Efficient Private Matching and Set Intersection

Michael J. Freedman[1⋆], Kobbi Nissim[2⋆⋆], and Benny Pinkas[3]

[1] New York University (mfreed@cs.nyu.edu)
[2] Microsoft Research SVC (kobbi@microsoft.com)
[3] HP Labs (benny.pinkas@hp.com)

Abstract. We consider the problem of computing the intersection of private datasets of two parties, where the datasets contain lists of elements taken from a large domain. This problem has many applications for online collaboration. We present protocols, based on the use of homomorphic encryption and balanced hashing, for both semi-honest and malicious environments. For lists of length k, we obtain $O(k)$ communication overhead and $O(k \ln \ln k)$ computation. The protocol for the semi-honest environment is secure in the standard model, while the protocol for the malicious environment is secure in the random oracle model. We also consider the problem of approximating the size of the intersection, show a linear lower-bound for the communication overhead of solving this problem, and provide a suitable secure protocol. Lastly, we investigate other variants of the matching problem, including extending the protocol to the multi-party setting as well as considering the problem of approximate matching.

1 Introduction

This work considers several two-party set-intersection problems and presents corresponding secure protocols. Our protocols enable two parties that each hold a set of inputs – drawn from a *large* domain – to jointly calculate the intersection of their inputs, without leaking any additional information. The set-intersection primitive is quite useful as it is extensively used in computations over databases, *e.g.*, for data mining where the data is vertically partitioned between parties (namely, each party has different attributes referring to the same subjects).

One could envision the usage of efficient set-intersection protocols for online recommendation services, online dating services, medical databases, and many other applications. We are already beginning to see the deployment of such applications using either trusted third parties or plain insecure communication.

Contributions. We study private two-party computation of set intersection, which we also denote as *private matching* (PM):

– Protocols for computing private matching, based on homomorphic encryption and balanced allocations: (i) a protocol secure against semi-honest adversaries; and (ii) a protocol, in the random oracle model, secure against

⋆ Research partially done while the author was visiting HP Labs.
⋆⋆ Research done while the author was at NEC Labs.

malicious adversaries.[4] Their overhead for input lists of length k is $O(k)$ communication and $O(k \ln \ln k)$ computation, with small constant factors. These protocols are more efficient than previous solutions to this problem.
- Variants of the private matching protocol that (i) compute the intersection *size*, (ii) decide whether the intersection size is greater than a threshold, or (iii) compute some other function of the intersection set.
- We consider private approximation protocols for the intersection size (similar to the private approximation of the Hamming distance by [10]). A simple reduction from the communication lower-bound on disjointness shows that this problem cannot have a sublinear *worst-case* communication overhead. We show a sampling-based private approximation protocol that achieves instance-optimal communication.
- We extend the protocol for set intersection to a multi-party setting.
- We introduce the problem of secure approximate (or "fuzzy") matching and search, and we present protocols for several simple instances.

2 Background and Related Work

Private equality tests (PET). A simpler form of private matching is where each of the two datasets has a single element from a domain of size N. A circuit computing this function has $O(\log N)$ gates, and therefore can be securely evaluated with this overhead. Specialized protocols for this function were also suggested in [9, 18, 17], and they essentially have the same overhead. A solution in [3] provides fairness in addition to security.

A circuit-based solution for computing PM of datasets of length k requires $O(k^2 \log N)$ communication and $O(k \log N)$ oblivious transfers. Another trivial construction compares all combinations of items from the two datasets using k^2 instantiations of a PET protocol (which itself has $O(\log N)$ overhead). The computation of this comparison can be reduced to $O(k \log N)$, while retaining the $O(k^2 \log N)$ communication overhead [18]. There are additional constructions that solve the private matching problem at the cost of only $O(k)$ exponentiations [12, 8]. However, these constructions were only analyzed in the random oracle model, against semi-honest parties.

Disjointness and set intersection. Protocols for computing (or deciding) the intersection of two sets have been researched both in the general context of communication complexity and in the context of secure protocols. Much attention has been given to evaluating the communication complexity of the disjointness problem, where the two parties in the protocol hold subsets a and b of $\{1, \ldots, N\}$. The disjointness function $\textsc{Disj}(a, b)$ is defined to be 1 if the sets a, b have an empty intersection. It is well known that $R_\epsilon(\textsc{Disj}) = \Theta(N)$ [14, 22]. An immediate implication is that computing $|a \cap b|$ requires $\Theta(N)$ communication. Therefore, even without taking privacy into consideration, the communication complexity of private matching is at least proportional to the input size.

[4] For malicious clients, we present a protocol that is secure in the standard model.

One may try and get around the high communication complexity of computing the intersection size by approximating it. In the context of secure protocols, this may lead to a sublinear *private approximation* protocol for intersection size.[5] If one settles for an approximation up to additive error ϵN (for constant ϵ), it is easy to see that very efficient protocols exist, namely $O(\log N)$ bits in the private randomness model [16, Example 5.5]. However, if we require *multiplicative* error (*e.g.*, an (ϵ, δ)-approximation), we show a simple reduction from disjointness that proves that a lower-bound of $\Omega(N)$ communication bits is necessary for any such approximation protocol. See Section 6 for details.

3 Preliminaries

3.1 Private matching (PM)

A **private matching** (PM) scheme is a two-party protocol between a client (chooser) $\mathcal{C}$ and a server (sender) $\mathcal{S}$. $\mathcal{C}$'s input is a set of inputs of size $k_{\mathcal{C}}$, drawn from some domain of size N; $\mathcal{S}$'s input is a set of size $k_{\mathcal{S}}$ drawn from the same domain. At the conclusion of the protocol, $\mathcal{C}$ learns which specific inputs are shared by both $\mathcal{C}$ and $\mathcal{S}$. That is, if $\mathcal{C}$ inputs $X = \{x_1, \ldots, x_{k_{\mathcal{C}}}\}$ and $\mathcal{S}$ inputs $Y = \{y_1, \ldots, y_{k_{\mathcal{S}}}\}$, $\mathcal{C}$ learns $X \cap Y$: $\{x_u | \exists v, x_u = y_v\} \leftarrow \mathsf{PM}(X, Y)$.

PM Variants. Some variants of the private matching protocol include the following. (i) Private cardinality matching (PM_C) allows $\mathcal{C}$ to learn *how many* inputs it shares with $\mathcal{S}$. That is, $\mathcal{C}$ learns $|X \cap Y|$: $|\mathsf{PM}| \leftarrow \mathsf{PM}_C(X, Y)$. (ii) Private threshold matching (PM_t) provides $\mathcal{C}$ with the answer to the *decisional problem* whether $|X \cap Y|$ is greater than some pre-specified threshold t. That is, $1 \leftarrow \mathsf{PM}_t(X, Y)$ if $\mathsf{PM}_C > t$ and 0 otherwise. (iii) Generalizing PM_C and PM_t, one could define arbitrary private-matching protocols that are simple functions of the intersection set, *i.e.*, based on the output of PM or PM_C.

Private Matching and Oblivious Transfer. We show a simple reduction from oblivious transfer (OT) to private matching. The OT protocol we design is a 1-out-of-2 bit-transfer protocol in the semi-honest case. The sender's input contains two bits b_0, b_1. The chooser's input is a bit σ. At the end of the protocol the chooser learns b_σ and nothing else, while the sender learns nothing.

