The example heuristic in the previous section B_{juv} had a simple True/False outcome - either the wallet waited 10 blocks, or it didn't. Now consider a defect like unlock time that contains any uint64 value, i.e. integer on [0, 18446744073709551615]. Let $U(t)$ be the value of the unlock time field in transaction t. We can make strong and weak counters.

The weak counter resembles the previous section, we'll start with defining B_f, in this case we'll take any non-zero value to be a 'defect'.

$$B_{lock}(t) = \begin{cases} \text{True} & \text{if } U(t) > 0 \\ \text{False} & \text{if } U(t) = 0 \end{cases} \tag{4}$$

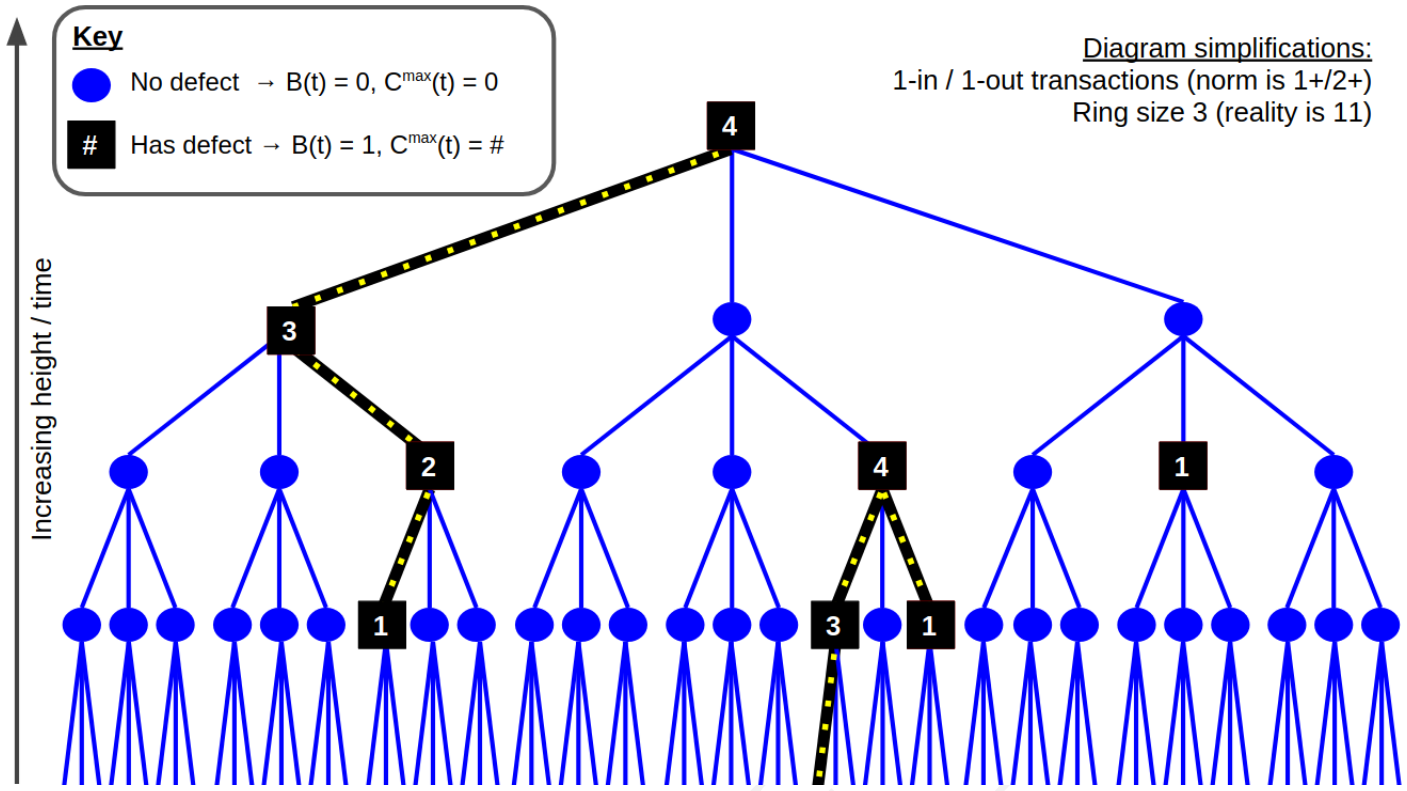

Figure 1: Applying a $B_f(t)$ Boolean value and $C_f^{max}(t)$ counter function to the transaction graph

The counter C_f as described in the previous section will keep track of the length of chained reactions with *any* nonzero lock time (the weak constraint).

$$C_{lock}^{max}(t) = \begin{cases} 0 & \text{if } B_{lock}(t) \text{ is False} \\ 1 + max(C_{lock}^{max}(P(t)) & \text{otherwise} \end{cases} \tag{5}$$

However, we can also build a counter with stronger constraints such as exact matches, by (generally) mapping the transaction to a single value $V_f(t)$. In specific example of unlock time, this value is simply the unlock time, $V_{lock}(t) = U(t)$. Analogous to the weak match counter C_f we will define an exact match counter E_f,

$$E_f^{max}(t) = \begin{cases} 0 & \text{if } \nexists p_i \in P(t) \text{ such that } V_f(p_i) = V_f(t) \\ 1 + max(E_f^{max}(P(t)) & \text{otherwise} \end{cases} \tag{6}$$

Thus if $E_{lock}(t) = 5$, this indicates 5 chained transactions with the exact same value for the unlock time. This type of situation ($V_f(t)$ value and $E_f^{max}(t)$ exact match counter) is shown in figure 2.

Note that there is lots of flexibility in how these constraints are designed. For example, we could define yet another unlock time heuristic ($lkht$) and counter by specifying a specific wallet fingerprint, such as using the unlock time field to represent a block height ($250 \leq U(t) < 500,000,000$). This is accomplished by defining

$$B_{lkht}(t) = \begin{cases} \text{True} & \text{if } 250 \leq U(t) < 500,000,000 \\ \text{False} & \text{otherwise} \end{cases} \tag{7}$$

and using $C_{lkht}^{max}(t)$ as defined in equation 2 as the counter.

3 Statistical analysis

3.1 Defect analysis (B and V)

This section describes general tips and general first steps exploratory data analysis of fungibility defects, examining whether they leak information, and the degree of impact.

