Efficient Set Intersection with Simulation-Based Security

Michael J. Freedman[*] Carmit Hazay[†] Kobbi Nissim[‡] Benny Pinkas[§]

September 4, 2014

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. In this work we present protocols based on the use of homomorphic encryption and different hashing schemes for both the semi-honest and malicious environments. 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. Our protocols obtain linear communication and computation overhead. We further implement different variants of our semi-honest protocol. Our experiments show that the asymptotic overhead of the protocol is affected by different constants. (In particular, the degree of the polynomials evaluated by the protocol matters less than the number of polynomials that are evaluated.) As a result, the protocol variant with the best asymptotic overhead is not necessarily preferable for inputs of reasonable size.

1 Introduction

In the setting of secure two-party computation, two parties with private inputs wish to jointly compute some function of their inputs while preserving certain security properties like privacy, correctness and more. The standard way of defining security in this setting is via the so-called ideal/real model paradigm [Bea91, GL90, MR91, Can00]. Here, an ideal model is first defined where an uncorrupted trusted party is used to compute the function for the parties. Then, a real protocol is said to be secure if no adversary can do more harm in a real protocol execution than in an ideal one (where by definition no harm can be done). Starting with the work of [Yao82, GMW87, CCD88, BGW88], it is by now well known that (in various settings, and considering semi-honest and malicious adversaries) any polynomial-time computation can be generically compiled into a secure function evaluation protocol with polynomial complexity. However, more often than not, the resulting protocols are inefficient for practical uses and hence attention was given to constructing efficient protocols for specific functions.

In this work we consider the basic two-party set intersection problem, where two parties P_1 and P_2 hold input sets X, Y, respectively, and wish to compute $X \cap Y$. The secure variant of this computation guarantees that nothing beyond $X \cap Y$ leaks within the protocol execution. This problem has been widely studied due

[*] Dept. of Computer Science, Princeton University. Email: mfreed@cs.princeton.edu.

[†] Faculty of Engineering, Bar-Ilan University. Email: hazay@biu.ac.il. Research partially supported by a grant from the Israel Ministry of Science and Technology (grant No. 3-10883).

[‡] Dept. of Computer Science, Ben-Gurion University. Email: kobbi@cs.bgu.ac.il. Research partially supported by the Israel Science Foundation (grant No. 860/06).

[§] Dept. of Computer Science, Bar Ilan University. Email: benny@pinkas.net. Research partially supported by the European Unions 7th Framework Program (FP7/2007-2013) under grant agreement n. 609611 (PRACTICE), and by a grant from the Israel Ministry of Science and Technology (grant No. 3-9094).

to its extensive usage 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 easily envision other potential applications for secure set intersection such as online recommendation services, online dating services, medical databases, and many other applications.

We continue with a survey of the current state of affairs with respect to secure two-party computation of set intersection.

1.1 Background and Related Work

Private equality tests (PET). The simplest non-trivial form of set intersection is where each of the two datasets consists of a single element. A circuit computing this function has $O(\log N)$ gates (where N is the domain size) and therefore can be securely evaluated with this overhead. Specialized protocols for this function, denoted by private equality test (PET) or the socialist millionaires problem, were also suggested in [FNW96, NP99, BST01, Lip03] with essentially the same overhead. We note that the solution in [BST01] also provides fairness. A related problem is that of private authenticated key exchange (PAKE), in which two parties agree on a secure key if both share the same password, taken from a small domain, see [KOY01] and references within.

Private set intersection. The following intuitive "solution" is often suggested for private set intersection. Assume the two parties agree on some cryptographic function $H(\cdot)$ such as a one-way hash function or a pseudorandom function. Alice sends to Bob the results of applying $H(\cdot)$ to each element of her input set. Bob then compares these values to the results of applying the function to the elements in his input set. If Bob identifies that, for an input x of his, the value $H(x)$ appears in the list of values sent by Alice, then x must be in the intersection. This simple solution is unfortunately *insecure*. The reason is that given Alice's hashed values Bob can test whether an element x appears in her set by searching for $H(x)$ in Alice's hashed set. In particular, when Alice's set comes from a polynomial domain Bob can recover her entire input set.

Denoting by n the number of elements in each dataset and by N the domain size from which the elements are picked, a trivial solution compares all combinations of items from the two datasets using n^2 instantiations of a PET protocol (that incurs $O(\log N)$ overhead). The computation of this comparison can be reduced to $O(n \log N)$, while retaining the $O(n^2 \log N)$ communication overhead [NP99]. A circuit with a smaller asymptotic size can be constructed by sorting the inputs of the two parties using a sorting network and then comparing every two adjacent items. The size of a sorting network is $O(n \log n)$ comparators, when using the AKS sorting network [AKS83] (which has a large constant factor) or $O(n \log^2 n)$ comparators when using the Batcher sorting network [Bat68], whose overhead induces a more reasonable constant factor. Since our circuit has to handle N bit long inputs, the size of the circuit would be $O(n \log n \log N)$ when using the AKS sorting network or $O(n \log^2 n \log N)$ when using the Batcher sorting network. The $\log N$ factor can be somewhat reduced using Hashing.

These solutions can be implemented securely by applying Yao's construction of garbled circuit [Yao82]. For the semi-honest setting this results in a protocol with communication complexity of the same order as that of the circuit's size times the security parameter [LP09]. Coping with malicious adversaries is more challenging and requires additional tools for ensuring correctness such as cut-and-choose [LP12] or zero-knowledge proofs [JS07, DPSZ12, DZ13], which inflate the communication/computation costs. In this work we avoid both costly techniques.

The first concrete construction solving the set intersection problem securely had a cost of $O(n)$ exponentiations [HFH99, EGS03]. However, these constructions were only analyzed in the random oracle model against semi-honest parties and did not provide a full proof with simulation based security. In [FNP04], Freedman et al. studied set intersection in the standard model and presented a construction for the semi-honest setting, utilizing oblivious polynomial evaluation and balanced allocation hash functions. Their protocol exhibits linear communication and $O(n \log \log n)$ computation overhead (counting modular exponentiations). In addition to their semi-honest protocol Freedman et al. presented variants of the above protocol for the case where one of the parties is malicious and the other is semi-honest. The protocol for a malicious P_1 and a semi-honest P_2 utilizes a cut-and-choose strategy and therefore the communication/computation costs are enhanced by a statistical security parameter.

Kissner and Song [KS05] used polynomials to represent multi-sets. Letting the roots of $Q_X(\cdot)$ and $Q_Y(\cdot)$ coincide with elements of the multi-sets X and Y. They observed that if $r(\cdot), s(\cdot)$ are polynomials chosen at random then the roots of $r(\cdot) \cdot Q_X(\cdot) + s(\cdot) \cdot Q_Y(\cdot)$ coincide with high probability with the multi-set $X \cap Y$. Their result is a set intersection protocol for the semi-honest case, where the parties use an additively homomorphic encryption scheme to perform the polynomial multiplication, introducing quadratic computation costs in the set sizes. For the security of the protocol, it is crucial that no party should be able to decrypt on its own. Hence, the secret key must be shared and joint decryption must be deployed. For malicious parties [KS05] introduced generic zero-knowledge proofs for proving adherence to the prescribed protocol (e.g., zero-knowledge proofs of knowledge for the multiplication of the encrypted $Q_x(\cdot)$ with a randomly selected $r(\cdot)$). The costs of these proofs as well as those of setting the shared key for the Paillier scheme are not specified explicitly and can be relatively high. An improved protocol for the malicious setting using more efficient zero-knowledge proofs was presented in [CJS10].

Another tool that can be used for computing set intersection is secure implementation of oblivious pseudorandom function evaluation (OPRF). Namely, having one party hold the keys to a PRF $f_k()$ while enabling the other party, who has an input x, to compute $f_k(x)$ without learning anything else. The idea of using this tool for set intersection was first described in [FIPR05]. In [HL08] Hazay and Lindell presented two OPRF based protocols for set intersection, one achieving security in the presence of malicious adversaries with one-sided simulatability, whereas the other is secure in the presence of covert adversaries [AL07]. Having P_1, P_2 hold sets of sizes m_1, m_2 respectively, both protocols in [HL08] are constant round and incur communication of $O(m_1 \log N + m_2)$ group elements and computation of $O(m_1 \log N + m_2)$ modular exponentiations. We note that the protocols in [HL08] can be made secure in the malicious setup by introducing a secure key selection step for the oblivious PRF and by adding zero-knowledge proofs of knowledge to show correctness at each step. Namely, for proving that the *same* PRF key is used by party P_1 for all PRF evaluations and to enable the extraction of the preimages (as a pseudorandom function is not necessarily invertible). While this would preserve the complexity of these protocols asymptotically (in m_1, m_2), introducing such proofs would probably make this protocol impractical since there is no efficient known way to design such proofs.

Jarecki and Liu [JL09] generalized the technique of [HL08] and presented a very efficient protocol for computing a pseudorandom function with a committed key (informally, this means that the same key is used in all invocations), and showed that it yields an efficient set intersection protocol. The main restriction of this construction is that the input domain size of the PRF must be polynomial in the security parameter (since the proof of security for the set intersection protocol makes use of the ability to exhaustively search over the input domain). Their protocol is secure in the CRS model with a safe RSA modulus placed in the CRS, and relies on the Decisional q-Diffie-Hellman Inversion assumption. In a followup work [JL10], Jarecki and Liu presented a protocol for set intersection that is secure against malicious adversaries under the interactive One-More Gap Diffie-Hellman assumption in the random oracle model. Their protocol computes an adaptive variant of set intersection for which a receiver is allowed to make adaptive queries, each time revealing whether an item y_i belongs to a set X. On the other hand, their protocol takes only one round of interaction and its total computational cost is under $3(|X| + |Y|)$ exponentiations (which is better than our protocol presented here).

Dachman-Soled et al. [DSMRY09] presented a protocol for set intersection in the presence of malicious adversaries without restricting the domain. Their construction uses polynomial evaluation and secret sharing of the inputs. They avoid generic zero-knowledge by utilizing the fact that Shamir's secret sharing implies Reed Solomon code. Their protocol incurs communication of $O(nk^2 \log^2 n + kn)$ group elements and $O(n^2 k \log n + nk^2 \log^2 n)$ exponentiations where k is the security parameter.

Finally, Hazay and Nissim [HN10] investigated protocols in the malicious setting for constructing efficient secure two-party protocols for set intersection and set union. They designed constant-round protocols that exhibit linear communication and a (practically) linear number of exponentiations with simulation based security. More explicitly, they get that for sets $X, Y \subseteq \{0,1\}^{\log N}$ of m_1, m_2 elements respectively, the costs are of sending $O(m_1 + m_2 \log N)$ group elements, and the computation of $O(m_1 + m_2(\log\log m_1 + \log N))$ modular exponentiations. In the heart of these constructions is a technique based on a combination of a perfectly hiding commitment and an oblivious pseudorandom function evaluation protocol with the aim to replace the random oracle used in [FNP04]. Their work does not consider Cuckoo hashing but our ideas can be applied to their construction as well.

Other variants of the problem were also investigated. Ateniese et al. [ACT11] discussed size-hiding set intersection, where one of the parties can hide the size of its set. Camenisch and Zavrucha [CZ09] investigated the problem of set intersection inputs that are certified by a third party.

Disjointness and set intersection. Much attention has been given to bounding the communication complexity of the disjointness function, defined as

$$\textsc{Disj}(A, B) = 1 \text{ if } A \cap B = \phi.$$

It is known [KS92, Raz90] that if A, B can be arbitrary subsets of $[n]$ then the randomized communication complexity of the disjointness function is $\Theta(n)$. An immediate implication is that computing the intersection of two sets $|X| = |Y| = n$ (over a large enough domain) requires $\Omega(n)$ communication. This follows by a reduction from $\textsc{Disj}(\cdot,\cdot)$ to set intersection over domain of size $N \geq 3n$.[1] Let $Z = \{z_1, \ldots, z_{3n}\}$ be a subset of the domain and let ϕ be a one-to-one mapping from $[n]$ to $\{z_1, \ldots, z_n\}$. Given a subset $A \subseteq [n]$ let $\hat{X} = \{\phi(a) : a \in A\}$ and set $X = \hat{X} \cup \{z_{n+1}, \ldots, z_{2n-|\hat{X}|}\}$. Similarly, Given $B \subseteq [n]$ let $\hat{Y} = \{\phi(b) : b \in B\}$ and set $Y = \hat{Y} \cup \{z_{2n+1}, \ldots, z_{3n-|\hat{Y}|}\}$. Note that $|X \cap Y| = |\hat{X} \cap \hat{Y}| = |A \cap B|$ and hence $\textsc{Disj}(A, B) = 1$ iff $X \cap Y = \emptyset$. We thus obtain a lower bound of $\Omega(n)$ on the communication needed for computing the intersection even without taking privacy and security into consideration.

Set intersection and oblivious transfer. The bit oblivious transfer functionality is defined as

$$((b_0, b_1), \sigma) \mapsto (\lambda, b_\sigma),$$

where $(b_0, b_1) \in \{0,1\} \times \{0,1\}$ is the sender's input, σ is the receiver's input bit and λ denotes the empty string (meaning that the sender has no output). A simple reduction from oblivious transfer (OT) to set intersection shows that implementing set intersection implies implementing oblivious transfer. Given its input (b_0, b_1) the sender generates a set of two strings $X = \{0|b_0, 1|b_1\}$. The receiver generates the set $Y = \{\sigma|0, \sigma|1\}$. The parties run the set intersection protocol on X, Y where at the end the receiver learns $X \cap Y = \{\sigma|b_\sigma\}$. By the results of Impagliazzo and Rudich [IR89] it follows that there is no black-box reduction of oblivious transfer to one-way functions, and therefore the same holds with regards to reductions of set intersection to one-way functions. Indeed, our set intersection protocols use a stronger primitive – an additively homomorphic encryption scheme.

[1] The reduction is to a domain of size $3n$ in order to ensure that the inputs to the set intersection problem are always of size $\Theta(n)$. This simplifies the description of the result that is proved by the reduction.

1.2 Our Contributions

This paper is an extended and improved version of [FNP04]. We present secure protocols for set intersection in the presence of semi-honest and malicious adversaries with linear costs (with respect to the sets sizes). Our results include the following,

1.2.1 Protocols for Computing Set Intersection

These protocols employ a homomorphic encryption scheme, and in particular the Paillier or ElGamal encryption schemes: (**i**) a protocol with security against semi-honest adversaries (cf. Section 3.1); and (**ii**) a protocol in the random oracle model with security against malicious adversaries (cf. Section 5). Our protocols have simulation based security (unlike the protocols in [FNP04]), assuming the hardness of DDH/DCR problems. Moreover, they introduce linear (or nearly linear) computation and communication overheads with small constant factors, where the analysis depends on the type of the hash scheme we use. The most efficient result is achieved using a new protocol based on Cuckoo hashing. We further analyze efficiency based on simple hashing and balanced allocation schemes. Our analysis is presented in details in Section 3.2.

The semi-honest setting. The high-level description of our semi-honest protocol follows by having party P_1 generate a polynomial $Q(\cdot)$ of degree m_1, with roots set to the m_1 elements of X, and send the encrypted coefficients to P_2 (using a homomorphic encryption). Then, for each element $y \in Y$, P_2 replies with the encryption of $r \cdot Q(y) + y$ for a random r. This immediately implies that for $y \in X \cap Y$ the result plaintext would be y. Otherwise, the plaintext equals a random value that does not leak any information about y. Note first that the communication complexity of this protocol is linear in $m_1 + m_2$, yet the work performed by P_2 is high, as each of the m_2 oblivious polynomial evaluations includes performing $O(m_1)$ exponentiations totaling in $O(m_1 \cdot m_2)$ exponentiations.

To save on computational work we use hashing to map the items into different bins. In that case the items mapped by P_1 to a certain bin must only be compared to those mapped by P_2 to the same bin. Thus the number of comparisons can be reduced to be in the order of the number of P_2's inputs times the maximum number of items mapped to a bin. In this work, we describe and compare modifications of the basic protocol based on the following different hash schemes: simple hashing, balanced allocations, and Cuckoo hashing. We also provide in Section 4 results of experiments with each of these schemes.

The malicious setting. Introducing security in the malicious setting raises new concerns. First, the basic scheme introduced above is not secure any longer in the malicious setting since, for instance, a malicious P_2 can compute the encryption of $r \cdot Q(y) + y'$ for distinct y and y' (which implies the ciphertext is decrypted to y' if and only if $y \in X$). In addition, using hashing introduces new attacks since we must ensure that P_1 computes the small polynomials correctly and that its input is well defined.

Our protocol for the malicious setting avoids the standard solutions that involve zero-knowledge proofs or the cut-and-choose technique for demonstrating correct behavior. Instead, it enables party P_1 to redo the entire computation supposedly carried out by P_2 on each element and verify that its outcome is consistent with the messages received from P_2. In the proof we show that the probability that P_2 convinces P_1 of a correct behavior even though it is not the case, is negligible. This technique is implemented in the random oracle model. Importantly, we *do not* rely on the programmability property of the random oracle which weakens the security random oracle notion that we require. In particular, our simulator only need to observe the adversary's random oracle queries.

We note that using hash functions to reduce communication cost in the malicious setting introduces new problems as the parties must prove that they used a correct mapping for each element without leaking anything about it. Our solution also deals with this challenge, ensuring that each element is mapped to the correct bin. A more subtle problem that we deal with that was overlooked in prior work, is that with some homomorphic PKEs P_1 may construct $Q(\cdot)$ such that the evaluation of $r \cdot Q(y)$ (and hence also of $r \cdot Q(y) + y$) is far from being random in the plaintext space even though $Q(y) \neq 0$ and r is chosen at random. This attack can be carried out with respect to PKEs for which the plaintext space is *not* a cyclic group of prime order, implying that $r \cdot Q(y)$ may be a random element within a smaller subgroup.