First, the parties generate their respective PM inputs: The sender generates a list of two strings, $\{0|b_0, 1|b_1\}$, and the chooser generates the list $\{\sigma|0, \sigma|1\}$. Then, they run the PM protocol, at the end of which the chooser learns $\sigma|b_\sigma$. It follows by the results of Impagliazzo and Rudich [13] that there is no black-box reduction of private matching from one-way functions.

Since the reduction is used to show an impossibility result, it is sufficient to show it for the simplest form of OT, as we did above. We note that if one actually wants to build an OT protocol from a PM primitive, it is possible to directly construct a 1-out-of-N bit transfer protocol. In addition, the `PM-Semi-Honest` protocol we describe supports OT of strings.

[5] Informally, a private approximation is an approximation that does not leak information that is not computable given the exact value. See the definition in [10].

3.2 Adversary models

This paper considers both semi-honest and malicious adversaries. Due to space constraints, we only provide the intuition and informal definitions of these models. The reader is referred to [11] for the full definitions.

Semi-honest adversaries. In this model, both parties are assumed to act according to their prescribed actions in the protocol. The security definition is straightforward, particularly as in our case where only one party ($\mathcal{C}$) learns an output. We follow [18] and divide the requirements into (i) protecting the client and (ii) protecting the sender.
The client's security – indistinguishability: Given that the server $\mathcal{S}$ gets no output from the protocol, the definition of $\mathcal{C}$'s privacy requires simply that the server cannot distinguish between cases in which the client has different inputs.
The server's security – comparison to the ideal model: The definition ensures that the client does not get more or different information than the output of the function. This is formalized by considering an ideal implementation where a trusted third party (TTP) gets the inputs of the two parties and outputs the defined function. We require that in the real implementation of the protocol—that is, one without a TTP—the client $\mathcal{C}$ does not learn different information than in the ideal implementation.

Malicious adversaries. In this model, an adversary may behave arbitrarily. In particular, we cannot hope to avoid parties (i) refusing to participate in the protocol, (ii) substituting an input with an arbitrary value, and (iii) prematurely aborting the protocol. The standard security definition (see, *e.g.*, [11]) captures both the correctness and privacy issues of the protocol and is limited to the case in which only one party obtains an output. Informally, the definition is based on a comparison to the ideal model with a TTP, where a corrupt party may give arbitrary input to the TTP. The definition also is limited to the case where at least one of the parties is honest: if $\mathcal{C}$ (resp. $\mathcal{S}$) is honest, then for any strategy that $\mathcal{S}$ (resp. $\mathcal{C}$) can play in the real execution, there is a strategy that it could play in the ideal model, such that the real execution is computationally indistinguishable from execution in the ideal model. We note that main challenge in ensuring security is enforcing the protocol's correctness, rather than its privacy.

3.3 Cryptographic primitives – Homomorphic encryption schemes

Our constructions use a semantically-secure public-key encryption scheme that preserves the group homomorphism of addition and allows multiplication by a constant. This property is obtained by Paillier's cryptosystem [20] and subsequent constructions [21, 7]. That is, it supports the following operations that can be performed without knowledge of the private key: (i) Given two encryptions $\mathsf{Enc}(m_1)$ and $\mathsf{Enc}(m_2)$, we can efficiently compute $\mathsf{Enc}(m_1+m_2)$. (ii) Given some constant c belonging to the same group, we can compute $\mathsf{Enc}(cm)$. We will use the following corollary of these two properties: Given encryptions of the coefficients $a_0, \ldots, a_k$ of a polynomial P of degree k, and knowledge of a plaintext value y, it is possible to compute an encryption of $P(y)$.[6]

4 The Semi-Honest Case

4.1 Private Matching for set intersection (PM)

The protocol follows the following basic structure. $\mathcal{C}$ defines a polynomial P whose roots are her inputs:

$$P(y) = (x_1 - y)(x_2 - y) \ldots (x_{k_\mathcal{C}} - y) = \sum_{u=0}^{k_\mathcal{C}} \alpha_u y^u$$

She sends to $\mathcal{S}$ homomorphic encryptions of the coefficients of this polynomial. $\mathcal{S}$ uses the homomorphic properties of the encryption system to evaluate the polynomial at each of his inputs. He then multiplies each result by a fresh random number r to get an intermediate result, and he adds to it an encryption of the value of his input, *i.e.*, $\mathcal{S}$ computes $\mathsf{Enc}(r \cdot P(y) + y)$. Therefore, for each of the elements in the intersection of the two parties' inputs, the result of this computation is the value of the corresponding element, whereas for all other values the result is random.[7] See Protocol `PM-Semi-Honest`.[8]

4.2 Efficiently evaluating the polynomial

As the computational overhead of exponentiations dominates that of other operations, we evaluate the computational overhead of the protocol by counting exponentiations. Equivalently, we count the number of *multiplications* of the homomorphically-encrypted values by constants (in Step 2(a)), as these multiplications are actually implemented as exponentiations.

Given the encrypted coefficients $\mathsf{Enc}(\alpha_u)$ of a polynomial P, a naive computation of $\mathsf{Enc}(P(y))$ as $\mathsf{Enc}(\sum_{u=0}^{k_\mathcal{C}} y^u \alpha_u)$ results in an overhead of $O(k_\mathcal{C})$ exponentiations, and hence in a total of $O(k_\mathcal{C} k_\mathcal{S})$ exponentiations for `PM-Semi-Honest`.

The computational overhead can be reduced by noting that the input domain is typically much smaller than the modulus used by the encryption scheme.

[6] We neglect technicalities that are needed to make sure the resulting ciphertext hides the sequence of homomorphic operations that led to it. This may be achieved, *e.g.*, by multiplying the result by a random encryption of 1.

[7] This construction can be considered a generalization of the oblivious transfer protocols of [19, 1, 17]. In those, a client retrieving item i sends to the server a predicate which is 0 if and only if $i = j$ where $j \in [N]$.

[8] It is sufficient for Step 3 of the protocol that $\mathcal{C}$ is able to decide whether some ciphertext corresponds to $x \in X$ (*i.e.*, decryption is not necessary). This weaker property is of use if, for example, one uses the El Gamal encryption scheme and encodes an element x by g^x (to allow the homomorphic properties under addition). This may prevent $rP(y) + y$ from being recovered in the decryption process, yet it is easy for $\mathcal{C}$ to decide whether $rP(y) + y = x$. The Paillier [20] homomorphic encryption scheme recovers $rP(y) + y$.

Protocol PM-Semi-Honest

INPUT: $\mathcal{C}$'s input is a set $X = \{x_1, \ldots, x_{k_\mathcal{C}}\}$, $\mathcal{S}$'s input is a set $Y = \{y_1, \ldots, y_{k_\mathcal{S}}\}$. The elements in the input sets are taken from a domain of size N.

