Fast, Lightweight, Secure Record Set Intersection Without Homomorphic Encryption

Michael Brand

November 8, 2023

Executive Summary

This document details a new algorithm proposal, not based on homomorphic encryption, to discover the intersection of two entity sets, for use in FIU-to-Reporting-Entity queries. The proposal requires a trusted third party (so is open to collusion attacks) but is otherwise provably secure, under the appropriate assumptions. The third party is only able to determine the size of the intersection, and this, too, has workarounds to mitigate it.

As opposed to previous algorithmic suggestions, this one is extremely lightweight, requiring less than 20GB of communication altogether and being computable in seconds.

1 Background

We are concerned with the problem of secure set intersection between an Entity of Interest (EoI) list held by the Financial Intelligence Unit (FIU), L_F, and a customer list held by a Reporting Entity (RE), L_R. The FIU would like to determine which entities of interest in L_F are also customers of the RE.

For the purposes of measuring actual communication and computation costs in this document, we take $|L_A| = 10,000$ and $|L_R| = 36,000,000$. Both numbers appear to be reasonable upper bounds on the problem. For $|L_R|$ this is also likely to be a typical size, whereas mostly the true size of $|L_A|$ will be in the hundreds. In the proposed algorithm, we will pad the lists with random elements in order to hide their actual size. The sizes used, in all communications and computations are therefore as though the lists were at their maximal sizes. (See Section 4 for detailed descriptions regarding how to perform this padding efficiently.)

Our existing solutions for private set intersection turns out to be not quite lightweight enough to be convenient, and not quite secure enough to be reassuring, and any attempt to strengthen either of these parameters pushes the other into complete infeasibility. This is partly due to the intrinsic heaviness of the basic building block used to provide security in these algorithms, being Homomorphic Encryption (HE)[1]. It therefore became imperative to widen the search for feasible algorithms.

The present algorithmic proposal uses nothing but straightforward, classic cryptography, and nevertheless provides theoretical privacy guarantees and sub-second computing times.

One problem is that because the FIU has multiple roles in the secure set intersection algorithm (e.g., it is both a contributor of a set and a receiver of the set intersection), this gives it too much power, ultimately enabling it to reveal information about the other party's set. Technically, in our HE algorithms, this manifests as follows. The similarity metric we use to gauge the match quality of two records is ultimately a bilinear function between a cryptographically non-secure hash of L_F and a cryptographically non-secure hash of L_R. Because the FIU has full knowledge of L_F, for it the function is a linear function in L_R. Because it has visibility to this computation's results, this gives it a set of linear equations. Solving them, reveals the hash of L_R, leaking information about L_R itself.

[1]The specific HE algorithm under discussion is Brakerski/Fan-Vercauteren (BFV), which is a relatively lightweight HE algorithm. Nevertheless, it remains too heavy for large-scale usage.

To mitigate this problem we (re-)introduce into the computation a trusted third party, or *Linkage Unit*, (LU). The LU does not know either set, and so can compute the similarity metric without this being quite so revealing, and can then communicate back to FIU only the final match data, without this revealing to FIU anything beyond the information it is required and eligible to obtain.

The need for an LU has the intrinsic cost of introducing the potential for collusion attacks. We aim for an algorithm where collusion attacks are the only possible attacks.

We describe a secure set intersection that uses an LU. It is exposed to collusion attacks, but barring collusion attacks it provides strong privacy guarantees. The only information "leakage" in the algorithm is that the size of the intersection is visible to the LU, and even that can be mitigated.

We describe this algorithm in several parts. In Section 2, we first introduce the basic building blocks of this algorithm in the context of an exact match. In Section 3, we then show how to adapt this algorithm to the actual scenario at hand, where matches can be very partial. In Section 4, we describe a further improvement over the algorithm, that hides from the LU information regarding the intersection result size. In Section 5, we analyse the security of this new algorithm.

2 Exact set intersection

2.1 Algorithm description

Let us begin with the following simplified problem: FIU has a list L_F, RE has a list L_R, and we are trying to determine their intersection, in the sense that FIU will ultimately want to know, regarding every element in L_F whether it is also an element of L_R. The problem is simplified in that we require an exact match between elements of L_F and L_R.

