Fair Private Set Intersection with a Semi-trusted Arbiter

Changyu Dong[1], Liqun Chen[2], Jan Camenisch[3], and Giovanni Russello[4]

[1] Department of Computer and Information Sciences, University of Strathclyde
changyu.dong@strath.ac.uk
[2] Hewlett-Packard Laboratories – Bristol, United Kingdom
liqun.chen@hp.com
[3] IBM Research – Zurich, Switzerland
jca@zurich.ibm.com
[4] Department of Computer Science, University of Auckland
g.russello@aucklanduni.ac.nz

Abstract. A private set intersection (PSI) protocol allows two parties to compute the intersection of their input sets privately. Most of the previous PSI protocols only output the result to one party and the other party gets nothing from running the protocols. However, a mutual PSI protocol in which both parties can get the output is highly desirable in many applications. A major obstacle in designing a mutual PSI protocol is how to ensure *fairness*. In this paper we present the first fair mutual PSI protocol which is efficient and secure. Fairness of the protocol is obtained in an optimistic fashion, i.e. by using an offline third party arbiter. In contrast to many optimistic protocols which require a fully trusted arbiter, in our protocol the arbiter is only required to be semi-trusted, in the sense that we consider it to be a potential threat to both parties' privacy but believe it will follow the protocol. The arbiter can resolve disputes without knowing any private information belongs to the two parties. This feature is appealing for a PSI protocol in which privacy may be of ultimate importance.

1 Introduction

An interesting problem in secure computation is private set intersection (PSI). Namely, how to enable two mutually untrusted parties to compute jointly the intersection of their private input sets. PSI has many potential applications in private data mining, online recommendation services, online dating services, medical databases and so on. There have been many protocols proposed to solve the PSI problem [1–10]. The majority of them are single-output protocols, i.e. only one party obtains the intersection and the other party gets nothing. However, there are many motivating scenarios in which both parties want to know the intersection. Several examples have been given in [6] to demonstrate the need for such *mutual* PSI protocols:

- *Two real estate companies would like to identify customers (e.g., homeowners) who are double-dealing, i.e., have signed exclusive contracts with both companies to assist them in selling their properties.*
- *A government agency needs to make sure that employees of its industrial contractor have no criminal records. Neither the agency nor the contractor are willing to disclose their respective data-sets (list of convicted felons and employees, respectively) but both would like to know the intersection, if any.*

L. Wang and B. Shafiq (Eds.): DBSec 2013, LNCS 7964, pp. 128–144, 2013.

A mutual PSI protocol must be fair, i.e. if one party knows the intersection, the other party should also know it. However fairness is hard to achieve in cryptographic protocols (see Section 2 for a brief overview). To efficiently achieve fairness, most fair cryptographic protocols are *optimistic* which requires help from an offline arbiter who is a trusted third party. The arbiter only participates if one party unfairly aborts the protocol and can recover the output from the protocol for the honest party. Incorporating optimistic fairness in PSI protocols is not easy for two reasons: firstly, although there is a generic structure, there is no generic construction for optimistic fair protocols. Secondly, the arbiter usually has to get access to some private information and therefore has to be fully trusted. However, in reality it is hard to find such a fully trusted third party. Think about the examples above: an independent entity, e.g. an auditing service provider, could be well qualified to resolve the disputes, however giving a third party access to private data may raise privacy concerns. We can find more cases in which the two parties may trust a third party for fairly resolving disputes, but may not trust it for privacy.

In this paper, we present the first fair mutual PSI protocol. The protocol has built-in support for optimistic fairness and does not require setup assumptions such as certified input sets. In addition, the third party acting as the arbiter can resolve disputes without knowing the private inputs or the output of the PSI protocol. Hence we can significantly reduce the trust placed on the arbiter. This makes the protocol more flexible in terms of practical usage as any third party can become an arbiter as long as they are believed to be able to correctly carry out instructions.

2 Related Work

Private Set Intersection (PSI) protocols allow two parties, each with a private set, to securely compute the intersection of their sets. It was first introduced by Freedman et al in [1]. Their protocol is based on oblivious polynomial evaluation. Dachman-Soled et al [2], Hazay and Nissim [3] followed the oblivious polynomial evaluation approach and proposed protocols which are more efficient in the presence of malicious adversaries. Hazey and Lindell [4] proposed another approach for PSI which is based on oblivious pseudorandom function evaluation. This approach is further improved by Jarecki and Liu [5]. De Cristofaro et al [6, 7] proposed PSI protocols with linear communication and computational complexities. Huang et al [11] presented a PSI protocol based on garble circuits, and shows in the semi-honest model the protocol can be very efficient. There are also protocols based on commutative encryption [12, 13].