Variants of set intersection. Finally, we present in Section 3.4 a protocol for computing the cardinality of the intersection. This protocol uses our protocol for computing the set intersection as a main building block. The computation/communication overheads do not change.

1.2.2 Experimental Results

We implement and test the different variants of the semi-honest protocol and analyze their overhead (cf. Section 4). These experiments are new to this work and were not conducted in [FNP04]. Throughout our experiments we gain some insights regarding the practicality of the cryptographic primitives we use in our constructions and the relation between the asymptotic and actual overhead of the protocols. Somewhat surprisingly, we get that the variant with the best asymptotic overhead is not necessarily preferable for inputs of reasonable size. In particular, the degree of the polynomials evaluated by the protocol matters less than the number of polynomials that are evaluated. More specifically, recalling that the random hashing construction evaluates a single polynomial of degree $O(\log m_1)$, whereas the balanced allocations construction evaluates two polynomials of degree $O(\log \log m_1)$ and the Cuckoo hashing construction evaluates three polynomials: two linear polynomials and a polynomial of degree 2. Asymptotically, the performance of the constructions based on balanced allocations and Cuckoo hashing is preferable, but since these two constructions use more polynomials than the first construction, their overhead is higher than that of random hashing for the input sizes that we tested.

1.2.3 A Roadmap

In Section 2 we present definitions and tools that are useful for our constructions. In Section 3 we present our first construction for the semi-honest setting and generalizations for two related problems. In Section 4 we present our performance evaluation, and in Section 5 we describe our protocol for the malicious setting and its proof.

2 Definitions and Tools

Basic notations. The security parameter is denoted by k and, although not explicitly specified, input lengths are always assumed to be bounded by some polynomial in k. A probabilistic machine is said to run in *polynomial-time* (PPT) if it runs in time that is polynomial in the security parameter k. A function $\mu(k)$ is called *negligible in* k (*negligible* for short) if for every polynomial $p(\cdot)$ there exists a value $k_0 = k_0(p)$ such that $\mu(k) < \frac{1}{p(k)}$ for all $k > k_0$; i.e., $\mu(k) = k^{-\omega(1)}$. Let $X = \{X(k,a)\}_{k \in \mathbb{N}, a \in \{0,1\}^*}$ and $Y = \{Y(k,a)\}_{k \in \mathbb{N}, a \in \{0,1\}^*}$ be distribution ensembles (over strings of length polynomial in k). We say that X and Y are *computationally indistinguishable*, denoted $X \overset{c}{\equiv} Y$, if for every polynomial non-uniform distinguisher D there exists a negligible $\mu(\cdot)$ such that

$$\Big|\Pr[D(X(k,a)) = 1] - \Pr[D(Y(k,a)) = 1]\Big| < \mu(k)$$

for every $k \in \mathbb{N}$ and $a \in \{0,1\}^*$.

2.1 Secure Two-Party Computation

We briefly present the standard definition for secure multiparty computation and refer to [Gol04, Chapter 7] for more details and motivating discussions.

Two-party computation. A two-party protocol problem is cast by specifying a random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a *functionality* and denote it $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. That is, for every pair of inputs (x, y), the output-vector is a random variable $(f_1(x, y), f_2(x, y))$ ranging over pairs of strings where P_1 receives $f_1(x, y)$ and P_2 receives $f_2(x, y)$. We use the notation $(x, y) \mapsto (f_1(x, y), f_2(x, y))$ to describe a functionality. For example, the Oblivious Transfer functionality is written $((x_0, x_1), \sigma) \mapsto (\lambda, x_\sigma)$, where (x_0, x_1) is the first party's input, σ is the second party's input, and λ denotes the empty string (meaning that the first party has no output). A special case for a two-party functionality is that of zero-knowledge proof of knowledge for a relation $\mathcal{R}_{\text{ZK}}$. This relation can be defined by the inputs $(x, (x, w))$ that are mapped into $(1, \lambda)$ if $\mathcal{R}_{\text{ZK}}(x, w) = 1$, or into $(\perp, \lambda)$ otherwise.

Security of protocols. We prove the security of our protocols in the settings of *semi-honest* and *malicious* computationally bounded adversaries. Loosely speaking, the adversary in the semi-honest setting is assumed to act according to its prescribed actions in the protocol, whereas in the malicious setting it may arbitrarily deviate from the specified protocol. Security is analyzed by comparing what an adversary can do in a *real* protocol execution to what it can do in an *ideal* scenario. In the ideal scenario, the computation involves an incorruptible *trusted third party* to whom the parties send their inputs. The trusted party computes the functionality on the inputs and returns to each party its respective output. Informally, the protocol is secure if any adversary interacting in the real protocol (i.e., where no trusted third party exists) can do no more harm than what it could do in the ideal scenario. There are technical issues that arise, such as that it may be impossible to achieve fairness or guaranteed output delivery. E.g., it is possible for the an adversarial party to prevent an honest party from receiving outputs.

2.1.1 The Semi-Honest Setting

In this model the adversary controls one of the parties and follows the protocol specification. However, it may try to learn more information than allowed by looking at the transcript of messages that it received and its internal state. The following definition is according to [Gol04].

Let $f = (f_1, f_2)$ be a two-party functionality and let π be a two-party protocol for computing f. The *view* of the first party in an execution of π on inputs (x, y) is

$$\mathbf{View}_{\pi,1}(x, y) = (x, r_1, m_1, \ldots, m_t),$$

where r_1 is the content of the first party's internal random tape, and m_i represents the ith message that it received. The *output* of the first party in an execution of π on (x, y) is denoted $\mathbf{Output}_{\pi,1}(x, y)$ and can be computed from $\mathbf{View}_{\pi,1}(x, y)$. Similarly, $\mathbf{View}_{\pi,2}(x, y)(y, r_2, m_1, \ldots, m_t)$ where r_2 is second party's randomness and m_i is the ith message it received. The output of the second party can be computed from her view and is denoted $\mathbf{Output}_{\pi,2}(x, y)$.

Definition 2.1 *Let f and π be as above. Protocol π is said to securely compute f in the presence of* semi-honest *adversaries if there exist probabilistic polynomial-time algorithms $\mathcal{S}_1$ and $\mathcal{S}_2$ such that*

$$(\mathcal{S}_1(x, f_1(k, x, y)), f_2(k, x, y))_{k\in\mathbb{N}, x,y\in\{0,1\}^*} \stackrel{c}{\equiv} \{(\mathbf{View}_{\pi,1}(k, x, y), \mathbf{Output}_{\pi,2}(k, x, y))\}_{k\in\mathbb{N}, x,y\in\{0,1\}^*}$$

$$(f_1(k,x,y), \mathcal{S}_2(y, f_2(k,x,y)))_{k\in\mathbb{N},x,y\in\{0,1\}^*} \overset{\text{c}}{\equiv} \{(\mathbf{Output}_{\pi,1}(k,x,y), (\mathbf{View}_{\pi,2}(k,x,y)))\}_{k\in\mathbb{N},x,y\in\{0,1\}^*}$$

where k is the security parameter.

2.1.2 The Malicious Setting

Execution in the ideal model. In an ideal execution, the parties submit inputs to a trusted party, that computes the output. An honest party receives its input for the computation and just directs it to the trusted party, whereas a corrupted party can replace its input with any other value of the same length. Since we do not consider fairness, the trusted party first sends the outputs of the corrupted parties to the adversary, and the adversary then decides whether the honest parties would receive their outputs from the trusted party or an *abort* symbol $\perp$. Let f be a two-party functionality where $f = (f_1, f_2)$, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine, and let $I \subset [2]$ be the set of corrupted parties (either P_1 is corrupted or P_2 is corrupted or neither). Then, the ideal execution of f on inputs (x,y), auxiliary input z to $\mathcal{A}$ and security parameter k, denoted $\mathbf{Ideal}_{f,\mathcal{A}(z),I}(k,x,y)$, is defined as the output pair of the honest party and the adversary $\mathcal{A}$ from the above ideal execution.

Execution in the real model. In the real model there is no trusted third party and the parties interact directly. The adversary $\mathcal{A}$ sends all messages in place of the corrupted party, and may follow an arbitrary polynomial-time strategy. The honest parties follow the instructions of the specified protocol π.

Let f be as above and let π be a two-party protocol for computing f. Furthermore, let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine and let I be the set of corrupted parties. Then, the real execution of π on inputs (x,y), auxiliary input z to $\mathcal{A}$ and security parameter k, denoted $\mathbf{Real}_{\pi,\mathcal{A}(z),I}(k,x,y)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the real execution of π.

Security as emulation of a real execution in the ideal model. Having defined the ideal and real models, we can now define security of protocols. Loosely speaking, the definition asserts that a secure party protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that adversaries in the ideal model are able to simulate executions of the real-model protocol.

Definition 2.2 *Let f and π be as above. Protocol π is said to* securely compute f with abort in the presence of malicious adversaries *if for every non-uniform probabilistic polynomial-time adversary $\mathcal{A}$ for the real model, there exists a non-uniform probabilistic polynomial-time adversary $\mathcal{S}$ for the ideal model, such that for every $I \subset [2]$,*

$$\left\{\mathbf{Ideal}_{f,\mathcal{S}(z),I}(k,x,y)\right\}_{k\in\mathbb{N},x,y,z\in\{0,1\}^*} \overset{\text{c}}{\equiv} \left\{\mathbf{Real}_{\pi,\mathcal{A}(z),I}(k,x,y)\right\}_{k\in\mathbb{N},x,y,z\in\{0,1\}^*}$$

where k is the security parameter.

The f-hybrid model. In our constructions we will use secure two-party protocols as sub-protocols. A standard way of abstracting out the details of the sub-protocols, is to work in a *"hybrid model"* where the two parties interact with each other (as in the real model) and also use trusted help (as in the ideal model). Specifically, an execution of a protocol π that uses a sub-protocol for securely computing some functionality f is modeled as if the parties run π and issue "ideal calls" to a trusted party for computing f instead of invoking the protocol for f. In these calls to f the parties send inputs to the trusted party, which, upon receiving the inputs from the parties, computes f and sends each party its corresponding output. After receiving these outputs the protocol π continues.

We stress that we use the $\boldsymbol{f}$-hybrid model in a *sequential* composition, i.e., the parties do not send messages in π between the time that they send input to the trusted party and the time that they receive back output. The trusted party may be used a number of times throughout the execution of π. Each time is *independent* in the sense that the trusted party does not maintain any state between these calls. We call the regular messages of π that are sent amongst the parties standard messages and the messages that are sent between parties and the trusted party ideal messages.

Let f be a functionality and let π be a two-party protocol that uses ideal calls to a trusted party computing f. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time machine and let I be the set of corrupted parties. Then, the f-hybrid execution of π on inputs (x, y), auxiliary input z to $\mathcal{A}$ and security parameter k, denoted $\mathbf{Hybrid}^{f}_{\pi,\mathcal{A}(z),I}(k, x, y)$, is defined as the output vector of the honest parties and the adversary $\mathcal{A}$ from the hybrid execution of π with a trusted party computing f.

Let f and π be as above, and let ρ be a protocol. Consider the real protocol π^{ρ} that is defined as follows. All standard messages of π are unchanged. When a party P_i is instructed to send an ideal message α_i to the trusted party, it begins a real execution of ρ with input α_i instead. When this execution of ρ concludes with output β_i, party P_i continues with π as if β_i was the output received by the trusted party (i.e. as if it were running in the f-hybrid model). Then, the composition theorem of [Can00] states that if ρ securely computes f, then the output distribution of a protocol π in a hybrid execution with f is computationally indistinguishable from the output distribution of the real protocol π^{ρ}. Thus, it suffices to analyze the security of π when using ideal calls to f; security of the real protocol π^{ρ} is derived via this composition theorem.

2.2 Hardness Assumptions

Our constructions rely on the following hardness assumptions.

Definition 2.3 (DDH) *We say that* the decisional Diffie-Hellman (DDH) problem is hard relative to $\mathbb{G} = \{\mathbb{G}_k\}$ *if for all polynomial-sized circuits* $\mathcal{A} = \{\mathcal{A}_k\}$ *there exists a negligible function* negl *such that*

$$\left| \Pr\left[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1\right] - \Pr\left[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1\right] \right| \leq \mathsf{negl}(k),$$

where q is the order of $\mathbb{G}$ and the probabilities are taken over the choices of g and $x, y, z \in \mathbb{Z}_q$.

We require the DDH assumption to hold for prime order groups.

Definition 2.4 (DCR) *We say that* the decisional composite residuosity (DCR) problem is hard *if for all polynomial-sized circuits* $\mathcal{A} = \{\mathcal{A}_k\}$ *there exists a negligible function* negl *such that*

$$\left| \Pr\left[\mathcal{A}(N, z) = 1 | z = y^N \bmod N^2\right] - \Pr\left[\mathcal{A}(\mathcal{A}(N, z) = 1 | z = (1+N)^r \cdot y^N \bmod N^2\right] \right| \leq \mathsf{negl}(k),$$

where N is a random k-bit RSA composite, r is chosen at random in $\mathbb{Z}_N$, and the probabilities are taken over the choices of N, y and r.

2.3 Public Key Encryption Schemes

We begin with the definitions of public key encryption and semantic security. We then specify the definition of homomorphic encryption and two encryption schemes that meet this definition.

Definition 2.5 (PKE) *We say that* $\Pi = (G, E, D)$ *is a* public key encryption scheme *if G, E, D are polynomial-time algorithms specified as follows:*

- *G, given a security parameter n (in unary), outputs keys (pk, sk), where pk is a public key and sk is a secret key. We denote this by $(pk, sk) \leftarrow G(1^k)$.*
- *E, given the public key pk and a plaintext message m, outputs a ciphertext c encrypting m. We denote this by $c \leftarrow E_{pk}(m)$; and when emphasizing the randomness r used for encryption, we denote this by $c \leftarrow E_{pk}(m; r)$.*
- *D, given the public key pk, secret key sk and a ciphertext c, outputs a plaintext message m s.t. there exists randomness r for which $c = E_{pk}(m; r)$ (or $\perp$ if no such message exists). We denote this by $m \leftarrow D_{pk,sk}(c)$.*

For a public key encryption scheme $\Pi = (G, E, D)$ and a non-uniform adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we consider the following *Semantic security game*:

$$\begin{aligned}
&(pk, sk) \leftarrow G(1^k).\\
&(m_0, m_1, history) \leftarrow \mathcal{A}_1(pk), \text{ s.t. } |m_0| = |m_1|.\\
&c \leftarrow E_{pk}(m_b), \text{ where } b \leftarrow_R \{0, 1\}.\\
&b' \leftarrow \mathcal{A}_2(c, history).\\
&\mathcal{A} \text{ wins if } b' = b.
\end{aligned}$$

Denote by $\mathsf{Adv}_{\Pi,\mathcal{A}}(n)$ the probability that $\mathcal{A}$ wins the semantic security game.

Definition 2.6 (Semantic security) *A public key encryption scheme $\Pi = (G, E, D)$ is* semantically secure*, if for every polynomial non-uniform adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ there exists a negligible function* negl *such that $\mathsf{Adv}_{\Pi,\mathcal{A}}(k) \le \frac{1}{2} + \mathsf{negl}(k)$.*

2.3.1 Building Blocks: Additively Homomorphic PKE

Intuitively, a public key encryption scheme is additively homomorphic if given two ciphertexts $c_1 = E_{pk}(m_1; r_1)$ and $c_2 = E_{pk}(m_2; r_2)$ it is possible to efficiently compute $E_{pk}(m_1 + m_2; r)$ with independent r, and without the knowledge of the secret decryption key. Clearly, this assumes that the plaintext message space is a group; we actually assume that both the plaintext and ciphertext spaces are groups (with respective group operations + and $\cdot$). We abuse notation and use $E_{pk}(m)$ to denote the random variable induced by $E_{pk}(m; r)$ where r is chosen uniformly at random. We have the following formal definition,

Definition 2.7 (Homomorphic PKE) *A public key encryption scheme (G, E, D) is* homomorphic *if for all k and all (pk, sk) output by $G(1^k)$, it is possible to define groups $\mathbb{M}, \mathbb{C}$ such that:*

- *The plaintext space is $\mathbb{M}$, and all ciphertexts output by $E_{pk}(\cdot)$ are elements of $\mathbb{C}$.*[2]
- *For every $m_1, m_2 \in \mathbb{M}$ it holds that*

$$\{pk, c_1 = E_{pk}(m_1), c_1 \cdot E_{pk}(m_2)\} \equiv \{pk, E_{pk}(m_1), E_{pk}(m_1 + m_2)\} \tag{1}$$

where the group operations are carried out in $\mathbb{C}$ and $\mathbb{M}$, respectively and the randomness for the distinct ciphertexts are independent.

Note that any such scheme supports a multiplication of a plaintext by a scalar, that can be achieved by computing multiple additions. In particular, this homomorphic property allows oblivious evaluation of any polynomial $Q(x) = \sum_i Q_i x^i$, given its encrypted coefficients $\{E_{pk}(Q_i)\}_i$.

[2] The plaintext and ciphertext spaces may depend on pk; we leave this implicit.

