LineTracer: Finer control in privacy-preserving typology definition

Michael Brand

November 23, 2023

Executive Summary

We describe a new algorithm for the FinTracer privacy-preserving algorithm suite, which we refer to as LineTracer. LineTracer enables matching of typologies that involve complex relationships between the various entities in the typology structure, and can be executed efficiently by leveraging it with SoNaR.

1 Introduction

The FinTracer algorithm is the core of the suite of algorithms we have developed for privacy-preserving intel investigation of financial transactions .

The main problem with the FinTracer algorithm is that it is an algorithm that sacrifices detail in favour of speed. In particular, it provides only the simplest answers to the simplest connectivity questions.

This has led to multiple follow-up algorithms meant to fill in gaps, such as [2, 5, 6].

In [6], we introduced a new privacy model called SoNaR that allows heavier algorithms to once again become applicable.

In this document we describe one such algorithm, LineTracer, which performs an operation similar to that of FinTracer, but in much more complicated scenarios.

Consider the following motivating example, which we will use as a running example throughout this document to show how the many variations of LineTracer can be put to good use.

A classic typology for the Purgles Platform is the "cash-funded overseas money transfer". One example of such a typology is a situation in which some party A_0 receives a cash deposit of some substantial amount x (through one or many deposits). From account A_0, this money, or a substantial portion (say, 90%+ of it) gets transferred to another account A_1 (which may also happen through one or more transfers), and this happens fairly quickly (say, within 24 hours), and this process is then repeated again and again through multiple other accounts, $A_2, \ldots, A_k$, until, finally, A_k (again, within 24 hours of money arriving to it) transmits a substantial portion of the money (again, say 90%+ of it) overseas.

The problem is that FinTracer alone does not provide good means to describe and detect this typology. In this document, we introduce LineTracer and show how it does, and where its advantage is compared with FinTracer.

In Section 2 we explain the basic algorithm. In Section 3 we explain how it can be used in the SoNaR model to powerful effect. In Section 4 and Section 5 we introduce two generalisations over the algorithm that make it even more powerful.

In each section, we demonstrate how LineTracer can be put to use to tackle the cash-funded-IFTI problem, and how each improvement consecutively makes the algorithm's solution of the problem more exact, until it can solve not just the problem as described above, but also more complicated incarnations of it.

2 The algorithm

Consider a two-sided viewable FinTracer propagation graph, G. This graph has as its vertices accounts, and as its directed edges any pair (a, b) matching some criterion of choice that can be ascertained by both RE(a) and RE(b), where RE(a) and RE(b) denote the REs managing accounts a and b, respectively.

In graph theory, the *line graph* of G, denoted $L(G)$, is a graph such that $V(L(G)) = E(G)$ and $E(L(G)) = \{((a, b), (b, c))|(a, b), (b, c) \in E(G)\}$.

The most concise way to describe LineTracer is as a variation of FinTracer that propagates its tags on $L(G)$ rather than on G.

More explicitly:

- LineTracer keeps an ElGamal ciphertext for every edge (a, b) in $E(G)$, where each such ciphertext is stored at RE(b),
- Each RE, R, retains a list of triplets (a, b, c), where RE(b) = R, and where (a, b) and (b, c) are both in $E(G)$, and
- Propagation is done by R propagating the tag from each (a, b) to its respective (b, c), then transmitting the final resulting value to RE(c).[1]

Notably, in this initial description it is not clear why to use LineTracer at all, because FinTracer can perform essentially the same task, and using much less computation and memory resources. (Much of the efficiency of FinTracer comes from the ability to compact communication, as exemplified in [4] in the FTIL scripts `PrepIncomingComp` and `PrepOutgoingComp`. Such compaction does not exist in LineTracer.)

The reason to use LineTracer is that not all triplets of $E(L(G))$ have to be represented. A user defining the graph can choose to keep any subset of them, based on any criteria. This is the LineTracer equivalent of FinTracer's use of one-sided conditions to supplement G's two-sided-viewable definition. However, the flexibility afforded by LineTracer's triplet approach is much greater.