All of the above protocols are single-output, i.e. one party gets the output and the other party gets nothing. This is a traditional way to simplify protocol design in the malicious model because it removes the need for fairness, i.e. how to prevent the adversary from aborting the protocol pre-maturely after obtaining the output (and before the other party obtains it) [14].

Nevertheless, there have been a few mutual PSI protocols which are designed to output the intersection to both parties. Kissner and Song [8] proposed the first mutual PSI protocol. The protocol itself does not guarantee fairness, but relies on the assumption that the homomorphic encryption scheme they use has a fair threshold decryption

protocol. However, unless there is an online trusted third party, it is also non-trivial to achieve fairness in threshold decryption protocols. On the other hand, if an online trust third party is available, the PSI functionality can be trivially computed by giving the input sets to the trusted party. Camenisch and Zaverucha [9] sketched a mutual PSI protocol which requires the input sets to be signed and certified by a trusted party. Their mutual PSI protocol is obtained by weaving two symmetric instances of a single-output PSI protocol with certified input sets. Fairness is obtained by incorporating an optimistic fair exchange scheme. However this protocol does not work in general cases where inputs are not certified because it is hard to force the two parties to use the same inputs in the two instances. Another mutual PSI protocol is proposed by Kim et al [10], but they specifically state that fairness is not considered in their security model.

Fairness is a long discussed topic in cryptographic protocols. Cleve [15] showed that *complete fairness* is impossible in two-party protocols in the malicious model. However, *partial fairness* can be achieved. Partial fairness means that one party can have an unfair advantage, but the advantage is computationally insignificant. Many protocols achieve partial fairness by using the gradual release approach [16–18]. However, this approach is very inefficient in nature. The *Optimistic* approach, which uses an offline trusted third party, has been widely used to obtain fairness efficiently. It is called optimistic because it cannot prevent the unfair behaviour but later the trusted third party can recover the output for the honest party. There has been a long line of research in this direction [19–25]. Previously, the trusted third party in an optimistic fair protocol which requires non-trivial computation on the inputs needs to be fully trusted and can get the output or inputs of the protocol if one party raises a dispute. This might not be desirable when the output or inputs should be strictly kept private. There are also other approaches for achieving partial fairness efficiently. But usually they work only for a specific problem. For example, the concurrent signatures protocol [26] allows two parties to produce and exchange two ambiguous signatures until an extra piece of information (called keystone) is released by one of the parties. The two parities obtain the signature from the other party concurrently when the keystone is released and therefore fairness is achieved. Kamara el al [27] proposed a new computation model in which a non-colluding server is involved. Fairness can be achieved in this model if there is a semi-trusted server, but the server has to be online during the computation. In our protocol we also require a semi-trusted server but it can be offline most of the time.

3 Building Blocks

3.1 Homomorphic Encryption

A semantically secure homomorphic public key encryption scheme is used as a building block in the protocol. There are two types of homomorphic encryption, additive and multiplicative. The additive homomorphic property can be stated as follows: (1) given two ciphertexts $E_{pk}(m_1), E_{pk}(m_2)$, $E_{pk}(m_1 + m_2) = E_{pk}(m_1) \cdot E_{pk}(m_2)$; (2) given a ciphertext $E_{pk}(m_1)$ and a constant c, $E_{pk}(c \cdot m_1) = E_{pk}(m_1)^c$. The multiplicative homomorphic property can be stated as follows: (1) given two ciphertexts $E_{pk}(m_1), E_{pk}(m_2)$, $E_{pk}(m_1 \cdot m_2) = E_{pk}(m_1) \cdot E_{pk}(m_2)$; (2) given a ciphertext $E_{pk}(m_1)$ and a constant c, $E_{pk}(m_1^c) = E_{pk}(m_1)^c$.