The Paillier PKE. An example of an encryption scheme that meets Definition 2.7 is the encryption scheme of Paillier [Pai99]. In this scheme, the public key pk is an RSA composite N and the corresponding secret key sk is $\phi(N)$. Then, given a message $m \in \mathbb{Z}_N$, the encryption procedure selects $r \leftarrow_R \mathbb{Z}_N^*$ (in practice $r \leftarrow_R \mathbb{Z}_N$) and computes $E_{pk}(m;r) = (1+N)^m \cdot r^N \mod N^2$. Conversely, given a ciphertext $c \in \mathbb{Z}_{N^2}^*$, the decryption procedure computes

$$D_{sk}(c) = \frac{[c^{\phi(N)} \mod N^2] - 1}{N} \cdot \phi(N)^{-1} \mod N.$$

This scheme is semantically secure, assuming hardness of the decisional composite residuosity problem. Note that, the Paillier scheme is homomorphic with respect to addition modulo N, as $D_{sk}[E_{pk}(m_1) \cdot E_{pk}(m_2)] = m_1 + m_2 \mod N$. Furthermore, $D_{sk}[(E_{pk}(m))^c \mod N^2] = cm \mod N$. Hence, given encryptions $\{c_i\}$ of messages $\{m_i\}$ and pk one can compute an encryption of any linear combination $\sum_i \alpha_i \cdot m_i \mod N$.

The ElGamal PKE. Another encryption scheme suitable for our needs is the ElGamal encryption [ElG85]. Namely, let $\mathbb{G}$ be a group generated by g of prime order q, in which the decisional Diffie-Hellman (DDH) problem is hard. A public key is then a pair $pk = \langle g, h \rangle$ and the corresponding secret key is $s = \log_g(h)$, i.e. $g^s = h$. Then, an encryption of a message $m \in \mathbb{Z}_q$ is defined by $E_{pk}(m;r) = \langle g^r, h^r \cdot g^m \rangle$ where r is picked uniformly at random from $\mathbb{Z}_q$. The decryption algorithm follows by outputting c_2/c_1^s, decrypting a ciphertext $\langle c_1, c_2 \rangle$. This scheme does not directly support the necessary homomorphic operations since the decryption yields g^m rather than m. Fortunately, the lack of "full" decryption is not an issue since P_1 only needs to distinguish between an encryption of a uniformly picked m and the case where $m \in X$ (i.e., when m is part of its input).

Our experiments, detailed in Section 4, show that the ElGamal PKE is much faster than the Paillier PKE with comparable security, and therefore one might prefer using the ElGamal-based version of our protocols.

3 Secure Set Intersection in the Semi-Honest Setting

In this section we present in detail our construction for a protocol realizing $\mathcal{F}_\cap$ in the presence of semi-honest adversaries. The main tool used in our construction is oblivious polynomial evaluation, implemented based on an additively homomorphic PKE. We consider the functionality of *set intersection* where each party's input consists of a set and the size of the *other party's* input set (the reason for including the size of the other party's set in each party's input is to model the fact that the protocol leaks the sizes of the input sets). More formally,

Definition 3.1 *Let X and Y be subsets of a predetermined domain,*[3] *the functionality $\mathcal{F}_\cap$ is:*

$$((X, m_2), (Y, m_1)) \mapsto \begin{cases} (X \cap Y, \lambda) & \textit{if } |X| = m_1 \textit{ and } |Y| = m_2, \\ (\lambda, \lambda) & \textit{otherwise.} \end{cases}$$

We continue with a high-level description of the basic semi-honest protocol as given in [FNP04]. We then discuss techniques to improve the computational overhead (Section 3.2) and provide a formal description of our protocol together with a detailed simulation based security proof (Section 3.3). In Section 3.4 we consider two variants of the set intersection problem and discuss solutions for these problems.

[3] W.l.o.g., we assume $X, Y \subseteq \{0,1\}^{p(k)}$ for some polynomial $p(\cdot)$ such that $2^{p(k)}$ is super-polynomial in k.

3.1 A High Level Description

Recall that P_1 has m_1 elements and P_2 has m_2 elements. The basic protocol works as follows,

1. Party P_1 chooses a pair of encryption/decryption keys $(pk, sk) \leftarrow G(1^k)$ for a homomorphic PKE (G, E, D) and sends pk to P_2.

2. P_1 computes the coefficients of a polynomial $Q(\cdot)$ of degree m_1, whose roots are set to be the m_1 elements of X. P_1 sends the encrypted coefficients of $Q(\cdot)$ to P_2.

3. For each element $y \in Y$ (in a random order), party P_2 chooses r at random (from the appropriate plaintext space $\mathbb{M}$), and uses the homomorphic properties of the encryption scheme to compute an encryption of $r \cdot Q(y) + y$. P_2 sends the encrypted values to P_1.

4. Upon receiving these encrypted values, P_1 extracts $X \cap Y$ by decrypting each value and then checking if the result is in X.

Security argument (informal). This argument is split into two parts. First note that if $y \in X \cap Y$ then by the construction of the polynomial $Q(\cdot)$ we get that $r \cdot Q(y) + y = r \cdot 0 + y = y$. On the other hand, if $y \notin X \cap Y$ we require that the product $r \cdot Q(y)$ corresponds to a random value within the plaintext group so that it reveals no information about y and (with high probability) is not in X.[4] This requirement is obtained almost immediately when using the ElGamal PKE since the plaintext space (excluding zero), $\mathbb{Z}_q^*$, is a multiplicative group. Therefore, for any element $m \in \mathbb{Z}_q^*$ the probability that $r \cdot Q(y) = m$ is $1/(q-1)$ since $r = m/Q(y)$, given that $Q(y) \neq 0$. This, however, is not immediately true when using the Paillier PKE with plaintext space $\mathbb{Z}_N$ since N is not a prime. Fortunately, the fraction of elements outside the multilicative subgroup $\mathbb{Z}_N^*$ is negligible in k (furthermore, finding an element in $\mathbb{Z}_N \setminus \mathbb{Z}_N^*$ when the secret key is unknown amounts to factoring N). We thus get that for any element $y \notin X \cap Y$ the probability that $Q(y) \notin \mathbb{Z}_N^*$ is negligible in k.

More generally, we formalize the requirements from the homomorphic PKE as follows:

Definition 3.2 *We say that homomorphic PKE is* good for plaintext space *if the following holds:*

1. *The plaintext space $\mathbb{M}$ contains the input domain space $\{0,1\}^{p(k)}$.*

2. *There exists a multiplicative group $\mathbb{G}$ for which all but a negligible fraction of $\mathbb{M}$ is in $\mathbb{G}$. Specifically, for any $X, Y \subseteq \{0,1\}^{p(k)}$ the probability that $Q(y) \notin \mathbb{G}$ for all $y \notin X \cap Y$ is negligible, where the probability is taken over the choice of the parameters for $\mathbb{G}$.*

Efficiency. The communication complexity of this simple scheme is linear in m_1 and m_2, as $m_1 + 1$ ciphertexts are sent from P_1 to P_2 (these are the encrypted coefficients of $Q(\cdot)$), and m_2 encrypted values are sent from P_2 to P_1 (i.e., $Q(y)$ for every $y \in Y$). However, the work performed by P_2 is relatively high as each of the m_2 oblivious polynomial evaluations includes performing $O(m_1)$ exponentiations, totaling in $O(m_1 \cdot m_2)$ exponentiations.

A first improvement in the computational overhead can be obtained by applying Horner's rule to the evaluation of $Q(\cdot)$. Any polynomial $P(y) = \alpha_0 + \alpha_1 y + \alpha_2 y^2 + \cdots + \alpha_{m_1} y^{m_1}$ can be evaluated "from the inside out" as follows,

$$\alpha_0 + y(\alpha_1 + y(\alpha_2 + y(\alpha_3 + \cdots y(\alpha_{m_1 - 1} + y\alpha_{m_1}) \cdots))).$$

[4] This construction can be considered a generalization of the oblivious transfer protocols of [NP01, AIR01, Lip03]. In those, a chooser retrieving item i sends to the sender a predicate which is 0 if and only if $i = j$, where $j \in [N]$.

This method of computation employs m_1 multiplications by y, compared to the straightforward polynomial computation method which multiplies the coefficients with the values $y, y^2, \ldots, y^{m_1}$. The improvement is significant when the length of y is much shorter than the length of a full exponent. Recall that with the textbook algorithm for modular exponentiation the computational overhead is linear in the length of the exponent. The gain may be significant even when fine-tuned exponentiation algorithms such as Montgomery's method or Karatsuba's technique are used.

For example, the length of the exponent is typically at least 1024 bits in the case of Paillier, and could be as short as 160 bits in the case of ElGamal. There are many applications in which the input y can be rather short, e.g., if y is a social security number then it can be encoded using only 30 bits. When the domain from which the inputs y are taken is large standard hashing techniques can be used as long as collision probability is negligible. Based on the birthday paradox it is sufficient to hash the values to strings of length about $2\max(\log m_1, \log m_2)$ bits. For the textbook exponentiation algorithm the improvement is by a factor of $k/2\max(\log m_1, \log m_2)$, where k is the length of the exponent. For $m_1, m_2 \approx 100,000$, this factor is about 30 for Paillier with $k = 1024$ or about 5 for ElGamal with $k = 160$.

3.2 Using Hashing to Reduce the Computational Overhead

The main computational overhead of the basic protocol is the work of P_2, which essentially has to do $m_1 \cdot m_2$ comparisons, in order to compare each of its inputs to each of the inputs of P_1. This overhead can be reduced using hashing, if both parties use the same hash scheme to map their respective items into different bins. In that case, the items mapped by P_1 to a certain bin must only be compared to those mapped by P_2 to the same bin. Thus the number of comparisons can be reduced to be in the order of the number of P_2's inputs times the maximum number of items mapped to a bin. (Of course, care must be taken to ensure that the result of the hashing does not reveal information about the inputs.) We describe and compare modifications of the basic protocol based on the following different hash schemes: simple hashing, balanced allocations, and Cuckoo hashing. We also provide in Section 4 results of experiments with each of these schemes.

Using simple hashing. Suppose that the items are hashed into one of B bins as follows. Let h be a randomly chosen hash function mapping elements into bins numbered $1, \ldots, B$. It is well known that if the hash function h maps m_1 items to random bins, then, if $m_1 \geq B \log B$, each bin contains with high probability at most $M = \frac{m_1}{B} + \sqrt{\frac{m_1 \log B}{B}}$ (see, e.g., [RS98, Wie07]). Setting $B = m_1 / \log m_1$ and applying the Chernoff bound shows that $M = O(\log m_1)$ except with probability $(m_1)^{-s}$, where s is a constant that depends on the exact value of M.[5]

In the protocol, the hash function h will be known to both parties. We would therefore like to hide from each party how many of the inputs of the other party are mapped by h to each bin. Thus, the modified protocol works by having P_1 define a polynomial of degree M for each bin by fixing its mapped elements to be the set of roots. In addition, P_1 adds the root 0 sufficiently many times, so that the total degree of the polynomial is M. That is, if h maps ℓ items to the bin, P_1 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 degree M, with exactly m_1 non-zero roots. Note that although only a few of the bins have M items, we have to set all polynomials to this degree, and pay the associated overhead, in order to hide the exact number of items that are mapped to every bin.

[5] By setting $B = m_1 \log\log m_1 / \log m_1$ we can make the error probability negligible in m_1. However, any actual implementation will have to examine the exact value of B which results in a sufficiently small error probability for the input sizes that are expected. As for theoretical analysis, the subsequent construction, based on balanced allocation hashing, presents a negligible error probability.

Finally, P_1 sends to P_2 the encrypted coefficients of the polynomials, and the mapping h from elements to bins. (For our purposes, it is sufficient that the mapping is selected pseudo-randomly, either jointly or by either party.) For every $y \in Y$, P_2 finds the bin into which y is mapped and evaluates the polynomial of this bin. It proceeds as before and responds to P_1 with the encryptions of $rP(y) + y$ for all y. Note that the communication complexity is not affected, as P_1 sends $BM = O(m_1)$ items and p_2 replies with m_2 values. The main gain is that P_2 now has to perform only $m_2 M = m_2 O(\log m_1)$ exponentiations (rather than $m_1 m_2$ exponentiations if no hashing is used).

Using balanced allocations. To save on the computational work even more, we introduce a balanced allocation scheme for randomly mapping elements into bins. Loosely speaking, we use the balanced allocation scheme of [ABKU99] where elements are inserted into B bins as follows. Let h_0, h_1 be two randomly chosen hash functions mapping elements into bins numbered $1, \ldots, B$. An element x is inserted into the less occupied bin from $\{h_0(x), h_1(x)\}$, where ties are broken arbitrarily. If m_1 elements are inserted, then, except with negligible probability over the choice of the hash functions h_0, h_1, the maximum number of elements allocated to any single bin is at most $M = O(m_1/B + \log\log B)$. (The exact probability is described in a note below.) Setting $B = \frac{m_1}{\log\log m_1}$ we get that $M = O(\log\log m_1)$.[6] Note that in this case the degree of the polynomials, and the associated overhead, are much smaller than the $\log m_1$ degree that was set when simple hashing was used.

Then, upon receiving the encrypted polynomials, party P_2 obliviously evaluates, for each of its inputs y, the encryption of $r_0 \cdot Q_{h_0(y)}(y) + y$ and $r_1 \cdot Q_{h_1(y)}(y) + y$ for each of the two bins $h_0(y), h_1(y)$ in which y can be allocated. (Note that here P_2 must evaluate *two* polynomials for each of its inputs since it has no way of knowing what is the right bin.) Finally, P_1 decrypts both evaluations and performs the same check as in the high-level description above, twice. Setting $B = \frac{m_1}{\log\log m_1}$ and $M = O(\log\log m_1)$, we get that the communication complexity is not affected (neglecting constant factors), as P_1 now sends $BM = O(m_1)$ encrypted values and P_2 replies with $2m_2$ encrypted values. There is, however, a reduction in the work performed by P_2, as each of the oblivious polynomial evaluations amounts to performing $O(M)$ exponentiations, and so P_2 now performs only $2m_2 M = 2m_2 O(\log\log m_1)$ exponentiations overall.

Finally, one may worry about the case that P_1 is unlucky in its choice of hash functions such that more than M items are mapped to some bin. The bound of [ABKU99] only guarantees that this happens with probability $o(1)$. However, Broder and Mitzenmacher [BM01] have shown that asymptotically, when we map n items into n bins, the number of bins with i or more items falls approximately like $2^{-2.6^i}$. Formally, this means that if $M = \omega(\log\log n)$ then except with negligible probability no bin will be of size greater than M. Practically, this means that a bound of $M = 5$ suffices with probability $1 - 10^{-35}$. The authors also provide experimental results that confirm the asymptotic bound for the case of $n = 32,000$.

Using Cuckoo hash. Cuckoo hash [PR04] is a multiple-choice hashing scheme with evictions. In its simplest form, n items are mapped to $2(1+\varepsilon)n$ bins (where ε is a small constant which affects the error probability), using two hash functions h_0, h_1. When item x is inserted, then if either bin $h_0(x)$ or bin $h_1(x)$ are free, x is mapped to the free bin. Otherwise, we put x in bin $h_0(x)$, evict the item x' that is already in that location, and try to move x' to the location defined by the other hash function (i.e., if $h_0(x) = h_0(x')$ then we try to move x' to $h_1(x')$). If that location is occupied as well, we evict the item that is located there and try to relocate it, and so on. A version of this algorithm described in [PR04] continues with this process, until there is a series of $O(\log n)$ evictions, in which case it chooses new hash functions h_0, h_1 and tries

[6] A constant factor improvement is achieved using the *Always Go Left* scheme in [VÖ3] where $h_0 : \{0,1\}^* \to [1, \ldots, \frac{B}{2}], h_1 : \{0,1\}^* \to [\frac{B}{2}+1, \ldots, B]$. In that scheme, an element x is inserted into the less occupied bin from $\{h_0(x), h_1(x)\}$; in case of a tie x is inserted into $h_0(x)$.

to re-insert all items. It was shown in [PR04] that the expected time, i.e. the expected number of insertion trials, needed by the scheme to insert all items is $O(n)$.

The use of Cuckoo hash for our purposes is appealing, since at most a single item is mapped to every bin. Therefore the number of comparisons can be reduced to be *linear* in the uninput size. A major obstacle, though, is that there is a pretty high probability, of $1/n$, that a specific input set of n elements cannot be inserted in the table using a specific pair of hash functions h_0, h_1, and then a new pair of hash functions must be chosen. Therefore P_2, which knows the hash functions and knows that they were chosen at random from the set of pairs of functions which do not cause an abort for P_1's input, can identify with probability of about $1/n$ whether a certain potential input set of P_1 cannot be the actual input of P_1 in a specific run of the protocol. This problem occurs even if P_1 rearranges its input element in a random order.

One approach to reducing the effect of this problem is to use the "Cuckoo hashing with a stash" solution of [KMW08]: In addition to the hash table, this solution keeps a small amount of additional memory, namely a stash of s items. If the insertion algorithm encounters an infinite cycle of evictions, then an element in that cycle is moved to the stash. When we look for an item x in the table, we search for it in locations $h_0(x)$ and $h_1(x)$ as well as in the stash. It was shown in [KMW08] that, for any constant s, using a stash of size s fails with probability $O(n^{-s})$ (taken over the choice of the hash functions). Therefore, if we change the algorithm so that it aborts if the original choice of hash functions results in more than s items being moved to the stash, then the algorithm aborts with probability of at most $O(n^{-s})$. Consequently, P_2 can identify with probability $O(n^{-s})$ whether a specific potential input of P_1 does not agree with the hash functions h_0, h_1. This is a low (albeit not negligible) probability which we further characterize experimentally in Section 4 for various settings of hash table size $(1 + \epsilon)$, n and stash size s.