Note 2.1. A potential question is why to actually store the list of triplets and not compute at propagation time which triplets to actually propagate over. Both solutions are feasible. Note, however, that

1. In terms of complexity, tag propagation already requires as much work as there are triplets, so at best this is a trade-off of space for time, at equal rates, and
2. It only really makes sense to use LineTracer when the required triplets are sparse (e.g., at the order of a total of $|E(G)|$ or less). In this case, the amount of space needed is no larger, in terms of orders of magnitude, than what is required by a standard FinTracer run. If the triplets are *very* sparse (e.g., at an order of $|V(G)|$ or less) the dominant space requirements will be for the tags themselves, and not for the triplets, in any case.

To explain the rationale for LineTracer, consider once again the motivating example of cash-funded-IFTIs.

In FinTracer, a nonzero tag indicates an account that is of interest, and we need to define our propagation rules in a way that what is an account of interest after k rounds can be propagated to reveal what is an account of interest after $k+1$ rounds. Our only ability to define the propagation rules is by defining which edges exist in the propagation graph G.

Suppose that a match for our typology exists, where cash is deposited to account A, is then moved to account B, then to C, and from C is then transferred overseas. RE(B) needs to decide,

[1] The triplet (a, b, c) is really just a compact way to store the pair of edges $((a, b), (b, c))$. Later on, we will discuss cases where the way to store edges is necessarily different, and the way to store edge pairs efficiently is necessarily different, but the principle presented here remains.

separately, whether the potential edge (A, B) should be part of G and whether the potential edge (B, C) should be part of G. What could such conditions be?

The only two-sided viewable condition for an edge in this case is to limit to only edges that carry a substantial-enough amount of money over a 24-hour period.

RE(B) can also apply two additional one-sided conditions:

1. For (A, B) to be included in G there must be some date d during which A transferred to B a substantial-enough amount x, *and* within the 24 hours following d between $\frac{9}{10}x$ and x left B to exactly one other account.

2. For (B, C) to be included in G there must be some date d during which B transferred to C a substantial-enough amount x, *and* within the 24 hours preceding d between x and $\frac{10}{9}x$ was transferred into B from exactly one other account.

Using the former one-sided condition alone defines a propagation graph G_1 that allows FinTracer to use outgoing-compacting. Using the latter alone defines a propagation graph G_2 that allows FinTracer to use incoming-compacting. Using both together defines a propagation graph G_{12} that is non-compactable. This means that G_{12} cannot be scaled up efficiently over the entire economy. We can approximate it with, e.g., the quasi-propagation mechanism

$$T[k] = G_1T[k-1] \cap G_2T[k-1],$$

but note that this is not a true propagation: there is no such propagation matrix, and no corresponding inverse-matrix (meaning that the direction of propagation cannot be inverted).

Without LineTracer, even G_{12} (the propagation matrix so heavy to compute that it is impractical to scale up) is a very partial description of what we are looking for. Suppose money flowed from A to B to some D on some date d_1 and then from some E to B to C on some completely unrelated date d_2. Graph G_{12} would still include both the edge (A, B) and the edge (B, C), making it erroneously connect A to C. Even if the A-to-B-to-D dynamic was the movement of deposited cash and even if the E-to-B-to-C dynamic was the movement of funds offshore, the two are in no way related. FinTracer will erroneously declare a match.

LineTracer, however, defines its connections by the relationship between edges, and in this case the relationship between (A, B) and (B, C) can be defined by the condition that on some date d some substantial amount x was moved from A to B, *and* in the following 24 hours an amount between $\frac{9}{10}x$ and x was moved from B to C.

This description eliminates the false match described.

3 LineTracer with SoNaR

The main problem with using an algorithm like LineTracer, or, in fact, any algorithm that is not lightweight like FinTracer, is that it is difficult to scale it up. A compacted FinTracer requires computation and propagation that is proportional in amount to the number of accounts in $V(G)$. LineTracer, by contrast, like an uncompacted FinTracer run, requires communication amounts proportional to $E(G)$, and potentially even more computation, which is beyond what we would like to scale up nationwide.