3.2 The Freedman-Nissim-Pinkas (FNP) Protocol

Our starting point is the PSI protocol in the semi-honest model proposed by Freedman et al. [1], which is based on oblivious polynomial evaluation. In this protocol, one party A has an input set X and another party B has an input set Y such that $|X| = |Y| = n$.[1] The two parties interact as follows

1. A chooses a key pair (pk, sk) for an additive homomorphic encryption scheme and makes the public key pk available to B.
2. A defines a polynomial $Q(y) = (y-x_1)(y-x_2)\dots(y-x_n) = \sum_{i=0}^{n} d_i y^i$, where each element $x_i \in X$ is a root of $Q(y)$. A then encrypts each coefficient d_i using the public key chosen in the last step and sends the encrypted coefficients $E_{pk}(d_i)$ to B.
3. For each element $y_j \in Y$, B evaluates $Q(y_j)$ obliviously using the homomorphic property $E_{pk}(Q(y_j)) = \prod_{i=0}^{n} E_{pk}(d_i)^{y_j^i}$. B also encrypts y_j using A's public key. B then chooses a random r_j and uses the homomorphic property again to compute $E_{pk}(r_j \cdot Q(y_j) + y_j) = E_{pk}(Q(y_j))^{r_j} \cdot E_{pk}(y_j)$. B sends each $E_{pk}(r_j \cdot Q(y_j) + y_j)$ to A. [2]
4. A decrypts each ciphertext received from B. If $y_j \in X \cap Y$, then $Q(y_j) = 0$, thus the decryption will be y_j which is also an element in X, otherwise, the decryption will be a random value. By checking whether the decryption is in X, A can output $X \cap Y$ while learns nothing about other elements in Y but not in X.

3.3 Zero Knowledge Proof

A zero knowledge proof protocol allows a prover to prove the validity of a statement without leaking any other information. The protocol presented in Section 3.2 is secure against semi-honest adversaries. However, in the presence of malicious adversaries we have to prevent the adversaries from deviating from the protocol. We enforce this by requiring each party to use zero knowledge proofs to convince the other party that it follows the protocol correctly. We will name the protocols as $PK_{(\dots)}$ and use the notation introduced in [28] to present the protocols in the rest of the paper

$$\text{ﻨ}\{\omega_i \in \mathcal{I}^*(m_{\omega_i})\}_{i=1}^{n} : \exists\{\chi_j \in \mathcal{I}^*(m_{\chi_j})\}_{j=1}^{m} : \phi(\omega_1, \dots, \omega_n, \chi_1, \dots, \chi_m)$$

In short, the prover is proving the knowledge of $\omega_1, \dots, \omega_n$ and the existence of $\chi_1, \dots, \chi_m$ such that these values satisfy certain predicate $\phi(\omega_1, \dots, \omega_n, \chi_1, \dots, \chi_m)$. Each ω_i and χ_j belongs to some integer domain $\mathcal{I}^*(m_{\omega_i})$ and $\mathcal{I}^*(m_{\chi_j})$. Each predicate is a boolean formula built from atomic predicates of discrete logarithms $y = \prod_{i=1}^{n} g_i^{F_i(\omega_1,\dots,\omega_n)}$, where F_i is an integer polynomial. All quantities except $\omega_1, \dots, \omega_n$ are assumed to be publicly known.

[1] In our protocol described in 4, we have a different requirement on the size of the input sets. This is due to the fact that the FNP protocol is a single output PSI protocol and ours is a mutual PSI protocol.

[2] For the sake of simplicity, we neglect the optimisations made in the paper to polynomial evaluation by using balanced allocation scheme and Horner's rule.

For example, the following means that given a certain group structure and a tuple (α, β, g, h), the prover can prove in zero knowledge that it knows the discrete logarithm x of α and there exists some s such that $\beta = h^x g^s$.

$$\text{ɪ}\, x \in \mathbb{Z}_q : \exists s \in \mathbb{Z}_q : \alpha = g^x \wedge \beta = h^x g^s$$

3.4 Verifiable Encryption

In a nutshell, a verifiable encryption scheme is a public key encryption scheme accompanied by an efficient zero knowledge proof of the plaintext satisfies certain properties [29]. It has numerous applications in key escrow, secret sharing and optimistic fair exchange. In optimistic fair exchange protocols, a convention is to let a party create a verifiable escrow of a data item. The escrow is essentially an encryption of the escrowed item under the offline arbiter's public key. A public data called a *label* is attached so that the arbiter can verify the decryption against the label to ensure certain properties hold. It also allows efficient zero knowledge proof of correct decryption to be constructed.