When the protocol is implemented using Cuckoo hash, then when P_2 processes an input x it must evaluate the polynomials of the two bins to which x might have been mapped (bins $h_0(x)$ and $h_1(x)$), as well as the polynomial representing the stash. The advantage is that the polynomials of the two bins are linear and the polynomial representing the stash is of small degree. This should be compared to the balanced allocation solution in which the polynomials representing the bins are of degree $\log \log m_1$.

3.3 The Detailed Protocol

We are now ready to formally present our protocol. We describe the protocol based on balanced allocations, since for that hashing scheme there is a negligibly small bound on the failure probability. We remind the reader that our construction is secure with respect of both Paillier and ElGamal, and any homomorphic PKE that meets Definition 3.2. We describe our construction using a general notation for the homomorphic encryption scheme. In Section 4 we describe the results of experiments with each of the hashing schemes.

Protocol 1 ($\pi_{\cap}^{\text{SH}}$ - secure set intersection in the semi-honest model):

- **Inputs:** *The input of P_1 is m_2 and a set $X \subseteq \{0,1\}^{p(k)}$ containing m_1 items; the input of P_2 is m_1 and a set $Y \subseteq \{0,1\}^{p(k)}$ containing m_2 items (hence, both parties know m_1 and m_2). Recall from definition 3.2 that $\{0,1\}^{p(k)}$ is contained in the plaintext space of a homomorphic PKE with a good plaintext space.*
- **Auxiliary inputs:** *A security parameter 1^k.*
- **Convention:** *Both parties check every received ciphertext for validity, and abort if an invalid ciphertext is received.*
- **The protocol:**

  1. **Key setup:** *P_1 chooses $(pk, sk) \leftarrow G(1^k)$ and sends pk to P_2.*

2. **Setting the balanced allocation scheme:** *P_1 computes the parameters B, M for the scheme and chooses two randomly chosen hash functions $h_0, h_1 : \{0,1\}^{p(k)} \to [B]$.*[7] *It sends B, M, h_0, h_1 to P_2.*
3. **Creating polynomials for the set X:** *For every $x \in X$, P_1 maps x into the less occupied bin from $\{h_0(x), h_1(x)\}$ (ties broken arbitrarily). Let $\mathcal{B}_i$ denote the set of elements mapped into bin i and let $Q_i(x) \overset{\text{def}}{=} \sum_{j=0}^{M} Q_{i,j} \cdot x^j$ denote a polynomial with the set of roots $\mathcal{B}_i$ (if $\mathcal{B}_i = \emptyset$ then P_1 sets $Q_i(x) = 1$). If $|\mathcal{B}_i| < M$ then P_1 multiplies $Q_i(\cdot)$ by $x^{M-|B_i|}$.*

   P_1 encrypts the coefficients of the polynomials, setting $q_{i,j} = E_{pk}(Q_{i,j}; r_{i,j})$. It sends the encrypted coefficients to P_2.
4. **Substituting in the polynomials:** *Let $y_1, \ldots, y_{m_2}$ be a random ordering of the elements of set Y. P_2 does the following for all $\alpha \in \{1, \ldots, m_2\}$:*
   - *It sets $\hat{h}_0 = h_0(y_\alpha)$, $\hat{h}_1 = h_1(y_\alpha)$ (these values denote the bin number). Then it chooses random plaintexts r_0, r_1 in the domain of $E_{pk}()$ and uses the homomorphic properties of the encryption scheme to evaluate $e_\alpha^0 = E_{pk}(r_0 \cdot Q_{\hat{h}_0}(y_\alpha) + y_\alpha)$, and $e_\alpha^1 = E_{pk}(r_1 \cdot Q_{\hat{h}_1}(y_\alpha) + y_\alpha)$. Finally, it sends e_α^0, e_α^1 to P_1.*
5. **Computing the intersection:** *For each received e_α^0, e_α^1 party P_1 checks if for some $x \in X$, $x = D_{sk}(e_\alpha^0)$ or $x = D_{sk}(e_\alpha^1)$. In this case P_1 records x as part of its output.*

Correctness. We first note that when both parties follow the protocol P_1 outputs $X \cap Y$ with probability negligibly close to 1: (**i**) For elements $x \in X \cap Y$ there exists $y_\alpha \in Y$ that zeros at least one of the polynomials $Q_{h_0(y_\alpha)}(\cdot)$, $Q_{h_1(y_\alpha)}(\cdot)$. Hence, getting the messages sent for y_α, P_1 records y_α. (**ii**) For elements $x \in X \setminus Y$, P_1 records x only if either $x = D_{sk}(e_\alpha^0)$ or $x = D_{sk}(e_\alpha^1)$ which, occurs with negligible probability due to the randomness of r_0 and r_1 and the requirement that the homomorphic PKE is good for plaintext space.

Theorem 3.3 *Assume that (G, E, D) is a semantically secure homomorphic PKE that is good for plaintext space (cf. Definition 3.2). Then Protocol 1 securely computes $\mathcal{F}_\cap$ in the presence of semi-honest adversaries (cf. Definition 2.1).*

Proof: We consider two corruption cases.

Simulating the view of party P_1. The simulation is based on the fact that messages received by P_1 are encryptions either of elements of the intersection set or are random elements of the message space of the encryption scheme. We construct a simulator $\mathcal{S}$ as follows:

1. $\mathcal{S}$ is given X, m_2, $X \cap Y$ and 1^k and sets $m_1 = |X|$.
2. $\mathcal{S}$ receives from P_1 its public key pk.
3. $\mathcal{S}$ receives from P_1 the parameters B, M and the seeds for the two chosen hash functions $h_0, h_1 : \{0,1\}^{p(k)} \to [B]$ used in the balanced allocation scheme.
4. $\mathcal{S}$ receives from P_1 the encrypted polynomials $\{q_{i,j}\}_{i \in \{1,\ldots,B\}, j \in \{0,\ldots,M\}}$.
5. $\mathcal{S}$ completes $X \cap Y$ to size m_2 by adding random elements from $\{0,1\}^{p(k)}$. Let $\tilde{Y}$ denote this recorded set.
6. $\mathcal{S}$ now plays the role of the honest P_2 for the rest of the execution, using $\tilde{Y}$ as the input for P_2.

[7] Any implementation of a hashing scheme must replace the idealized random hash function (that is used for the analysis) with an actual construction of a hash function that works well in practice. See,. e.g., [PT12] and related work.

7. $\mathcal{S}$ outputs the view of P_1.

We claim that the joint distributions of P_1's view and P_2's output are statistically close in the real and simulated executions. The potential difference between these executions is in Step 5 of the simulation, where $\mathcal{S}$ plays the role of the honest P_2 with $\tilde{Y}$ instead of Y. The difference is that the random elements in $\tilde{Y}' = \tilde{Y} \setminus (Y \cap X)$ are used in the simulated execution where members of $Y' = Y \setminus X = Y \setminus (Y \cap X)$ are used in the real execution.

Let $y \in Y'$ (respectively, $\tilde{y} \in \tilde{Y}'$) be elements considered in the real (respectively, simulated) execution. Then P_1 receives the encryptions $e^0 = E_{pk}(r_0 \cdot Q_{h_0(y)}(y) + y)$, and $e^1 = E_{pk}(r_1 \cdot Q_{h_1(y)}(y) + y)$ in the real execution, and the encryptions $\tilde{e}^0 = E_{pk}(\tilde{r}_0 \cdot Q_{h_0(\tilde{y})}(\tilde{y}) + \tilde{y})$, and $\tilde{e}^1 = E_{pk}(\tilde{r}_1 \cdot Q_{h_1(\tilde{y})}(\tilde{y}) + \tilde{y})$ in the simulated execution. Note first that due to the requirement that the input domain size is super-polynomial in k, the probability that elements from $\tilde{Y}$ are in X as well is negligible in k. Moreover, $Q_{h_0(y)}(y), Q_{h_1(y)}(y), Q_{h_0(\tilde{y})}(\tilde{y}), Q_{h_1(\tilde{y})}(\tilde{y}) \in \mathbb{G}$ with overwhelming probability (for $\mathbb{G} \subseteq \mathbb{M}$ multiplicative group as specified in Definition 3.2). We thus get that all the encrypted plaintexts sent to P_1 (by both P_2 and $\mathcal{S}$) are uniformly distributed in $\mathbb{G}$ and therefore P_1 cannot distinguish between the two cases since they are identical with overwhelming probability.

Simulating the view of party P_2. The simulation is based on the fact that as P_2 only receives encrypted values it cannot detect whether these are encryption of values sent in the real protocol or encryptions of arbitrary messages sent by the simulator. Construct a simulator $\mathcal{S}$ as follows:

1. $\mathcal{S}$ is given Y, m_1, $X \cap Y$ and 1^k and invokes P_2 on these inputs. $\mathcal{S}$ sets $m_2 = |Y|$.

2. $\mathcal{S}$ chooses $(pk, sk) \leftarrow G(1^k)$ and sends pk to P_2.

3. $\mathcal{S}$ computes the parameters B, M for the balanced allocation scheme and chooses random seeds for the hash functions h_0, h_1. These are then sent to P_2.

4. $\mathcal{S}$ sends to P_2, BM encryptions of the value 0, under the key pk (i.e., encryptions of the coefficients of B polynomials which are identically equal to 0). Each encryption is done with fresh randomness.

5. $\mathcal{S}$ completes the execution and outputs whatever P_2 does.

In the following we define a sequence of hybrid games and denote by the random variable $\mathbf{H}_\ell(k, X, Y)$ (for a fixed k) the joint output of P_2 and P_1 in hybrid game $\mathbf{H}_\ell$.

Game $\mathbf{H}_0$: The simulated execution.

Game $\mathbf{H}_1$: The simulator $\mathcal{S}_1$ acts identically to $\mathcal{S}$ except that in Step 2 of the simulation S_1 does not get to know the secret key sk. The random variables $\mathbf{H}_0(k, X, Y)$ and $\mathbf{H}_1(k, X, Y)$ are identical as $\mathcal{S}$ ignores sk.

Game $\mathbf{H}_2$: In this game there is no trusted party and no honest P_1. Instead, the simulator $\mathcal{S}_2$ is given as input P_1's real input X. $\mathcal{S}_2$ works exactly like $\mathcal{S}_1$ except that instead of sending encryptions of zero polynomials it computes the polynomials as in the real execution using the set X.

We prove that the random variables $\mathbf{H}_1(k, X, Y)$ and $\mathbf{H}_2(k, X, Y)$ are computationally indistinguishable via a reduction to the security of (G, E, D).[8] Assume, for contradiction, the existence of a distinguisher circuit D for the distribution induced by games $\mathbf{H}_1(k, X, Y)$ and $\mathbf{H}_2(k, X, Y)$ and choose hash functions h_0, h_1 that maximize D's distinguishing advantage. Construct a distinguisher circuit D_{E} that distinguishes

[8] We consider an extension of the semantic security game (cf. Definition 2.6), where a distinguisher D_{E} is given a public key pk, outputs two vectors of plaintexts $\overline{m}_0, \overline{m}_1$, and receives back a vector of ciphertexts $\overline{c}$ comprised of the encryptions of $\overline{m}_b$ under E_{pk}, where $b \leftarrow_R \{0, 1\}$. D_{E} then outputs a bit b' and we say that it distinguishes successfully in this game if $b' = b$.

	Encrypt	Decrypt	HAdd
Paillier	3742	273	4
El Gamal	266	155	2

Table 1: Average speed (μs) of public-key operations with a 1024-bit El Gamal modulus and a 2048-bit Paillier modulus.

between the encryptions of two sets of messages: (**i**) coefficients of zero-polynomials as constructed in game $\mathbf{H}_1$, and (**ii**) coefficients of polynomials corresponding to X and hash functions h_0, h_1 as constructed in game $\mathbf{H}_2$. D_{E} receives a public key pk and vector of encryptions $\overline{c}$ under pk and works exactly like $\mathcal{S}_2$ except that it uses the public key given to it as input in the semantic security game, instead of generating it by itself, and forwards to P_2 the vector of encryptions $\overline{c}$ instead of the encrypted polynomials. Note that the output distribution generated by D_{E} is either identical to $\mathbf{H}_1(k, X, Y)$ or to $\mathbf{H}_2(k, X, Y)$ (conditioned on our choice of h_0, h_1), and hence the existence of D contradicts the indistinguishability of ciphertexts of the encryption scheme (G, E, D).

Game $\mathbf{H}_3$: The simulator $\mathcal{S}_3$ acts identically to $\mathcal{S}_2$ except that $\mathcal{S}_3$ is given sk. The random variables $\mathbf{H}_2(k, X, Y)$ and $\mathbf{H}_3(k, X, Y)$ are identical as $\mathcal{S}_2$ and $\mathcal{S}_3$ do not use sk.

Note that game $\mathbf{H}_3$ is identical to the real execution with respect to the adversary's view. ■

3.4 Computing Set Intersection Cardinality

In a protocol for computing the cardinality of the set intersection, P_1 learns $|X \cap Y|$ but not the actual elements of this set. In order to compute this functionality, P_2 needs only slightly change his behavior from that in Protocol 1. That is, instead of computing the encryptions of $r_0 \cdot Q_{\hat{h}_0}(y_\alpha) + y_\alpha$ and $r_1 \cdot Q_{\hat{h}_1}(y_\alpha) + y_\alpha$, P_2 now only encodes $r_0 \cdot Q_{\hat{h}_0}(y_\alpha)$ and $r_1 \cdot Q_{\hat{h}_1}(y_\alpha)$. Then, in Step 5 of the protocol, P_1 counts the number of ciphertexts received from P_2 that are decrypted to zero, and locally outputs this number. The proof of security for this protocol trivially follows from that of Protocol 1.

4 Performance Evaluation

In order to give concrete performance numbers to our protocol, we provide in this section performance benchmarks for an implementation of the private matching protocol described in Section 3.1. As described below, the benchmarks show the advantage of the hashing-based approaches over the basic approach which uses no hashing. On the other hand, the performance benefit of using the solutions based on balanced allocations and on Cuckoo hashing, over that of a single hash function, does not become noticeable until the input sets are pretty large, due to additional factors which affect the overhead.

Our implementation uses the cryptographic libraries that are part of the SFS toolkit [Maz01] (although we wrote the implementation of the Paillier cryptosystem), which in turn uses the GNU Multi-Precision library 4.3.0 [GMP09] for large arithmetic operations. Performance benchmarks were performed on a Sun X2200 M2 server running Fedora core 12 in 64-bit mode, using a single core of an AMD Opteron 2376 processor (2.3 GHz).

Table 1 shows the performance of the encryption schemes of Paillier [Pai99] in a fast decryption mode, as described in Section 6 of [Pai99], and ElGamal, using the variant described in Section 2.3.1. (Recall that in that variant of ElGamal we encrypt values in the exponent, to support the homomorphic addition property. This makes decryption harder, but that issue is irrelevant to our protocol since we only need to check whether the decrypted plaintext is equal to 0.) We used key and modulus sizes that offered comparable security: a 1024-bit value for N in the Paillier cryptosystem (and thus a 2048-bit modulus for N^2), and a 160-bit key and

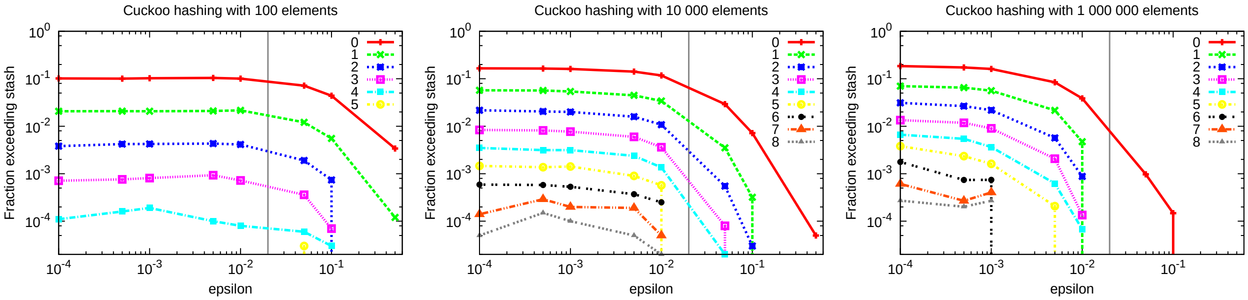

Figure 1: In Cuckoo hashing, fraction of randomized trials that exceed a given stash size, for varying hash table sizes $2 \cdot (1+\epsilon) \cdot n$. Solid vertical line at 0.02 corresponds to the ϵ used in set intersection experiments.

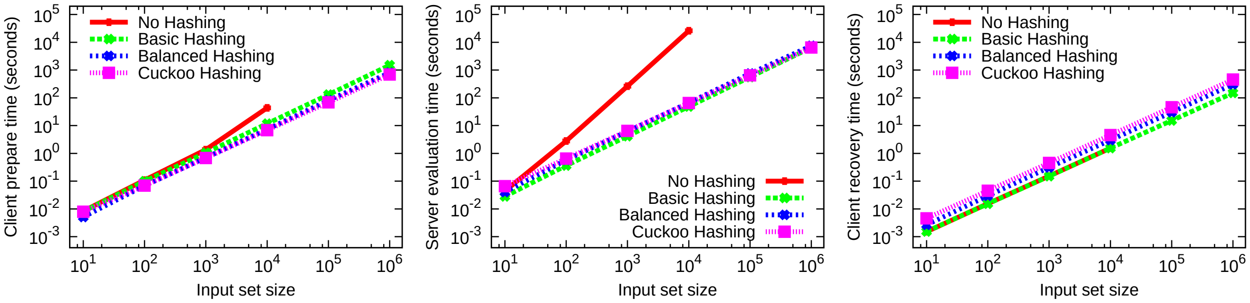

Figure 2: Computational set intersection performance for P_1's setup and precomputation (left), P_2's evaluation (center), and P_1's corresponding recovery of the intersection (right).

