Efficient Robust Private Set Intersection

Dana Dachman-Soled[1], Tal Malkin[1], Mariana Raykova[1], and Moti Yung[2]

[1] Columbia University
{dglasner,tal,mariana}@cs.columbia.edu
[2] Columbia University and Google Inc.
moti@cs.columbia.edu

Abstract. Computing Set Intersection privately and efficiently between two mutually mistrusting parties is an important basic procedure in the area of private data mining. Assuring robustness, namely, coping with potentially arbitrarily misbehaving (i.e., malicious) parties, while retaining protocol efficiency (rather than employing costly generic techniques) is an open problem. In this work the first solution to this problem is presented.

Keywords: Set Intersection, Secure Two-party Computation, Cryptographic Protocols, Privacy Preserving Data Mining.

1 Introduction

Constructing an efficient, robust two-party protocol for computing set intersection that is secure and realizable given current encryption methods is an open question first introduced in the work of Freedman, Nissim and Pinkas [9]. Here we solve this problem and present a protocol that allows two mutually distrustful parties holding private inputs to compute the intersection of their inputs without revealing any additional information. We prove the security of our protocol in the standard Ideal/Real Model. The Set Intersection primitive is widely used in the area of privacy preserving data mining ([19]); the prototypical application involve secure sharing of information in areas like personal health and finance.

Although generic robust methods (c.f., [20]) based on Yao's general two-party computations [25] are sufficient for computing any two-party functionality, here we are after efficient methods. Since the size of the naive circuit needed to compute Set Intersection is at least $\Omega(m \cdot n)$ (where n is the input size of the party that receives output and m is the input size of the other party) any generic construction for semi-honest two-party computation will have communication complexity $\Omega(m \cdot n)$, even without robustness. In contrast, our protocol's communication complexity is $O(mk^2 \log^2 n + kn)$ ciphertexts, where k is a security parameter (i.e. logarithm of the size of the field, where we allow sets that are arbitrary but are representable in this field). Additional properties of our solution are worth mentioning: First, the number of exponentiations needed by our protocol increases only by a poly-logarithmic (i.e. a $k^2 \log^2 n$) factor in comparison to the number of exponentiations required by the semi-honest protocol of [9]

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 125–142, 2009.

over domains as above. Secondly, our construction is fully-black box assuming the existence of a homomorphic encryption scheme. Finally, the encryption is only required to possess a few natural properties, which are discussed in the following section (and satisfied by known homomorphic encryption schemes, e.g., based on DDH).

Our Methodology and Techniques. Our starting point is the semi-honest protocol of [9] that computes set intersection via polynomial evaluation. In this protocol, the Server must evaluate an encrypted polynomial of degree n on each of his inputs and send the (encrypted) results back to the Client. In the malicious adversary case, though, to achieve security, the Client must be able to verify that the Server evaluated the polynomial honestly with respect to its input. To ensure this, we use techniques that add redundancy to the representation of the inputs (this is motivated by techniques in Choi et. al [6]). More specifically, we employ a Server that shares its input via a Shamir secret-sharing [23] threshold scheme using a degree k polynomial, where k is the security parameter, and then commits to shares of its input. Note that a Shamir secret-sharing can also be viewed as a Reed-Solomon encoding of the input. What we would like to do next is have the server evaluate the encrypted polynomial on each share of its input and send the resulting shares to the Client. Note that due to the fact that polynomials are closed under composition, the above yields a valid-secret sharing (and a valid Reed-Solomon encoding) of the output value. However, the resulting polynomial is now of degree $n \cdot k$, and so we need at least $n \cdot k + 1$ shares to recover the secret. To improve our efficiency, we apply input-preprocessing technique that allows us to reduce the degree of the output polynomial to $d = k(\lfloor \log n \rfloor + 1) + k$, and thus we need only $O(k(\lfloor \log n \rfloor + 1))$ shares in order to recover the shared value.

Next, we ensure that the Server acted honestly for a large-fraction of the shares by executing a cut-and-choose protocol that forces the Server to open k random shares for each committed input value, thus allowing the Client to verify that the corresponding output share was computed correctly. Due to the information-theoretic security of the secret-sharing scheme, no information about the input is leaked by opening these shares. Additionally, the Client checks that all the output shares he received indeed lie on the same polynomial of degree d. Due to the large distance between codewords in a Reed-Solomon code this ensures that, in fact, *all* the shares were computed exactly correctly. Finally, the Client reconstructs the secret, which is now guaranteed to be consistent with the Server's inputs. Note that in the two-party case, we only need to either complete the computations if the other party acts honestly, or detect cheating. This allows us to use Lagrange interpolation and a consistency check as an error detection code, rather than error-correction (implied by techniques such as Berlkamp-Welch). As is noted in the sequel, this is important to the realization of the encryption schemes, since the interaction of the algebra of secret sharing methods and the algebra of concrete encryption schemes is a subtle issue (not treated in earlier work). We ensure that every algebraic operation used in our protocol is realizable given a concrete encryption scheme.

Related Work. Multiple papers address the problem of secure set intersection and suggest various solutions [1,9,18,12]. (We remark that, in addition, several works deal with variants of set intersection such as the private equality test for input sets of size one [8,21,3,14] or the problem of disjointness that asks whether the intersection of two sets is empty ([17])). However, none of the protocols for set intersection that have been suggested thus far are secure in the scenario of arbitrarily malicious adversaries and arbitrary input domains. Freedman et. al([9]) present a protocol claimed (without details or proofs) to be secure in the presence of a malicious Client in the standard Ideal/Real model, and secure in the presence of a malicious Server only in the Random Oracle model. Hazay and Lindell ([12]), in turn, adopt a different approach based on secure pseudorandom function evaluation that does not use random oracles but they only achieve security against a malicious Client (and semi-honest Server), or security against two *covert* parties, where covert is a new non-standard model that is stronger than semi-honest, but weaker than malicious. Recently (and independently of our work), Jarecki and Liu ([15]) extend the approach of [12] to provide a protocol secure against two malicious parties, when the input sets are chosen from a polynomial-sized domain and based on the Decisional q-Diffie-Hellman Inversion Assumption. We also note the work of Kissner and Song ([18]) which presents multi-party protocols that are secure in the presence of semi-honest adversaries for several set operations including Set Intersection. Additionally, they briefly address achieving security in the presence of malicious adversaries, but their method relies on inefficient generic zero-knowledge proofs. Also independently, Camenisch and Zaverucha ([4]) extend the protocol of [9] in a different direction where they assume the presence of a certifying third party that signs the input sets of the two participants. This provides guarantees that the set intersection functionality is computed correctly with respect to the signature certified input sets in the presence of malicious adversaries.

Organization. In section 2 we present definitions and known building blocks, while in Section 3 we present our protocol steps and protocol. In Section 4 we present some intuition for the proof of security, and discuss our complexity.

2 Definitions and Building Block Protocols

We use a standard simulation-based definition of security from [5], and follow the definitions of zero knowledge proofs of knowledge and commitment schemes from [10]. We denote Com_B a perfectly binding commitment scheme and Com_H a perfectly hiding commitment scheme.