3.5 Perfectly Hiding Commitment

In our protocol, we also use a perfectly hiding commitment scheme [30] in zero knowledge proof protocols. Generally speaking, a commitment scheme is a protocol between two parties, the committer and the receiver. The committer can commit to a value v by generating a commitment $com(v)$ and sends it to the receiver. The commitment can be used as input to zero knowledge proof protocols. The commitment has two properties: *hiding* which means it is infeasible for the receiver to find v; *binding* which means it is infeasible for the committer to find another v' such that $com(v') = com(v)$. The strength of hiding and binding can be perfect or computational. In our case, we want a perfectly hiding commitment scheme which means the receiver cannot recover the value committed, even with unbounded computational power.

4 Overview of the Protocol

In this section, we give a high level view of the protocol as depicted in Fig. 1. The protocol has two sub-protocols: a PSI protocol to compute the set intersection between A and B and a dispute resolution protocol. Note in our protocol, all encryptions are in exponential form, i.e. rather than encrypting directly a message m, we encrypt g^m where g is a generator of a certain group. This modification is necessary to allow zero knowledge proof, and the modification does not affect the correctness or security of the encryption schemes. With this modification, oblivious polynomial evaluation is still possible if we use a multiplicative homomorphic encryption scheme rather than an additive one. The polynomial is moved to the exponent and the evaluation is done by operations on exponents. This is a standard technique in homomorphic encryption. For example, given $E_{pk}(g^a)$, $E_{pk}(g^b)$ and x, we can evaluate $ax + b$ obliviously and get $E_{pk}(g^{ax+b})$ by computing $(E_{pk}(g^a))^x \cdot E_{pk}(g^b)$. Having polynomial evaluation results on exponents is sufficient for our protocol, as the parties only need to test whether for certain y, $Q(y)$ is 0. This can be done effectively because $Q(y) = 0$ iff $g^{Q(y)} = 1$.