1024-bit prime modulus for ElGamal. The numbers correspond to the mean value (in microseconds) of 100 runs across five different keys. The column HAdd corresponds to the cost of performing a homomorphic addition of plaintexts, which is akin to modular multiplication. Due to its superior performance, we use ElGamal to instantiate our set intersection protocol in this section's subsequent benchmarks. We also note that the ElGamal encryption can be implemented even more efficiently based on elliptic curve cryptography.

For the Cuckoo hash construction, we need to instantiate the protocol with appropriate parameters for the stash size s and table size $2 \cdot (1+\epsilon) \cdot n$. As stated earlier, it was shown in [KMW08] that using a stash of size of size s fails with probability $O(n^{-s})$; here, we also characterize this probability experimentally. Figure 1 shows the fraction of randomized trials (out of a total of 100,000 trials) for which the stash was not large enough to prevent cycles, for varying choices of ϵ and s. We evaluate input sizes $n = 10^i$ for $i = 1 \dots 6$, of which three sizes are shown in the figure. In our set intersection experiments, we instantiate the Cuckoo hashing version with $\epsilon = 0.02$ (shown by the solid vertical line in the figure) and a stash size of 2. With this configuration, $n = 1000$ has the highest fraction of failed constructions at ~0.55%, while inputs that were very small or large had a much lower rate of failure ($n = 10$ has ~0.005%, $n = 10^6$ has none). We conclude that these chosen parameters are adequate to balance safety and performance across various values of n, although an implementation might choose to vary both parameters as a function of n, in order to better tune this tradeoff to a particular setting. As we will see next, however, this still will not result in any real benefit compared to the simpler hashing strategies.

Recall that the set intersection protocol consists of three main stages: (1) P_1's setup to construct an encrypted polynomial. The overhead of this step is $O(m_1)$ exponentiations in the basic protocol, and $O(BM) = O(m_1)$ exponentiations in each of the hashing-based constructions, where B is the number of bins, and m_1 is the size of P_1's input. (2) P_2's evaluation of this polynomial on its inputs. This step takes $O(m_1 m_2)$ exponentiations in the basic method, $O(m_2 \log m_1)$ exponentiations in the random hash method,

Construction	Stage 1	Stage 2	Stage 3
	P_1's setup	P_2's evaluation	decryption by P_1
Random hash	10 936	52 334	2 153
Balanced allocations	7 833	68 336	4 692
Cuckoo hash	8 503	86 193	6 375

Table 2: Speed (msec) of each of the hashing-based solutions, for $m_1 = m_2 = n = 10000$, computed as the average of 10 runs. The random hash construction required a single evaluation of a polynomial of degree 26. The balanced allocations construction evaluated two polynomials of degree 6. The Cuckoo hash construction evaluated two linear polynomials and a degree two polynomial (representing a stash of size 2).

$O(m_2 \log\log m_1)$ exponentiations in the balanced allocations construction, and only $O(m_2)$ exponentiations in the Cuckoo hash construction. (3) P_1's subsequent recovery of the intersection. This step requires $O(m_2)$ decryptions in each of the methods.

Measured performance trends. First, let us analyze the high-level performance trends of our various set intersection constructions. Figure 2 shows the performance of the set intersection protocol for these three stages in log-log scale, where both client and server have the same input set size ranging from 10 to 1 million elements. The figure demonstrates the poor performance implications of using a single polynomial constructed across all of P_1's input (i.e., "No Hashing"), as compared to the hashing-based schemes. The latter's performance is almost linear in n.

Explaining asymptotic performance. We now explain the measured performance trends of the three hashing-based approaches, as it differs from their asymptotic computational overhead. To simplify the notation, we consider the case of $m_1 = m_2 = n$. Recall that asymptotically, the computation overhead of the constructions based on random hashing, balanced allocations, and Cuckoo hashing, is $O(n \log n)$, $O(n \log\log n)$, and $O(n)$, respectively, where the bulk of the computation is done in Step 2. Our experiments, however, show the opposite: The run time of the random hashing based approach, in Stages 2 and 3, is better than that of the balanced allocations scheme, which is in turn better than the run time of the Cuckoo hashing approach. In order to explain these results, we must examine the constants that affect the overhead.

Table 2 details the run times for the case of $n = 10000$. We ran experiments for values of n up to $n = 100000$, where we parameterized the experiments based on the results from initial experiments to find values which do not result in the hash functions overflowing the capacity of the bins. The results for $n = 10000$ are representative of the results in the range we examined. The random hash scheme used $B = n/\log n = 753$ bins, each of size 26. The balanced allocation scheme used $B = n/\log\log n = 2680$ bins, each of size 6. The Cuckoo hashing scheme used $2 \cdot 1.02 \cdot n = 20400$ bins, each with a single element, and a stash of size 2.

In Stage 1, P_1 constructs a polynomial for each bin in the hash table (corresponding to polynomials of degree 26, 6, or 1, respectively for each scheme). The task of P_1 is to encrypt each of the coefficients of the polynomials and therefore the overhead per polynomial is linear in the degree. An analysis based on the number of polynomials used in every experiment and their degrees, shows that the overhead per polynomial of degree d is about $0.55d$ msec.

Let us now focus on the overhead of Stage 2, which consists of the evaluation of the polynomials by P_2 and consumes the bulk of the overhead. Recall that P_2 must compute expressions of the form $E_{pk}(r \cdot P(y) + y)$. The overhead of computing these expressions includes a component that is linear in the degree of P, and some constant overhead, which does not depend on the degree and is caused by multiplying $P(y)$ by r

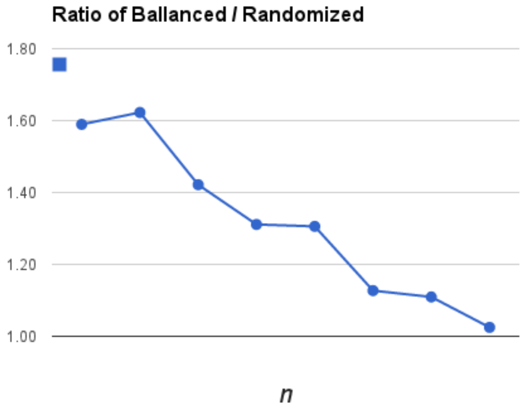

Figure 3: The ratio of the runtimes of Step 2 in the randomized hashing and balanced allocations constructions, for $n = 100, \dots, 100000$.

and adding y. The random hash based construction requires computing n polynomials of a relatively high degree (26 when $n = 10000$). The balanced allocations experiment requires computing $2n$ polynomials of a lower degree (degree 6 when $n = 10000$), whereas the Cuckoo hash experiment has $2n$ evaluations of linear polynomials and n evaluations of polynomials of degree 2 (corresponding to the stash).

Linear regression based on the experiment results shows that the evaluation time of a polynomial of degree d is about $2.8 + 0.1d$ ms. Namely, that it has a large additive constant, which is comparable to increasing the degree by 28. This large additive constant is mostly due to the combined effect of Horner's rule and of the short length of value y used as input to the polynomials (which is only 5 bytes long). The Horner's rule based implementation uses homomorphic multiplications by y, that are implemented as exponentiations with a short, 5 bytes long exponent, whereas the multiplication by r (that is done once per polynomial) is implemented as a full exponentiation. Therefore the effect of that single exponentiation is quite dominant.

As a result, for $n = 10000$ the runtime of Step 2 in our implementation for the random hash experiment, which evaluates a single polynomial of degree 26, is about $(2.8{+}2.6)\cdot n = 5.4n$. The runtime of the balanced allocations experiment, which evaluates two polynomials of degree 6, is about $2 \cdot (2.8 + .6) \cdot n = 6.8n$. The runtime of the Cuckoo hashing experiment, which evaluates for each input two linear polynomials and one polynomial of degree 2, is $(2 \cdot (2.8 + .1) + (2.8 + .2)) \cdot n = 8.8n$. The balanced allocations construction is therefore slower than the random hashing construction, and the run time of the Cuckoo hashing construction is slower than that of the balanced allocations construction.

When n increases, the degrees of the polynomials of the randomized hashing and balanced allocations constructions increase, and dominate the constant additive factor. As a result, the runtime should approach its asymptotic behavior, where the Cuckoo hashing construction is favorable, followed by the balanced allocations construction. To verify that this phenomenon indeed takes place, we charted that ratio of the runtimes of the randomized hashing construction and the balanced allocations construction for different values of n ranging from $n = 100$ to $n = 100000$. This ratio is decreasing and is almost 1 when $n = 100000$, and we assume that it will become smaller than 1 for larger values of n. See Figure 3.

As for Stage 3, note that in the three constructions P_2 sends back n, $2n$, and $3n$ encryptions, respectively. P_1 must decrypt all these encryptions, and its actual run time in this stage is indeed linear in the number of decryptions that it must perform.

We note that the running time might be further optimized by increasing the sizes of the tables, and by that reducing the size of the bins. This will increase the overhead of Stages 1 and 3, as well the communication overhead, but should decrease the overhead of Stage 2.

To summarize our observations about the actual runtime, we found that the running time of evaluating each polynomial is linear in the degree, but it also has a constant factor which increases the degree by about 28. As a result, each additional polynomial that is evaluated has a considerable effect. The random hashing construction evaluates a single polynomial of degree $O(\log n)$, whereas the balanced allocations construction evaluates two polynomials of degree $O(\log \log n)$, and the Cuckoo hashing construction evaluates three polynomials: two linear polynomials and a polynomial of degree 2. Asymptotically, the performance of the constructions based on balanced allocations and Cuckoo hashing is preferable, but since these two constructions use more polynomials than the first construction, their overhead is higher than that of random hashing for the input sizes that we tested.

5 Security in the Presence of Malicious Adversaries

We now modify the secure set intersection protocol to accommodate malicious adversaries. We present in detail a protocol that is based on a balanced allocation scheme. A similar transformation can be applied to the protocol based on Cuckoo hashing. The overhead of the resulting protocol is of the same order as that of the corresponding semi-honest protocol. Considering our construction for semi-honest parties a number of problems need to be taken care of:

1. P_1 can construct polynomials that would enable it to learn the value of elements that are not in the intersection. For instance, by setting the polynomial Q_i to be identically zero P_1 learns all elements $\{y \in Y : h_0(y) = i$ or $h_1(y) = i\}$. We solve this problem by presenting a zero-knowledge protocol for verifying that $\sum_{i\in\{1,\ldots,B\}} \deg(Q_i) = m_1$, and $Q_i(\cdot) \not\equiv 0$ for all $i \in \{1,\ldots,B\}$ (Section 5.1).

2. A more subtle problem that was overlooked in prior work is that with some homomorphic encryption PKEs, P_1 may construct $Q(\cdot)$ such that the evaluation of $r \cdot Q(y)$ (and hence also of $r \cdot Q(y) + y$) is far from being random in $\mathbb{M}$, even though $Q(y) \neq 0$ and r is chosen at random. This attack can be carried out with respect to encryption schemes for which the plaintext space is *not* a cyclic group of prime order. Therefore in this case, $r \cdot Q(y)$ may be a random element within a smaller subgroup.

   For a concrete example consider the Paillier encryption scheme and note that since P_1 knows both the public and secret keys $(pk = N, sk = \phi(N))$, it can construct a polynomial $Q(\cdot)$ such that $Q(y) \notin \mathbb{Z}_N^*$ for some specific value(s) of y of its interest.[9] This implies that $r \cdot Q(y) \notin \mathbb{Z}_N^*$, and so is very far from being random in $\mathbb{Z}_N$, hence fails to hide y. This problem does not exists when using ElGamal.

   We are aware of two ways to address this problem: The first solution would be to have P_1 generate the polynomials after it learns pk but *before* it learns sk. That is, the parties first run a secure protocol for a mutual generation of the public key in which the secret key is shared between them. Then P_1 generates its polynomials and finally, P_2 reveals its share so that P_1 will be able to learn the secret key. For the Paillier encryption scheme we get that coming up with a polynomial $Q(\cdot)$ and y such that $Q(y) \notin \mathbb{Z}_N^*$ amounts to factoring the product N. The public key generation can be computed using the efficient protocol of Hazay et al. [HMRT12] that is proven with simulation based security in the malicious setting. To the best of our knowledge, this is the only non-generic protocol that guarantees simulation based security in the two-party setting. We further note that generating such a public key for the ElGamal scheme is easier following the underlying ideas of Diffie and Hellman [DH76].

[9] Learning sk allows P_1 to efficiently decrypt messages it receives from P_2. Otherwise P_1 and P_2 would have to engage in a protocol for a joint decryption.

Another solution by [LL07] would be to have P_2 encrypt some padding of the payload value added to the polynomial evaluation. This corresponds to a randomized encoding in order to compensate the loss of entropy. Looking ahead, this implies that P_2 first encodes s and then adds it to the encrypted polynomial evaluation. This yields an encryption to an element in $\mathbb{Z}_N^*$ that is statistically close to a random element in this group, where the statistical difference depends on the encoding parameters. For simplicity we employ the former solution in our protocol below.

3. A malicious P_1 may choose the hash functions adversarially, e.g., so that an overflow is caused conditioned on P_2 having certain input set. This issue can be overcome by having the two parties choose the hash functions jointly so that as long as one of them is honest the result is a randomly chosen hash function. There exist well known efficient protocols for joint coin tossing, secure against malicious adversaries. We therefore reduce the task of choosing the hash function to joint coin tossing. It is known in advance that the hash function will be applied to some n values, and we wish to ensure that the hash function that is chosen would be close to uniform on the set of n inputs to which it will be applied. A straightforward method of defining this function is by choosing at random the coefficients of an $(n-1)$ degree polynomial. However, this function has the drawback that each evaluation of the polynomial takes $O(n)$ time. A result of [PP08] describes a function defined by $O(n \log n)$ random bits (actually $O(n \log |V|)$ bits, where $|V|$ is the size of the range of function), where the function can be computed in $O(1)$ time, and has the property that for any set S of n items the function is uniform on S except with probability that can be bounded by an arbitrarily small polynomial in n. A very recent construction for the case of a Cuckoo hash with a stash, shows that the hash functions used can be defined by only $2n^{1/2} \log n + O(s \log n)$ bits, where s is the size of the stash, which is also $O(1)$ [ADW12]. The run time of the resulting hash function was further improved in [DW03, Woe06], although it still requires the same number of random bits.

4. Lastly, while party P_2 is supposed to send m_2 pairs of encryptions resulting from substituting a value y (known to P_2) in the (encrypted) polynomials $Q_{h_0(y)}$ and $Q_{h_1(y)}$, it may deviate from its prescribed computation. Thus, its input to the protocol may be ill defined.

   The standard solution to this problem involves the usage of zero-knowledge proofs for demonstrating correct behavior by P_2. The cost of such a proof for each $y \in Y$ would be proportional to the size of X since P_2 cannot disclose the identity of y's bin. Therefore, the overall cost would be quadratic in the size of X. Instead, we introduce a technique that enables P_1 to redo the entire computation supposedly carried out by P_2 on any value y supposedly in the intersection and verify that its outcome is consistent with the messages received from P_2. This is where we incorporate the usage of a random oracle $\mathcal{H}$ in our protocol.

   We first describe how to use this technique in the case where P_2 evaluates a single polynomial first (the "No Hashing" case), and then extend it for the hashing based protocols.

   For each $y \in Y$ in P_2's input set it chooses a random element s and computes the encryption of $r \cdot Q(y) + s$, where the randomness used for this computation is taken from $\mathcal{H}(s)$. Clearly, if $Q(y) = 0$, then P_1 learns s, and can easily verify whether there exists $x \in X$ such that together with s yields exactly the same encryption. The security argument shows that this is true only for elements in the intersection set. In particular, when P_2 deviates from the prescribed computation, P_1 records an incorrect output with only a negligible probability.

   When using a balanced allocation scheme (or Cuckoo hashing), P_2 needs to evaluate more than one polynomial on the *same* value y. Furthermore, when y is in the intersection exactly one of these polynomials evaluates to zero. To force P_2 to act according to the protocol we let P_2 repeat the above computation twice so that it ends up sending two encryptions for $r_0 \cdot Q_{h_0(y)}(y) + s_1$ and

$r_1 \cdot Q_{h_1(y)}(y) + s_0$. By learning $s_b \in \{s_0, s_1\}$, P_1 is able to extract s_{1-b} and hence verify P_2's computation. Loosely speaking, by applying the random oracle on s_b the adversary learns r_{1-b} and thus is able to extract s_{1-b} from the plaintext that is combined with s_{1-b}. This method is different than the one applied in [HN10] that uses a single value of s for both bins, where the randomness is extracted from the outcome of a PRF evaluated on s. Specifically, in [HN10] P_2 commits to each y and the payload value s first. Later, P_2 uses s to generate the randomness used to evaluate the corresponding polynomial with y.

5.1 Properties of Homomorphic PKE

In this section we consider the properties of the homomorphic PKE that we need for proving the security of our protocol. We demonstrate that these properties hold for both Paillier and ElGamal PKEs.