The classic solution to this problem is a Bloom filter. This is a solution that affords zero false negatives (which is a great plus in our scenario) and a controllable amount of false positives. Let E be the largest expected number of false positives we are willing to have. For the purpose of later computations, we will take $E = 0.5$. Let us suppose that s and h are, respectively, the Bloom filter size and the hash Hamming weight that guarantee this number of false positives.

The algorithm now proceeds as follows.

Preprocessing: FIU and the RE coordinate between themselves (secretly, without this being exposed to the LU) a cryptographically-secure hash function, H, that can be used to map records to bit-vectors of size s and Hamming weight h (such as, e.g., using a secure hash function into $h \log s$ bits). We will henceforth assume that H maps records into this space uniformly and independently. Coordinating a new H must be re-done on every run of the algorithm, but we refer to it as 'preprocessing' because it can be performed before the identity of either L_R or L_F is known.

Step 1) RE Challenge: The RE computes the Bloom filter of L_R as the logical "OR" of the individual H results of L_R's elements. The RE communicates the result to the LU as a bit-vector of size s.

Step 2) FIU Challenge: FIU computes the H hash of each member of L_F and sends the results to LU. FIU sends this information as a list of "1"-bit positions.

Step 3) Response: The LU determines for each element received from FIU whether it is in the RE's Bloom filter. It returns to FIU a "match"/"no match" result for each element.

2.2 Algorithm costs

Following standard Bloom filter calculations, in order to ensure the desired E, we choose

$$s = \frac{\log(|L_A|/E)}{\log^2 2}|L_R|$$

and

$$h = \log_2(|L_A|/E).$$

Here and everywhere, logarithms are natural logarithms unless specified otherwise.

We will ignore the communication cost of coordinating H between FIU and RE. This is a classic problem, well understood, and known to be lightweight.

Next, RE sends s bits to LU, whereas FIU sends $h|L_A|\log_2 s$. LU's response can be either a bit-string of size $|L_A|$ or a list of the matched indices. Either way, this communication is of negligible size.

The LU computation is standard Bloom filter processing, and is, again, extremely lightweight. All the LU needs to do is to retrieve $h|L_A|$ bit positions in the s-sized vector sent to it by RE and report the results to FIU.

3 Fuzzy set intersection

3.1 Algorithm description

The similarity metric we use is defensible from a statistical and from an intelligence point of view. Essentially, in order to determine whether two records match, we extract multiple features from each record; each feature is then assigned an l_m value, and the overall similarity is computed as the sum of the l_m values corresponding to features that match between the two records. Two records are considered to be a "match", if this similarity exceeds a certain threshold, t.

The value of l_m for each feature is computable independently by both parties (e.g., by use of a reference corpus). It is identical to both parties.

For a given record, let F be the set of features extracted from it, and let $l_m(f)$, for $f \in F$, be the associated l_m values. Similarly, for a set of features, s, let $l_m(s) = \sum_{f \in s} l_m(f)$. To handle fuzzy set intersection, we begin by calculating S, the set of minimal feature-sets whose total l_m weight exceeds t. In a formula:

$$S = \{s \subseteq F | l_m(s) > t, \forall f \in s, l_m(s \setminus \{f\}) \leq t\}.$$

The way the fuzzy set intersection algorithm works is that instead of hashing the original record (or F) into a single Bloom filter entry, it hashes each element of S separately. This increases the effective size of L_F and L_R by a factor equal to the expected $|S|$, but does not have any other impact on the algorithm.

The only other change is that LU no longer reports a "match"/'no-match" to FIU for every signature that has been matched. Rather, it continues to give a single "match"/"no-match" indication per EoI.

3.2 Algorithm costs

Clearly, computing the full costs of the algorithm requires estimating the expected size of S, which we shall name γ. Given that we do not have at this time a relevant reference corpus for which we computed all l_m values, judging the size of γ requires us to make a guesstimate. In particular, we will assume that the minimal sets which will ultimately be in S are not very different to the signature sets presently being used in the p-signature algorithm.