For this, however, the SoNaR model provides a solution.

Consider first running a pass of regular FinTracer, with G_1, G_2, or some form of intersection of their constraints. While this run does not compute the set of accounts that we are after, it does compute a superset of it. Furthermore, even though the superset computed may still be much too large to be returned to AUSTRAC without this becoming privacy overreach, it is likely to still be much smaller than the grand total number of accounts in the overall economy.

We therefore want to reveal this interim result set, S, to the REs, and then run LineTracer only over the subgraph of G induced by S.

The main problem with doing so is that the induced graph is not a two-sided-viewable graph. If we try to propagate naively across it, either there will not be any saving in run-time complexities,

or there will be information leakage: each RE will receive information about the accounts that are in S at the other REs. This, clearly, is not a type of information leakage that we wish to allow.

To solve the problem, we simply use FinTracer DARK [**?**, 3] propagation, using a hash of the edge identifier (a, b) as our DARK address.

Using this methodology, communication is only related to the total number of edges connecting to those accounts that are in S, rather than to all of $E(G)$. REs still need to send data for edges connecting (a, b) if a is in S and is an account that they, themselves manage, but b is an account not managed by them regarding which they therefore do not know whether it is in S or not, and the same also for the same situation but with a and b reversed, but in total this is still a much smaller amount of computation, and, for reasonable-sized S values, should still remain feasible.

From this point, when considering LineTracer, we will always assume that the user first runs a pass of FinTracer to narrow down the account set, and then uses LineTracer in SoNaR mode, using DARK propagation.

4 LineTracer on a multigraph

The use of LineTracer described in Section 2 eliminates some (and potentially a great many) false matches, but not all of them.

Consider a situation where there was a cash deposit into account A followed by a transfer from A to B, and there was also a transfer from A to B followed by a transfer from B to C, and there was also a transfer from B to C followed by an IFTI. If the two transfers from A to B are the same transfer, and the two transfers from B to C are the same transfer, this is the typology we are looking for, but the three sequences of events listed may have all happened at different dates, involved completely different monetary amounts, and did not even happen in the order presented. In such a case, this would not be an instance of our target typology.

In [**?**, Section 3.6], it is suggested to encode the date, d, by having a different propagation graph, G_d, for each date, and then to perform propagation on each G_d sequentially, thus ensuring that the typology matched happens in the desired chronological order. This is only a partial solution, in that it still does not cover the problem of unrelated events that happen to occur in chronological order, nor does it solve the problem of unrelated monetary quantity. On the other hand, it is a very heavy solution, requiring a full FinTracer run (not just a single propagation) for each date inspected. Even if the time window in question is only a couple of months, this already accumulates to an unreasonably heavy operation.

In LineTracer, we can overcome this difficulty without paying nearly as much extra cost in performance by considering G as a multigraph rather than as a graph. Instead of having only one edge from a to b, we can have one per date d in which a substantial transfer of funds from a to b occurred. In terms of LineTracer's edge-pair propagation rules, each such (a, b, d) is distinct, and has its own propagation conditions. In such an implementation, the extra cost for searching over a longer time window is no longer related to the raw time-period over which computation needs to happen, but rather solely to the increase in the number of interesting transaction-date combinations the algorithm needs to analyse.

To exemplify this in a more straightforward scenario, assume, for the moment, that instead of looking for total transfers from a to b over 24 hours, we are now interested in matching this typology only for cases where the transfer from a to b and from b to c was done in one go.

Now, we can have a graph G that has an edge per interesting transaction (i.e., transaction over a predefined monetary amount), and each such edge has its own, unique ID. Suppose that α and β are two such IDs, then the propagation rule of $L(G)$ is a sequence of (α, β) pairs (replacing the original triplet structure). Here, having (α, β) as a propagation rule means not only that the destination of α is the source of β, but also that if α transferred x dollars at time t then β transfers between $\frac{9}{10}x$ and x dollars at a time between t and $t + 1$ day.

Running LineTracer now, over the multigraph, for a single propagation step, will discover only exactly those transactions moving funds from a to b to c that exactly match our entire typology.

