Oblivious (and non-oblivious) discovery of sparse matches

Michael Brand

November 23, 2023

Executive Summary

A new oblivious algorithm is described, which allows an FIU to obliviously detect which accounts report a FinTracer "match", even when searching among a large number of accounts, given that the total number of matches is limited. This account number retrieval technique requires less total communication than previous non-oblivious, non-sparse techniques. In an appendix, we show how the algorithm can be adapted to deliver the same communication saving also in a non-oblivious scenario.

1 Introduction

The FinTracer algorithm is a privacy-preserving protocol for matching typologies across financial data. It allows an FIU to define a typology in relation to accounts of interest, and then to initiate a computation to find the accounts that match said typology. The computation is a distributed, privacy-preserving algorithm that is run jointly both by the FIU and by the Reporting Entities (REs). The REs' role in the computation is to provide the relevant transactional and account data. The protocol is privacy preserving in the sense that none of the REs discovers any new information during the execution of the algorithm, and the FIU discovers only which accounts have matched the typology. It may be desirable that the algorithm then informs each RE of accounts that they hold which match the typology.

One element that is not kept private by the basic FinTracer protocol is the choice of typology, and specifically the choice of sets of accounts of interest. However, in scenarios of real intel investigations it is often the case that these lists of accounts of interest are privacy-sensitive lists.

To address this gap, two "oblivious" algorithms were previously presented. In [1], an algorithm was described that allows the FIU to specify accounts of interest without the REs learning the identity of these accounts. In [2], an algorithm was described that allows the FIU to determine regarding selected accounts whether or not they match a chosen typology, where this is done, again, without the REs receiving any information regarding which accounts were checked. (As a safety measure, however, the REs do get notified by the algorithm regarding how many accounts have been checked.)

The present document describes yet a third algorithm from this "oblivious" family. It allows the FIU to determine, obliviously, which accounts have matched a typology, in a situation where a typology search was performed over a large number of target accounts, but only very few "wrongdoer" accounts are expected to report matches.

This algorithm is the oblivious equivalent of the FIU scanning the results for matches, and then asking the RE to reveal which account is associated with any given positive match, except that in an oblivious setting the RE will not know which account numbers have been in this way revealed.

2 Privacy guarantees

Before going over the details of the algorithm, let us enumerate the privacy guarantees it is designed to provide. Notably, this algorithm is unusual compared to other algorithms in the FinTracer suite in that some of its guarantees are not categorical, but rather involve soft limits. Specifically, the algorithm guarantees the following.

1. The FIU will not discover the identity of any account that does not match the query, nor the total number of such accounts.
2. The approximate maximum number of items retrievable, N, is a privacy policy parameter known to both the FIU and the REs. The FIU will not be able to retrieve account numbers for matches if the total number of matches significantly exceeds this limit. (The value of N can be set on a per-query basis, or may be a global constant.)
3. The REs will not discover which accounts matched the query.
4. The REs will not discover any information about the true number of matches. (If the number of matches significantly exceeds N, the FIU will not be able to retrieve the account numbers for matches, but the REs will not learn about this.)

3 The algorithm

3.1 The setup

The algorithm can be divided to two parts. First, we encode all account numbers in a (decodable) way that is more conducive to the algorithm. Then, we communicate the account numbers. We separate the discussion of these into two sections.

As was done with all previous oblivious algorithms, all descriptions will assume only a single RE. If the FIU wants to repeat the process several times for multiple REs, it can.

As is the standard case with FinTracer, our assumption is that the RE has a FinTracer tag, t, associated with its accounts, which holds a value, encrypted using ElGamal encryption with a private key known only to the FIU, that is zero if the account is not matched by the target query and non-zero if it is matched.

In this particular algorithm, the FIU merely discovers the identities of the matched accounts, not their actual tag values. However, once the FIU has the account numbers, it can use the algorithm of [2] to retrieve the values if it so desires.

