Massively parallel FinTracer path finding

Michael Brand

November 23, 2023

Executive Summary

We describe a new algorithm for determining all minimum length paths from all source accounts to all destination accounts in a FinTracer pairwise run.

1 Introduction

Let G be a directed graph. We will refer to it as the *propagation graph*, and its vertex set as the *account set*, V.

The FinTracer algorithm, was initially described as a privacy-preserving tool that, given a *source* account set $A \subseteq V$, a *destination* account set $B \subseteq V$ and a natural k, it answers the question which $B' \subseteq B$ accounts can be reached from any account in A by a path of length at most k along the propagation graph G.

We have shown that this basic building block can also be used in other ways in order to determine other properties. For example, the algorithm was able to find $d_A(b)$, the *distance* of an account $b \in B'$ from the set A, which is the minimal length for a path from any $a \in A$ to b.

Similarly, the document detailed an algorithm to determine for such a vertex b all minimum length paths from A to b.

Focusing now on this problem of path finding, which is a problem of significant practical importance in financial crime investigation, the down-side of the FinTracer path-finding algorithm is that it is quite slow, determining each vertex along each path in order, one by one. If there are many paths to b, each has to be explored individually. If one wants to discover all minimum-length paths from A to any vertex in B', these have to be found individually.

Furthermore, the algorithm requires every account to remember not just its own full history of tag values, but also each tag value ever used to update it and which other account the update came from. This bloats the memory needed per account, clashes with attempts to reduce the total number of such updates by collating many together, and may require a special preparatory step in which old updates get re-sent.

This document introduces a new path-finding algorithm that finds in parallel and with no additional restrictions on the propagation graph or on the manner of propagation, for every $b \in B'$, every minimum length path from the set A to b.

2 Background and terminology

The basic FinTracer algorithm is based on the idea that we can associate to each account in V an encrypted "tag value".

Conceptually, consider all accounts in V as though they were arranged in some order, so that each account, $x \in V$, is associated with a vector position $\tilde{x}$. A *tag*, in this representation, is a vector, v, such that the vector position $v[\tilde{x}]$ contains the encrypted *tag value* of x, which is a semi-homomorphically encrypted value. Its ciphertext is known only to $R(x)$, the RE managing x; its plaintext value is not necessarily known to anyone. (The decryption key is held exclusively by the FIU, but the FIU does not normally have access to the encrypted value so cannot decipher the information.)

The FinTracer algorithm is an algorithm allowing us to begin with any tag, v, and compute from it Mv, where M is an adjacency matrix computed from a propagation graph. The only condition M needs to satisfy is that whether $M[\tilde{y}, \tilde{x}] = 1$, which is to say whether in the propagation graph there is an edge from x to y, is a fact known to both $R(x)$ and $R(y)$.

Let $\hat{w}$ be the Boolean vector whose element $\hat{w}[\tilde{x}]$ is True if and only if $w[\tilde{x}]$ is nonzero. We refer to such $\hat{w}$ as *reduced tags*.

A separate process to the calculation of $w \leftarrow Mv$ allows also privacy-preserving computation of any set of elementwise Boolean functions on reduced tags. The method to do this is described in [3].

Importantly, these are all performed without revealing essentially any information to the FIU. The FIU can request, at any time, to retrieve the value of any reduced tag, but will only do so when there is a legitimate legislative reason for it. It can also request to retrieve only certain rows of a reduced tag, such as only those corresponding to accounts in the destination set B. We use $v[\tilde{B}]$ to denote the tag v restricted only to the rows related to accounts in B.

Note 2.1. Though we may say that $v[\tilde{B}]$ is communicated to the FIU, in fact whenever any tag value is communicated to the FIU it is first multiplied by a randomly-chosen nonzero group member, selected from the uniform distribution over these. This is a process we refer to as *sanitisation*. It ensures that no information is communicated to the FIU other than the information of $\hat{v}[\tilde{B}]$, i.e. the associated Boolean values of the tag. Tags received *from* the FIU are similarly sanitised upon receipt.

Once such $w \leftarrow Mv$ propagation is possible, it is also possible to propagate tags along adjacency matrices, N, that are not immediately tied to a propagation graph. In [1], for example, it is noted that if we perform $w \leftarrow Mv$ followed by $u \leftarrow Mw$, then we effectively propagated along an adjacency matrix $N = M^2$. It is also possible in this way to propagate along the product of two matrices, as well as along M^k. Similarly, if we perform $w \leftarrow M_1 v$ and $u \leftarrow M_2 v$, then propagating from v to $w + u$ is effectively propagating along the sum $N = M_1 + M_2$.

Consider, again, the problem of propagating a tag, v, k steps along M from a source account set, A, to a destination account set B, of which only the accounts in B' can actually be reached in k steps.

If v is a tag such that $v[\tilde{x}]$ is nonzero if and only if $x \in A$, the nonzero positions in $M^k v$ correspond to those accounts that can be reached from A by a walk of length *exactly* k. Similarly, the tag $(M^k + M^{k-1} + \cdots + M + I)v$, with I being the identity matrix—a tag that for purposes of zero-vs-nonzero distinction is identical to $(M + I)^k v$—is the tag whose reduced value determines which accounts are at distance *at most* k from any account in A.

In research over the past year, multiple new algorithms were designed that allow these basic building blocks to be combined in ways that deliver the answers to other, related, important intel questions. For example, in [2], it was recently demonstrated that if we have a matrix N for which we are able to compute $w \leftarrow Nv$, it is possible to reconstruct the nonzero positions of N by propagating several different v and observing w. Typically, this would be done only for particular rows and columns in N, where legitimate legislative justifications for the query exist.