We follow the standard definitions of semantically-secure encryption schemes and homomorphic encryption schemes given in [16]. We assume the plaintexts of the semantically-secure encryption scheme ENC are elements of a finite group P with group operation '+' and that the ciphertexts are elements of a finite group C with group operation '$\cdot$'. Since ENC is a homomorphism from P to C, the homomorphic property of an encryption scheme ENC can be stated as follows:

Property 1 (Homomorphic Encryption).

$$\mathrm{ENC}(X, r_1)\cdot\mathrm{ENC}(Y, r_2) = \mathrm{ENC}(X+Y, r) \qquad (\mathrm{ENC}(X, r_3))^{\lambda} = \mathrm{ENC}(\lambda\cdot X, r').$$

We will also require that r can be computed in polynomial-time given r_1, r_2, X, Y, r' can be computed in polynomial time given r_3, X, λ, and that r, r' are uniformly distributed when r_1, r_2, r_3 are (so the encryptions after applying a homomorphic operation are distributed as random encryption). It turns out that known homomorphic encryption schemes typically satisfy the above requirements, and actually possess the following, stronger, property:

Property 2.

$$\mathrm{ENC}(X, r_1)\cdot\mathrm{ENC}(Y, r_2) = \mathrm{ENC}(X+Y, r_1+r_2);\ (\mathrm{ENC}(X, r_3))^{\lambda} = \mathrm{ENC}(\lambda\cdot X, \lambda\cdot r_3).$$

This property is satisfied by most known homomorphic encryption schemes, such as Paillier [22], ElGamal [7], and Goldwasser-Micali [11] encryption schemes.

We also present the Additive El-Gamal Encryption scheme, which we will use to concretely instantiate our protocol:

Definition 1 (Additive El Gamal Encryption Scheme: AEG_{enc}).

- GEN*: on input* 1^n *generate* (G, q, g) *where* q *is prime,* G *is a cyclic group of order* q *and* g *is a generator. Then choose a random* $x \leftarrow Z_q$ *and compute* $h = g^x$*. The public key is* $\langle G, q, g, h\rangle$ *and the private key is* $\langle G, q, g, x\rangle$*.*
- ENC*: on input a public key* $pk = \langle G, q, g, h\rangle$ *and a message* $m \in Z_q$*, choose a random* $y \leftarrow Z_q$ *and output the ciphertext*

$$\langle g^y, h^y \cdot g^m\rangle$$

- DEC*: on input a private key* $sk = \langle G, q, g, x\rangle$ *and a ciphertext* $\langle c_1, c_2\rangle$*, output*

$$g^m = c_2/c_1^x$$

Unlike regular Additive El Gamal decryption, here we can recover g^m and not necessarily know m. However, this will be sufficient for our application and we are able to handle plaintexts that come from a large domain.

Now we proceed to define several auxiliary protocols that will be used in our main protocols.

2.1 Homomorphic Encryption Proof of Knowledge

This protocol will be used by both the Server and Client when a party P_0 sends to a party P_1 a public key pk and several values encrypted under ENC_{pk}. In the malicious case, we require P_0 to prove that he knows the corresponding plain text values and randomness and additionally that the encrypted plaintexts are "valid" (i.e. belong to a particular language). This protocol is similar to the polynomial time provers in [13].

If P_1 is behaving honestly, its input should be a member of the language L whose membership can be determined in polynomial time and is closed under addition and subtraction. The NP-language L', is defined as follows:

$$L' = \{\overline{C} = (pk1, c_1, \ldots, c_\alpha) \mid \ c_i = \mathrm{ENC}_{pk1}(x_i; r_i), \text{ for some } x_i, r_i, 1 \leq i \leq \alpha, \text{ and } (x_1, \ldots, x_\alpha) \in L\}$$

Homomorphic Encryption Proof of Knowledge and Plaintext Verification (HEPKPV) Protocol: Π_{POK}

Input: $P_0 \leftarrow \overline{C} = (pk1, c_1, \ldots, c_\alpha)$, $(x_1, \ldots x_\alpha) \in L$, $(r_1, \ldots, r_\alpha)$ where $c_i = \mathrm{ENC}_{pk1}(x_i; r_i)$ for $1 \leq i \leq \alpha$;
$P_1 \leftarrow \overline{C} = (pk1, c_1, \ldots, c_\alpha)$
Output: P_1 outputs *Accept* if $\overline{C} \in L'$, and *Reject* otherwise.

1. P_0 chooses k random vectors $(e_{11}, \ldots e_{1\alpha}), \ldots, (e_{k1}, \ldots, e_{k\alpha})$ such that for $1 \leq i \leq k$, $(e_{i1}, \ldots, e_{i\alpha}) \in L$. and another k vectors $(r_{11}, \ldots r_{1\alpha}), \ldots, (r_{k1}, \ldots, r_{k\alpha})$ of random numbers.
2. P_0 computes the encryptions $(c_{i1}, \cdots, c_{i\alpha}) = (\mathrm{ENC}(e_{i1}, r_{i1}), \ldots, \mathrm{ENC}(e_{i\alpha}, r_{i\alpha}))$ for $1 \leq i \leq k$ and sends them to the Server.
3. P_1 chooses a sequence of k bits $b'_1 \ldots b'_k$ and sends to P_1 a commitment to those bits: $Com_H(b'_1 \ldots b'_k)$, along with the public parameters for the commitment scheme.
4. P_0 chooses a sequence of k bits $b''_1 \ldots b''_k$ and sends to P_0 a commitment to those bits: $Com_B(b''_1 \ldots b''_k)$, along with the public parameters for the commitment scheme.
5. P_0 and P_1 decommit the value $b''_1 \ldots b''_k$ and $b'_1 \ldots b'_k$, respectively.
6. P_0, P_1 verify that the bits received correspond to the commitments that were sent. If the check fails, they abort the protocol. Otherwise both P_0 and P_1 compute $b_1 \ldots b_k = b'_1 \ldots b'_k$ **XOR** $b''_1 \ldots b''_k$.
7. For each $1 \leq i \leq k$:
   (a) if $b_i = 0$, P_0 sends to P_1 $\overline{M} = (e_{i1}, \cdots e_{i\alpha})$ and $\overline{R} = (r_{i1}, \cdots r_{i\alpha})$;
   (b) if $b_i = 1$, P_0 sends to P_1 $\overline{M} = (x_1 + e_{i1}, \cdots, x_n + e_{i\alpha})$ and $\overline{R} = (r_1 + r_{i1}, \cdots, r_\alpha + r_{i\alpha})$.
8. For each $1 \leq i \leq k$:
   (a) if $b_i = 0$, P_1 verifies that $(c_{i1}, \cdots, c_{i\alpha}) = (\mathrm{ENC}(e_{i1}, r_{i1}), \cdots \mathrm{ENC}(x_{i\alpha}, r_{i\alpha}))$;
   (b) if $b_i = 1$, P_1 verifies that $(c_1 c_{i1}, \cdots c_\alpha c_{i\alpha}) = (\mathrm{ENC}(x_1 + e_{i1}, r_1 + r_{i1}), \cdots, \mathrm{ENC}(x_\alpha + e_{i\alpha}, r_\alpha + r_{i\alpha}))$.
9. P_1 verifies that $(\overline{M}) \in L$.
10. If any of the verifications steps of P_1 fail, abort the protocol. Otherwise, accept.