Obviously, running over a multigraph is even more costly than running standard LineTracer. One possible solution for this is to use another iteration of SoNaR: we begin with a run of FinTracer, then, using the narrowing down of the potential account set provided by FinTracer, we run a pass of standard LineTracer. Then, we communicate to the REs the still-suspicious accounts after the LineTracer run, and perform on the result a multi-graph LineTracer (again, using DARK propagation).

What we cannot do, unfortunately, is use the result from the first LineTracer in order to communicate to the REs which inter-RE transactions are not relevant, because such communication will leak inter-RE information. This is analogous to what the situation was when communicating the results of the initial FinTracer run, and in both cases what needs to be communicated back to the REs is the still-relevant account set only. An account is defined as relevant if it is a party of at least one transaction that is "of interest". This can be computed easily by summing together the tag values associated with all transactions that each individual account is party to. This computes the logical OR of the tag values associated with these transactions (each corresponding to whether the respective transaction is considered to be "of interest", individually).

5 Non-linear LineTracer

Consider, as a final complication to our running example, that we also want the use-case to handle the consolidation of funds from multiple accounts. This is to say that we allow funds to B to arrive from both A (say, $\$x$) and from A' (say, $\$y$), as long as within 24 hours all funds arrive and are dispatched from B to C. The only criterion is that both x and y satisfy our criteria of being "substantial enough".

This typology cannot be caught by standard LineTracer, because the relationship considered is not just between two edges. Standard LineTracer can handle two-edge relationships, as well as a logical OR between multiple incoming edges and one outgoing edge (because this is a linear relationship). It cannot, however, handle non-linear relationships like the AND between the (A, B) edge and the (A', B) edge.

To address nonlinearity, we can use a non-linear variation of the LineTracer algorithm, as follows.

For each outgoing edge (B, C), we can define a logical operation based on its incoming edges, to determine whether it is of interest. The way to compute this is to gather the value of all incoming edges, then use the nonlinear computation mechanism of [1, 5] in order to compute the result. Note that this mechanism does not require that the same Boolean operation be applied to each output, nor even that the number of inputs is the same for each output. However, when actually implementing such an algorithm one should consider how to define such a per-edge operation efficiently, and how to execute it efficiently.

This operation, because it requires the computation of a nonlinear function after each propagation, is necessarily even heavier and slower than the previous algorithmic variations. Again, SoNaR can be used in exactly the same way, narrowing down the set of accounts that are of interest before applying heavier algorithms.

Note 5.1. The one scenario for which LineTracer does not provide a solution, and for which we currently have no algorithm, is the scenario of funds being split. For example, B may forward funds to both C and C', in amounts z and w totalling $x+y$, and each of C and C' may subsequently transmit these funds off-shore.

Our closest approximation to this, presently, is similar to what we did with FinTracer, initially, in our G_1 and G_2 construction. We can search for two partial typologies: the first, as described in Section 5, that labels both the z and the w transfers as interesting, if both incoming edges transferring x and y are deemed interesting, and then a second partial typology working in the opposite direction, making x and y as both interesting if both z and w are deemed interesting. The intersection of the interesting edges under both these partial typologies is an approximate solution for the matching of the full typology in question.

References

[1] Brand, M. (2020), Improved differential privacy for FinTracer Boolean queries. (Available as: `boolean_queries.pdf`)

[2] Brand, M. (2020), Massively parallel FinTracer path finding. (Available as: `path_finding.pdf`)

[3] Brand, M. (2021), Extreme DARK: A more secure DARK variant. (Available as: `extreme.pdf`)

[4] Brand, M. (2022), *The Complete, Authorised and Definitive FinTracer Compendium (Volume 1, Part 1, 3rd Revision [a.k.a. "Purgles" edition]): A work in progress.* (Available as: `FT_combined.pdf`)

[5] Brand, M. (2022), Some incremental progress in FinTracer technology. (Available as: `small_updates.pdf`)

[6] Brand, M. (2022), Fast privacy-preserving edge-finding, SoNaR, and their uses in FinTracer pair problems. (Available as: `edge_finding.pdf`)