We expect the algorithm to work on records with 7 fields of identify information. We standardise these fields, and for two of them (name and address) tokenize them into words. Assuming that name and address are the most complete fields, the selected features are then the sets of (1) a pair of tokens from the name followed by a pair of sequential tokens from the address, (2) a pair of tokens from the name followed by one of the 5 other fields, (3) a pair of sequential tokens from the address followed by one of the 5 other fields.

Any overly-frequent signatures are then omitted. In order to get an upper bound on γ, however, we will ignore this filtering step.[2]

We estimate the number of unordered token pairs in the name field is 6.44. Let us assume the average address length is 9 tokens (which is an overestimate), and so there are 8 possibilities for sequential address tokens. In total, the number of signatures used by the algorithm is on average no more than $6.44 \times 8 + 6.44 \times 5 + 8 \times 5$, so $\gamma < 124$.

Inputting these numbers into the cost formulas derived in Section 2.2, we get $s \approx 1.37 \times 10^{11}$, this translating to a bit-vector of size 17.1GB to be sent from the RE to the LU. We also get $h < 22$, so the size communicated from FIU to the LU is approximately 126MB. The communication from the LU to FIU is less than 20KB.

The computation at the LU will be to retrieve approximately 27 million bit-positions in the RE's 17.1GB string. Assuming we're using a machine that can store such a string in memory (or are using multiple machines that, jointly, can store it in memory), this retrieval can be performed in sub-second speeds.

(As my laptop only has 16GB of memory, I was not able to run a complete timing experiment, but I simulated the experiment with a 1GB Bloom filter, instead. On a single-thread, single-CPU, 64-bit, 1.8GHz CPU, this experiment concluded in 0.56 seconds.)

4 Securing the intersection size

Because in this algorithm the LU reports to FIU direct match/mismatch information, it has full knowledge of the exact intersection size. In this section we describe a small tweak to the algorithm that allows hiding this value.

We first point out that while we took $|L_A| = 10,000$, in reality the largest FIU lists are more likely to be under $|L'_A| = 300$. To get from $|L'_A|$ to L_F, we pad by (cryptographically secure) random signatures. Note that when FIU communicates these signatures to LU, it gives regarding each signature the information of which EoI it belongs to. The new random signatures will partly be used to pad $|L'_A|$, by adding fake new elements into the set, and partly will add more signatures to existing L'_A elements, so as to hide information regarding how many unknown fields each element has.

For the RE, the situation is similar. While we took $|L_R| = 36,000,000$, the reality is that the list will never be in this exact size. The actual L'_R representing the bank's active client list is almost certainly substantially smaller than the maximal size bound.

In the case of RE, we do not hide L'_R inside L_R by introducing fake signatures, as was the case with L'_A. Rather, we know that the final bit-vector of the Bloom filter behaves as a vector of s i.i.d. Bernoulli random variables with $p = 1/2$. If L'_R is smaller than L_R, we simply add additional "1"s to the Bloom filter in random, uniform and independent positions, until reaching a predetermined total Hamming weight, which is computed beforehand by sampling from the distribution $\mathrm{Bin}(s, 1/2)$.

We now tweak the algorithm to hide the intersection size, as follows.

First, choose a security parameter M, e.g., $M = 300$. Now, when RE fills its Bloom filter, it does not just fill it with $\gamma|L'_R|$ elements plus padding, but rather with $\gamma|L'_R| + M$ elements plus padding. The M new elements are ones that are chosen randomly (and securely), but whose identity is communicated to FIU ahead of time.

Now, FIU can choose $M' \in [0, M]$, and place M' of the pre-communicated signatures as part of the extra signatures it uses to pad its $|L'_A|$ into $|L_A|$. (The number of signatures from the M-sized list used in any given new fake entity in L_F should be distributed according to the same distribution as the number of signature matches in a true entity match.)