Lemma 1. *Assume that $Hom_{enc} = (Gen, Enc, Dec)$ is a CPA-secure homomorphic encryption scheme, Com_H is a perfectly hiding commitment scheme, and Com_B is a perfectly binding commitment scheme. Then protocol Π_{POK} is a Zero Knowledge Proof of Knowledge for L'.*

See full version for proof.

Now we define several languages that we will use in the main protocols in the context of the above HEPKPV protocol:

– Language consisting of points that lie on some polynomial of degree ℓ

$$L_{poly}(t,u,\ell) = \{m_{i,j} \mid 1 \le i \le t,\ 1 \le j \le u; \\ \text{for each } j \text{ the points } ((1,m_{1,j}),\ldots,(t,m_{t,j})) \\ \text{lie on a polynomial of degree } \ell\}.$$

– Language consisting of points that lie on some polynomial of degree ℓ that has zero free coefficient

$$L_{poly,0}(t,u,\ell) = \{m_{i,j} \mid 1 \le i \le t,\ 1 \le j \le u; \\ \text{for each } j \text{ the points } ((1,m_{1,j}),\ldots,(t,m_{t,j})) \\ \text{lie on a polynomial } P_j \text{ of degree } \ell \text{ and } P_j(0) = 0\}.$$

– Language consisting of points that lie on some polynomial of degree ℓ that has free coefficient equal to m'_j.

$$L_{eq}(t,u,\ell) = \{m_{i,j}, m'_j \mid 1 \le i \le t; 1 \le j \le u, \\ \text{for each } j \text{ the points } ((1,m_{1,j}),\ldots,(t,m_{t,j})) \\ \text{lie on a polynomial } P_j \text{ of degree } \ell,\ \text{where } P_j(0) = m'_j\}$$

– The following language consists of several tuple of pairs of the form $(m_{i,j}, m'_{i,j})$. For each i the points $(1,m_{i,1}),\ldots,(t,m_{i,t})$ lie on a polynomial P_i of degree ℓ and the points $(1,m'_{i,1}),\ldots,(t,m'_{i,t})$ lie on a polynomial R_i of degree 2ℓ and additionally, for each i, $P_{i+1}(0) = R_i(0)$.

$$L_{sq}(t,u,\ell) = \{(m_{i,j}, m'_{i,j}) \mid 1 \le i \le u,\ 1 \le j \le t;\ \text{for all } i, \\ \text{the points } ((1,m_{i,1}),\ldots,(t,m_{i,t})) \text{ lie on } P_i \text{ of degree } \ell; \\ \text{the points } ((1,m'_{i,1}),\ldots,(t,m'_{i,t})) \text{ lie on } R_i \text{ of degree } 2\ell; \\ \text{and for } 1 \le i \le u-1, P_{i+1}(0) = R_i(0)\}$$

Membership in all of the above languages can be determined in polynomial time. Also these languages are closed under addition and can be used in the context of the HEPKPV protocol.

2.2 Coin Tossing

The following protocol is run by the Server S and Client C in order to select a random number within a given range $[0, s-1]$ known to both of them. At the end of the protocol both parties obtain the same random number. The private input of the two parties is $(\perp, \perp)$ and the output to each party is $(rand, rand)$, where $rand$ is a uniformly random number chosen from $[0, s-1]$.

1. S chooses a random value $R' \in [0, s-1]$ and sends a commitment $C_1 = Com_H(R')$ to P_1.
2. C chooses a random value $R'' \in [0, s-1]$ and sends a commitment $C_2 = Com_B(R'')$ to P_0.
3. C opens the commitments C_2.
4. S opens the commitment C_1.
5. The two parties output $R = R' + R'' \mod s$.

We note that both statistically hiding and statistically binding commitments can be constructed using a homomorphic encryption scheme.

Lemma 2. *Assume that Com_H is a statistically hiding commitment scheme and Com_B is a perfectly binding commitment scheme. Then protocol Π_{Coin} is simulatable for Malicious C and Honest S.*

See full version for proof.

3 Set Intersection Protocol

We now describe the setting for the Set Intersection protocol. There are two participants in the protocol: Client, C and Server, S. The Client has an input set $X, |X| = n$ of size at most $n \leq max_c$ and the Server has an input set $Y, |Y| = m$ of size at most $m \leq max_s$. Both parties know a homomorphic encryption scheme $Hom_{enc} = (\text{GEN}, \text{ENC}, \text{DEC})$. Further the Client and the Server choose a security parameter k. The goal of the protocol is that the Client learns the intersection of their sets: $X \bigcap Y$ and nothing else while the Server learns nothing. Now if the pair (K_c, K_s) represents knowledge of the Client (K_c) and the Server (K_s), the input and output of the set intersection protocol can be summarized as follows:

$$\begin{matrix}(\{X, max_c, max_s, Hom_{enc}, k\},\\ \{Y, max_s, max_c, Hom_{enc}, k\})\end{matrix} \rightarrow \begin{cases} (X \bigcap Y, \perp), & \text{if } |X| \leq max_c,\ |Y| \leq max_s \\ (\perp, \perp) & \text{otherwise} \end{cases}$$

Our idea starts with the approach of [9]: the Client constructs a polynomial P of degree n over a finite field such that $P(x) = 0$ if and only if $x \in X$. The Client encrypts the coefficients of P using a homomorphic encryption scheme and sends them to the Server. Due to the homomorphic properties of the encryption scheme, the Server is now able to evaluate the polynomial at each of its inputs. Thus, for $1 \leq \ell \leq m$, the Server sends the encryption of the following output back to the Client: $r_j \cdot P(y_j) + y_j$, where r_j is chosen randomly. Thus, we have that if $y_j \in X \cap Y$ then the Client receives y_j. If $y_j \notin X \cap Y$ then the Client receives a random value.

Before presenting our main protocol, we define three protocols that are used as building blocks for the main protocol. They implement the two main ideas that we use to achieve security against malicious parties.

3.1 Input Sharing via Enhanced Shamir Scheme

In the set intersection protocol we "share" function evaluation by secret sharing the arguments of the function and evaluating the function on corresponding shares in order to obtain shares of the final value of the function. We use Shamir's secret sharing ([23]) but for the purposes of efficiency we apply the following further transformation on the inputs.

Let f be a polynomial of degree n over a single variable: $f = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_0$. For a given z and r, we would like to obtain shares of the result $g(z,r) = r \cdot f(z) + z$ by secret-sharing the inputs z and r. For this purpose we choose random polynomials P_z, P_r of degree k for such that $P_z(0) = z$ and $P_r(0) = r$ and evaluate g on m shares of the input to obtain $g(P_z(1), P_r(1)), \ldots g(P_z(m), P_r(m))$. We now define a new single variable polynomial $g'(i) = g(P_z(i), P_r(i))$. Note that the degree of g' is $n \cdot k + k$ and that $g'(0) = g(P_z(0), P_r(0)) = g(z,r)$ and thus given $g(P_z(1), P_r(1)), \ldots, g(P_z(m), P_r(m)) = g'(1), \ldots, g'(m)$ we can reconstruct $g'(0) = g(z,r)$ when $m \geq n \cdot k + k$. This means that the number of shares needed is at least $n \cdot k + k$.