1. Given a random pk (but not sk), it is infeasible to come up with a message $m \neq 0$ such that $m \notin \mathbb{G}$, for $\mathbb{G} \subseteq \mathbb{M}$ a multiplicative group for which only a negligible fraction of $\mathbb{M}$ is not in $\mathbb{G}$ (as specified in Definition 3.2). For Paillier $\mathbb{G}$ denotes $\mathbb{Z}_N^*$, whereas for ElGamal $\mathbb{G}$ denotes $\mathbb{Z}_q^*$.

2. There exists a polynomial-time algorithm that on input (pk, c) outputs 1 iff c is *valid*, i.e., in the range of $E_{pk}(\cdot;\cdot)$. For Paillier one only needs to check that $c \in \mathbb{Z}_{N^2}^*$. For ElGamal one can use the subgroup of quadratic residues modulo $q' = 2q + 1$.

3. There exists a protocol π_{KEY} for securely computing the key generation functionality $\mathcal{F}_{\mathrm{KEY}}$ in the presence of malicious adversary, where $\mathcal{F}_{\mathrm{KEY}}$ is defined by

$$(1^k, 1^k) \mapsto \Big((pk, sk_1), (pk, sk_2)\Big), \tag{2}$$

where $(pk, sk) \leftarrow G(1^k)$, sk_1 and sk_2 are random shares of sk that do not leak any information about the secret key, and (efficient) reconstruction of sk is with respect to the specific PKE via a secure two-party protocol. We also require that given sk, a simulator can efficiently compute shares sk_1, sk_2 such that the distribution over these shares when output by the protocol and by a simulator is computationally indistinguishable.

For Paillier one can use the protocol of [HMRT12] for which its efficiency is dominated by the number of trial divisions for testing the candidates composites. This protocol ensures an improved analysis based on the analysis from [BF01] for which a random number of length 1024 is a prime with probability $1/44$, condition that it passed a trial division with some threshold parameter B. Thus the expected number of attempts is 1936. For ElGamal, the key setup protocol by Diffie and Hellman [DH76] (that can be made secure for the malicious setting) can be used here.

4. There exists an efficient computational zero-knowledge proof of knowledge π_{MULT} for proving the multiplication of two plaintexts. The relation $\mathcal{R}_{\mathrm{MULT}}$ is formalized as follows,

$$\mathcal{R}_{\mathrm{MULT}} = \left\{ \Big((pk, (e_a, e_b, e_c))\Big) \Big| \exists (a, r_a, b, r_b, r_c) \ s.t \ \begin{array}{c} e_a = E_{pk}(a; r_a) \wedge e_b = E_{pk}(b; r_b) \\ \wedge\ e_c = E_{pk}(ab; r_c) \end{array} \right\}.$$

A constant-round zero-knowledge proof for $\mathrm{L}_{\mathrm{MULT}}$ with 15 exponentiations can be found in [DJ01].[10]

[10] The original construction of [DJ01] is presented in the honest verifier setting. Deriving a statistical zero-knowledge proof can be achieved by instantiating Pedersen's commitment scheme [Ped91] with the technique of Goldreich and Kahan [GK96]. The analysis of 15 exponentiations already takes into account this adjustment.

5. There exists an efficient computational zero-knowledge proof $\pi_{\rm NZ}$ for proving that some ciphertext encrypts a non-zero plaintext. Formally,

$$\mathrm{L}_{\rm NZ} = \{(pk, e_a) \mid e = E_{pk}(a; r_a) \text{ for some } a \neq 0, r_a\}\,.$$

A simple reduction to $\pi_{\rm MULT}$ is as follows. The prover chooses random b, r_b, r_c, sets $e_b = E_{pk}(b; r_b)$, $e_c = E_{pk}(ab; r_c)$ and proves that $((pk, (e_a, e_b, e_c)), (a, r_a, b, r_b, r_c)) \in \mathcal{R}_{\rm MULT}$. If the proof is accepted, then the prover sends ab, r_c, and the verifier accepts if $e_c = E_{pk}(ab; rc)$ and $ab \neq 0$.

6. There exists an efficient computational zero-knowledge proof $\pi_{\rm ZERO}$ for proving that some ciphertext encrypts the zero plaintext. Formally,

$$\mathrm{L}_{\rm ZERO} = \{(pk, e_a) \mid e = E_{pk}(0; r_a) \text{ for some } r_a\}\,.$$

A constant-round zero-knowledge proof for $\mathrm{L}_{\rm ZERO}$ with 13 exponentiations for ElGamal can be found in [CP92] and with 8 exponentiations for Paillier [DJ01].[11]

5.2 Zero-Knowledge Proof of Knowledge for $\mathcal{R}_{\rm POLY}$

In addition to the above, we will need a zero-knowledge proof of knowledge for proving the correctness of the polynomials sent by P_1. We recall that in the set intersection protocol presented below, P_1 generates B polynomials $\{Q_i(\cdot)\}_{i\in\{1,\ldots,B\}}$ representing its input set X and sends their encryptions to P_2. We present an efficient zero-knowledge proof of knowledge $\pi_{\rm POLY}$ that is used for checking these encrypted polynomials.[12] The basic idea of our proof is to count the number of non-zero coefficients and compare this result with $m-B$ (we subtract B since there are B bins and the number of roots of a polynomial $Q_i(\cdot)$ is upper bounded by $\deg Q_i(\cdot) - 1$). To prove this, the prover sends an encryption $Z_{i,j}$ of 1 for every $0 \leq j \leq \deg(Q_i(\cdot))$, and 0 otherwise. It then proves that it computed these encryptions correctly by proving that $Z_{i,j}$ is indeed an encryption of a value within $\{0, 1\}$, and that $Q_{i,j}\cdot(1-Z_{i,j}) = 0$ for all $j \in \{0, \ldots, M\}$. However, a problem arises if $Q_i(\cdot)$ has zero coefficients. In this case the prover can convince the verifier that $\sum_i \deg(Q_i(\cdot)) = m$ even if it is actually larger (as we only count the non-zero elements). To solve this problem we add an additional check that the set $Z_{i,0}, Z_{i,1}, \ldots, Z_{i,M}$ is monotonically non-increasing, which guarantees that $\forall i$, and $j \geq \deg(Q_i(\cdot))$, the event in which $Z_{i,j} = 0$ and $Z_{i,j+1} = 1$ does not happen. The proof is concluded by having the parties sum up these values using homomorphic addition, and having the prover prove that the result is an encryption of $m - B$. We note that the attack in which P_1 encrypts the zero polynomials is prevented by assuming that the coefficient of the highest degree equals 1.

Formally, the relation is defined by,

$$\mathcal{R}_{\rm POLY} = \left\{ \Big(\{q_{i,j}\}_{i,j}, m, pk\Big), \Big(\{Q_{i,j}, r_{i,j}\}_{i,j}\Big) \middle| \begin{array}{c} \forall i,j\ q_{i,j} = E_{pk}(Q_{i,j}; r_{i,j}) \wedge \\ \sum_i \deg(Q_i(\cdot)) = m\ \wedge \forall i,\ Q_i(\cdot) \not\equiv 0 \end{array} \right\}$$

where $Q_i(x) = \sum_{j=0}^{M} Q_{i,j} \cdot x^j$, and $i \in \{1, \ldots, B\}, j \in \{0, \ldots, M\}$. A protocol for $\mathcal{R}_{\rm POLY}$, as well as a complete proof, can be found in [HN10]. For the sake of completeness, we give a slightly modified description of their protocol.

Protocol 2 (zero-knowledge proof of knowledge $\pi_{\rm POLY}$ for $\mathcal{R}_{\rm POLY}$):

Joint statement: *A collection of B sets, each set is of $M + 1$ encryptions $\{q_{i,j}\}_{i\in\{1,\ldots,B\},j\in\{0,\ldots,M\}}$, a public key pk and an integer m.*

[11] See previous footnote.

[12] We will use the convention that the degree of a polynomial $Q_i(\cdot)$ can be chosen to be any integer j' such that $Q_{i,j} = 0$ for all $j \geq j'$, hence equality with m can always be achieved.

- **Auxiliary inputs for the prover:** *A collection of B sets, each set is of $M+1$ values $\{Q_{i,j}, r_{i,j}\}_{i\in\{1,\dots,B\},j\in\{0,\dots,M\}}$ such that the conditions in $\mathcal{R}_{\mathrm{POLY}}$ are met.*
- **Convention:** *Both parties check every received ciphertext for validity, and abort if an invalid ciphertext is received. Unless written differently, $i \in \{1,\dots,B\}$ and $j \in \{0,\dots,M\}$.*
- **The protocol:**
  1. *For every i let $Q_i(x) = \sum_{j=0}^{M} Q_{i,j}\cdot x^j = \sum_{j=0}^{M} D_{sk}(q_{i,j})\cdot x^j$, and recall that by our convention (cf. Footnote 12), $\sum_i \deg(Q_i(\cdot)) = m$. Let $Z_{i,j} = 1$ for $0 \le j \le \deg(Q_i(\cdot))$, and otherwise $Z_{i,j} = 0$ (i.e., $Z_{i,0} = 1, Z_{i,1},\dots,Z_{i,M}$ is monotonically non-increasing, and $\sum_{j\in\{0,\dots,M\}} Z_{i,j} = \deg(Q_i(\cdot)) + 1$).*
  2. *The prover P computes $z_{i,j} = E_{pk}(Z_{i,j};\tilde{r}_{i,j})$ and sends $\{z_{i,j}\}_{i,j}$ to the verifier V.*
     $\forall i$, P performs the following:
     (a) *For each $z_{i,j}$ encrypting a value $Z_{i,j}$ it proves that $Z_{i,j}\cdot(1-Z_{i,j}) = 0$ (and hence $Z_{i,j}\in\{0,1\}$).*[13] *Let $m_{i,j} = E_{pk}(0;\hat{r}_{i,j})$. Then P proves that $\big((pk, z_{i,j}, z'_{i,j}, m_{i,j}), (Z_{i,j}, \tilde{r}_{i,j}, 1-Z_{i,j}, \tilde{r}'_{i,j}, \hat{r}_{i,j})\big) \in \mathcal{R}_{\mathrm{MULT}}$ and that $m_{i,j}$ is an encryption of zero by decrypting $m_{i,j}$.*
     (b) *P proves that $Z_{i,0}, Z_{i,1},\dots,Z_{i,M}$ is monotonically non-increasing, i.e., that $Z_{i,j} = 0$ and $Z_{i,j+1} = 1$ does not happen for any value of $j \in \{\deg(Q_i(\cdot)),\dots,M-1\}$. For that, P and V compute an encryption of $2Z_{i,j} + Z_{i,j+1} - 1$ (note that both parties can compute this encryption) and P proves in zero-knowledge that $(pk, E_{pk}(2Z_{i,j}+Z_{i,j+1}-1)) \in \mathrm{L}_{\mathrm{NZ}}$.*
     (c) *P completes the proof that the values $Z_{i,j}$ were constructed correctly by proving (similarly to Step 2a above) that $Q_{i,j}\cdot(1-Z_{i,j}) = 0$ for all $j\in\{0,\dots,M\}$.*
  3. *Finally, to prove that the sum of degrees of the polynomials $\{Q_i(\cdot)\}_i$ equals m, both parties compute an encryption t of $T = \sum_{i,j} Z_{i,j} - B$, and P proves that $(pk, t/E_{pk}(m)) \in \mathrm{L}_{\mathrm{ZERO}}$.*
  4. *V verifies all the zero-knowledge proofs and decryptions. If any of the verifications fails, V outputs 0, otherwise, it outputs 1.*

Note that Protocol π_{POLY} runs in a constant number of rounds because each of the zero-knowledge proofs can be implemented in constant rounds and can be invoked in parallel. The parties compute and exchange $O(BM) = O(m)$ encryptions and execute $O(BM) = O(m)$ zero-knowledge proofs (for $\mathcal{R}_{\mathrm{MULT}}, \mathrm{L}_{\mathrm{NZ}}, \mathrm{L}_{\mathrm{ZERO}}$). Overall, these amount to performing $O(m)$ exponentiations and exchanging $O(m)$ group elements.

Proposition 5.1 *Assume that π_{MULT}, π_{ZERO} and π_{NZ} are as described above and that (G,E,D) is homomorphic PKE that is good for plaintext space (cf. Definition 3.2). Then Protocol 2 is a computational zero-knowledge proof of knowledge for $\mathcal{R}_{\mathrm{POLY}}$ with perfect completeness.*

A complete proof can be found in [HN10]. Intuitively, the zero-knowledge property follows from the fact that all the sub-protocols are zero-knowledge as well. Knowledge extraction follows from the fact that π_{MULT} is a proof of knowledge proof, so that the plaintexts and randomness can be extracted in Step 2c. Finally, soundness follows from soundness of sub-protocols, ensuring that the degree of the polynomial cannot exceed M.

[13] The proof can be easily computed since both parties can compute an encryption z' of $1-Z_{i,j}$, where P can recover randomness $\tilde{r}'_{i,j}$ that is consistent with this encryption. We also assume that P and V agree on an encryption of 1, for which both know the randomness.

5.3 Secure Set Intersection in the Random Oracle Model

We are now ready to present a formal description of our protocol in a setting where the parties are malicious and have oracle access to a random function $\mathcal{H} : \mathbb{G} \to \mathbb{M}\|\mathbb{M}\|R$, i.e., the outcome is split into three values: (1) The first value is used to mask the encrypted polynomial evaluation (for the message space specified for either Paillier or ElGamal), (2) the second value is used to mask the input to the polynomial, (3) and the third value is used to encrypt the values s_0, s_1 (to rerandomize the ciphertext encrypting the outcome of the polynomial evaluation), for R denoting the randomness space from which the random coins for the specified PKE are sampled from. A high-level description is presented in Figure 4.

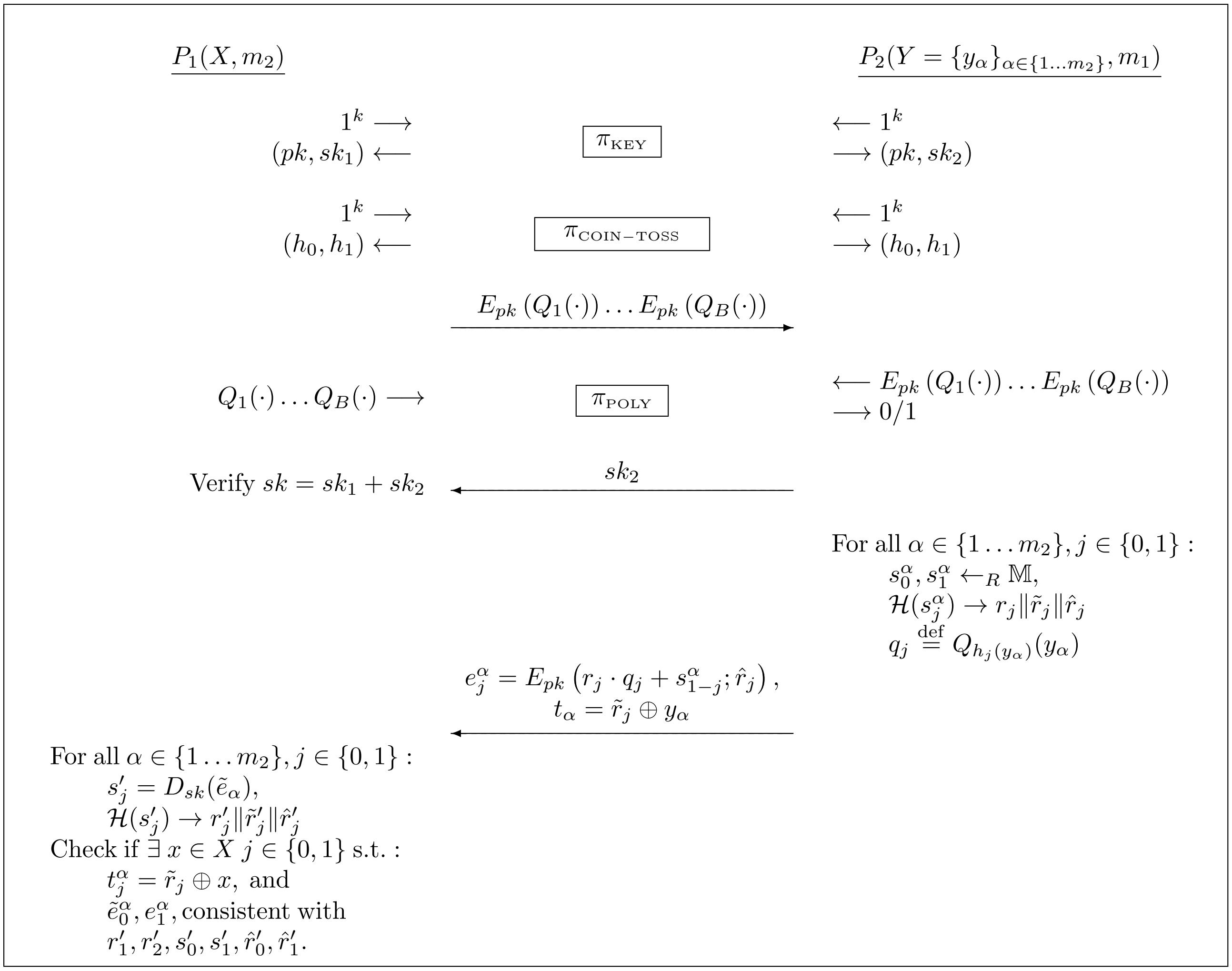
$P_1(X, m_2)$
$P_2(Y = \{y_\alpha\}_{\alpha\in\{1\ldots m_2\}}, m_1)$
$1^k \longrightarrow$
$(pk, sk_1) \longleftarrow$
π_{KEY}
$\longleftarrow 1^k$
$\longrightarrow (pk, sk_2)$
$1^k \longrightarrow$
$(h_0, h_1) \longleftarrow$
$\pi_{\text{COIN-TOSS}}$
$\longleftarrow 1^k$
$\longrightarrow (h_0, h_1)$
$E_{pk}(Q_1(\cdot)) \ldots E_{pk}(Q_B(\cdot))$
$Q_1(\cdot) \ldots Q_B(\cdot) \longrightarrow$
π_{POLY}
$\longleftarrow E_{pk}(Q_1(\cdot)) \ldots E_{pk}(Q_B(\cdot))$
$\longrightarrow 0/1$
Verify $sk = sk_1 + sk_2$
sk_2
For all $\alpha \in \{1 \ldots m_2\}, j \in \{0,1\}$:
$s_0^\alpha, s_1^\alpha \leftarrow_R \mathbb{M}$,
$\mathcal{H}(s_j^\alpha) \to r_j\|\tilde{r}_j\|\hat{r}_j$
$q_j \overset{\text{def}}{=} Q_{h_j(y_\alpha)}(y_\alpha)$
$e_j^\alpha = E_{pk}\left(r_j \cdot q_j + s_{1-j}^\alpha; \hat{r}_j\right)$,
$t_\alpha = \tilde{r}_j \oplus y_\alpha$
For all $\alpha \in \{1 \ldots m_2\}, j \in \{0,1\}$:
$s'_j = D_{sk}(\tilde{e}_\alpha)$,
$\mathcal{H}(s'_j) \to r'_j\|\tilde{r}'_j\|\hat{r}'_j$
Check if $\exists\, x \in X\ j \in \{0,1\}$ s.t. :
$t_j^\alpha = \tilde{r}_j \oplus x$, and
$\tilde{e}_0^\alpha, e_1^\alpha$, consistent with
$r'_1, r'_2, s'_0, s'_1, \hat{r}'_0, \hat{r}'_1$.