Another standard FinTracer assumption made here is that the tag values are *general*, in the sense that any sums calculated over them by the algorithm will be zero only if the operands are all zero, other than with negligible probability. If an RE has reason to suspect the generality of their tags, they can always sanitise t prior to the start of the algorithm (i.e., multiply each tag value by an independent random nonzero value) but that is a relatively heavy operation so best avoided if not necessary.

3.2 Encoding the account numbers

Australian accounts are typically identified by 9 digits of account number and 6 digits of BSB, although there are only roughly 2500 unique BSBs used. This gives an upper limit of $2.5 \times 10^{12} < 2^{42}$ to the total number of accounts.

It is important at this point to represent the account numbers in as compact a format as possible, because if any potential value (say, a 42-bit value) is equally likely to represent a true account number, this protects against any form of downstream "account number guessing" on the part of an adversarial FIU.

We now map the account numbers, using an invertible mapping, to bit strings of the minimal needed length that are 0/1-balanced (i.e., have the same number of 0 bits as 1 bits).

In order to map 2.5×10^{12} account numbers in this way, the minimal length of the balanced bit-string is $L = 46$ bits (of which 23 will be 0s and 23 will be 1s). Demonstrably, the blow-up in string length required to store an account number in this balanced way is minimal.

We will from this point throughout assume that all account numbers are represented in this balanced way.

Note 3.1. The assumption here, for simplification, is that the FIU searches matches within all of the RE's accounts. In other oblivious algorithm documents, it was always the case that the FIU first communicates a description for some set of accounts $\hat{A}$, and the algorithm runs entirely within $\hat{A}$. This can also be done in this algorithm, lowering L and lowering all costs associated with the algorithm. The details of this are straightforward and are left for the reader.

3.3 Communicating the account numbers

As before, let N be the rough upper bound on the number of accounts the FIU wishes to be able to retrieve. We assume that this is a privacy policy parameter already known to both sides.

The FIU generates a random number r (with enough randomness to be used to choose hash functions as per Algorithm 1), and sends it to the RE.

Once the RE receives r, it sends to the FIU data according to Algorithm 1.

Algorithm 1 Data sent by the RE to the FIU

```
1: Let $C$ be $1 + \log_2 N$.
2: Let $S$ be $N/\log(2)$.                                        ▷ Natural log.
3: Let $H_r^1, \ldots, H_r^C$ be statistically-independent hash functions from account numbers to $[1, S]$,
   chosen based on the random value $r$.
4: for $i \in 1, \ldots, L$ do
5:     for $j \in 1, \ldots, C$ do
6:         Let $v_{ij}[1], \ldots, v_{ij}[S]$ be fresh encrypted zeroes.
7:         for $a$ in accounts do
8:             if the $i$'th balanced-representation bit of $a$ is 1 then
9:                 $v_{ij}[H_r^j(a)] \leftarrow v_{ij}[H_r^j(a)] + t[a]$
10:            end if
11:        end for
12:        The RE sanitises and sends $v_{ij}$ to the FIU.
13:    end for
14: end for
```

3.4 Reconstructing the account numbers

Once all v_{ij} have been received at the FIU, the set of matching accounts can be reconstructed using Algorithm 2. Here, we assume that C, S, L and the H functions have all been previously computed by the FIU, using the same calculation as was done on the RE side.

4 Algorithm analysis

4.1 Correctness

Given that for any non-matched account a, the value of $t[a]$ is zero, its existence does not affect the calculation. One can assume, therefore, that only the n matched accounts are used.

As was previously proved in [1], the structure described ensures that except for a maximum expected 0.5 accounts, each account a of the n matched accounts has a hash function H_r^j, such that $H_r^j(a)$ is unique among all n matched accounts.

For such a j, the i values for which $v_{ij}[H_r^j(a)]$ is nonzero are the positions in which the balanced representation of a is 1, thus allowing us to reconstruct a.

The FIU can tell which v_{*j} values describe an actual a because these and only these will have exactly $L/2$ nonzero values in v_{*j}.