We extend the above idea further in order to decrease the degree of the final sharing polynomial of the result. For a given z and r, we obtain shares of the result $g(z,r)$ in the following way. For $0 \leq \ell \leq \lfloor \log n \rfloor$ we secret share the value z^{2^ℓ} using a random polynomial $P_{z^{2^\ell}}$, of degree k, such that $P_{z^{2^\ell}}(0) = z^{2^\ell}$. Let $s[i]$ indicate the i^th bit of a number s. We now define a new polynomial $g''(i) = P_z(i) + P_r(i) \cdot \Sigma_{s=1}^{n} a_s \cdot \Pi_{\ell=1}^{\lfloor \log n \rfloor + 1}(P_{z^{2^\ell}}(i))^{s[\ell]}$. Note that $g''(0) = g(z,r)$ and that g'' has degree $(\lfloor \log n \rfloor + 1)k + k$. We have thus drastically reduced the number of shares necessary to recover $g''(0) = g(z,r)$.

The above idea for function transformation guarantees correct evaluation of shares of the functional value if the party is following the protocol honestly. In the malicious case a party needs to prove that the sharing functions that it is using for the new arguments $z, z^2, \ldots, z^{2^\ell}$ have been constructed correctly. The following protocol allows a party to generate shares of an input z using the preprocessing idea and then prove that these shares were computed correctly without revealing any information about z.

Efficient Preprocessing of Input:

1. For each $y_j \in Y$, $1 \leq j \leq m$ S chooses a random polynomial P_{y_j} of degree k such that $P_{y_j}(0) = y_j$, and computes shares of the form $P_{y_j}(i)$ for $1 \leq i \leq 10k(\lfloor \log n \rfloor + 1)$ and the corresponding encryptions $\mathrm{ENC}_{pk}(P_{y_j}(i))$.
2. For each y_j, $1 \leq \ell \leq m$ S and C run the HEPKPV protocol with $L_{poly}(m,n,k) = \{m_{i,j} = P_{y_j}(i)\}$ in order for S to prove the correctness of his sharing.
3. For each y_j, for $\ell = 0$ to $\lfloor \log n \rfloor$, for $i = 0$ to $10k(\lfloor \log n \rfloor + 1)$, S computes the following:
   - Local Computation on Shares: a polynomial $P^2_{y_j^{2^\ell}}$ of degree $2k$ such that $P^2_{y_j^{2^\ell}}(i) = (P_{y_j^{2^\ell}}(i))^2$
   - Degree Reduction Step: a random polynomial $P_{y_j^{2^{\ell+1}}}$ of degree k such that $P_{y_j^{2^{\ell+1}}}(0) = P^2_{y_j^{2^\ell}}(0)$.

4. For each y_j, for $\ell = 0$ to $\lfloor \log n \rfloor$, for $i = 1$ to $10k(\lfloor \log n \rfloor + 1)$, S computes the following commitments:
   - New input shares: $\mathrm{ENC}_{pk}(P_{y_j^{2^{\ell+1}}}(i))$ and
   - Intermediate shares: $\mathrm{ENC}_{pk}(P^2_{y_j^{2^{\ell}}}(i))$

   and sends those commitments to C

We now describe how C verifies S's computation of its new shares.

Let J be the ordered set of all elements of $\{0,1\}^{10k(\lfloor \log n \rfloor + 1)}$ that contain exactly k ones. Note that given R, an index of a string in the set J, we can efficiently reconstruct the Rth string, j_R. Let $J_R = \{i | j_R[i] = 1\}$, where $j_R[i]$ denotes the ith position of the string j_R.

Preprocessing Verification:

Common Inputs: The commitments: $[C_{j,\ell,i}]_{1 \le j \le m, 0 \le \ell \le \lfloor \log n \rfloor + 1, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, $[C^2_{j,\ell,i}]_{1 \le j \le m, 0 \le \ell \le \lfloor \log n \rfloor, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, and a number $R \in [|J|]$ chosen using the Coin-Tossing protocol after S committed to its inputs.

Private Inputs of S: Decommitments to the above values.

1. For all $i \in J_R, 1 \le j \le m, 0 \le \ell \le \lfloor \log n \rfloor$ S opens the commitment $C^2_{j,\ell+1,i}$ to $C'^2_{j,\ell+1,i}$, $C_{j,\ell,i}$ to $C'_{j,\ell,i}$.
2. For all $i \in J_R, 1 \le j \le m, 0 \le \ell \le \lfloor \log n \rfloor$ C checks that $C'^2_{j,\ell+1,i} = (C'_{j,\ell,i})^2$.
3. S and C run the HEPKPV protocol with S's private inputs, the common commitment inputs and language $L_{sq}(\lfloor \log n \rfloor, 10k(\lfloor \log n \rfloor + 1), m, k) = \{m_{\ell,i,j} = P_{y_j^{2^\ell}}(i), m'_{\ell,i,j} = P^2_{y_j^{2^\ell}}(i)\}$

In the first step of the above protocol S first proves that for all $y_j \in Y, 0 \le \ell \le \lfloor \log n \rfloor$ he has computed correctly $P^2_{y_j^{2^\ell}}(i)$ for at least a .9-fraction of the shares correctly. In the second step of the protocol S proves that for all $y_j \in Y, 0 \le \ell \le \lfloor \log n \rfloor$ he has computed correctly the new sharing polynomials for the values $y_j^{2^{\ell+1}}$ and that both $P_{y_j^{2^\ell}}$ and $P^2_{y_j^{2^\ell}}$ are polynomials. Since any 2 polynomials of degree at most $2k$ must disagree on at least a .8-fraction of the shares, the combination of the above two statements implies that with probability at least $1 - m \cdot (\lfloor n \rfloor + 2)^2 \cdot (1/2^k + .9^k)$, all the sharings were, in fact, computed exactly correctly. For detailed analysis of the above intuition, see the proof sketch in section 4 and the full version.

3.2 Cut-and-Choose on Computations on Input Shares

Common Input: The encryptions: $b_{n+1}, \ldots, b_0$, The commitments: $[M_{i,j,\ell}]_{1 \le j \le m, 0 \le \ell \le \lfloor \log n \rfloor, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, $[R_{i,j}]_{1 \le j \le m, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, $[0_{i,j}]_{1 \le j \le m, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, $[C_{i,j}]_{1 \le \ell \le m, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, and a number $R \in [|J|]$ chosen using the Coin-Tossing protocol after S committed to the above.

Private input of S:Decommitments to $[M_{i,j,\ell}]_{1 \le j \le max_S, 0 \le \ell \le \lfloor \log n \rfloor, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, $[R_{i,j}]_{1 \le j \le m, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, $[Z_{i,j}]_{1 \le j \le m, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$, and the values $[r_{i,j}]_{1 \le j \le m, 1 \le i \le 10k(\lfloor \log n \rfloor + 1)}$.