Figure 4: A high-level diagram of $\pi_\cap^{\text{RO}}$ (random oracle model).

Protocol 3 ($\pi_\cap^{\text{RO}}$ – secure set intersection in the random oracle model):

- **Inputs:** *The input of P_1 is m_2 and a set $X \subseteq \{0,1\}^{p(k)}$ containing m_1 items; the input of P_2 is m_1 and a set $Y \subseteq \{0,1\}^{p(k)}$ containing m_2 items (hence, both parties know m_1 and m_2).*
- **Auxiliary inputs:** *A security parameter 1^k.*
- **Convention:** *Both parties check every received ciphertext for validity, and abort if an invalid ciphertext is received.*
- **The protocol:**

1. **Key setup:** *P_1 and P_2 run an execution of π_{KEY} for generating pk and random shares sk_1, sk_2 of sk. Let (pk, sk_1) and (pk, sk_2) denote the respective outputs of P_1 and P_2 from this execution.*
2. **Setting the balanced allocation scheme:** *P_1 computes the parameters B, M for the scheme and the parties run a coin tossing protocol in order to choose the seeds for two randomly chosen hash functions $h_0, h_1 : \{0,1\}^{p(k)} \to [B]$. It sends B, M, h_0, h_1 to P_2. P_2 checks that the parameters B, M and the seeds h_0, h_1 were computed correctly, and aborts otherwise.*
3. **Creating polynomials for the set X:** *For every $x \in X$, P_1 maps x into the less occupied bin from $\{h_0(x), h_1(x)\}$ (ties broken arbitrarily). Let $\mathcal{B}_i$ denote the set of elements mapped into bin i and let $Q_i(x) \stackrel{\text{def}}{=} \sum_{j=0}^{M} Q_{i,j} \cdot x^j$ denote a polynomial with the set of roots $\mathcal{B}_i$ (if $\mathcal{B}_i = \emptyset$ then P_1 sets $Q_i(x) = 1$). If $|\mathcal{B}_i| < M$ then P_1 sets the $M + 1 - |\mathcal{B}_i|$ highest-degree coefficients of $Q_i(\cdot)$ to zero.*

   P_1 encrypts the coefficients of the polynomials, setting $q_{i,j} = E_{pk}(Q_{i,j}; r_{i,j})$. It sends the encrypted coefficients to P_2.
4. **Checking the polynomials:** *P_1 and P_2 engage in an execution of π_{POLY} for which P_1 enters pk and the sets $\{Q_{i,j}, r_{i,j}\}_{i \in \{1,\ldots,B\}, j \in \{0,\ldots,M\}}$, and P_2 enters the sets $\{q_{i,j}\}_{i \in \{1,\ldots,B\}, j \in \{0,\ldots,M\}}$. If the outcome is not 1 then P_2 aborts.*
5. **Key setup completion:** *P_2 sends his share of the private key sk_2, as well as the randomness used for generating this share, to P_1. P_1 reconstructs sk from the shares sk_1, sk_2 and checks that the result is a valid private-key that corresponds to pk, and that P_2 sent the correct share. If it is not, it aborts.*
6. **Substituting in the polynomials:** *Let $y_1, \ldots, y_{m_2}$ be a random ordering of the elements of set Y. P_2 does the following for all $\alpha \in \{1, \ldots, m_2\}$:*
   (a) *It sets $\hat{h}_0 = h_0(y_\alpha)$, $\hat{h}_1 = h_1(y_\alpha)$ (these values denote the bin number). Then it chooses $s_0^\alpha, s_1^\alpha \leftarrow_R \mathbb{M}$ and parses $\mathcal{H}(s_j^\alpha)$ to obtain random strings $r_j, \tilde{r}_j, \hat{r}_j$ of appropriate lengths for their usage below, for all $j \in \{0,1\}$ (i.e., $r_j, \tilde{r}_j, \hat{r}_j = \mathcal{H}(s_j^\alpha)$).*

   (b) *P_2 uses the homomorphic properties of the encryption scheme to evaluate $E_{pk}(r_0 \cdot Q_{\hat{h}_0}(y_\alpha))$ and $E_{pk}(r_1 \cdot Q_{\hat{h}_1}(y_\alpha))$. It then applies an homomorphic operation on these ciphertexts and $E_{pk}(s_1^\alpha; \hat{r}_0)$ and $E_{pk}(s_0^\alpha; \hat{r}_1)$, respectively. This results in $e_0^\alpha = E_{pk}(r_0 \cdot Q_{\hat{h}_0}(y_\alpha) + s_1^\alpha)$ and $e_1^\alpha = E_{pk}(r_1 \cdot Q_{\hat{h}_1}(y_\alpha) + s_0^\alpha)$.*
   Finally, P_2 sends e_0^α, e_1^α together with $t_0^\alpha = \tilde{r}_0 \oplus y_\alpha$, $t_1^\alpha = \tilde{r}_1 \oplus y_\alpha$ to P_1.
7. **Computing the intersection:** *P_1 checks that P_2 sent m_2 tuples and aborts otherwise. For each received $e_0^\alpha, e_1^\alpha, t_0^\alpha, t_1^\alpha$ party P_1 computes $s'_0 = D_{sk}(e_1^\alpha)$ and $s'_1 = D_{sk}(e_0^\alpha)$, and sets $r'_j, \tilde{r}'_j, \hat{r}'_j = \mathcal{H}(s'_j)$ for all j. If $t_j^\alpha = \tilde{r}'_j \oplus x$ for some $x \in X$ and $j \in \{0,1\}$, then P_1 records $s_j^x = s'_j$ and $s_{1-j}^x = s'_{1-j} - r'_j \cdot Q_{h_j(x)}(x)$. P_1 uses these recorded values and x in order to recompute the set $e_0^\alpha, e_1^\alpha, t_0^\alpha, t_1^\alpha$. If its outcome agrees with P_2's messages it records x.*

   Namely, P_1 recomputes e_j^α and t_j^α using P_2's procedure in Step 6, but with s_0^x, s_1^x in the role of s_0^α, s_1^α.

Correctness. Notice first that if both parties are honest, then P_1 outputs $X \cap Y$ with probability negligibly close to 1: (**i**) For elements $x \in X \cap Y$ there exists a y_α that zeros at least one of the polynomials $Q_{h_0(y_\alpha)}(\cdot)$, $Q_{h_1(y_\alpha)}(\cdot)$, say w.l.o.g. the first polynomial. Hence, getting the messages sent for y_α, P_1 recovers $s'_1 = s_1^\alpha$ immediately. Moreover, since it already knows y_α from t_1^α, and r_1 from $\mathcal{H}(s_1^\alpha)$, it can recover s_0^α as well by simply computing $s_0^\alpha = s'_0 - r_1 \cdot Q_{h_1(y_\alpha)}(y_\alpha)$, and verify that the recomputed encryptions $e_0^\alpha, e_1^\alpha, t_0^\alpha, t_1^\alpha$ agree with P_2's messages. Consequently, P_1 records x in its output. (**ii**) For elements $x \in X \setminus Y$, P_1 records x only if for some y_α and j the values $r'_j, \tilde{r}'_j, \hat{r}'_j = \mathcal{H}(s'_j)$ are such that $t_j^\alpha = \tilde{r}'_j \oplus x$ and furthermore, the conditions on $e_0^\alpha, e_1^\alpha, t_0^\alpha, t_1^\alpha$ are met. As proven below, this occurs with only a negligible probability.

Theorem 5.1 *Assume that π_{KEY} and π_{POLY} are as described above, that (G, E, D) is a semantically secure homomorphic PKE that is good for plaintext space (cf. Definition 3.2), and that $\mathcal{H}$ is a random oracle. Then Protocol 3 securely computes $\mathcal{F}_\cap$ in the presence of malicious adversaries in the random oracle model.*

Proof: We separately prove security in the case that P_1 is corrupted and the case that P_2 is corrupted. Our proof is in a hybrid model where a trusted party computes the ideal functionalities $\mathcal{F}_{\text{KEY}}$ and the zero-knowledge proof of knowledge $\mathcal{R}_{\text{POLY}}$.

***P_1* is corrupted.** Intuitively, all a corrupted P_1 can do in protocol $\pi_{\cap}^{\text{RO}}$ is to try and construct the polynomials $Q_i(\cdot)$ such that it gains information about P_2's input Y. There are two provisions in the protocol against such an attempt: (i) The application of π_{POLY} ensures that as long as all the homomorphic operations are within elements of $\mathbb{G}$ (as specified in Definition 3.2), P_1 should fail; and (ii) Letting P_1 learn sk only after it constructs the polynomials ensures that this would indeed be the case. The following proof makes this intuition formal. Let $\mathcal{A}$ denote an adversary controlling P_1. Construct a simulator $\mathcal{S}$ for $\mathcal{A}$ as follows:

> **Convention:** During the entire execution $\mathcal{S}$ evaluates queries to the random oracle $\mathcal{H}$. Such queries are made by $\mathcal{A}$ or by $\mathcal{S}$ during its simulation of P_2. To evaluate $\mathcal{H}(s)$, $\mathcal{S}$ first checks if it has already recorded a pair (s, r), in which case $\mathcal{H}(s)$ evaluates to the value r. Otherwise, $\mathcal{S}$ chooses a random string r of the appropriate length, records (s, r) and evaluates $\mathcal{H}(s)$ to r.

1. $\mathcal{S}$ is given X, m_2 and 1^k and invokes $\mathcal{A}$ on these inputs. $\mathcal{S}$ sets $m_1 = |X|$.

2. $\mathcal{S}$ receives from $\mathcal{A}$ its input 1^k for the ideal functionality $\mathcal{F}_{\text{KEY}}$ and computes $(pk, sk) \leftarrow G(1^k)$. Next, $\mathcal{S}$ computes random shares sk_1, sk_2 such that the reconstruction with these shares yields sk, and sends (pk, sk_1) to $\mathcal{A}$, emulating the response of a trusted party for $\mathcal{F}_{\text{KEY}}$.

3. Upon engaging in a coin tossing protocol with $\mathcal{A}$, $\mathcal{S}$ receives from $\mathcal{A}$ the parameters B, M and the seeds for the two random chosen hash functions $h_0, h_1 : \{0,1\}^{p(k)} \to [B]$ used in the balanced allocation scheme. If the parameters B, M or h_0, h_1 were not computed correctly, $\mathcal{S}$ sends $\perp$ to the trusted party for $\mathcal{F}_\cap$ and aborts.

4. $\mathcal{S}$ receives from $\mathcal{A}$ the encrypted polynomials $\{q_{i,j}\}_{i\in\{1,\ldots,B\},j\in\{0,\ldots,M\}}$.

5. $\mathcal{S}$ receives from $\mathcal{A}$ its input $\{Q_{i,j}, r_{i,j}\}_{i\in\{1,\ldots,B\},j\in\{0,\ldots,M\}}$ and pk for the ideal computation $\mathcal{R}_{\text{POLY}}$. If the conditions of $\mathcal{R}_{\text{POLY}}$ for outputting $(\lambda, 1)$ are not met then $\mathcal{S}$ sends $\perp$ to the trusted party for $\mathcal{F}_\cap$ and aborts.

6. $\mathcal{S}$ sends sk_2 to $\mathcal{A}$.

7. $\mathcal{S}$ sets $\tilde{X} = \cup_{i=1}^{B}\{x : Q_i(x) = 0 \wedge (h_0(x) = i \vee h_1(x) = i)\}$ (note that for Paillier, $\mathcal{S}$ would know the factorization p and q of the public key N, so that it can factor $Q_i(\cdot)$ over the fields $\mathbb{Z}_q$ and $\mathbb{Z}_p$, recovering the roots over $\mathbb{Z}_N$ using the Chinese Reminder Theorem), and completes $\tilde{X}$ to size m_1 by adding random elements from $\{0,1\}^{p(k)}$. $\mathcal{S}$ sends $\tilde{X}$ to the trusted party for $\mathcal{F}_\cap$ and receives as answer a set $Z = \tilde{X} \cap Y$. $\mathcal{S}$ sets $\tilde{Y}$ to Z and completes $\tilde{Y}$ to the size m_2 by adding random elements from $\{0,1\}^{p(k)}$.

8. $\mathcal{S}$ plays the role of the honest P_2 for the rest of the execution, using $\tilde{Y}$ as the input for P_2, with the exception that $\mathcal{S}$ itself evaluates queries to $\mathcal{H}$ as described above.

9. $\mathcal{S}$ outputs whatever $\mathcal{A}$ does.

We claim that $\mathcal{A}$'s output distributions in the hybrid and the simulated executions are statistically close. The potential difference between these executions is in Step 7 of the simulation, where $\mathcal{S}$ plays the role of the honest P_2 with $\tilde{Y}$ instead of Y. Ignoring the case where $Y \cap \tilde{X} \neq Y \cap X$ (which happens with only a negligible probability), the difference is that members of $\tilde{Y}' = \tilde{Y} \setminus (Y \cap X)$ are used in the simulated execution where members of $Y' = Y \setminus X = Y \setminus (Y \cap X)$ are used in the hybrid execution. Let $y \in Y'$ ($y \in \tilde{Y}'$) be the αth element considered in the hybrid (simulated) execution, then $\mathcal{A}$ receives four values: $e_0^\alpha, e_1^\alpha, t_0^\alpha, t_1^\alpha$. Note first that unless $\mathcal{A}$ recovers either s_0^α or s_1^α the plaintexts encrypted under e_0^α, e_1^α are statistically close to random messages. Similarly, the strings t_0^α and t_1^α are statistically close to uniformly selected strings in $\mathbb{M}$. Hence, it suffices to show that s_0^α, s_1^α cannot be recovered, unless with negligible probability.

Recall that $e_j^\alpha = E_{pk}(r_j \cdot Q_{h_j(y)}(y) + s_{1-j}^\alpha; \hat{r}_j)$. Then if $Q_{h_0(y)}(y), Q_{h_1(y)}(y) \in \mathbb{G}$, except with negligible probability, r_j is uniformly distributed in $\mathbb{G}$ and we get that $r_j \cdot Q_{h_j(y)}(y)$ is uniformly distributed in $\mathbb{G}$. Therefore the probability of guessing s_{1-j}^α is negligible in the order of $\mathbb{G}$. Moreover, the probability that $Q_{h_0(y)}(y) \notin \mathbb{G}$ or $Q_{h_1(y)}(y) \notin \mathbb{G}$ is negligible both in the hybrid and in the simulated executions by Item 1 in Section 5.1. This concludes the proof.

***P_2* is corrupted.** Intuitively, we should prevent a corrupted P_2 from learning about P_1's input, and from making P_1 output a wrong output. The latter concern is dealt with using a technique we mentioned above, i.e., by having P_1 recover P_2's randomness from $\mathcal{H}(s)$ and verify that P_2's messages that result in an element supposedly in the intersection are in accordance with what a honest P_2 would have sent.

Getting to the formal proof, let $\mathcal{A}$ denote an adversary controlling P_2, and construct a simulator $\mathcal{S}$ for $\mathcal{A}$ as follows:

> **Convention:** During the entire execution $\mathcal{S}$ evaluates queries to the random oracle $\mathcal{H}$. Such queries are made by $\mathcal{A}$ or by $\mathcal{S}$ during its simulation of P_1. To evaluate $\mathcal{H}(s)$, $\mathcal{S}$ first checks if it has already recorded a pair (s, r), in which case $\mathcal{H}(s)$ evaluates to the value r. Otherwise, $\mathcal{S}$ chooses a random string r of the appropriate length, records (s, r) and evaluates $\mathcal{H}(s)$ to r.