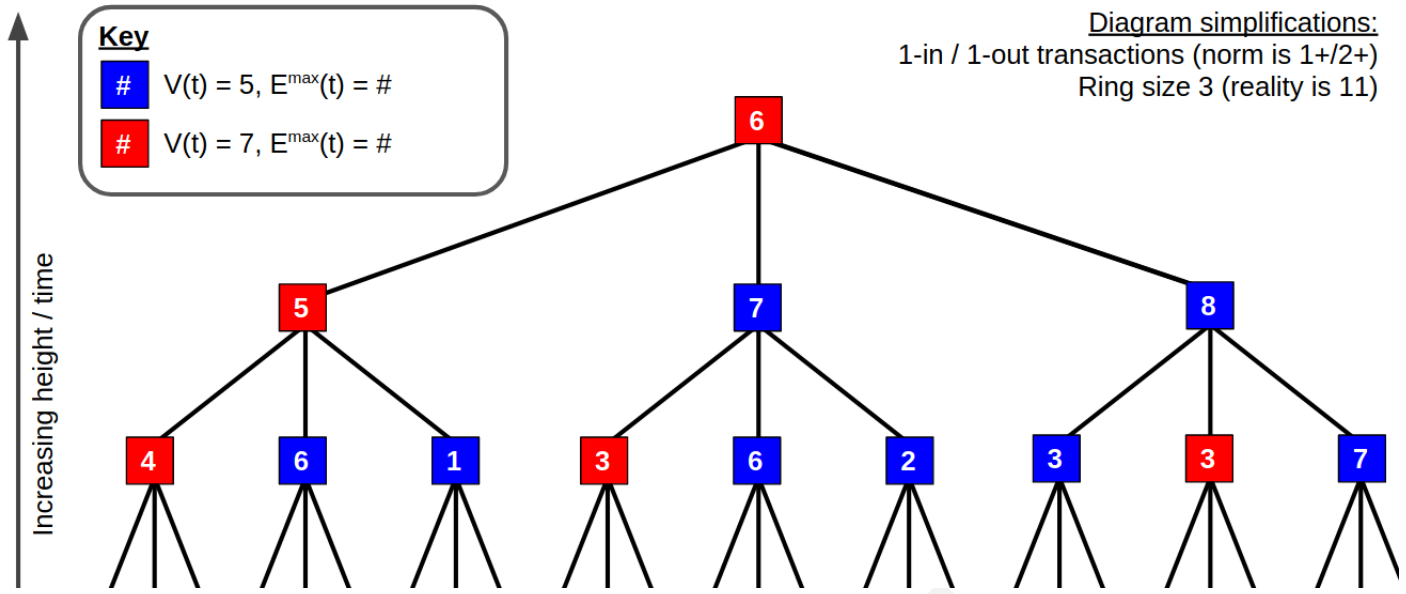

Figure 2: Applying a $V_f(t)$ value and $E_f^{max}(t)$ exact match counter function to the transaction graph.

3.1.1 Studying Boolean defects (B)

Let $< B >$ be the total number of transactions included, and $< B_f >$ be the number of transactions for which $B_f(t) = True$.

First note $< B_f > / < B >$. If 0% or 100% of transactions exhibit the same characteristic, then there is nothing to worry about - the transactions all resemble each other (with respect to this feature) which is the goal for fungibility.

For visualizations, consider a scatter plot showing ($y =$) fraction of transactions exhibiting a characteristic versus ($x =$) time (or height). Note that calculation of the trace's y value could be accomplished either by binning (e.g. by month) or a rolling average over $< B_f > / < B >$.

3.1.2 Studying valued defects (V)

First, plot a histogram of $V_f(t)$ to get a sense of the distribution. Indistinguishability can be achieved if all transactions have the same $V_f(t)$, which appears as a single spike on the histogram. Alternately, fungibility may be achieved if the $V_f(t)$ is uniformly distributed, which appears as a horizontally flat histogram. Distributions/histograms that suggest an information leak include those with multiple peaks (e.g. fees), or a nearly-uniform distribution that exhibits some deviations (e.g. CryptoNote nonces).

To see trends that change over time, consider a heatmap where the x-axis is time (or height), the y-axis is V_f, and the z-axis (or color) showing the count in that bin.

3.2 Studying chains of defects (C and E)

Examine the distribution of chain lengths for any heuristic. We desire a list of Q_f values which describe the lengths of chains observed in the transaction graph (keeping only the longest value from each chain). This Q_f list can be generated from either C_f of E_f (whichever is applicable). For respective examples, note that the set of chain lengths shown in figure 1 is $(4, 4, 1)$, and for 2 is $(3, 3, 6)$.

While some chains of fungibility defects will represent repeated transactions generated by a custom or malfunctioning wallet (consider this a true positive) there will occasionally be chains that arise by statistical chance through serendipitous decoys selection (false positive).

The false positive rate could be determined by asking how statistically likely it is for a given chain (length) to arise from random selection of outputs. However this is not a trivial exercise, since it must take into account both the decoy selection algorithm and the empirical distribution of the defects in the chain history.

For defects that are observed in less than half of the outputs (under weighted consideration by the decoy selection algorithm) the false positive rate will decrease as chain length increases. (In other words, defect chains of length 2 are statistically noisy, but (for example) a chain of length of 20 for a defect with a 2% incidence rate is quite statistically powerful!