We use the cut-and-choose technique to prove the correctness of evaluation of a specific function on committed inputs $[M_{i,j,\ell}], [R_{i,j}], [Z_{i,j}]$ that results in the commited outputs $[C_{i,j}]$. The function we use is:

$$C_{i,j} = \mathrm{ENC}(0; r_{i,j}) \cdot \mathrm{ENC}_{pk1}(Z'_{i,j}; 0) \cdot \mathrm{ENC}_{pk1}(M'_{i,j,0}; 0) \cdot \left(\Pi_{s=0}^{n} b_s^{\Pi_{\ell=0}^{\lfloor \log n \rfloor} (M'_{i,j,\ell})^{s[\ell]}} \right)^{R'_{i,j}}$$

where $s[\ell]$ denotes the ℓ^{th} bit of s.

We will explain why this is the function we need in the next section.

The steps of the protocols are the following:

1. For each $i \in J_R, 1 \leq j \leq m, 0 \leq \ell \leq \lfloor \log n \rfloor$ S opens the commitments $M_{i,j,\ell}, R_{i,j}, Z_{i,j}$ to $M'_{i,j,\ell}, R'_{i,j}, Z'_{i,j}$ and produces the random value $r_{i,j}$.
2. For $i \in J_R, 1 \leq j \leq m$, C verifies the following: $C_{i,j} =$ $\mathrm{ENC}_{pk1}(0; r_{i,j}) \cdot \mathrm{ENC}_{pk1}(Z'_{i,j}; 0) \cdot \mathrm{ENC}_{pk1}(M'_{i,j,0}; 0) \cdot \left(\Pi_{s=0}^{n} b_s^{\Pi_{\ell=0}^{\lfloor \log n \rfloor} (M'_{i,j,\ell})^{s[\ell]}} \right)^{R'_{i,j}}$
3. If any of these verifications fail, C outputs Reject. Otherwise, C outputs Accept.

3.3 Reconstruction and Set Membership Test Protocol

We describe here how the Client reconstructs and checks whether Server's input y_j is in his input set X and consequently in the intersection set using the output shares $[C_{i,j}]_{1 \leq i \leq 10k(\lfloor \log n \rfloor + 1)}$.

1. The Client decrypts the output shares $[C_{i,j}]_{1 \leq i \leq 10k(\lfloor \log n \rfloor + 1)}$ to obtain plaintexts $[C'_{i,j}]_{1 \leq i \leq 10k(\lfloor \log n \rfloor + 1)}$.
2. The Client uses the points $(1, C'_{1,j}), \ldots, (k + k(\lfloor \log n \rfloor + 1), C'_{k+k(\lfloor \log n \rfloor + 1),j}$ and the Lagrange interpolation polynomial to check that for $1 \leq i \leq 10k$ $(\lfloor \log n \rfloor + 1)$
$$C'_{i,j} = L_j(i) = \Sigma_{v=1}^{1+k+k(\lfloor \log n \rfloor + 1)} C'_{j,v} \ell_v(i)$$
where $\ell_v(x) = \Pi_{w=1, w \neq v}^{1+k+k(\lfloor \log n \rfloor + 1)} \frac{x-w}{v-w}$. Otherwise, abort.
3. The Client reconstructs the shared value:
$$C'_{0,j} = L_j(0) = \Sigma_{v=1}^{1+k+k(\lfloor \log n \rfloor + 1)} C'_{j,v} \ell_v(0)$$
and checks whether $C'_{0,j} = x$ for some $x \in X$. If it does, output x.

In the following, we give a concrete implementation of the Reconstruction Protocol using additive El Gamal encryption. We note the following subtlety due to the interaction of the algebraic properties needed to realize the protocol and the properties of the El Gamal encryption scheme. Due to the algebraic properties of the encryption scheme, we are able to compute the Lagrange interpolation polynomial and thus detect errors; however, we cannot run the Berlekamp-Welch algorithm to correct the errors in the codeword. This is due to the fact that the Client can only obtain pairs of the form $(i, g^{m_{i,j}})$ and we are interested in reconstructing a polynomial such that $P(i) = m_{i,j}$ (for a large fraction of i's). The Berlekamp-Welch algorithm requires us to solve a system of linear equations, which we do not know how to do efficiently when we know only $g^{m_{i,j}}$ and not $m_{i,j}$ itself (This issue was ignored in earlier work).

Reconstruction and Set Membership Test via Additive El Gamal Encryption

1. The Client decrypts the output shares $[C_{i,j}]_{1\leq i\leq 10k(\lfloor \log n\rfloor+1)}$ to obtain plaintexts $[g^{m_{i,j}}]_{1\leq i\leq 10k(\lfloor \log n\rfloor+1)}$.
2. The Client uses the points $(1, g^{m_{1,j}}), \ldots, (1 + k + k(\lfloor \log n\rfloor + 1), g^{m_{k+k(\lfloor \log n\rfloor+1),j}}$ and the Lagrange interpolation polynomial to check that for $1 \leq i \leq 10k(\lfloor \log n\rfloor + 1)$

$$g^{m_{i,j}} = L_j(i) = \Pi_{v=1}^{1+k+k(\lfloor \log n\rfloor+1)}(g^{m_{j,v}})^{\ell_v(i)}$$

where $\ell_v(x) = \Pi_{w=1,w\neq v}^{1+k+k(\lfloor \log n\rfloor+1)}\frac{x-w}{v-w}$. Otherwise, abort.
3. The Client reconstructs the shared value:

$$g^{m_{0,j}} = L(0) = \Pi_{v=1}^{1+k+k(\lfloor \log n\rfloor+1)}(g^{m_{j,v}})^{\ell_v(0)}$$

and checks whether $g^{m_{0,j}} = g^x$ for some $x \in X$. If it does, output x.

3.4 The Full Protocol

We start with an overview description of the main steps in the protocol, followed by the detailed specification of our set intersection protocol.

1. The Client runs $\mathrm{GEN}(1^k)$ to obtain a secret key sk and a public key pk for Hom_{enc} and sends pk to the Server.
2. The Client computes a polynomial $P(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$ of degree the size of his input n over a finite field such that $P(x) = 0$ if and only if $x \in X$.
3. The Client encrypts the coefficients of P, $b_i = \mathrm{ENC}(a_i)$ and sends them to the Server.
4. For each $y_j \in Y$ S chooses a random value r_j and constructs the function

$$\begin{aligned} F(y_j) &= \mathrm{ENC}_{pk_1}(r_j \cdot (y_j) + y_j + 0) = \\ &= \mathrm{ENC}_{pk_1}(0) \cdot \mathrm{ENC}_{pk_1}(y_j) \cdot (\prod_{s=0}^{n}(\mathrm{ENC}_{pk_1}(a_s))^{y_j^s})^{r_j} = \\ &= \mathrm{ENC}_{pk_1}(0) \cdot \mathrm{ENC}_{pk_1}(y_j) \cdot (\prod_{s=0}^{n}(b_s)^{y_j^s})^{r_j} \end{aligned}$$

The above function has the property that it maps the values in the intersection set of the two parties to themselves and values not in the intersection to random numbers.
5. The Server replaces each of its inputs y_j with new variables $c_\ell = y_j^{2^\ell}$ for $0 \leq \ell \leq \lfloor \log n - 1\rfloor$ and transforms the above function to

$$F(y_j) = \mathrm{ENC}_{pk_1}(0) \cdot \mathrm{ENC}_{pk_1}(y_j) \cdot \left(\prod_{s=0}^{n}(b_s)^{\Pi_{\ell=0}^{\lfloor \log n\rfloor}(y_j^{2^\ell})^{s[\ell]}}\right)^{r_j}$$

Note: The exponent of each b_s is y_j^s , however, in the form where s is written in binary and $s[1], ..., s[\lfloor \log n \rfloor + 1]$ are its binary digits and the power of y_j for each digit is substituted with the corresponding new variable from the efficient preprocessing of Servers inputs.