Algorithm 2 FIU reconstruction of account numbers

```
1: $\tilde{A} \leftarrow \{\}$.                                  ▷ The reconstructed account set.
2: for $s \in 1, \ldots, S$ do
3:     for $j \in 1, \ldots, C$ do
4:         Let $a$ be the value whose bit-position $i$ is 1 if and only if $v_{ij}[s]$ is nonzero.
5:         if $a$ has exactly $L/2$ nonzero bits then
6:             Add $a$ to $\tilde{A}$.
7:         end if
8:     end for
9: end for
10: The retrieved set is $\tilde{A}$.
11: Reconstruction is flagged incomplete if there are additional nonzero $v_{ij}[s]$ that are not explained by the elements of $\tilde{A}$.
```

Note 4.1. In Algorithm 1 we've used $C = 1 + \log_2 N$. This ensures that the expected number of unrecovered account numbers is at most 0.5 (noting that the FIU can tell, almost deterministically, whether any accounts remain unrecovered). The 1 in this formula can be replaced by any other constant number c, with the effect that the expected number of unrecovered accounts (and therefore also the probability that any such account exists) is bounded by 2^{-c}.

We are aware that in practice an intel analyst using the system will want to retrieve 100% of the matched accounts. However, we do not advocate a method in which querying is repeated in some way until all accounts have been retrieved because this reveals to the RE information about the matching set. (Given that the RE is aware of the value of r, it is also quite difficult to make the number of rounds satisfy differential privacy constraints.) Instead, the FIU should simply choose c to be high enough so as to guarantee that the probability that any unrecovered accounts exist is negligible.

4.2 Privacy guarantees

In Section 2, we enumerate the desired privacy guarantees of the algorithm. Mostly, these are met in obvious ways. (E.g., the RE receives no feedback after sending the v_{ij} information, so cannot learn anything about which accounts were matched.)

The one requirement that deserves a closer look is our guarantee that the FIU cannot retrieve account numbers if there are significantly more than N matches. Here is what happens in such a scenario.

Suppose that the actual number of matches is $n = \alpha N$. We will show that the FIU's ability to retrieve matches deteriorates rapidly as α increases. At $\alpha = 10$, we will show that it is already the case that the FIU cannot rule out the majority of potential values from being matched accounts.

To show this, consider some arbitrary account number, a, whose $t[a]$ value is zero, in its balanced representation. In order to exclude the possibility that a is a match, the most one can do is look at $v_{*j}[H_r^j(a)]$ for every j and determine whether in each case all $L/2$ of the positions that are 1 bits in a are nonzero values in $v_{*j}[H_r^j(a)]$. Any a value that passes this test cannot be excluded by the FIU.

For each j, the probability that exactly k accounts out of the n are mapped by H_r^j to $H_r^j(a)$ is binomially distributed, but given that S is large, we can approximate it as a Poisson with the same expectation.

For any given k, the expected proportion, among the $L/2$ positions that are 1 for a, which are zeros in v is 0.5^k. In total, this expected proportion is therefore

$$\sum_{k=0}^{\infty} \frac{\left(\frac{n}{S}\right)^k e^{-\frac{n}{S}}}{k!} \times 0.5^k = \frac{e^{-\frac{n}{S}}}{e^{-\frac{n}{2S}}} \times \sum_{k=0}^{\infty} \frac{\left(\frac{n}{2S}\right)^k e^{-\frac{n}{2S}}}{k!} = e^{-\frac{n}{2S}}.$$

The proportion of nonzeroes is therefore $1 - e^{-\frac{n}{2S}}$ in expectation, and the proportion of those that are nonzeroes in all C hashes is in expectation $\left(1 - e^{-\frac{n}{2S}}\right)^C$, making their expected number $\frac{L}{2}\left(1 - e^{-\frac{n}{2S}}\right)^C$.

In the algorithm, we've set $S = N/\log(2)$, so $\frac{n}{2S} = \alpha\frac{\log 2}{2}$, and $1 - e^{-\frac{n}{2S}} = 1 - 2^{-\alpha/2}$.