1. $\mathcal{C}$ performs the following:
   (a) She chooses the secret-key parameters for a semantically-secure homomorphic encryption scheme, and publishes its public keys and parameters. The plaintexts are in a field that contains representations of the N elements of the input domain, but is exponentially larger.
   (b) She uses interpolation to compute the coefficients of the polynomial $P(y) = \Sigma_{u=0}^{k_\mathcal{C}} \alpha_u y^u$ of degree $k_\mathcal{C}$ with roots $\{x_1, \ldots, x_{k_\mathcal{C}}\}$.
   (c) She encrypts each of the $(k_\mathcal{C} + 1)$ coefficients by the semantically-secure homomorphic encryption scheme and sends to $\mathcal{S}$ the resulting set of ciphertexts, $\{\mathsf{Enc}(\alpha_0), \ldots, \mathsf{Enc}(\alpha_{k_\mathcal{C}})\}$.
2. $\mathcal{S}$ performs the following for every $y \in Y$,
   (a) He uses the homomorphic properties to evaluate the encrypted polynomial at y. That is, he computes $\mathsf{Enc}(P(y)) = \mathsf{Enc}(\Sigma_{u=0}^{k_\mathcal{C}} \alpha_u y^u)$. See Section 4.2.
   (b) He chooses a random value r and computes $\mathsf{Enc}(rP(y) + y)$. (One can also encrypt some additional payload data p_y by computing $\mathsf{Enc}(rP(y) + (y|p_y))$. $\mathcal{C}$ obtains p_y iff y is in the intersection.)

   He randomly permutes this set of $k_\mathcal{S}$ ciphertexts and sends the result back to the client $\mathcal{C}$.
3. $\mathcal{C}$ decrypts all $k_\mathcal{S}$ ciphertexts received. She locally outputs all values $x \in X$ for which there is a corresponding decrypted value .

Hence one may encode the values x, y as numbers in the smaller domain. In addition, Horner's rule can be used to evaluate the polynomial more efficiently by eliminating large exponents. This yields a significant (large constant factor) reduction in the overhead.

We achieve a more significant reduction of the overhead by allowing the client to use multiple low-degree polynomials and then allocating input values to polynomials by hashing. This results in reducing the computational overhead to $O(k_\mathcal{C} + k_\mathcal{S} \ln \ln k_\mathcal{C})$ exponentiations. Details follow.

Exponents from a small domain. Let λ be the security parameter of the encryption scheme (*e.g.*, λ is the modulus size). A typical choice is $\lambda = 1024$ or larger. Yet, the input sets are usually of size $\ll 2^\lambda$ and may be mapped into a small domain—of length $n \approx 2\log(\max(kc, ks))$ bits—using pairwise-independent hashing, which induces only a small collision probability. The server should compute $\mathsf{Enc}(P(y))$, where y is n bits long.

Using Horner's rule. We get our first overhead reduction by applying Horner's rule: $P(y) = \alpha_0 + \alpha_1 y + \alpha_2 y^2 + \cdots + \alpha_{k_\mathcal{C}} y^{k_\mathcal{C}}$ is evaluated "from the inside out" as $\alpha_0 + y(\alpha_1 + y(\alpha_2 + y(\alpha_3 + \cdots y(\alpha_{k_\mathcal{C}-1} + y\alpha_{k_\mathcal{C}}) \cdots)))$. One multiplies each intermediate result by a *short* y, compared with y^i in the naive evaluation, which results in $k_\mathcal{C}$ *short* exponentiations.

When using the "text book" algorithm for computing exponentiations, the computational overhead is linear in the length of the exponent. Therefore, Horner's rule improves this overhead by a factor of λ/n (which is about 50 for $k_\mathcal{C}, k_\mathcal{S} \approx$

1000). The gain is substantial even when fine-tunes exponentiation algorithms—such as Montgomery's method or Karatsuba's technique—are used.

Using hashing for bucket allocation. The protocol's main computational overhead results from the server computing polynomials of degree $k_{\mathcal{C}}$. We now reduce the degree of these polynomials. For that, we define a process that throws the client's elements into B bins, such that each bin contains at most M elements.

The client now defines a polynomial of degree M for each bin: All items mapped to the bin by some function h are defined to be roots of the polynomial. In addition, the client adds the root $x = 0$ to the polynomial, with multiplicity which sets the total degree of the polynomial to M. That is, if h maps ℓ items to the bin, the client first defines a polynomial whose roots are these ℓ items, and then multiplies it by $x^{M-\ell}$. (We assume that 0 is not a valid input.) The process results in B polynomials, all of them of degree M, that have a total of $k_{\mathcal{C}}$ non-zero roots.

$\mathcal{C}$ sends to $\mathcal{S}$ the encrypted coefficients of the polynomials, and the mapping from elements to bins.[9] For every $y \in Y$, $\mathcal{S}$ finds the bins into which y could be mapped and evaluates the polynomial of those bins. He proceeds as before and responds to $\mathcal{C}$ with the encryptions $rP(y) + y$ for every possible bin allocation for all y.

Throwing elements into bins – balanced allocations. We take the mapping from elements to bins to be a random hash function h with a range of size B, chosen by the client. Our goal is to reduce M, the upper bound on the number of items in a bin. It is well known that if the hash function h maps each item to a random bin, then with high probability (over the selection of h), each bin contains at most $k_{\mathcal{C}}/B + O(\sqrt{(k_{\mathcal{C}}/B)\log B} + \log B)$ elements. A better allocation is obtained using the balanced allocation hashing by Azar et al. [2]. The function h now chooses *two* distinct bins for each item, and the item is mapped into the bin which is *less occupied* at the time of placement. In the resulting protocol, the server uses h to locate the two bins into which y might have been mapped, evaluates both polynomials, and returns the two answers to $\mathcal{C}$.

Theorem 1.1 of [2] shows that the maximum load of a bin is now exponentially smaller: with $1 - o(1)$ probability, the maximum number of items mapped to a bin is $M = (1 + o(1)) \ln\ln B/\ln 2 + \Theta(k_{\mathcal{C}}/B)$. Setting $B = k_{\mathcal{C}}/\ln\ln k_{\mathcal{C}}$, we get $M = O(\ln\ln k_{\mathcal{C}})$.

A note on correctness and on constants. One may worry about the case that $\mathcal{C}$ is unlucky in her choice of h such that more than M items are mapped to some bin. The bound of [2] only guarantees that this happens with probability $o(1)$. However, Broder and Mitzenmacher [4] have shown that asymptotically, when we map n items into n bins, the number of bins with i or more items falls approximately like $2^{-2.6^i}$. This means that a bound of $M = 5$ suffices with probability 10^{-58}. Furthermore, if the hashing searches for the emptiest in three bins, then $M = 3$ suffices with probability of about 10^{-33}. The authors also

[9] For our purposes, it is sufficient that the mapping is selected pseudo-randomly, either jointly or by either party.

provide experimental results that confirm the asymptotic bound for the case of $n = 32,000$. We conclude that we can bound $\ln \ln k_{\mathcal{C}}$ by a small constant in our estimates of the overhead. Simple experimentation can provide finer bounds.

Efficiency. The communication overhead, and the computation overhead of the client, are equal to the total number of coefficients of the polynomials. This number, given by $B \cdot M$, is $O(k_{\mathcal{C}})$ if $B = k_{\mathcal{C}} / \ln \ln k_{\mathcal{C}}$. If $k \leq 2^{24}$, then using $B = k_{\mathcal{C}}$ bins implies that the communication overhead is at most 4 times that of the protocol that does not use hashing.