6. The Server shares each of his input $y \in Y$ with polynomial P_{y_j} and each of the random values r_j with a polynomial P_{r_j}.
7. Additionally the Server computes m random polynomials $P_{0,j}$ that have constant coefficient zero. These are used to "rerandomize" the output shares so that they give no information about the input.
8. Using all of the above shares and a random $r_{i,j}$ (to "rerandomize" the encryption) the Server computes shares of the values $F(y_j)$:

$$Out_{i,j} = (F(y_j))(i) = \mathrm{ENC}_{pk_1}(0; r_{i,j}) \cdot \mathrm{ENC}_{pk_1}(P_{0,j}(i); 0) \cdot \mathrm{ENC}_{pk_1}(P_{y_j}(i); 0) \cdot \left(\prod_{s=0}^{n} (b_s)^{\Pi_{\ell=0}^{\lfloor \log n \rfloor} (P_{y_j^{2^\ell}}(i))^{s[\ell]}} \right)^{P_{r_j}(i)}$$

and sends them to the Client.

9. The Client decrypts the values that he received from the Server, verifies that they are valid, and uses them to reconstruct the shared values. He concludes that the obtained values that are in his input set are the values in the intersection set.

The above protocol ensures privacy in the presence of semi-honest parties, but is not secure in the presence of malicious parties. The following are several basic additional conditions that must hold in order that the above protocol will be secure in the presence of malicious parties.

The first condition is that the coefficients that the Client sends to the Server are values encrypted with Hom_{enc} under the key pk. We guarantee this by making the Client prove that he knows the encrypted values with HEPKPV.

Additionally, the Client must be sure that the Server correctly shared his inputs using the secret-sharing scheme. This will be guaranteed by HEPKPV showing that all the shares of one input lie on some polynomial of degree k.

The correctness of the protocol also depends on the Server evaluating F honestly. We apply the cut-and-choose protocol on the shares of the Server's inputs to ensure that the computation on a large fraction of final output shares was done correctly.

The last change that we apply improves the efficiency of the protocol. Since the number of shares needed to reconstruct $F_j(0)$ will depend on the degree of F, we reduce its degree by introducing new variables of the form $a_i = y^{2^i}$ for $1 \leq i \leq \lfloor \log n - 1 \rfloor$ for $y \in Y$. Here we need to prove that the computation of the new variables and their shares was done correctly with the Preprocessing Verification Protocol.

We present the full set intersection protocol below.

Set Intersection Protocol Π

Input: $C \leftarrow \{X, max_c, max_s, Hom_{enc}, k\}, S \leftarrow \{Y, max_s, max_c, Hom_{enc}, k\}$
Output: $C \rightarrow X \cap Y, S \rightarrow \perp$
Protocol:

1. The Client runs $\text{GEN}(1^k)$ to obtain a secret key $sk1$ and a public key $pk1$ for Hom_{enc} and sends $pk1$ to the Server.
2. C computes a polynomial $P(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ of degree $n = |X|$ such that $P(x_i) = 0$ if and only if $x_i \in X$. Let $a_n = 1$.
3. C computes $b_i = \text{ENC}_{pk1}(a_i)$ for all $0 \leq i \leq n-1$ and sends to S $\{b_{n-1}, \cdots, b_0\}$,
4. C and S run the HEPKPV Protocol presented in Section 2 as P_0 and P_1 respectively with common input: $\overline{B} = \{b_{n-1}, \cdots, b_0\}$ and $L = \{0,1\}^q$, in order that the C proves that it knows the decryptions of $\{b_{n-1}, \cdots, b_0\}$.
5. The Server runs $\text{GEN}(1^k)$ to obtain a secret key $sk2$ and a public key $pk2$ for Hom_{enc} and sends $pk2$ to the Client.
6. For each $y_j \in Y$ S runs the Efficient Preprocessing protocol to obtain the new variables $c_\ell = y_j^{2^\ell}$ for $0 \leq \ell \leq \lfloor \log n - 1 \rfloor$ and the corresponding sharing polynomials $P_{y_j^{2^\ell}}$ such that $P_{y_j^{2^\ell}}(0) = y_j^{2^\ell}$. During the protocol S commits to $P_{y_j^{2^\ell}}(i)$ for $1 \leq j \leq |Y|, 1 \leq \ell \leq \lfloor \log n \rfloor + 1, 1 \leq i \leq 10k(\lfloor \log n \rfloor + 1)$.
7. For each $y_j \in Y$ S chooses a random value r_j and selects a random polynomial P_{r_j} of degree k with constant coefficient equal to r_j, shares r_j into into $10k(\lfloor \log n \rfloor + 1)$ shares, and sends the following share commitments to C: $(\text{ENC}_{pk2}(P_{r_j}(1)), \ldots, \text{ENC}_{pk2}(P_{r_j}(10k(\lfloor \log n \rfloor + 1)))$
8. For each $y_j \in Y$ S chooses a random polynomial $P_{0,j}$ of degree $k + k(\lfloor \log n \rfloor + 1)$ with constant coefficient equal to 0, computes $10k(\lfloor \log n \rfloor + 1)$ shares, and sends the following share commitments to C: $(\text{ENC}_{pk2}(P_{0_j}(1)), \ldots, \text{ENC}_{pk2}(P_{0_j}(10k(\lfloor \log n \rfloor + 1)))$
9. For each $y_j \in Y$, for $1 \leq i \leq 10k(\lfloor \log n \rfloor + 1)$ using the sharing polynomials obtained in Steps 6, 7, 8, and a random value $r_{i,j}$ S computes:

$$Out_{i,j} = \text{ENC}_{pk_1}(0; r_{i,j}) \cdot \text{ENC}_{pk_1}(P_{0,j}(i); 0) \cdot \text{ENC}_{pk_1}(P_{y_j}(i); 0) \cdot \left(\prod_{s=0}^{n} (b_s)^{\Pi_{\ell=0}^{\lfloor \log n \rfloor} (P_{y_j^{2^\ell}}(i))^{s[\ell]}} \right)^{P_{r_j}(i)}$$

   where $s[\ell]$ denotes the ℓ^{th} bit of s and sends the obtained values to C.
10. C and S run the coin tossing protocol to choose a random number $R \in [1, |J|]$.
11. S and C run the Preprocessing Verification protocol with the share commitments that S computed in Step 6 in order for S to prove to C that it correctly computes the new variables and their shares.
12. S and C run the HEPKPV protocol as P_0 and P_1 respectively so that S proves to C knowledge and validity of the commitments $L_{poly}(10k(\lfloor \log n \rfloor + 1), |Y|, k) = \{m_{i,j} = P_{r_j}(i)\}$, $L_{poly,0}(10k(\lfloor \log n \rfloor + 1), |Y|, k + k(\lfloor \log n \rfloor + 1)) = \{m_{i,j} = P_{0,j}(i)\}$.
13. C and S run the cut-and-choose protocol to prove that S correctly computed $[Out_{i,j}]$.
14. C runs the Reconstruction Protocol to obtain the final output.