Substituting in $\alpha = 10$ and $C = 1 + \log_2 N$, we get that the expected proportion of positions that are nonzero in all C hashes is

$$\left(1 - 2^{-\alpha/2}\right)^{1+\log_2 N} = 0.96875^{1+\log_2 N}.$$

For all but unreasonably high N values ($N > 1,867,015$), the result is greater than 0.5. This means that the probability that the number of such nonzero positions is 0 is less than 0.5. Thus, a randomly chosen account number can only be ruled out as a match with probability less than 0.5.

We conclude that if $n \gg N$, the FIU will not be able to effectively use the algorithm.

4.3 Algorithm costs

Let us consider the costs of the algorithm in the context of a particularly heavy usage of it. We will run it over the set of all accounts, A, with $|A| = 2^{25}$ and will use a particularly large N, $N = 3000$.

We consider first the algorithm's computational costs.

In computing the algorithm's computational costs, we assume the bulk of the costs to be those associated with the ElGamal encryption, with all other costs being negligible in comparison.

The costs associated with the ElGamal encryption are two. First, there are the costs of performing CSL sanitisations over the cells of v. Second, there are the costs of performing $CL|A|$ summation operations. Given that $S \ll |A|$, we further assume that the bulk of the computational costs are the summation costs.

We have $CL|A| \approx 2 \times 10^{10}$. According to previous documents, each summation operation takes $0.36\mu s$ for a CPU and $0.0044\mu s$ for a GPU. In total, this very heavy query will require 1.5 minutes of GPU time or 2 hours of CPU time for all its summations.

In terms of communication costs, consider first what a standard, non-oblivious query of this sort requires. In a non-oblivious scenario, the RE simply communicates all tag values to the FIU in a random order, and the FIU replies with the positions corresponding to non-zero tags (matches) so that the RE can reveal the account numbers they relate to. Here, the bulk of communication is the sending of all tag values. We require, namely, $|A| = 2^{25}$ tags to be sent.

Considering, for an order of magnitude assessment, uncompressed (projective) FinTracer tags that are 256 bytes each, this is a total of 8GB.

Compare this now to the oblivious algorithm. Assuming $N = 3000$, the algorithm sends $CSL \approx 2.6 \times 10^6$ tags. Using, again, the uncompressed (projective) representation, the total size sent is roughly 632MB. In other words, by assuming that the tag is sparse, we were able to considerably reduce communication costs, namely to only $CSL/|A|$ of the original amount, which in this particular example is only 7.7% (and is even less for smaller N).

We conclude that the algorithm is practically feasible.

Note 4.2. We considered also variants of this algorithm in which N is determined by a measurement of the actual number of matches, rather than by a policy parameter. However, given that determining the number of matches seems to require much more communication than what is potentially being saved by a lower N, no such variants are presented in this document.

A A non-oblivious sparse algorithm

We have shown that by utilising the sparseness of matches, this oblivious algorithm allows retrieval of account numbers using much less communication than even the non-oblivious, non-sparse algorithm.

To adapt this algorithm to a non-oblivious (but still sparse) scenario, one can make the following modifications.

1. Instead of the FIU choosing r randomly, the RE should choose it as a secret key, thus making the H_r^j hashes into keyed hashes. Optionally, the H functions can be made into encryption functions, so as to enable decryption.
2. If necessary, the value of L should be increased, so that a randomly chosen (balanced) number will have only a very low probability of being a true account number. This provides protection against account number fishing.
3. Once the FIU retrieves the hashed/encrypted matching account numbers, it will communicate these back to the REs, which will, in turn, respond by sending the FIU the associated plaintext account numbers. (If the RE discovers that the FIU has asked about a value that does not correspond to a real account, it should raise an alert and abort the execution. Such an eventuality cannot occur if both sides have followed the protocol correctly.)

References

[1] Brand, M. (2020), Oblivious setting of source accounts in FinTracer. (Available as: `oblivious_a.pdf`)

[2] Brand, M. (2021), Oblivious querying of FinTracer tag values. (Available as: `oblivious_b.pdf`)