The server computes, for each item in his input, M exponentiations with a small exponent, and one exponentiation with a full-length exponent (for computing $r \cdot P(y)$). Expressing this overhead in terms of full-length exponentiations yields an overhead of $O(k_{\mathcal{S}} + k_{\mathcal{S}} \frac{\ln \ln k_{\mathcal{C}} \cdot n}{\lambda})$ for $B = k_{\mathcal{C}} / \ln \ln k_{\mathcal{C}}$. In practice, the overhead of the exponentiations with a small exponent has little effect on the total overhead, which is dominated by $k_{\mathcal{S}}$ full-length exponentiations.

4.3 Security of `PM-Semi-Honest`

We state the claims of security for PM in the semi-honest model.

Lemma 1 (Correctness). *Protocol* `PM-Semi-Honest` *evaluates the* PM *function with high probability.*

(The proof is based on the fact that the client receives an encryption of y for $y \in X \cap Y$, and an encryption of a random value otherwise.)

Lemma 2 ($\mathcal{C}$'s privacy is preserved). *If the encryption scheme is semantically secure, then the views of $\mathcal{S}$ for any two inputs of $\mathcal{C}$ are indistinguishable.*

(The proof uses the fact that the only information that $\mathcal{S}$ receives consists of semantically-secure encryptions.)

Lemma 3 ($\mathcal{S}$'s privacy is preserved). *For every client $\mathcal{C}^*$ that operates in the real model, there is a client $\mathcal{C}$ operating in the ideal model, such that for every input Y of $\mathcal{S}$, the views of the parties $\mathcal{C}, \mathcal{S}$ in the ideal model is indistinguishable from the views of $\mathcal{C}^*, \mathcal{S}$ in the real model.*

(The proof defines a polynomial whose coefficients are the plaintexts of the encryptions sent by $\mathcal{C}^*$ to $\mathcal{S}$. The $k_{\mathcal{C}}$ roots of this polynomial are the inputs that $\mathcal{C}$ sends to the trusted third party in the ideal implementation.)

Security of the hashing-based protocol. Informally, the hashing-based protocol preserves $\mathcal{C}$'s privacy since (i) $\mathcal{S}$ still receives semantically-secure encryptions, and (ii) the key is chosen independently of $\mathcal{C}$'s input. Thus, neither the key nor h reveal any information about X to $\mathcal{S}$. The protocol preserves $\mathcal{S}$'s privacy since the total number of non-zero roots of the polynomials is $k_{\mathcal{C}}$.

4.4 Variant: Private Matching for set cardinality (PM_C)

In a protocol for private cardinality matching, $\mathcal{C}$ should learn the cardinality of $X \cap Y$, but not the actual elements of this set. $\mathcal{S}$ needs only slightly change his behavior from that in Protocol `PM-Semi-Honest` to enable this functionality. Instead of encoding y in Step 2(b), $\mathcal{S}$ now only encodes some "special" string, such as a string of 0's, *i.e.*, $\mathcal{S}$ computes $\mathsf{Enc}(rP(y) + 0^+)$. In Step 3 of the protocol, $\mathcal{C}$ counts the number of ciphertexts received from $\mathcal{S}$ that decrypt to the string 0^+ and locally outputs this number c. The proof of security for this protocol trivially follows from that of `PM-Semi-Honest`.

4.5 Variants: Private Matching for cardinality threshold (PM_t) and other functions

In a protocol for private threshold matching, $\mathcal{C}$ should only learn whether $c = |X \cap Y| > t$. To enable this functionality, we change `PM-Semi-Honest` as follows. (i) In Step 2(b), $\mathcal{S}$ encodes random numbers instead of y in PM (or 0^+ in PM_C). That is, he computes $\mathsf{Enc}(rP(y)+r_y)$, for random r_y. (ii) Following the basic PM protocol, $\mathcal{C}$ and $\mathcal{S}$ engage in a secure circuit evaluation protocol. The circuit takes as input $k_{\mathcal{S}}$ values from each party: $\mathcal{C}$'s input is the ordered set of plaintexts she recovers in Step 3 of the PM protocol. $\mathcal{S}$'s input is the list of random payloads he chooses in Step 2(b), in the same order he sends them. The circuit first computes the equality of these inputs bit-by-bit, which requires $k_{\mathcal{S}}\lambda'$ gates, where λ' is a statistical security parameter. Then, the circuit computes a threshold function on the results of the $k_{\mathcal{S}}$ comparisons.

Hence, the threshold protocol has the initial overhead of a PM protocol plus the overhead of a secure circuit evaluation protocol. Note, however, that the overhead of circuit evaluation is not based on the input domain of size N. Rather, it first needs to compute equality on the input set of size $k_{\mathcal{S}}$, then compute some simple function of the size of the *intersection set*. In fact, this protocol can be used to compute any function of the intersection set, *e.g.*, check if c within some range, not merely the threshold problem.

5 Security against Malicious Parties

We describe modifications to our PM protocol in order to provide security in the malicious adversary model. Our protocols are based on protocol `PM-Semi-Honest`, optimized with the balanced allocation hashing.

We first deal with malicious clients and then with malicious servers. Finally, we combine these two protocols to achieve a protocol in which either party may behave adversarially. We take this non-standard approach as: (i) It provides conceptual clarity as to the security concerns for each party; (ii) These protocols may prove useful in varying trust situations, *e.g.*, one might trust a server but not the myriad clients; and (iii) The client protocol is secure in the standard model, while the server protocol is analyzed in the random oracle model.

Protocol `PM-Malicious-Client`

INPUT: $\mathcal{C}$ has input X of size $k_{\mathcal{C}}$, and $\mathcal{S}$ has input Y of size $k_{\mathcal{S}}$, as before.

1. $\mathcal{C}$ performs the following:
   (a) She chooses a key for a pseudo-random function that realizes the balanced allocation hash function h, and she sends it to $\mathcal{S}$.
   (b) She chooses a key s for a pseudo-random function F and gives each item x in her input X a new pseudo-identity, $F_s(G(x))$, where G is a collision-resistant hash function.
   (c) For each of her polynomials, $\mathcal{C}$ first sets roots to the pseudo-identities of such inputs that were mapped to the corresponding bin. Then, she adds a sufficient number of 0 roots to set the polynomial's degree to M.
   (d) She repeats Steps (b),(c) for L times to generate L copies, using a different key s for F in each iteration.
2. $\mathcal{S}$ asks $\mathcal{C}$ to open $L/2$ of the copies.
3. $\mathcal{C}$ opens the encryptions of the coefficients of the polynomials for these $L/2$ copies to $\mathcal{S}$, but does not reveal the associated keys s. Additionally, $\mathcal{C}$ sends the keys s used in the unopened $L/2$ copies.
4. $\mathcal{S}$ verifies that the each opened copy contains $k_{\mathcal{C}}$ roots. If this verification fails, $\mathcal{S}$ halts. Otherwise, $\mathcal{S}$ uses the additional $L/2$ keys he receives, along with the hash function G, to generate the pseudo-identities of his inputs. He runs the protocol for each of the polynomials. However, for an input y, rather than encoding y as the payload for each polynomial, he encodes $L/2$ random values whose exclusive-or is y.
5. $\mathcal{C}$ receives the results, organized as a list of $k_{\mathcal{S}}$ sets of size $L/2$. She decrypts them, computes the exclusive-or of each set, and compares it to her input.

5.1 Malicious clients

To ensure security against a malicious client $\mathcal{C}$, it must be shown that for any possible client behavior in the real model, there is an input of size $k_{\mathcal{C}}$ that the client provides to the TTP in the ideal model, such that his view in the real protocol is efficiently simulatable from his view in the ideal model.