Given this, visualizing the distribution (histogram of Q_f values) of chain length provides an intuitive way to estimate a significance threshold for a given fungibility defect. Due to false positives, low values of chain length will contain chance matches, but this should drop off relatively quickly for longer chains. Outliers at high values (long chain lengths) are

statistically likely to represent true spend patterns, often due to change outputs (TXOs) that link many transactions generated by the same wallet.

To see how the statistics around chain length for a given fungibility defect change over time, consider a scatter plot of $(y =)\ Q_f$ values versus $(x =)$ the height of the last transaction in the chain.

4 Comments

4.1 Shape of the data

This framework is designed to be implemented in a relational database, e.g. SQL where each row is a transaction and columns can be added as necessary to calculate and store the flags and counters such as B_f, C_f, V_f, E_f.

4.2 Efficiency

Note that calculating each set of features for the entire blockchain only requires a single $O(N)$ pass from the genesis block up to the current height. No recursion is necessary to calculate chain lengths leading to a given transaction.

4.3 Implications for blockchain analysis

If one is applying probabilistic methods for graph matching on the transaction tree, some efficiency may be gained by initializing the weights based on defect chains where possible (then apply other heuristic(s) for the remaining edges based on the decoy selection algorithms and/or spend time distribution priors).

4.4 Windowed implementation

For some analyses/heuristics, it might make sense to only look at a given period or subset of transactions, for example 'RingCT transactions' or 'Transactions after plaintext payment IDs were deprecated'. This can be handled by defining how C_f and/or E_f handle references to transactions outside the set under study. In most cases, a simple approach is to simply ignore those ring members, i.e. drop them from $P(t)$, though this will not make sense in all contexts.

5 Example Applications

Here we note how various fungibility defects can be formulated in this framework, by defining $B_f(t)$ or $V_f(t)$ which are typically paired with C_f^{max} and E_f^{max} respectively.

5.1 Incorrect decoy selection algorithm.

Note: let $F(H(R(t))) = F(\{H(r_1), H(r_2), H(r_3), ..., H(r_k)\})$ where $F()$ is $max()$ or $median()$.

5.1.1 Uniform or old-weighted decoy selection

Apply some threshold ϵ e.g. 1 week

$$B_{algo}(t) = \begin{cases} \text{True} & \text{if } max(H(R(t))) - median(H(R(t))) > \epsilon \\ \text{False} & \text{otherwise} \end{cases} \tag{8}$$

5.1.2 Juvenile spending

$$B_{juv}(t) = \begin{cases} \text{True} & \text{if } \exists x \in R(t) \text{ such that } H(t) - H(T(x)) < 10 \\ \text{False} & \text{otherwise} \end{cases} \tag{9}$$

5.1.3 Cached ring members

Transactions that reference the same handful of outputs $G = g_1, g_2, ...g_6$. This has been observed in the wild, anecdotally attributed to a particular exchange.

$$B_{cache}(t) = \begin{cases} \text{True} & \text{if } (\forall g \in G) g \in R(t) \\ \text{False} & \text{otherwise} \end{cases} \tag{10}$$

5.2 Unlock time L

5.2.1 Resembling height differences

Note that the Monero protocol does *not* support height differences in the protocol, so this is an incorrect use of the field. Observed low values:

The usage distribution of low values is strange (top row `unlock_time` values, bottom row counts of their occurrence):

1	2	3	4	5	6	10	12	13	15
8141	279	677	266	6	1	297	2600	1	1

$$B(t) = \begin{cases} \text{True} & \text{if } U(t) < 250 \\ \text{False} & \text{if } U(t) = 0 \end{cases} \tag{11}$$

5.2.2 Resembling heights

$$B(t) = \begin{cases} \text{True} & \text{if } 250 \leq U(t) < 500,000,000 \\ \text{False} & \text{if } U(t) = 0 \end{cases} \tag{12}$$