4 Analysis

Our main theorem is the following:

Theorem 1. *If the Decisional Diffie-Hellman problem is hard in G with generator q and protocol Π is instantiated with the additive El-Gamal encryption scheme such that $Hom_{enc} = AEG_{enc}$, then Π securely computes the Set Intersection functionality in the presence of malicious adversaries.*

We note that Π is also secure when instantiated with any homomorphic encryption scheme satisfying property 2, and allowing to solve the Lagrange interpolation, as discussed in Sections 2 and 3.3. In particular, we can securely instantiate the protocol with a properly modified version of Paillier encryption (but the details are left out of this abstract). The complete proof of Theorem 1 is in the full version of our paper. Here we give some intuition and a proof sketch.

4.1 Client-Side Simulator

We consider the case in which the Client is corrupted and the Server is honest.

Let A_C be a non-uniform probabilistic polynomial-time real adversary that controls the Client. We construct a non-uniform probabilistic expected polynomial-time ideal model adversary simulator S_C.

The idea behind how S_C works is that it first extracts the Malicious Client's inputs using the extractor for the HEPKPV protocol. S_C then plays the role of the Honest Server using dummy inputs that are all set to 0. When proving knowledge and validity of the Server's input, S_S uses the simulator for the HEPKPV protocol. Next, the Simulator chooses a random subset I' of size k such that $I' \subset [10k(\lfloor \log n \rfloor + 1)]$. When committing to the secret-sharing of its input, it places random values in the positions indexed by I'. S_C computes correctly all calculations that will be verified in the cut-and-choose step for elements in the subset I'. Then, S_C simulates the Coin-Tossing protocol to guarantee that the outcome of the Coin-Tossing protocol is $I = I'$. To ensure that the final output sent to the Client is correct, the Simulator utilizes the the Trusted Party to find out the elements in $X \cap Y$ and includes them in the Server's final output. Intuitively, because the Simulator is able to choose the set I ahead of time, the Simulator can run the protocol using the challenge ciphertext from a CPA-IND experiment as the inputs of the Server in indeces $i \notin I$, thereby reducing indistinguishability of the views to the semantic security of the encryption scheme AEG_{enc}. Therefore, we have that the Malicious Client cannot distinguish its view in the Ideal Model when interacting with a Simulator that chooses all 0 values as the Server's input for indeces $i \notin I$ and its view in the Real model when the Honest Server uses its actual input. This is due to the information-theoretic secrecy of the secret-sharing scheme and the semantic security of the encryption scheme.

We now describe in detail the Simulator for the case of the Malicious Client and Honest Server

1. S_C extracts the Client's inputs using the extractor for the HEPKPV protocol.
2. S_C uses the Berlekamp factoring algorithm ([2]) to factor the extracted polynomial and obtain the Malicious Client's input set X.
3. S_C sends X to the Trusted Party and receives back the set $Out = X \cap Y$.
4. S_C chooses a random subset $I' \subset [10k(\lfloor logn \rfloor + 1)]$ of size k, $I' = \{j_1, \ldots, j_k\}$
5. **Input Preprocessing:**
   - S_C chooses a random value $r_{i,j,l}$ and sets $P_{y_j^{2^i}}(l) = r_{i,j,l}$ for $1 \leq j \leq max_S, 0 \leq i \leq \lfloor \log n \rfloor, l \in I$.
   - S_C sets $P_{y_j^{2^i}}(l) = 0$ for $1 \leq j \leq max_S, 0 \leq i \leq \lfloor \log n \rfloor, l \in [10k(\lfloor logn \rfloor + 1)] \setminus I'$.
   - S_C sets $P^2_{y_j^{2^i}}(l) = (P_{y_j^{2^i}}(l))^2$ for $1 \leq j \leq max_S, 0 \leq i \leq \lfloor \log n \rfloor - 1, l \in I'$
   - S_C sets $P^2_{y_j^{2^i}}(l) = 0$ for $1 \leq j \leq max_S, 0 \leq i \leq \lfloor \log n \rfloor - 1, l \in [10k(\lfloor logn \rfloor + 1)] \setminus I'$
   - S_C commits to these inputs.
6. **Choosing Random Polynomials:**
   - S_C chooses a random value $r_{j,l}$ and sets $P_{r_j}(l) = r_{j,l}$ for $1 \leq j \leq max_S, l \in I'$.
   - S_C sets $P_{r_j}(l) = 0$ for $1 \leq j \leq max_S, l \in [10k(\lfloor logn \rfloor + 1)] \setminus I'$.
   - S_C commits to these inputs
7. **Choosing Zero Polynomials:**
   - S_C chooses a random value $r_{j,l}$ and sets $P_{0,j} = r_{j,l}$ for $1 \leq j \leq max_S, l \in I'$.
   - S_C sets $P_{0,j} = 0$ for $1 \leq j \leq max_S, l \in [10k(\lfloor logn \rfloor + 1)] \setminus I'$.
   - S_C commits to these inputs
8. For $1 \leq j \leq max_S$ and for $i \in I'$, S_C honestly computes the outputs $Out_{i,j} = \mathrm{ENC}_{pk1}(s_{i,j})$ based on the inputs committed to in the previous stages.
9. For each $y_j \in Out$, S_C chooses a random polynomial P_{Out_j} of degree $k + k(\lfloor \log n \rfloor + 1)$ such that $P_{Out_j}(i) = s_{i,j}$ for $i \in I'$ and $P_{Out_j}(0) = y_j$. Note that S_C can compute $s_{i,j}$ since it has extracted the coefficients of the Client's polynomial P.
10. For each $y_j \in V$, S_C chooses a random polynomial P_{Out_j} such that $P_{Out_j}(i) = s_{i,j}$ for $i \in I'$.
11. For $Out_{i,j}$, $i \in [10k(\lfloor logn \rfloor + 1)] \setminus I', 1 \leq j \leq max_S$ S_C computes a random encryption of $P_{Out_j}(i)$.
12. S_C commits to the shares of its inputs and sends the output computed above to A_C.
13. S_C simulates a run of the HEPKPV protocol with the committed inputs from above using the simulator for the HEPKPV protocol.
14. S_C simulates a run of the Coin-Tossing protocol to ensure the outcome is the set $I' = J_R$ using the simulator for the Coin-Tossing protocol.
15. S_C plays the role of the honest Server in the Preprocessing Verification protocol to prove the preprocessing was done correctly.
16. S_C plays the role of the honest Server in the the Cut-and-Choose protocol to prove output was calculated correctly.