We first describe a simple solution for the implementation that does not use hashing. We showed in Lemma 3 that if a value y is not a root of the polynomial sent by the client, the client cannot distinguish whether this item is in the server's input. Thus, we have to take care of the possibility that $\mathcal{C}$ sends the encryption of a polynomial with more than $k_{\mathcal{C}}$ roots. This can only happen if all the encrypted coefficients are zero (P's degree is indeterminate). We therefore modify the protocol to require that at least one coefficient is non-zero – in Step 1(b) of Protocol `PM-Semi-Honest`, $\mathcal{C}$ generates the coefficients of P with α_0 set to 1, then sends encryptions of the other coefficients to $\mathcal{S}$.

In the protocol that uses hashing, $\mathcal{C}$ sends encryptions of the coefficients of B polynomials (one per bin), each of degree M. The server must ensure that the *total* number of roots (different than 0) of these polynomials is $k_{\mathcal{C}}$. For that we use a cut-and-choose method, as shown in Protocol `PM-Malicious-Client`. With overhead L times that of the original protocol, we get error probability that is exponentially small in L.

Proof. (sketch) In our given cut-and-choose protocol, note that $\mathcal{C}$ learns about an item iff it is a root of all the $L/2$ copies evaluated by $\mathcal{S}$. Therefore, to learn about more than $k_{\mathcal{C}}$ items, she must have $L/2$ copies such that each has more than $k_{\mathcal{C}}$ roots. The probability that all such polynomials are not checked by $\mathcal{S}$ is exponentially small in L. This argument can be used to show that, for every adversarial $\mathcal{C}*$ whose success probability is not exponentially small, there is a corresponding $\mathcal{C}$ in the ideal model whose input contains at most $k_{\mathcal{C}}$ items.[10]

5.2 Malicious servers

Protocol `PM-Semi-Honest` of Section 4 enables a malicious server to attack the protocol *correctness*.[11] He can play tricks like encrypting the value $r \cdot (P(y) + P(y')) + y''$ in Step 2(b), so that $\mathcal{C}$ concludes that y'' is in the intersection set iff both y and y' are X. This behavior does not correspond to the definition of PM in the ideal model. Intuitively, this problem arises from $\mathcal{S}$ using two "inputs" in the protocol execution for input y—a value for the polynomial evaluation, and a value used as a payload—whereas $\mathcal{S}$ has a single input in the ideal model.[12]

We show how to modify Protocol `PM-Semi-Honest` to gain security against malicious servers. The protocol based on balanced allocations may be modified similarly. Intuitively, we force the server to run according to its prescribed procedure. Our construction, `PM-Malicious-Server`, is in the random oracle model.

The server's privacy is preserved as in `PM-Semi-Honest`: The pair (e, h) is indistinguishable from random whenever $P(y) \neq 0$. The following lemma shows that the client security is preserved under malicious server behavior.

Lemma 4 (Security for the client). *For every server $\mathcal{S}^*$ that operates in the real model, there is a server $\mathcal{S}$ operating in the ideal model, such that the views of the parties $\mathcal{C}, \mathcal{S}$ in the ideal model is computationally indistinguishable from the views of $\mathcal{C}, \mathcal{S}^*$ in the real model.*

Proof. (sketch) We describe how $\mathcal{S}$ works.

1. $\mathcal{S}$ generates a secret-key/public-key pair for the homomorphic encryption scheme, chooses a random polynomial $P(y)$ of degree $k_{\mathcal{C}}$ and gives $\mathcal{S}^*$ his encrypted coefficients. Note that $\mathcal{S}^*$ does not distinguish the encryption of $P(y)$ from the encryption of any other degree $k_{\mathcal{C}}$ polynomial.
2. $\mathcal{S}$ records all the calls $\mathcal{S}^*$ makes to the random oracles H_1, H_2. Let $\hat{S}$ be the set of input values to H_1 and $\hat{Y}$ be the set of y input values to H_2.

[10] In the proof, the pseudo-random function F hides from $\mathcal{S}$ the identities of the values corresponding to the roots of the opened polynomials. The collision-resistant hash function G prevents $\mathcal{C}$ from setting a root to which $\mathcal{S}$ maps two probable inputs.

[11] He cannot affect $\mathcal{C}$'s privacy as all the information $\mathcal{C}$ sends is encrypted via a semantically-secure encryption scheme.

[12] Actually, the number of "inputs" is much higher, as $\mathcal{S}$ needs to be consistent in using the same y for all the steps of the polynomial-evaluation procedure.

Protocol `PM-Malicious-Server`

INPUT: $\mathcal{C}$ has input X of size $k_{\mathcal{C}}$, and $\mathcal{S}$ has input Y of size $k_{\mathcal{S}}$, as before.
RANDOM ORACLES: H_1, H_2.

1. $\mathcal{C}$ performs the following:
   (a) She chooses a secret-key/public-key pair for the homomorphic encryption scheme, and sends the public-key to $\mathcal{S}$.
   (b) She generates the coefficients of a degree $k_{\mathcal{C}}$ polynomial P whose roots are the values in X. She sends to $\mathcal{S}$ the encrypted coefficients of P.
2. $\mathcal{S}$ performs the following for every $y \in Y$,
   (a) He chooses a random s and computes $r = H_1(s)$. We use r to "derandomize" the rest of $\mathcal{S}$'s computation for y, and we assume that it is of sufficient length.
   (b) He uses the homomorphic properties of the encryption scheme to compute $(e, h) \leftarrow (\mathsf{Enc}(r' \cdot P(y) + s), H_2(r'', y))$. In this computation, r is parsed to supply r', r'' and all the randomness needed in the computation.

   $\mathcal{S}$ randomly permutes this set of $k_{\mathcal{S}}$ pairs and sends it to $\mathcal{C}$.
3. $\mathcal{C}$ decrypts all the $k_{\mathcal{S}}$ pairs she received. She performs the following operations for every pair (e, h),
   (a) She decrypts e to get $\hat{s}$ and computes $\hat{r} = H_1(\hat{s})$.
   (b) She checks whether, for some $x \in X$, the pair (e, h) is consistent with x and $\hat{s}$. That is, whether the server yields (e, h) using her encrypted coefficients on $y \leftarrow x$ and randomness $\hat{r}$. If so, she puts x in the intersection set.

3. For every output pair (e, h) of $\mathcal{S}^*$, $\mathcal{S}$ checks whether it agrees with some $\hat{s} \in \hat{S}$ and $\hat{y} \in \hat{Y}$. We call such a pair a consistent pair. That is, $\mathcal{S}$ checks that (i) e is a ciphertext resulting from applying the server's prescribed computation using the encrypted coefficients, the value $\hat{y}$, and randomness r'; and (ii) $h = H_2(r'', \hat{y})$, where r', r'' and the randomness in the computation are determined by $H_1(\hat{s})$. If such consistency does occur, $\mathcal{S}$ sets $y = \hat{y}$, otherwise it sets $y = \perp$.
4. $\mathcal{S}$ sends the values y it computed to the TTP, and $\mathcal{S}$ outputs the same output as $\mathcal{S}^*$ in the real model.