5.2.3 Resembling timestamps

$$B(t) = \begin{cases} \text{True} & \text{if } U(t) \geq 500,000,000 \\ \text{False} & \text{if } U(t) = 0 \end{cases} \tag{13}$$

5.3 Transaction extra contents

Let $X(t)$ be a list of items in the transaction extra field for transaction t, such as plaintext payment identifiers ($pPID$), encrypted payment identifiers ($ePID$), public transaction key(s) (K), etc. Let $\#X(t).item$ resent the number of some item $item$ in the transaction extra payload $X(t)$. Let $X(t).item.data$ represent the data itself (this glosses over the edge case of duplicate items containing different data, such as two non-matching pPIDs in the same transaction.)

5.3.1 Intertransaction duplicates

Unexpectedly and statistically unlikely collisions between different transactions have been located in the wild for the $pPID$, $ePID$, and K fields. Of course chance collisions are possible by chance, but they are statically extremely unlikely (the birthday problem on a 8 byte or 32 byte field) and certainly cannot account for the large number of duplicates observed on the blockchain (included often repeated collisions on the same values). In general, we should identify instances where

$$B(t) = \begin{cases} \text{True} & \text{if } \exists(t') | t \neq t' \wedge X(t).item.data = X(t').item.data \\ \text{False} & \text{otherwise} \end{cases} \tag{14}$$

Checking the transactions pairwise would scale with $O(N^2)$ for number of transactions, but there is a faster approach. Consider a particular $item$ category, such as $pPID$. First, let $J_{item} = \{X(t_1).item.data, X(t_2).item.data, X(t_3).item.data, ..., X(t_m).item.data\}$ represent a list containing data from each of the m transactions the $item$, and let K_{item} be a list of the duplicates in J_{item}. Then our fast way to construct the weak (non-matching) $B(t)$ and $C(t)$ data features is:

$$B(t) = \begin{cases} \text{True} & \text{if } X(t).item.data \in K \\ \text{False} & \text{otherwise} \end{cases} \tag{15}$$

We could also make an exact match filter by declaring $V_{item}(t) = X(t).item.data$ and calculating $E^{max}_{item}(t)$ according to equation 6

5.3.2 Unencrypted payment ID after deprecation

$$B(t) = \begin{cases} \text{True} & \text{if } H(T) > uDepHeight \wedge \#X(t).pPID > 0 \\ \text{False} & \text{otherwise} \end{cases} \tag{16}$$

5.3.3 No payment ID after inclusion on all transactions by core software

$$B(t) = \begin{cases} \text{True} & \text{if } H(T) > eEnfHeight \wedge \#X(t).ePID = 0 \\ \text{False} & \text{otherwise} \end{cases} \tag{17}$$

5.3.4 Payment ID contents

(write this later)

5.3.5 Extra transaction public keys

Surprisingly, this happens, and we have seen $\#K(X(t))) = 1000$ in the wild. May be related to https://hackerone.com/reports/377592

$$B(t) = \begin{cases} \text{True} & \text{if } \#X(t).K > 0 \\ \text{False} & \text{otherwise} \end{cases} \tag{18}$$

5.3.6 Transaction extra ordering

Link to Ukoe issues and related research

5.4 Unusual fees

5.4.1 Outlier fees

Let $F(t)$ be the fee attached to a transaction t and let $\hat{F}(t)$ be the set of plausible fee values that the core wallet may have suggested (at any priority level) in the 1000 blocks leading up to $H(t)$. In this case, the exact match filter on $V(t) = F(t)$ could be useful. The weaker filter for any non-default fee is

$$B(t) = \begin{cases} \text{True} & \text{if } F(t) \notin \hat{F}(t) \\ \text{False} & \text{otherwise} \end{cases} \tag{19}$$

5.4.2 Round fees

Theoretically, we identify round fees by checking for transactions whose fees map to 0 modular some power of ten. The below equation uses XMR as the denomination, however note that the transaction & daemon store amounts as atomic units (each atomic unit corresponds to 10^{-12} XMR)

$$B(t) = \begin{cases} \text{True} & \text{if } F(t) = 0 \, (mod \, 0.001\text{XMR}) \\ \text{False} & \text{otherwise} \end{cases} \tag{20}$$

Since most round fees have been used in multiple transactions, another (perhaps even more reliable) method for fishing out these transactions is checking for fees that have been used more than some threshold, for example showing up in more than 100 transactions.