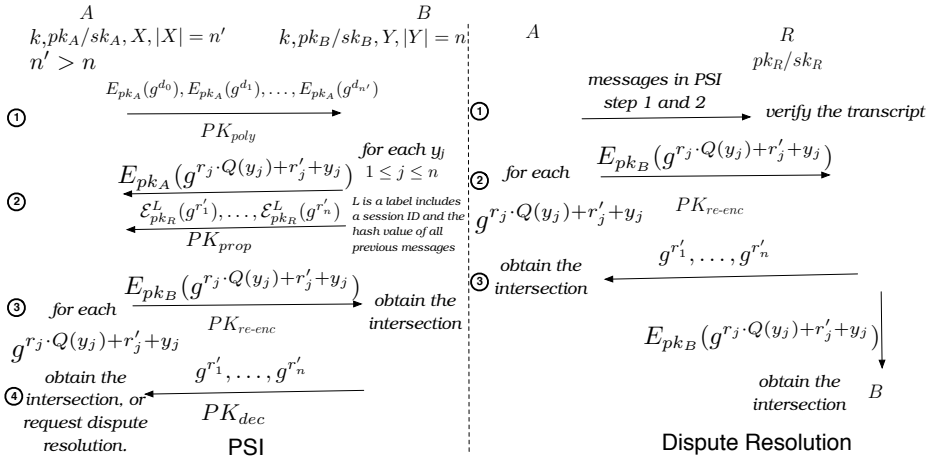

Fig. 1. Overview of the Fair PSI protocol

- **Setup:** Choose a homomorphic encryption scheme E, a verifiable encryption scheme $\mathcal{E}$, publish the public parameters. The offline arbiter R also generates a key pair for $\mathcal{E}$ and publishes the public key through a CA.
- **Private Set Intersection:** A and B are parties who engage in the computation of the set intersection, and each has a private input set X and Y respectively. In our protocol we require that A's set contains at least one random dummy element in each protocol execution. The sizes of X and Y are also required to be different. Namely, $|X| = n', |Y| = n$ such that $n' > n$. The requirements are placed to protect A's polynomial (see remark 1). A and B each also generates a random key pair for E and sends the public key to the other. They also negotiate a message authentication code (MAC) key k. This key is used by both parties to ensure the messages in the protocol execution comes from the other party. A general method to achieve this is using a MAC algorithm. To simplify presentation, we omit the MAC in the protocol description .
  1. A generates a polynomial based on A's set X as described in Section 3.2. If $d_{n'}$ is zero, regenerates the random dummy elements in X and the polynomial again until $d_{n'}$ is not zero. A encrypts all the coefficients as $E_{pk_A}(g^{d_0}), ...,$ $E_{pK_A}(g^{d_{n'}})$ and sends the ciphertexts to B. A then runs a zero knowledge proof protocol PK_{poly} to prove that the polynomial is indeed correctly constructed.
  2. For each element $y_j \in Y$, B evaluates the polynomial using the homomorphic property. Unlike in the FNP protocol that evaluates to $E_{pk_A}(r_j \cdot Q(y_j) + y_j)$, in our protocol, B also uses another random blinding factor r'_j to blind the result. So the polynomial evaluates to $E_{pk_A}(g^{r_j \cdot Q(y_j) + r'_j + y_j})$. B sends all ciphertexts to A. B then encrypts all the blinding factors r'_j using R's public key with a label L as $\mathcal{E}^L_{pk_R}(g^{r'_j})$. L includes a session ID and a hash value of all communication in the the protocol execution so far (see remark 2). B sends the encrypted

blinding factors to A, and uses PK_{prop} to prove that (1) the polynomial evaluation is properly done and (2) the encryption of blinding factors is properly done.

3. A decrypts each $E_{pk_A}(g^{r_j \cdot Q(y_j)+r'_j+y_j})$ and then encrypts each $g^{r_j \cdot Q(y_j)+r'_j+y_j}$ using B's public key. Each ciphertext $E_{pk_B}(g^{r_j \cdot Q(y_j)+r'_j+y_j})$ is sent to B and A must prove to B that the ciphertext is a correct re-encrypted ciphertext of the corresponding $E_{pk_A}(g^{r_j \cdot Q(y_j)+r'_j+y_j})$. B then decrypts each ciphertext and checks whether there is $g^{y_j+r'_j}$ that matches the decryption $g^{r_j \cdot Q(y_j)+r'_j+y_j}$, if so y_j is in $X \cap Y$.
4. B then sends $g^{r'_1}, ..., g^{r'_n}$ and proves they are correct with regard to the encryption sent in step 2. Then A will be able to test all combinations of $g^{x_i+r'_j}$ to see whether there is a match of a decryption $g^{r_j \cdot Q(y_j)+r'_j+y_j}$ it obtained in step 3, if so x_i is in $X \cap Y$. If B does not send $g^{r'_1}, ..., g^{r'_n}$ or fail to prove they are valid, A can raise a dispute with R by sending a dispute resolution request.

– **Dispute Resolution**:

1. A sends all messages sent and received in the first two setps of the PSI protocol execution to R. R verifies it by checking the consisitence between the messages and the label. If the transcript ends before the end of step 2 of the PSI protocol, R simply aborts as neither party gets any advantage.
2. A then encrypts each $g^{r_j \cdot Q(y_j)+r'_j+y_j}$ using B's public key. The ciphertext $E_{pk_B}(g^{r_j \cdot Q(y_j)+r'_j+y_j})$ is sent to R and A must prove to R that the ciphertext is a correct re-encrypted ciphertext of the corresponding $E_{pk_A}(g^{r_j \cdot Q(y_j)+r'_j+y_j})$ in the transcript.
3. R decrypts $\mathcal{E}^L_{pk_R}(g^{r'_1}), ..., \mathcal{E}^L_{pk_R}(g^{r'_n})$ and sends $g^{r'_1}, ..., g^{r'_n}$ to A, so that A can learn the intersection $X \cap Y$.
4. R also sends all $E_{pk_B}(g^{r_j \cdot Q(y_j)+r'_j+y_j})$ to B.

Remark 1: In the initialisation stage of the PSI protocol, we require A to randomise its set X by adding at least one random and secret dummy element, and make sure $|X| > |Y|$. This is to protect A's privacy. Plaintext in each $E_{pk_B}(g^{r_j \cdot Q(y_j)+r'_j+y_j})$ needs to be released to B in the PSI protocol. As r_j and r'_j are chosen by B, B might be able to recover $g^{Q(y_j)}$. B can recover A's polynomial if it can obtain at least n' $(g^{Q(y_j)}, y_j)$ pairs. In any execution of the protocol, B can recover at most n pairs. Because $n' > n$, the attack is not possible. Randomising the polynomial in each execution prevents B from pooling information gathered from different executions to recover A's polynomial.