It is easy, given the view of $\mathcal{S}^*$, to decide whether a pair is consistent. As $\mathcal{S}^*$ cannot distinguish the input fed to it by $\mathcal{S}$ from the input it receives from $\mathcal{C}$ in the real execution, we get that S^*'s distributions on consistent and inconsistent pairs, when run by the simulator and in the real execution, are indistinguishable.

Whenever (e, h) forms an inconsistent pair, giving an invalid symbol $\perp$ as input to the TTP does not affect its outcome. Let (e, h) be a consistent pair, and let y be the value that is used in its construction. In the real execution, $y \in X$ would result in adding y to the intersection set, and this similarly would happen in the simulation. The event that, in the real execution, an element $x \neq y$ would be added to the intersection set occurs with negligible probability.

We get that the views of the parties $\mathcal{C}, \mathcal{S}$ in the ideal model is computationally indistinguishable from the views of $\mathcal{C}, \mathcal{S}^*$ in the real model, as required.

5.3 Handling both malicious clients and servers

We briefly describe how to *combine* these two schemes yield a PM protocol fully secure in the malicious model. We leave the detailed description to the full version of this paper.

$\mathcal{C}$ generates B bins as before; for each bin B_i, she generates a polynomial of degree M with $P(z) = 0$, where $z \in B_i$ if it is (1) mapped to B_i by our hashing scheme (for $z = F_s(G(x))$ for $x \in X$) or (2) added as needed to yield M items. The latter should be set outside the range of F_s. For each polynomial, $\mathcal{C}$ prepares L copies and sends their commitments to $\mathcal{S}$.

Next, $\mathcal{S}$ opens the encryptions of $L/2$ copies and verifies them. If verification succeeds, $\mathcal{S}$ opens the F_s used in the other $L/2$ copies. He chooses a random s, splits it into $L/2$ shares, and then acts as in `PM-Malicious-Server`, albeit using the random shares as payload, $H_1(s)$ as randomness, and appending $H_2(r'', y)$.

Finally, $\mathcal{C}$ receives a list of the unopened $L/2$ copies. For each, she computes candidates for s's shares and recovers s from them. She uses a procedure similar to `PM-Malicious-Server` to check the consistency of the these $L/2$ shares.

6 Approximating Intersection

In this section, we focus on a problem related to private matching: set intersection and its approximation. Assume $\mathcal{C}$ and $\mathcal{S}$ hold strings X and Y respectively, where $|X| = |Y| = N$. Define $\textsc{Intersect}(X, Y) = |\{i : X_i = Y_i\}|$. Equivalently, $\textsc{Intersect}(X, Y)$ is the scalar product of X, Y. Let $0 < \epsilon, \delta$ be constants. An (ϵ, δ)-approximation protocol for intersection yields, on inputs X, Y, a value $\hat{\alpha}$ such that $\Pr[(1-\epsilon)\alpha < \hat{\alpha} < (1+\epsilon)\alpha] \geq 1 - \delta$ where $\alpha = |X \cap Y|$. The probability is taken over the randomness used in the protocol.

A lower bound. Let $0 < \eta \leq N$. It is easy to see that an (ϵ, δ)-approximation may be used for distinguishing the cases $|X \cap Y| \leq \eta$ and $|X \cap Y| \geq \eta(1+\epsilon)^2$, as (with probability $1 - \delta$) its output is less than $\eta(1+\epsilon)$ in the former case and greater than $\eta(1+\epsilon)$ in the latter.

A protocol that distinguishes $|X \cap Y| \leq \eta$ and $|X \cap Y| \geq \eta(1+\epsilon)$ may be used for deciding disjointness, as defined in Section 2. Given inputs a, b of length m for $\textsc{Disj}$, $\mathcal{C}$ sets her input to be $X = 1^\eta | a^{(2\epsilon+\epsilon^2)\eta}$ (*i.e.*, η ones followed by $(2\epsilon + \epsilon^2)\eta$ copies of a). Similarly, $\mathcal{S}$ sets $Y = 1^\eta | b^{(2\epsilon+\epsilon^2)\eta}$. The length of these new inputs is $N = |X| = |Y| = \eta + (2\epsilon + \epsilon^2)\eta m$ bits. Note that if a, b are disjoint, then $|X \cap Y| = \eta$; otherwise, $|X \cap Y| \geq \eta(1+\epsilon)^2$. Hence, for constant ϵ, it follows that the randomized communication complexity of distinguishing the two cases is at least $\Omega(m) = \Omega(N/\eta)$. By setting η to a constant, we get that the randomized communication complexity of an (ϵ, δ) approximation for $\textsc{Intersect}$ is $\Theta(N)$.

A private approximation protocol for intersection. We describe a protocol for the semi-honest case. Informally, a protocol realizes a private approximation to a function $f(X, Y)$ if it computes an approximation to $f(X, Y)$ and does not leak any information that is not efficiently computable from $f(X, Y)$. This

Protocol $\texttt{Private-Sample-}B$

INPUT: $\mathcal{C}$ and $\mathcal{S}$ hold N-bit strings X, Y, respectively.

1. $\mathcal{C}$ picks a random mask $m_{\mathcal{C}} \in_R \{0,1\}$ and shift amount $r_{\mathcal{C}} \in_R [N]$. She computes the N-bit string $X' = (X \ll r_{\mathcal{C}}) \oplus m_{\mathcal{C}}$ (*i.e.*, she shifts X cyclicly $r_{\mathcal{C}}$ positions and XORs every location in the resulting string with $m_{\mathcal{C}}$). Similarly, $\mathcal{S}$ picks $m_{\mathcal{S}}, r_{\mathcal{S}}$ and computes $Y' = (Y << r_{\mathcal{S}}) \oplus m_{\mathcal{S}}$.
2. $\mathcal{C}$ and $\mathcal{S}$ invoke two $\binom{N}{1}$-OT protocols where $\mathcal{C}$ retrieves $s_{\mathcal{C}} = Y'_{r_{\mathcal{C}}}$ and $\mathcal{S}$ retrieves $s_{\mathcal{S}} = X'_{r_{\mathcal{S}}}$.
3. $\mathcal{C}$ computes $T_{00} = B(m_{\mathcal{C}}, s_{\mathcal{S}}), T_{01} = B(m_{\mathcal{C}}, s_{\mathcal{S}} \oplus 1), T_{10} = B(m_{\mathcal{C}} \oplus 1, s_{\mathcal{S}}), T_{11} = B(m_{\mathcal{C}} \oplus 1, s_{\mathcal{S}} \oplus 1)$.
4. $\mathcal{C}$ and $\mathcal{S}$ invoke a $\binom{4}{1}$-OT protocol where $\mathcal{S}$ retrieves $T_{m_{\mathcal{S}}, s_{\mathcal{S}}}$. $\mathcal{S}$ sends $T_{m_{\mathcal{S}}, s_{\mathcal{S}}}$ back to $\mathcal{C}$.

is formulated by the requirement that each party should be able to simulate her view given her input and $f(X, Y)$. We refer the reader to [10] for the formal definition.

