Some incremental progress in FinTracer technology

Michael Brand

November 23, 2023

Executive Summary

We document some of the incremental algorithmic improvements and smaller new algorithms that have been developed during the period July 2021–February 2022, in the FinTracer privacy-preserving algorithm suite.

1 Introduction

This document describes some smaller tweaks to existing algorithms in the FinTracer suite.

2 The problem setup

Let G be a directed graph. The vertices of G, $V(G)$, represent accounts. Each account, x, is associated with a reporting entity (RE), $\mathrm{RE}(x)$.

The FinTracer algorithm suite is a suite of privacy preserving algorithms running in a distributed fashion over multiple computation nodes. One computation node is managed by AUSTRAC, the others—by the various REs.

AUSTRAC can initiate distributed protocols (which we refer to as *queries*), and these are executed in conjunction on all nodes.

At the beginning of the process each RE knows certain properties of the accounts they manage. For example, if we define account sets $A, B \subseteq V(G)$, each $\mathrm{RE}(x)$ will be able to resolve, independently, whether any given x belongs to such a set or not. Also, each of $\mathrm{RE}(x)$ and $\mathrm{RE}(y)$ is able to resolve independently whether $(x, y) \in E(G)$. This is true whether or not $\mathrm{RE}(x) = \mathrm{RE}(y)$.

The aim of each protocol is to deliver to AUSTRAC the information sought by the query, so that AUSTRAC learns as little as possible about G beyond what the query asks explicitly. The query result is typically in the form of a set of accounts or of properties determined for a given account. At the end of the protocol, each $\mathrm{RE}(x)$ must also learn this information about x (i.e. an RE learns if an account they hold is identified as suspicious at the completion of a query), but should learn no additional information (beyond what can be inferred from the information about x).

The above rules, however, are only for *final results*. Many of our protocols can also be used to generate *intermediate results* that can later on be used to compute other queries. Such intermediate results are ideally computed completely obliviously: AUSTRAC must receive as little as possible information, and the REs should receive no information whatsoever.

In terms of their usability, protocols that can be used to compute intermediate results are superior to those that can only compute final results.

In this document, all algorithms generate intermediate results, unless flagged otherwise.

3 Nonlinear functions

One recent addition to the FinTracer algorithm suite that improves almost all our algorithms is the improved ability to compute nonlinear functions. A fast and robust method to compute any

nonlinear Boolean function, as well as any number of such functions simultaneously, was already presented in [1]. The more recent improvement, however, is that we have found better ways of dealing with the addition of differential privacy noise.

In [3, Section 5.4], a better policy for dealing with differential privacy is discussed. FTIL code that implements the new policy is available in [3, Sections 4.3, 4.4, 4.8, 4.9 and 4.10]. In particular, Section 4.8 discusses the general differential privacy policy, Section 4.9 uses it for standard retrieval, and Section 4.10 demonstrates how to perform a negation operation. The algorithms of [1] are all built around computing all nonlinear functions through negations, so the code in Section 4.10 can be used as a simple drag-and-drop replacement for the negation methods used in [1], to create better function-computing algorithms.

Note 3.1. Readers are encouraged to also familiarise themselves with [3, Section 10.1], where the limitations of any use of differential privacy is discussed.

In [3], differential privacy is discussed in terms of a differential privacy *budget* that is agreed in advance with the REs (each RE possibly agreeing to a different budget), and each time differential privacy is needed the user is asked to specify how much of their budget they are willing to spend on the specific use. We add here that a reasonable policy for the management of the differential privacy budget is to set a particular proportion $0 < \alpha < 1$, and in each invocation to spend α of the remaining differential privacy budget. While this is not an optimal policy for a user who knows exactly what their future use of data will be, in real world situations users are unlikely to possess such absolute foresight, and the suggested policy ensures that users will always be able to continue querying any object, although the more an object is queried the higher the differential privacy noise costs that will be incurred in order to use it.

4 Distance finding

Let A and B be two subsets of $V(G)$.

A FinTracer run can tell which accounts in B can be reached from accounts in A by paths on G that are of length at most k. Let B' be this subset of B.

Distance finding is the problem of determining for each $b \in B'$ what is the length of the shortest path to it from A.

Although we have tackled this this problem previously, it was only as a query returning final results: the protocol divulged the distance to both AUSTRAC and the REs. Where such a distance needs to be computed as an intermediate result, the following would be a better approach.

Let $A[i]$ be the subset of $V(G)$ that can be reached after at most i FinTracer iterations (i.e., progression along the edges $E(G)$) beginning with A. Using linear algebra notation for linear operations, we can describe $A[i]$ as $A[i] = (G + I)^i A$. (Multiplying by G indicates propagation along the edges of G. Multiplying by I indicates tags remain where they are. The sum indicates that both possibilities are allowed. In total, this computes what happens when tags are either advanced or not, i times, beginning with A.)