Effectively, whereas before we were able to retrieve the identity of B', the subset of B for whose vertices $b \in B'$ an edge (a, b) exists such that $a \in A$ and $(a, b) \in N$, we can now tell which elements of A connect to which elements of B'.

We will now show how the FIU can retrieve all minimal length paths from A to B'. In the process, each RE will learn which of their accounts are on these paths. Unless the FIU decides to explicitly hide this information, the algorithm also reveals in which path position each such account appears. We assume that neither the identity of the accounts on the path nor their path positions is a secret from the REs.

3 Retrieving the set of intermediate accounts

The process of determining a path is a two-step process. The first step, described in this section, determines the set of all intermediate accounts on all minimal-length paths, as well as their path positions, but does not return how they are connected. The next step then completes the process by reconstructing the connections.

For example, if in our graph a is connected to b via two paths, one being "$a \to c \to d \to b$" and the other being "$a \to e \to f \to b$", this first step of the process will return the information that b can be reached from a by paths of length 3 which in their first hop go from a to either c or e and in their second hop go to either d or f before reaching b, but the algorithm will not, at this first step, be able to tell whether the paths being traversed are the two described or, for example, "$a \to c \to f \to b$" and "$a \to e \to d \to b$".

The algorithm is as follows. We will assume, for simplicity, that the basic FinTracer pairwise algorithm has already been run in advance, and that the FIU and the REs both already know B'.

1. Let v_0 be the tag that is nonzero at A.
2. For i in the range 1 through k, compute $v_i \leftarrow (M+I)v_{i-1}$.
3. Define a new set of tags, $f_0, \ldots, f_k$ such that $f_i[\tilde{x}]$ is nonzero if and only if $i = d_A(x)$. These can be calculated, simultaneously for all i and in a privacy-preserving manner, using the methods of [3], by describing f_i using the Boolean function $\hat{f}_i = \hat{v}_i \wedge \neg\hat{v}_{i-1}$.
4. Let b_0^i be the tag such that $b_0^i[\tilde{x}]$ is nonzero if and only if $f_i[\tilde{x}]$ is nonzero and $x \in B'$. (This tag can be computed from f_i independently by each RE, because B' is known to the REs.)
5. For each $i : 1 \le i \le k$ and each $j : 1 \le j \le i$, let $b_j^i \leftarrow M^{-1}b_{j-1}^i$.
6. Using the tools of [3] again, we calculate, in parallel, for each $i : 1 \le i \le k$ and each $j : 0 \le j \le i$, the tag m_j^i such that $\hat{m}_j^i = \hat{b}_j^i \wedge \hat{f}_{i-j}$ (noting that the case $j = 0$ has already been previously computed).
7. We can also define m_n as the sum over all m_j^i such that $n = i - j$, which would make $\hat{m}_n$ a disjunction over all such $\hat{m}_j^i$.

The accounts with nonzero tags for m_j^i are accounts that appear on the $i-j$'th step of an i-length path from an account in A to an account in B' that is of distance i from any account in A.

A proof of this claim is straightforward, in that the f_i tags mark the accounts that are at distance i from A, whereas the accounts with nonzero b_j^i are those accounts from which a path of length j leads to an element of B' that is of length i from A. Thus, the intersection between b_j^i and f_{i-j} is the set of accounts that are $i-j$ steps away from A on a walk of length i from A to an account in B' that is at distance i from A. Because the length of the walk equals the distance, the walk must be a minimum-length path.

The accounts with nonzero tags for m_n are therefore the accounts on step n of a path from A to B'.

4 Reconstructing the paths

Once all accounts involved in all paths have been discovered, we can connect them by use of the pair-finding algorithm of [2].

Specifically, we can do this in one of two ways.

If we are very concerned that the FIU does not learn *any* edge in the propagation graph other than those used in the desired paths, we can perform k pair-finding runs. On the i'th run, we find pairs over a single-step propagation on M, where the source accounts are defined by m_{i-1} and the destination accounts by m_i.

These are all edges used in all paths from A to B', because we have essentially reduced the propagation graph into a layer graph, where m_i represents all vertices (accounts) on layer i.

An alternative is to execute just one pair-finding run, where both the source set and the destination set are the disjunction of all $\hat{m}_i$ (representable in a tag as the sum of all m_i).

While this may save on run-time, it will find for the FIU not just the desired edges but also any other potential edges among the m_i sets.

By construction, if there is an edge in the propagation graph from an account x_i for which m_i is nonzero to an account x_j for which m_j is nonzero, it cannot be that $j > i + 1$. The edges where $j = i + 1$ are the ones that form the paths from A to B', so are the edges the FIU is seeking. However, there may still be edges going "backwards", against the progression direction of the embedded layer graph, and they will have $j \leq i$. These edges are not part of any minimal length path, but will still be revealed to the FIU.

In a real-world scenario, an intel investigation finding the accounts on the minimum-length paths from A to B' is likely to immediately proceed to running a radius-1 search around all such accounts, anyway, making the question of this extra information-reveal moot. Note, however, that if this is the case then it is not necessary to run the second part of the algorithm at all: the first part of the algorithm retrieves the accounts. One can, at this point, jump directly to a radius-1 search around these accounts without proceeding to the second part of the algorithm. This search will retrieve, as a subset of its results, all edges that are part of a minimum length path from A to B'.

References

[1] Brand, M. (2022), *The Complete, Authorised and Definitive FinTracer Compendium: A work in progress.* (Available as: `FT_combined.pdf`)

[2] Brand, M. (2020), Efficient connection-finding with FinTracer, AUSTRAC internal. (Available as: `pair_finding.pdf.pdf`)

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