[2] This filtering serves two purposes. First, overly-frequent signatures are assumed to be less informative for matching. Second, overly-frequent signatures are prone to algorithmic attacks. In this algorithm, the first problem is handled by the selection of feature sets s to hash using the rule $l_m(s) > t$, and the second—by the use of a Bloom filter, which omits any repetition information.

The result is that the LU no longer sees the set intersection size, but rather the set intersection size plus a number, settable by FIU, that provides an offset of $[0, M]$. Thus, the true number of matches can be any number between that discovered by the LU and M less.

While not as complete a solution as other parts of this algorithm, this does provide a measure of differential privacy to this number, and by changing the value of M the amount of privacy afforded can be set to any desired level.

5 Security analysis

We prodide a full security analysis.

5.1 Honest-but-curious attacks

The RE does not receive any information during the run of this protocol, and FIU only receives the match/no-match information that is the final output of the algorithm. Hence, any honest-but-curious attack must be by the LU.

Given that H is cryptographically secure, uniform, presumed independent, and chosen anew at each run of the protocol, there is no information that the LU can glean by comparing which of FIU's signatures overlap in which bits. The one exception to this rule is in cases where multiple EoIs in L_F share the same signatures. If overly-frequent signatures were allowed in L_F, this would have opened up an opportunity for a frequency-based attack. However, because of our condition "$l_m(s) > t$", signatures are intrinsically unique: they are unique enough to enable confident identification of an individual.

The same is true when adding the RE Bloom filter: the only information gained is where there is an intersection between an L_R signature and an L_F signature. Any of FIU's signatures that exists in the filter will have a full match in all of its h "on" positions, whereas the Hamming weight of the intersection for non-matching signatures will be distributed $\mathrm{Bin}(h, 1/2)$. However, this merely gives away the intersection size, which we've secured separately (See Section 4).

The most information that is leaked in this algorithm is that in expectation E of RE's random challenges will be reported as a match. This gives RE the identity of h bits known to be "1" in the Bloom filter. Given that h is very small (approximately 22 bits) and that s is large (approximately 1.4×10^{11}), it is almost inevitably the case that the h bits discovered do not belong to the same L_R element, and so it remains impossible to discover even those L_R elements that contributed to said bits.

Had it been the case that H is not refreshed from query to query, these 22 bits would have eventually accumulated, but with a new H each time, the information thus divulged is basically unusable.

5.2 Eavesdropping attacks

We are assuming all communications are over secure channels, which only intended recipients are privy to.

5.3 Non-data attacks

The key (e.g., H) exchange between FIU and the RE happens in preprocessing, so is not relevant for non-data attacks. Ignoring that, the only party that participates in more than a single communication is the LU. However, because the LU cannot get any information about the sets being intersected, neither can an attacker analysing its communications.

5.4 Cryptographic weaknesses

The only cryptographic algorithm used here is secure hashing, which is a classic, well-studied and well-understood problem.

5.5 Malicious attacks

Given that both FIU and the RE send to the LU challenge strings in the appropriate amounts and Hamming weights (which LU can easily verify), it does not seem possible for any malicious attacks to be launched. Signatures other than those queried about are all independent of the queried signatures.

It is possible for both RE and FIU to lie about the contents of their lists, but that is a verifiability issue, not an attack. FIU cannot determine the signatures of the RE without checking these signatures specifically, and the RE does not gain any information at all during the entire run of the protocol.

Similarly, the LU can lie in its response to FIU, but that cannot divulge for it any information because it does not receive from FIU any follow-up communications that may potentially carry such information.

5.6 Collusion

The protocol is unprotected against collusion. If the LU can get H by colluding with either other party, it can simply read the other party's signatures, potentially retrieving information about the original sets by a dictionary attack. (Assuming H is cryptographically secure, it remains the case that no attack other than a dictionary attack is possible.)

6 Conclusion

This Bloom-filter-based algorithm appears both feasible and secure under the assumption of no collusion, and is as lightweight as can be hoped, computable in seconds on a single CPU.

It is a reasonable conclusion that for another solution to out-do this algorithm, it will have to be a two-party protocol: excluding the risk of collusion there isn't much room for improvement over what this solution already affords.