Taking into account Monero's dynamic fee algorithm, high precision fees, and 10 block lock time, it is very unlikely that long chains of transactions with *exactly* identical fees will arise naturally. Thus we can define $V_{fee}(t) = F(t)$ and then calculating $E^{max}_{item}(t)$ according to equation 6.

5.5 Ring size (has been addressed)

Let $z(t)$ be the ring size of transaction t and let $\hat{z}(t)$ be the minimum ring size allowed by the protocol at the time that t was included in the blockchain. In this case, the exact match filter on $V(t) = z(t)$ could be useful (for example, when MyMonero users frequently fingerprinted themselves with $z = 41$ due to the defaults on the web app). The weaker filter for any non-default ring size is

$$B(t) = \begin{cases} \text{True} & \text{if } z(t) > \hat{z}(t) \\ \text{False} & \text{otherwise} \end{cases} \tag{21}$$

to be matched with C^{max} as described in equation 2.

5.6 Single output transactions (has been addressed)

$$B(t) = \begin{cases} \text{True} & \text{if } |O(t)| = 1 \\ \text{False} & \text{otherwise} \end{cases} \tag{22}$$

and utilize C^{max} as described in equation 2.

5.7 Network analysis (off chain)

If one is surveilling the network, the IP address could be treated as a valued trait. A hacky way to force the data into integer form by including leading 0s, prepending a 1, and retaining only the numerals. For example, in decimal:

$$\begin{aligned} 10.23.45.6 &\mapsto V_{IP}(t) = 1010023045006 \\ 10.234.5.6 &\mapsto V_{IP}(t) = 1010234005006 \end{aligned} \tag{23}$$

(There must be much better ways to do this, and I welcome suggestions.)

In terms of dragnet surveillance, IP analysis is likely to be fraught with false positives. However, it may be effective for targeted surveillance when combined with other intelligence and network monitoring. Let $IP(t)$ be the IP address from which a transaction appears to originate, and let L_{IP} be a list of IP address(es) attributed to some entity.

$$B_{IP}(t) = \begin{cases} \text{True} & \text{if } \exists i \in L_{IP} \text{ such that } i = IP(t) \\ \text{False} & \text{otherwise} \end{cases} \tag{24}$$

Monero's recent Dandelion integration increases the difficulty of the identifying transaction origins, and users concerned about network-layer surveillance can take additional measures to improve their security.

5.8 Age analysis

This is an partially unrelated topic, but the notation described above is convenient to describe it:

Any given transaction and its outputs has many possible histories, which initially grows exponentially the more hops considered - given ring size z we initially expect z^n possible histories looking n rounds deep. Of course, this does not increase indefinitely, as all paths eventually terminate at a coinbase.

Naturally, each possible history starts at some coinbase and ends at t, and we can ask which path is the shortest. We will refer to the length of that path as the 'youngest possible age', which could be measured in hops, height, or time. Here, we show derivation $Y_{hops}(t)$, which is the length of the shortest chain of transactions from all possible histories.