As opposed to linear operations, for nonlinear operations we will use set notation. These operations destroy the original numerical information a FinTracer tag holds about each account, mapping it to only a Boolean: each nonzero value is mapped to "True" and each zero value to "False".

The accounts in B that are at distance i from A exactly are $B[i] = B \cap (A[i] \setminus A[i-1])$, where $B[0] = B \cap A$. In this way, we can create $k+1$ FinTracer tags, $B[0], \ldots, B[k]$, each $B[i]$ signifying those accounts that are at distance i from A, and the process is done entirely obliviously.

5 In-between accounts

Once we know the distance from A to any $b \in B$, a logical follow-up question is: which are the accounts that are on any shortest path from any account in A to any account in B? We refer to these as *in-between accounts*.

In [2], this problem was solved by first partitioning the elements of B based on their distance from A as per Section 4, and then, for each distance d and every $i = 0, \ldots, d$ by running a FinTracer search along G^{-1} (i.e. the graph that has an edge (b, a) for every (a, b) in G) starting from $B[d]$.

Using matrix notation, if $G^m A$ is the set of accounts reached after m steps in G from A, and $G^{-m}B[d]$ is the set of accounts reached after m steps in G^{-1} from $B[d]$, then the intersection of $G^{d-i}A$ and $G^{-i}B[d]$ are exactly the set of accounts that are $d - i$ steps from A and i steps from $B[d]$ along a shortest path from A to $B[d]$. Enumerating over all d and all i, we get all in-between accounts on any shortest path from A to B'.

To this we now added the optimisation that the intersection computation should be done using the new method for computing nonlinear Boolean functions.

Note that in order to compute all the above only one forward FinTracer run was needed from A, and only one backwards FinTracer run was needed per each $B[d]$.

The results can be kept as separate M_{id}, but consider the common case where we only want to query $M[i] = \sum_d M_{id}$. When this is the case, we can offer the following additional optimisation, shown in Algorithm 1. (Here and throughout, we continue to use linear algebraic notation for linear operations and set notation for nonlinear operations.)

Algorithm 1 In-between accounts

$A[0] \leftarrow A$ ▷ Accounts at distance exactly i from A.
$A'[0] \leftarrow A$ ▷ All accounts reachable in at most i steps.
$B[0] \leftarrow A \cap B$
for $i = 1, \ldots, k$ **do**
    $A'[i] \leftarrow (G + I)A'[i-1]$
    $A[i] \leftarrow A'[i] \setminus A'[i-1]$
    $B[i] \leftarrow (A'[i] \cap B) \setminus B[i-1]$
end for
$M[k] \leftarrow B[k]$
for $i = 1, \ldots, k$ **do**
    $M[k-i] \leftarrow B[k-i] + (G^{-1}M[k+1-i] \cap A[k-i])$
end for

In this way, with only a single forward and a single backward FinTracer run, one can compute both all distances (where the result is stored in the form of the partition $B[i]$) and all in-between accounts $M[i]$.

6 Accounts on paths of length up to k

The problem described in Section 5, much like what was originally discussed in [2], was about finding shortest paths from A to B, which is a clean, graph-theoretical problem. From the standpoint of intel usefulness, however, it is usually more conducive to find all accounts on paths of length up to some k from A to B, regardless of whether they are shortest paths or not. We cannot solve this problem exactly, but can find all accounts on *walks* of length up to some k from A to B, which we think would be an adequate approximation.

Note 6.1. A *walk* on a graph G is a sequence of vertices, $v_1, v_2, \ldots$, where for each i, (v_i, v_{i+1}) is in $E(G)$. A *path* in a graph is the same thing, but with the added restriction that no vertex may repeat.

This variation on the idea of in-between accounts, which we'll call *k-path accounts*, can be discovered by Algorithm 2, which is a variation on Algorithm 1. The new result will be called $M'[i]$, instead of $M[i]$.

As before, the entire algorithm requires only one forward and one backwards FinTracer run, and all results are computed obliviously, with neither the REs nor AUSTRAC discovering the

Algorithm 2 k-path accounts

$A[0] \leftarrow A$
$A'[0] \leftarrow A$
for $i = 1, \ldots, k$ **do**
    $A'[i] \leftarrow (G + I)A'[i - 1]$
    $A[i] \leftarrow A'[i] \setminus A'[i - 1]$
end for
$B' \leftarrow B \cap A'[k]$
$M'[k] \leftarrow B'$
for $i = 1, \ldots, k$ **do**
    $M'[k - i] \leftarrow (B' + G^{-1}M'[k + 1 - i]) \cap A[k - i])$
end for

identity of the accounts in question. (AUSTRAC gets in the process only an approximate count of the number of accounts in some of the sets. Specifically, when computing $X \cap Y$, AUSTRAC gets an approximation of $|X| + |Y|$, as well as of $|V(G)|$.)