Our building block – protocol $\texttt{Private-Sample-}B$ – is a simple generalization of the private sampler of [10]. $\texttt{Private-Sample-}B$ samples a random location ℓ and checks if a predicate B holds on (X_ℓ, Y_ℓ). The location ℓ is shared by $\mathcal{C}$ and $\mathcal{S}$ as $\ell = r_{\mathcal{C}} + r_{\mathcal{S}} \pmod N$, with each party holding one of the random shares $r_{\mathcal{C}}, r_{\mathcal{S}}$ at the end of Step 1. Step 2 results in $\mathcal{C}$ and $\mathcal{S}$ holding random shares of $X_\ell = m_{\mathcal{C}} \oplus s_{\mathcal{S}}$ and $Y_\ell = m_{\mathcal{S}} \oplus s_{\mathcal{C}}$. Finally, both parties learn $B(m_{\mathcal{C}} \oplus s_{\mathcal{C}}, m_{\mathcal{S}} \oplus s_{\mathcal{S}}) = B(X_\ell, Y_\ell)$.

It is easy to see that the views of $\mathcal{C}$ and $\mathcal{S}$ in Protocol $\texttt{Private-Sample-}B$ are simulatable given $v = |\{i : B(X_i, Y_y)\}|$. It follows that any approximation based on the outcome of the protocol is a *private approximation* for v.

The communication costs of $\texttt{Private-Sample-}B$ are dominated by the cost of the $\binom{N}{1}$-OT protocol in use. Naor and Pinkas [19] showed how to combine a $\binom{N}{1}$-OT protocol with any computational PIR scheme, under the DDH assumption. Combining this result with PIR scheme of Cachin et al. [5] (or of Kiayias and Yung [15]) results in $\lambda\,\mathsf{polylog}(N)$ communication, for security parameter λ.

Our protocol $\texttt{Intersect-Approx}$ repeatedly invokes $\texttt{Private-Sample-}B$ with $B(\alpha, \beta) = \alpha \wedge \beta$, for a maximum of M invocations. We call an invocation *positive* if it concludes with B evaluated as 1. If T invocations occur in $t < M$ rounds, the protocol outputs T/t and halts. Otherwise (after M invocations) the protocol outputs 0 and halts.

The random variable t is the sum of T independent geometric random variables. Hence, $\mathbf{E}[t] = T/p$ and $\mathbf{Var}[t] = T(1-p)/p^2$, where $p = v/N$. Using the Chebyshev Inequality, we get that $\Pr\left[|t - T/p| \geq \beta T/p\right] \leq \left(T\frac{1-p}{p^2}\right) \Big/ \left((\beta\frac{T}{p})^2\right) \leq \frac{1}{\beta^2 T}$. Let $\beta = \frac{\epsilon}{1+\epsilon}$, taking $T = \frac{2}{\beta^2\delta}$ ensures that, if T positive invocations occur, then the protocol's output is within $(1-\epsilon)\frac{v}{N}$ and $(1+\epsilon)\frac{v}{N}$, except for $\delta/2$ probability. To complete the protocol, we set $M = N(\ln\delta + 1)$ so that if $v \neq 0$, the probability of not having T positive invocations is at most $\delta/2$.

Note that the number of rounds in protocol `Intersect-Approx` is not fixed, and depends on the exact intersection size v. The protocol is optimal in the sense that it matches the lower-bound for distinguishing inputs with intersection size k from inputs with intersection size $k(1+\epsilon)$ in an expected $O(N/k)$ invocations of `Private-Sample-`B.

Caveat. As the number of rounds in our protocol is a function of its outcome, an observer that only counts the number of rounds in the protocol, or the time it takes to run it, may estimate its outcome. The problem is inherent in our security definitions—both for semi-honest and malicious parties—as they only take into account the parties that "formally" participate in the protocol (unlike, *e.g.*, in universal composability [6]). In particular, these definitions allow for any information that is learned by all the participating parties to be sent in the clear. While it may be that creating secure channels for the protocol (*e.g.*, using encryption) prevents this leakage in many cases, this is not a sufficient measure in general nor specifically for our protocol (as one must hide the communication length of `Intersect-Approx`).

7 The Multi-Party Case

We briefly discuss computing the intersection in a multi-party environment. Assume that there are n parties, $P_1, \ldots, P_n$, with corresponding lists of inputs $X_1, \ldots, X_n$; w.l.o.g., we assume each list contains k inputs. The parties compute the intersection of *all* n lists. We only sketch a protocol for semi-honest parties, starting with a basic protocol that is secure with respect to client parties $P_1, \ldots, P_{n-1}$ and then modifying it get security with respect to all parties.

A strawman protocol. Let client parties $P_1, \ldots, P_{n-1}$ each generate a polynomial encoding their input, as for Protocol `PM-Semi-Honest` in the two-party case. Each client uses her own public key and sends the encrypted polynomials to P_n, which we refer to as the *leader*. This naming of parties as *clients* and the *leader* is done for conceptual clarity.

For each item y in his list, leader P_n prepares $(n-1)$ random shares that XOR to y. He then evaluates the $(n-1)$ polynomials he received, encoding the lth share of y as the payload of the evaluation of the lth polynomial. Finally, he publishes a shuffled list of $(n-1)$-tuples. Each tuple contains the encryptions that the leader obtained while evaluating the polynomials on input y, for every y in his input set. Note that every tuple contains exactly one entry encrypted with the key of client P_l, for $1 \leq l \leq n-1$.

To obtain the outcome, each client P_l decrypts the entries that are encrypted with her public key and publishes them. If XOR-ing the decrypted values results in y, then y is in the intersection.

Achieving security with respect to semi-honest parties. This strawman approach is flawed. The leader P_n generates the shares that the clients decrypt. Hence, he may recognize, for values y in his set but not in the intersection, which clients also hold y: these clients, and only these clients, would publish the right shares. We can fix this problem by letting each client generate k sets of random shares that XOR to zero (one set for each of the leader's inputs). Then, each client encrypts one share from each set to every other client. Finally, the clients publish the XOR of the original share from the leader with the new shares from other clients. If y is in the intersection set, then the XOR of all published values for each of the leader's k inputs is still y, otherwise it looks random to any coalition. More concretely, the protocol for semi-honest parties is as follows.

1. A client party P_i, for $1 \leq i \leq n-1$, operates as in the two-party case. She generates a polynomial Q_i of degree k encoding her inputs, and generates homomorphic encryptions of the coefficients (with her own public key). P_i also chooses k sets of $n-1$ random numbers, call these $\{s^i_{j,1}, \ldots, s^i_{j,n-1}\}_{j=1}^{k}$. We can view this as a matrix with k rows and $(n-1)$ columns: Each column corresponds to the values given to party P_l; each row corresponds to the random numbers generated for one of the leader's inputs. This matrix is chosen such that the XOR of each row sums to zero, *i.e.*, for $j=1\ldots k$, $\bigoplus_{l=1}^{n-1} s^i_{j,l} = 0$. For each column l, she encrypts the corresponding shares using the public key of client P_l. She sends all her encrypted data to a public bulletin board (or just to the leader who acts in such a capacity).
2. For each item y in his list X_n (the rows), leader P_n prepares $(n-1)$ random shares $\sigma_{y,l}$ (one for each column), where $\bigoplus_{l=1}^{n-1} \sigma_{y,l} = y$. Then, for each of the k elements of the matrix column representing client P_l, he computes the encryption of $(r_{y,l} \cdot Q_l(y) + \sigma_{y,l})$ using P_l's public key and a fresh random number $r_{y,l}$. In total, the leader generates k tuples of $(n-1)$ items each. He randomly permutes the order of the tuples and publishes the resulting data.
3. Each client P_l decrypts the n entries that are encrypted with her public key: namely, the lth column generated by P_n (of k elements) and the $(n-1)$ lth columns generated by clients (each also of k elements). P_l computes the XOR of each row in the resulting matrix: $(\bigoplus_{i=1}^{n-1} s^i_{j,l}) \oplus \sigma_{j,l}$. She publishes these k results.
4. Each P_i checks if the XOR of the $(n-1)$ published results for each row is equal to a value y in her input: If this is the case, $\bigoplus_{l=1}^{n-1} \left((\bigoplus_{i=1}^{n-1} s^i_{j,l}) \oplus \sigma_{j,l} \right) = y$, and she concludes that y is in the intersection.