The Boolean B_f simply indicates whether or not the transaction references a coinbase.

$$B_{hops}(t) = \begin{cases} \text{False} & \text{if } \exists x \in R(t) \text{ where } T(x) \text{ is a coinbase} \\ \text{True} & \text{otherwise} \end{cases} \tag{25}$$

Then $Y_{hops}(t)$ is simply $C_f^{min}(t)$ as described in equation 3

$$Y_{hops}(t) = C_{hops}^{min}(t) = \begin{cases} 0 & \text{if } B_{hops}(t) \text{ is False} \\ 1 + min(C_{hops}^{min}(P(t))) & \text{otherwise} \end{cases} \tag{26}$$

This application of a $C^{min}(t)$ style counter for evaluating shortest distance to a coinbase is shown in figure 3.

Youngest possible age in terms of height or days can be calculated in a related manner, and are left as an exercise to the reader.

Looking at the empirical distribution of $Y_{hops}(t)$ values observed on the blockchain in real life, it will become apparent that some transactions have a plausible history with (including) *no* history, if a transaction references a coinbase and thus has $Y_{hops}(t) = 0$. Other transactions that do not reference coinbases have $Y_{hops}(t) > 0$ and are thus the funds guaranteed to have been *received* rather than *mined*. Of course, protocol could enforce 1 coinbase per ring to remove this difference.

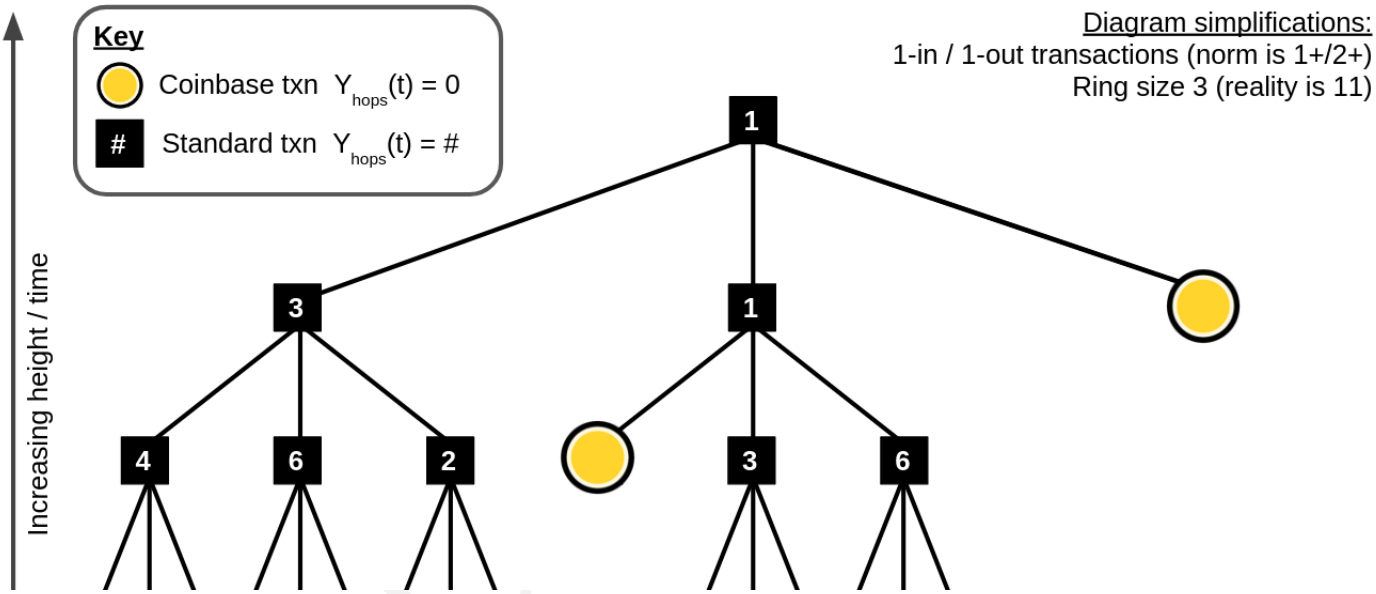

Figure 3: Counting $Y_{hops}(t)$, the shortest path from any given transaction to a coinbase