4.2 Sender-Side Simulator

We now consider the case in which the Sender is corrupted and the Receiver is honest.

Let A_S be a non-uniform probabilistic polynomial-time real adversary that controls the Server. We construct a non-uniform probabilistic expected polynomial-time ideal model adversary simulator S_S.

The idea behind how S_S works is that it plays the role of the Honest Client using dummy inputs: For the coefficients of the polynomial P, it sends n random encryptions of 0. Then, instead of playing the role of the prover in the zero knowledge proof of knowledge for the validity and knowledge of the coefficients of P, it invokes the Simulator for the HEPKPV protocol. In the second stage, S_S uses the extractability property of the HEPKPV to obtain the inputs of the Malicious Server from the Share Commitment Protocol. After obtaining all the input commitments and output from the Server, S_S continues to play the role of the Honest Client in the coin-tossing protocol to choose a random subset for the cut-and-choose test. If the Malicious Server passes the cut-and-choose test, then the inputs extracted earlier are submitted to the Trusted Party, otherwise the Simulator aborts (as the Honest Client does). The cut-and-choose test ensures that most of the shares of the output generated by the Malicious Server are consistent with the input extracted previously. Additionally, the honest Client in the Real Model checks that he has, in fact, received a polynomial. Due to the properties of the secret-sharing scheme, the above two points ensure that with all but negligible probability the same (correct) output will be obtained by an Honest Client in the Real and Ideal model.

We now describe in detail the Simulator for the case of the Malicious Server and Honest Client

1. S_S chooses n random encryptions of 0: $c_1, \ldots c_n$ and sends to Server.
2. S_S simulates a run of the HEPKPV protocol with input from above using the simulator for the HEPKPV protocol.
3. S_S extracts the Server's inputs using the extractor for the HEPKPV protocol.
4. A_S computes output $v_{i,j}$ for $i = 1$ to $10k(\lfloor \log n \rfloor + 1)$ and $j = 1$ to $|Y|$ and sends to Simulator
5. S_S plays the part of the Honest Client in the Coin-Tossing protocol.
6. S_S and A_S run the Cut-and-Choose protocol.
7. S_S rewinds A_S to the beginning of the Coin-Tossing protocol and re-runs the protocol with new randomness
8. S_S and A_S run the Cut-and-Choose protocol.
9. S_S repeats the previous two steps until all input indeces from 1 to $10k(\lfloor \log n \rfloor + 1)$ have been opened and the output $v_{i,j}$ has been shown to be computed correctly.
10. S_S submits the previously extracted inputs to the TP.

4.3 Computation and Communication Complexity

The communication complexity of our protocol is $O(mk^2 \log^2 n + kn)$ encryptions and the computational complexity is $O(mnk \log n + mk^2 \log^2 n)$ exponentiations.

The best known protocols for Set Intersection secure against malicious parties until now were generic protocols based on Yao's garbled circuit ([24,25]). Clearly, the communication complexity of these protocols must at least be the size of the circuit required for the functionality since all generic two-party protocols that are known require one party to send a (garbled) circuit for the functionality being evaluated.

The best known circuit for evaluating the Set Intersection functionality has size $O(m \cdot n)$, where m and n are the size of the Server and Client's inputs respectively, since we must have at least $O(m \cdot n)$ comparisons to compute the functionality. A secure implementation will require a bit-wise circuit of size at least $O(m \cdot n \cdot k)$, and this does not even take into account the costly zero-knowledge techniques that must be employed. Our communication complexity of $O(mk^2 \log^2 n + kn)$ is much smaller.

Additionally, our protocol accesses the underlying field in a black-box manner. This is in contrast to an implementation based on a Yao circuit (which must be binary) that is used in the generic protocols for 2-party computation. Therefore, our complexity scales much better as the size of the field increases.

Acknowledgment. We thank Stas Jarecki for very helpful discussions.

References

1. Agrawal, R., Evfimievski, A., Srikant, R.: Information sharing across private databases. In: SIGMOD 2003: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pp. 86–97. ACM, New York (2003)
2. Berlekamp, E.: Factoring polynomials over large finite fields. Mathematics of Computation 24, 713–735 (1970)
3. Boudot, F., Schoenmakers, B., Traoré, J.: A fair and efficient solution to the socialist millionaires problem. Discrete Applied Mathematics 111, 2001 (2001)
4. Camenisch, J., Zaverucha, G.: Private intersection of certified sets. In: Proceedings of Financial Cryptography 2009 (2009)
5. Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of Cryptology 13, 2000 (2000)
6. Choi, S., Dachman-Soled, D., Malkin, T., Wee, H.: Black-box construction of a non-malleable encryption scheme from any semantically secure one. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 427–444. Springer, Heidelberg (2008)
7. El Gamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakely, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985)
8. Fagin, R., Naor, M., Winkler, P.: Comparing information without leaking it. Communications of the ACM 39, 77–85 (1996)
9. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (2004)

10. Goldreich, O.: Foundations of cryptography: a primer. Found. Trends Theor. Comput. Sci. 1(1), 1–116 (2005)
11. Shafi, G., Silvio, M.: Probabilistic encryption & how to play mental poker keeping secret all partial information. In: STOC 1982: Proceedings of the fourteenth annual ACM symposium on Theory of computing, pp. 365–377. ACM, New York (1982)
12. Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 155–175. Springer, Heidelberg (2008)
13. Impagliazzo, R., Yung, M.: Direct minimum knowledge computations. In: Pomerance, C. (ed.) CRYPTO 1987. LNCS, vol. 293, pp. 40–51. Springer, Heidelberg (1988)
14. Jakobsson, M., Yung, M.: Proving without knowing: On oblivious, agnostic and blindfolded provers. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 186–200. Springer, Heidelberg (1996)
15. Jarecki, S., Liu, X.: Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In: TCC, pp. 577–594 (2009)
16. Katz, J., Lindell, Y.: Introduction to Modern Cryptography. Chapman & Hall/Crc Cryptography and Network Security Series. Chapman & Hall/CRC, Boca Raton (2007)
17. Kiayias, A., Mitrofanova, A.: Testing disjointness of private datasets. In: Patrick, A.S., Yung, M. (eds.) FC 2005. LNCS, vol. 3570, pp. 109–124. Springer, Heidelberg (2005)
18. Kissner, L., Song, D.X.: Privacy-preserving set operations. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 241–257. Springer, Heidelberg (2005)
19. Lindell, Y., Pinkas, B.: Privacy preserving data mining. Journal of Cryptology, 36–54 (2000)
20. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer, Heidelberg (2007)
21. Naor, M., Pinkas, B.: Oblivious transfer and polynomial evaluation. In: STOC 1999: Proceedings of the thirty-first annual ACM symposium on Theory of computing, pp. 245–254. ACM Press, New York (1999)
22. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
23. Shamir, A.: How to share a secret. Commun. ACM 22(11), 612–613 (1979)
24. Yao, A.C.-C.: Protocols for secure computations. In: FOCS, pp. 160–164 (1982)
25. Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In: FOCS, pp. 162–167 (1986)