1. $\mathcal{S}$ is given Y, m_1 and 1^k and invokes $\mathcal{A}$ on these inputs. $\mathcal{S}$ sets $m_2 = |Y|$.

2. $\mathcal{S}$ receives from $\mathcal{A}$ its input 1^k for the ideal functionality $\mathcal{F}_{\text{KEY}}$ and computes $(pk, sk) \leftarrow G(1^k)$. Next $\mathcal{S}$ chooses random shares such that the reconstruction with these shares yields sk, and sends (pk, sk_2) to $\mathcal{A}$, emulating the response of a trusted party for $\mathcal{F}_{\text{KEY}}$.

3. Upon engaging in a coin tossing protocol with $\mathcal{A}$, $\mathcal{S}$ computes the parameters B, M for the balanced allocation scheme and the seeds for the hash functions h_0, h_1. These are then sent to $\mathcal{A}$.

4. $\mathcal{S}$ sends to $\mathcal{A}$, BM encryptions of the value 0, under the key pk (i.e., encryptions of B zero-polynomials). Each encryption is done with fresh randomness.

5. $\mathcal{S}$ emulates the ideal computation of $\mathcal{R}_{\text{POLY}}$. It receives from $\mathcal{A}$ a set of BM coefficients and pk. If $\mathcal{A}$'s input is the exact set of encryptions that it received from $\mathcal{S}$ in the previous step, and pk then $\mathcal{S}$ returns 1, otherwise it returns 0.

6. $\mathcal{S}$ receives from $\mathcal{A}$ its private-key share. If $\mathcal{A}$ does not send sk_2, $\mathcal{S}$ aborts, sending $\perp$ to the trusted party for $\mathcal{F}_\cap$.

7. In case $\mathcal{A}$ sends more than m_2 tuples in Step 6 of the protocol $\mathcal{S}$ aborts, sending $\perp$ to the trusted party.

8. For every $e_0^\alpha, e_1^\alpha, t_0^\alpha, t_1^\alpha$ received from $\mathcal{A}$ in the simulation of Step 6 of the protocol, every $j \in \{0, 1\}$ and every pairs $(s_0, r_0), (s_1, r_1)$ recorded by $\mathcal{S}$ as part of its evaluation of the random oracle $\mathcal{H}$, $\mathcal{S}$ performs the following: (i) it parses r_j as $r_j' \| \tilde{r}_j \| \hat{r}_j$ and sets $y_j = t_j^\alpha \oplus \tilde{r}_j$. (ii) For all $j \in \{0, 1\}$ it checks if $e_0^\alpha, e_1^\alpha, t_0^\alpha, t_1^\alpha$ are consistent with what P_2 should have sent on $y_\alpha = y_j$ given $s_{1-j}^\alpha = s_{1-j}$ and $r_j', \tilde{r}_j, \hat{r}_j$. That is, $\mathcal{S}$ recomputes these encryptions using s, r and y_α as the honest P_2 would in

the real execution, and checks whether the result equals $e_0^\alpha, e_1^\alpha, t_0^\alpha, t_1^\alpha$. If the check succeeds $\mathcal{S}$ locally records the value y_α.

Naively implementing the above algorithm, the computational complexity of the simulator is quadratic in the number of queries. It is possible to reduce this cost into a linear overhead by additional encoding a common payload in the ciphertexts. Namely, the input to the random oracle would be the s_i's concatenated with additional string that is in common for both s_0 and s_1.

9. $\mathcal{S}$ sets $\tilde{Y}$ to the set of values recorded in the previous step, and completes $\tilde{Y}$ to the size m_2 by adding random elements from $\{0,1\}^{p(k)}$.

10. $\mathcal{S}$ sends $\tilde{Y}$ to the trusted party and outputs whenever $\mathcal{A}$ does.

In the following we define a sequence of hybrid games and denote by the random variable $\mathbf{H}_\ell^{\mathcal{A}(z)}(k, X, Y)$ (for a fixed k) the joint output of $\mathcal{A}$ and P_1 in hybrid game $\mathbf{H}_\ell$.

Game $\mathbf{H}_0$: The simulated execution.

Game $\mathbf{H}_1$: The simulator $\mathcal{S}_1$ acts identically to $\mathcal{S}$ except that it does not get to know sk. The random variables $\mathbf{H}_0^{\mathcal{A}(z)}(k, X, Y)$ and $\mathbf{H}_1^{\mathcal{A}(z)}(k, X, Y)$ are identical as $\mathcal{S}$ and $\mathcal{S}_1$ do not use sk.

Game $\mathbf{H}_2$: In this game there is no trusted party and no honest P_1. Instead, the simulator $\mathcal{S}_2$ is given as input P_1's real input X. $\mathcal{S}_2$ works exactly like $\mathcal{S}_1$, except that instead of sending zero polynomials, it computes the polynomials as in the hybrid execution using the set X. In addition, $\mathcal{S}_2$ does not send $\tilde{Y}$ to the trusted party, but uses X to compute and output $X \cap \tilde{Y}$. The random variables $\mathbf{H}_1^{\mathcal{A}(z)}(k, X, Y)$ and $\mathbf{H}_2^{\mathcal{A}(z)}(k, X, Y)$ are computationally indistinguishable via a reduction to the security of (G, E, D). Assume, for contradiction, the existence of a distinguisher circuit D for $\mathbf{H}_1^{\mathcal{A}(z)}(k, X, Y)$ and $\mathbf{H}_2^{\mathcal{A}(z)}(k, X, Y)$ and choose hash functions h_0, h_1 that maximize D's distinguishing advantage. Construct a distinguisher circuit D_{E} that distinguishes between the encryptions of two sets of messages: (i) coefficients of zero-polynomials as constructed in game $\mathbf{H}_1$, and (ii) coefficients of polynomials corresponding to X and hash functions h_0, h_1 as constructed in game $\mathbf{H}_2$. D_{E} receives a public key pk and a vector of encryptions $\overline{c}$ under pk and works exactly like $\mathcal{S}_2$ except that it uses the public key given to it as input instead of the public key it generated within $\mathcal{F}_{\mathrm{KEY}}$, and forwards to $\mathcal{A}$ the vector of encryptions $\overline{c}$ instead of the encrypted polynomials. Note that the output distribution generated by D_{E} is either identical to $\mathbf{H}_1^{\mathcal{A}(z)}(k, X, Y)$ or to $\mathbf{H}_2^{\mathcal{A}(z)}(k, X, Y)$ (conditioned on our choice of h_0, h_1), and hence the existence of D contradicts the indistinguishability of ciphertexts of the encryption scheme (G, E, D).

Game $\mathbf{H}_3$: The simulator $\mathcal{S}_3$ acts identically to $\mathcal{S}_2$ except that $\mathcal{S}_3$ is given sk. The random variables $\mathbf{H}_2^{\mathcal{A}(z)}(k, X, Y)$ and $\mathbf{H}_3^{\mathcal{A}(z)}(k, X, Y)$ are identical as $\mathcal{S}_2$ and $\mathcal{S}_3$ do not use sk.

Game $\mathbf{H}_4$: In this game $\mathcal{S}_4$ acts identically to $\mathcal{S}_3$ except that $\mathcal{S}_4$ performs in Step 6 of the protocol the same check as in the hybrid execution, i.e., first decrypt and then recompute. To conclude the proof, we show that the random variables $\mathbf{H}_3^{\mathcal{A}(z)}(k, X, Y)$ and $\mathbf{H}_4^{\mathcal{A}(z)}(k, X, Y)$ are statistically close. Neglecting the event that new elements added to $\tilde{Y}$ are in X as well and ignoring this set, observe first that if an element y_α satisfies the conditions for being included in the output in Game $\mathbf{H}_3$, it also satisfies the conditions for being included in the output in Game $\mathbf{H}_4$. Namely, if $\mathcal{S}_3$ outputs an element y_α then it must be that $y_\alpha \in X$, and either e_0^α or e_1^α encrypts s_j^α, and hence $\mathcal{S}_4$ would have outputted it.

Consider now the reverse direction. Let bad denote the event where there exists an element $x \in X$ such that $\mathcal{S}_4$ decided to output, but should not have been outputted by $\mathcal{S}_3$. We show that $\Pr[\mathsf{bad}]$ is negligible. Note that for bad to occur it must be that for some j, one of e_0^α, e_1^α is decrypted into s_j^α such that for some $x \in X$ the values $e_0^\alpha, e_1^\alpha, t_0^\alpha, t_1^\alpha$ are consistent with setting $y = x$ and the randomness obtained from $\mathcal{H}(s_0^\alpha)$

and $\mathcal{H}(s_1^{\alpha})$ and this occurs only with negligible probability due to the fact that the output by $\mathcal{H}$ is truly random.

Game $\mathbf{H}_5$: The hybrid execution. The random variables $\mathbf{H}_4^{\mathcal{A}(z)}(k, X, Y)$ and $\mathbf{H}_5^{\mathcal{A}(z)}(k, X, Y)$ are identical as the only difference between the executions is that in $\mathbf{H}_4$ the outcome of the random oracle is chosen upon request, whereas in $\mathbf{H}_5$ it is chosen before the execution. In both cases the outcome of the execution does not depend on entries of the random oracle which are not accessed. ■

Efficiency. Protocol $\pi_{\cap}^{\mathrm{RO}}$ runs in a constant number of rounds, because all the zero-knowledge proofs can be implemented in constant rounds, and also the implementations of π_{KEY} and π_{POLY} are constant round. Implementation of π_{KEY} depends on the homomorphic PKE that is used. For ElGamal the key setup of [DH76] can be used here with additional zero-knowledge proofs of knowledge for proving a knowledge of discrete logarithm. The overhead induced by this protocol is relatively small since each such zero-knowledge proof of knowledge requires 9 exponentiations. A protocol for Paillier requires an increased overhead since its costs depend on the number of attempts to successfully generate a pair of primes which takes an expected number of 1936 trials, where for each trial the number of exponentiations for generating and testing the composite is constant [BF01]. Clearly, ElGamal gives better efficiency with a shorter security parameter (i.e., 160 bits compared to 1024/2048).

The overhead induced by protocol π_{POLY} is $O(BM)$ which amounts to performing $O(m_1)$ exponentiations and exchanging $O(m_1)$ group elements. In the last two steps of the protocol P_2 substitutes m_2 values in the encrypted polynomials and sends the results of the substitution to P_1. Neglecting the costs of invoking the random oracle $\mathcal{H}$, this amounts to performing $O(m_2 M) = O(m_2 \log\log m_1)$ modular exponentiations and communicating $O(m_2)$ group elements. We get that the overall communication costs are of sending $O(m_1 + m_2)$ group elements, and the computation costs are of performing $O(m_1 + m_2 \log\log m_1)$ modular exponentiations.

References

[ABKU99] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. Balanced allocations. *SIAM Journal on Computing*, 29(1):180–200, 1999.

[ACM88] *Proc. Twentieth Annual ACM Symposium on Theory of Computing*, Chicago, Illinois, 2–4 May 1988.

[ACT11] Giuseppe Ateniese, Emiliano De Cristofaro, and Gene Tsudik. (if) size matters: Size-hiding private set intersection. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *Public Key Cryptography*, volume 6571 of *Lecture Notes in Computer Science*, pages 156–173. Springer, 2011.

[ADW12] Martin Aumller, Martin Dietzfelbinger, and Philipp Woelfel. Explicit and efficient hash families suffice for cuckoo hashing with a stash. 2012.

[AIR01] 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.

[AKS83] Miklós Ajtai, János Kolmós, and Endre Szemerédi. An $O(n \log n)$ sorting network. In *STOC*, pages 1–9, 1983.

[AL07] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 137–156. Springer, 2007.

[Bat68] Kenneth E. Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computing Conference*, pages 307–314, 32(1968).

[Bea91] Donald Beaver. Foundations of secure interactive computing. *CRYPTO*, 576:377–391, 1991.

[BF01] Dan Boneh and Matthew K. Franklin. Efficient generation of shared rsa keys. *J. ACM*, 48(4):702–722, 2001.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In ACM [ACM88], pages 1–10.

[BM01] Andrei Z. Broder and Michael Mitzenmacher. Using multiple hash functions to improve ip lookups. In *IEEE INFOCOM 2001. 20th Ann. Joint Conference of the IEEE Computer and Communications Societies. Proceedings*, 2001.

[BST01] 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.

[Can00] Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In ACM [ACM88], pages 11–19.

[CJS10] Jung Hee Cheon, Stanislaw Jarecki, and Jae Hong Seo. Multi-party privacy-preserving set intersection with quasi-linear complexity. Cryptology ePrint Archive, Report 2010/512, 2010.

[CP92] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *CRYPTO*, pages 89–105, 1992.

[CZ09] Jan Camenisch and Gregory M. Zaverucha. Private intersection of certified sets. In Roger Dingledine and Philippe Golle, editors, *Financial Cryptography*, volume 5628 of *Lecture Notes in Computer Science*, pages 108–127. Springer, 2009.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, November 1976.

[DJ01] 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.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662, 2012.

[DSMRY09] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS*, volume 5536 of *Lecture Notes in Computer Science*, pages 125–142, 2009.

[DW03] Martin Dietzfelbinger and Philipp Woelfel. Almost random graphs with simple hash functions. In *Proceedings of the 35th Annual ACM Symposium on Theory of Computing*, pages 629–638, 2003.

[DZ13] Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *TCC*, pages 621–641, 2013.

[EGS03] 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.

[ElG85] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer, 2005.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology—EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19. Springer-Verlag, 2–6 May 2004.

[FNW96] Ronald Fagin, Moni Naor, and Peter Winkler. Comparing information without leaking it. *Communications of the ACM*, 39(5):77–85, 1996.

[GK96] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–189, 1996.

[GL90] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. *CRYPTO*, 537:77–93, 1990.

[GMP09] GMP. GNU Multiple Precision Arithmetic Library. gmplib.org, 2009.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proc. Nineteenth Annual ACM Symposium on Theory of Computing*, pages 218–229, New York City, 25–27 May 1987.

[Gol04] Oded Goldreich. *Foundations of cryptography: Basic applications*. Cambridge Univ Pr, 2004.

[HFH99] 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.

[HL08] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC*, pages 155–175, 2008.

[HMRT12] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, and Tomas Toft. Efficient rsa key generation and threshold paillier in the two-party setting. In *CT-RSA*, pages 313–331, 2012.

[HN10] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. In *Public Key Cryptography*, pages 312–331, 2010.

[IR89] 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.

[JL09] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. *TCC*, 5444:577–594, 2009.

[JL10] Stanislaw Jarecki and Xiaomin Liu. Fast secure computation of set intersection. *SCN*, 6280:418–435, 2010.

[JS07] Stanislaw Jarecki and Vitaly Shmatikov. Efficient two-party secure computation on committed inputs. *EUROCRYPT*, 4515:97–114, 2007.

[KMW08] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. In *Proceedings of the 16th annual European symposium on Algorithms*, pages 611–622. Springer, 2008.

[KOY01] Efficient password-authenticated key exchange using human-memorable passwords. In *Advances in Cryptology - EUROCRYPT 2001, Innsbruck, Austria, May 6-10, 2001*, pages 475–494, 2001.

[KS92] Bala Kalyanasundaram and Georg Schnitger. The probabilistic communication complexity of set intersection. *SIAM J. Discrete Mathematics*, 5(4):545–557, 1992.

[KS05] Lea Kissner and Dawn Song. Private and threshold set-intersection. In *Proceedings of CRYPTO '05*, August 2005.

[Lip03] Helger Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In *Advances in Cryptology—ASIACRYPT 2003*, pages 416–433, Taipei, Taiwan, November 2003.

[LL07] Sven Laur and Helger Lipmaa. A new protocol for conditional disclosure of secrets and its applications. In *ACNS*, pages 207–225, 2007.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yaos protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[LP12] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *J. Cryptology*, 25(4):680–722, 2012.

[Maz01] David Mazières. A toolkit for user-level file systems. In *USENIX Technical Conference*, June 2001.

[MR91] Silvio Micali and Phillip Rogaway. Privacy preserving data mining. *Unpublished manuscript*, 576:392–404, 1991.

[NP99] 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.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SIAM Symposium on Discrete Algorithms (SODA)*, pages 448–457, Washington, D.C., January 2001.

[Pai99] 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.

[Ped91] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Advances in Cryptology—CRYPTO '91*, volume 576 of *LNCS*, pages 129–140. Springer-Verlag, 1992, 11–15 August 1991.

[PP08] Anna Pagh and Rasmus Pagh. Uniform hashing in constant time and optimal space. *SIAM J. Comput.*, 38(1):85–96, 2008.

[PR04] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.

[PT12] Mihai Patrascu and Mikkel Thorup. The power of simple tabulation hashing. *J. ACM*, 59(3):14, 2012.

[Raz90] Alexander A. Razborov. Application of matrix methods to the theory of lower bounds in computational complexity. *Combinatorica*, 10(1):81–93, 1990.

[RS98] Martin Raab and Angelika Steger. Balls into Bins - A Simple and Tight Analysis. *Randomization and Approximation Techniques in Computer Science*, pages 159–170, 1998.

[V03] Berthold Vöcking. How asymmetry helps load balancing. *Journal of the ACM (JACM)*, 50:568–589, July 2003.

[Wie07] Udi Wieder. Balanced allocations with heterogenous bins. In *Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*, page 193. ACM, 2007.

[Woe06] Philipp Woelfel. Asymmetric balanced allocation with simple hash functions. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, SODA '06, pages 424–433, 2006.

[Yao82] Andrew C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science*, pages 160–164, Chicago, Illinois, 3–5 November 1982. IEEE.