Intuitively, the values output by each client (Step 3) appear random to the leader, so he cannot differentiate between the output from clients with y in their input and those without, as he could in the strawman proposal.

Note that the communication involves two rounds in which $P_1, \ldots P_{n-1}$ submit data, and a round where P_n submits data. This is preferable to protocols consisting of many rounds with n^2 communication. The computation overhead of P_n can be improved by using the hashing-to-bins method of Section 4.2.

8 Fuzzy Matching and Fuzzy Search

In many applications, database entries are not always accurate or full (*e.g.*, due to errors, omissions, or inconsistent spellings of names). In these cases, it would be useful to have a private matching algorithm that reports a match even if two entries are only *similar*.

We let each database entry be a list of T attributes, and consider $X = (x_1, \ldots, x_T)$ and $Y = (y_1, \ldots, y_T)$ similar if they agree on (at least) $t < T$ attributes. One variant is fuzzy search, where the client specifies a list of attributes and asks for all the database entries that agree with at least t of the attributes. This may be achieved by a simple modification of our basic `PM-Semi-Honest` protocol, by letting the server reply with the encryptions of $r_i \cdot P_i(y_i) + s_i$, where t shares of $s_1, \ldots, s_T$ are necessary and sufficient for recovering Y. This fuzzy search scheme may be used to compare two "databases" each containing just one element comprised of many attributes.

The protocol may be modified to privately compute fuzzy matching in larger databases, *e.g.*, when a match is announced if entries agree on t out of T attributes. In this section, we present a scheme, in the semi-honest model, that considers a simple form of this fuzzy private matching problem.

A 2-out-of-3 fuzzy matching protocol A client $\mathcal{C}$ has $k_\mathcal{C}$ 3-tuples $X_1, \ldots, X_{k_\mathcal{C}}$. Let P_1, P_2, P_3 be polynomials, such that P_j is used to encode the jth element of the three tuple, X_i^j, for $1 \leq i \leq k_\mathcal{C}$. For all i, let $\mathcal{C}$ choose a new random value R_i and set $R_i = P_1(X_i^1) = P_2(X_i^2) = P_2(X_i^3)$. In general, the degree of each such polynomial is $k_\mathcal{C}$, and therefore, two non-equal polynomials can match in at most $k_\mathcal{C}$ positions. $\mathcal{C}$ sends (P_1, P_2, P_3) to $\mathcal{S}$ as encrypted coefficients, as earlier. The server $\mathcal{S}$, for every three-tuple Y_i in his database of size $k_\mathcal{S}$, responds to $\mathcal{C}$ in a manner similar to Protocol `PM-Semi-Honest`: He computes the encrypted values $r(P_1(Y_i^1) - P_2(Y_i^2)) + Y_i$, $r'(P_1(Y_i^2) - P_3(Y_i^3)) + Y_i$, and $r''(P_1(Y_i^1) - P_3(Y_i^3)) + Y_i$. If two elements in Y_i are the same as those in X_i, the client receives Y_i in one of the entries.

We leave as open problems the design of more efficient fuzzy matching protocols (without incurring a $\binom{T}{t}$ factor in the communication complexity) and of protocols secure in the malicious model.

References

1. Bill Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *Advances in Cryptology—EUROCRYPT 2001*, Innsbruck, Austria, May 2001.
2. Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, 1999.
3. Fabrice Boudot, Berry Schoenmakers, and Jacques Traore. A fair and efficient solution to the socialist millionaires' problem. *Discrete Applied Mathematics*, 111(1-2):23–036, 2001.
4. Andrei Z. Broder and Michael Mitzenmacher. Using multiple hash functions to improve ip lookups. In *IEEE INFOCOM'01*, pages 1454–1463, Anchorage, Alaska, April 2001.
5. Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *Advances in Cryptology—EUROCRYPT '99*, pages 402–414, Prague, Czech Republic, May 1999.

6. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science*, pages 136–145, Las Vegas, Nevada, October 2001.
7. Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *4th International Workshop on Practice and Theory in Public Key Cryptosystems (PKC 2001)*, pages 13–15, Cheju Island, Korea, February 2001.
8. Alexandre Evfimievski, Johannes Gehrke, and Ramakrishnan Srikant. Limiting privacy breaches in privacy preserving data mining. In *Proc. 22nd ACM Symposium on Principles of Database Systems (PODS 2003)*, pages 211–222, San Diego, CA, June 2003.
9. Ronald Fagin, Moni Naor, and Peter Winkler. Comparing information without leaking it. *Communications of the ACM*, 39(5):77–85, 1996.
10. Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin Strauss, and Rebecca N. Wright. Secure multiparty computation of approximations. In *Automata Languages and Programming: 27th International Colloquim (ICALP 2001)*, pages 927–938, Crete, Greece, July 2001.
11. Oded Goldreich. Secure multi-party computation. In *Available at Theory of Cryptography Library,* `http://philby.ucsb.edu/cryptolib/BOOKS`, 1999.
12. Bernardo A. Huberman, Matt Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *Proc. ACM Conference on Electronic Commerce*, pages 78–86, Denver, Colorado, November 1999.
13. Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *Proc. 21st Annual ACM Symposium on Theory of Computing*, pages 44–61, Seattle, Washington, May 1989.
14. Bala Kalyanasundaram and Georg Schnitger. The probabilistic communication complexity of set intersection. *SIAM J. Discrete Mathematics*, 5(4):545–557, 1992.
15. Aggelos Kiayias and Moti Yung. Secure games with polynomial expressions. In *Automata Languages and Programming: 27th International Colloquim (ICALP 2001)*, pages 939–950, Crete, Greece, July 2001.
16. Eyal Kushilevitz and Noam Nisan. *Communication Complexity.* Cambridge University Press, Cambridge, 1997.
17. Helger Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In *Advances in Cryptology—ASIACRYPT 2003*, pages 416–433, Taipei, Taiwan, November 2003.
18. Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proc. 31st Annual ACM Symposium on Theory of Computing*, pages 245–254, Atlanta, Georgia, May 1999.
19. Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SIAM Symposium on Discrete Algorithms (SODA)*, pages 448–457, Washington, D.C., January 2001.
20. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology—EUROCRYPT '99*, pages 223–238, Prague, Czech Republic, May 1999.
21. Pascal Paillier. Trapdooring discrete logarithms on elliptic curves over rings. In *Advances in Cryptology—ASIACRYPT 2000*, pages 573–584, Kyoto, Japan, 2000.
22. Alexander A. Razborov. Application of matrix methods to the theory of lower bounds in computational complexity. *Combinatorica*, 10(1):81–93, 1990.