7 Edge counting

Consider the following problem, which we'll refer to as the *connectivity* problem: Given $A, B \subseteq V(G)$, how many elements from A can reach any single $b \in B$?

In terms of intel use, consider a situation where AUSTRAC is aware of a certain set of accounts of interest, A, all suspected to be involved in money laundering, and AUSTRAC tries to answer the question of which business accounts are the most "friendly" to A. We can define B as the set of business accounts of interest, and then find those businesses that appear downstream from many different accounts in A.

In its general form, this is a difficult problem. However, it is easier when all edges in G map directly from A to B. This is a "close-up" version of the connectivity problem, in that we replaced the counting of long-distance connections from A to B with the counting of edges from A to B. We refer to it as the *edge counting* problem.

The close-up version is much simpler. It can be solved as described in Algorithm 3.

Algorithm 3 Edge counting

Initialise A as a FinTracer tag.
Propagate A through a single FinTracer round on G.
Send the resulting tag, reduced to B, to AUSTRAC without sanitisation.

In describing Algorithm 3 we use neither the linear algebra notation previously used for linear operations nor the set notation previously used for nonlinear operations. The reason for this is to highlight the fact that the tag's values are sent to AUSTRAC *without sanitisation*, i.e. they are revealed in their full numerical values, and not just in terms of zero vs. nonzero. This provides AUSTRAC with more information than we usually provide it.

Note 7.1. FinTracer uses additive ElGamal as its underlying encryption mechanism. For this reason, even once the tag is decrypted, AUSTRAC is not really able to just read its numerical values. These will still be Ed25519 group elements. The way to read the values is for AUSTRAC to pre-compute a hashmap, listing all Ed25519 group elements that map to integers in $[0, k]$ for some k. Following this, AUSTRAC will be able to determine what the number is via a single look-up, if it is in the mapped range, and otherwise will be able to conclude that the true number is greater than k.

This problem would not have existed if we had used multiplicative ElGamal for this algorithm instead of additive ElGamal, but do note that multiplicative ElGamal is orders of magnitude slower, and requires orders of magnitude larger ciphertexts.

As described, this algorithm is unsuitable for computing intermediate results, because the results are directly revealed to AUSTRAC.[1]

An even greater problem with it is, however, that one cannot use it if B is not known to AUSTRAC but is, rather, provided to the REs by means of a description. Attempting to use it directly will give AUSTRAC, for example, the full list of accounts in B. In fact, in order to minimise unnecessary information reveal to AUSTRAC, we will mostly be interested in merely reporting, for some threshold value k, only the set of those elements in B that are connected to from more than k elements of A.

Solving this on an implicitly-defined B can be done using Algorithm 4 (which we describe here using our usual notation, because no unsanitised information needs to reach AUSTRAC). Algorithm 4 is also suitable for producing intermediate results.

Algorithm 4 Edge thresholding

1: $X \leftarrow GA$
2: **return** $B \cap X \cap (X - I) \cap (X - 2I) \cap \cdots \cap (X - kI)$

Here, "$X - kI$" means that we reduced the tag's numerical value by k for all accounts in the set. Evaluating the nonlinear expression results in only the set of those accounts in B that can be reached by more than k separate accounts in A being revealed to AUSTRAC. The amount of communication required to compute the nonlinear expression is proportional to $|B|k$.

Notably, the same algorithm can also be used for longer walks, simply by switching G with G^t, where t is the length of the walk sought for. One can also use $(G + I)^t$ for walks of length *up to* t.

However, when using the algorithm for longer walks, the following points must be considered.

1. The algorithm will no longer count the number of accounts in A connecting to a $b \in B$. Rather, it will count the number of *walks* of the chosen lengths leading from A to b. The number of walks is an upper bound on the number of connecting accounts, but can be much greater.

2. The number of walks is far more revealing regarding the topology of G than the number of accounts, so the raw number should not be communicated to AUSTRAC, and even when communicating only a threshold, this should be done judiciously.

3. Once the number of walks to any given account rises above 1, it will tend to increase exponentially with t. If the threshold k is set to a reasonable value, accounts whose counters are above k will tend to store values that are much greater than k. Increasing k will therefore yield diminishing information returns.

4. It is possible to split B into $B[i]$ subsets, as was done in Section 5, and to consider only shortest paths. This may help to some extent to keep the path numbers from exploding, but they still can increase exponentially. When considering shortest paths, however, one is at least guaranteed not to be counting any non-path walk.

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`)

[1] We can, of course, simply not transmit the results to AUSTRAC, but in the FinTracer suite information held obliviously by the REs is almost exclusively in the form of Boolean indicators about individual accounts, as this is the form that makes it usable by downstream algorithms later on.

[3] Brand, M. (2021), *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`)