Remark 2: We let B to encrypt blinding factors with a label L in step 2. The label L is for two purposes: (1) to ensure timeliness of dispute resolution. A session ID is attached to each protocol execution and B uses it as an input when generating the label. We assume a standard format and semantics of the session ID have been agreed by all parities beforehand, so that R can verify the identities of the two parties involved and that the protocol execution is within a certain time window. (2) To ensure the integrity of the messages in the first two steps of the protocol. As only A can raise a dispute resolution, B needs to ensure A cannot get any advantage by modifying critical messages, e.g. the encrypted coefficients and polynomial evaluation results. By using the hash of past communication as an input for the label, B can ensure that. This is because

the ciphertext with the label is encrypted under R's public key so cannot be modified without R's private key, and any modification to the messages will invalidate the label so R can detect it.

Remark 3: In our protocol B adds an additional blinding factor r'_j when evaluating A's polynomial. This is because if we follow the FNP protocol and do not add this blinding factor, then there is no good way to deal the case in which A aborts after decrypting all $E_{pk_A}(g^{r_j \cdot Q(y_j)+y_j})$. In this case to maintain fairness, B needs R to recover the set intersection. A would have to to provide a verifiable encryption of its private key sk_A in order for R to decrypt $E_{pk_A}(g^{r_j \cdot Q(y_j)+y_j})$ for B. But that will violate A's privacy because given the private key R can also recover A's polynomial coefficients from the transcript. Our design is better because now R only gets random numbers $g^{r'_1}, ..., g^{r'_n}$ which contain no information about both parties' sets.

Remark 4: In the last step of the dispute resolution protocol, R sends $E_{pk_B}(g^{r_j \cdot Q(y_j)+r'_j+y_j})$ to B. This is needed because from the transcript, R cannot tell whether A has sent them to B or not. It is possible that A unfairly aborts the protocol after finishing step 2 and then uses R to recover the result. we add this step to make sure B also receives the output in this case. And because this is the only case that A can gain advantage by unfairly aborting the protocol, we do not need a dispute resolution protocol for B.

5 A Concrete Construction

5.1 Verifiable Encryption

As a setup requirement. the arbiter R must have a key pair of a verifiable encryption scheme. In the second step of the PSI protocol, B must encrypt the blinding factors $r'_1, r'_2, ..., r'_n$ under R's public key. The encryption scheme used by R is the Cramer-Shoup encryption [31] with a small modification. The system works in this way:

- **Setup:** On input 1^k, output two prime numbers p, q such that q divides $p-1$, a cyclic group $\mathbb{G}$ with two generator g, h such that $\mathbb{G}$ is the unique order q subgroup of $\mathbb{Z}_p^*$. Choose $u_1, u_2, v_1, v_2, w \overset{R}{\leftarrow} \mathbb{Z}_q$. Compute $a = g^{u_1}h^{u_2}, b = g^{v_1}h^{v_2}, c = g^w$. Then publish (a, b, c) along with $\mathbb{G}, q, g, h$ as the public key and retain (u_1, u_2, v_1, v_2, w) as the private key.
- **Encryption:** To encrypt a message m, calculate the following:
  - $e_1 = g^z, e_2 = h^z, e_3 = c^z m$ where $z \overset{R}{\leftarrow} \mathbb{Z}_q$.
  - $\sigma = H(e_1, e_2, e_3, L)$, where H is a hash function and L is the label.
  - $e_4 = a^z b^{z\sigma}$
  - The ciphertext is (e_1, e_2, e_3, e_4).
- **Decryption:** To decrypt, compute $\sigma = H(e_1, e_2, e_3, L)$, then verify $e_1^{u_1} e_2^{u_2} (e_1^{v_1} e_2^{v_2})^{\sigma} = e_4$. If the verification succeeds, then decrypt $m = e_3/(e_1^w)$

The only modification we made to the original Cramer-Shoup encryption is that L is added as an ingredient of σ. All security properties of the Cramer-Shoup encryption are inherited.

5.2 A Homomorphic Encryption Scheme

At the core of our construction is a semantically secure homomorphic encryption scheme. Our choice is the ElGamal [32] encryption scheme. This allows us to construct efficient zero knowledge proofs needed in the protocol. To simplify design, we share certain parameters between E and $\mathcal{E}$. The scheme is described as follows:

- **Setup:** Use the same group $\mathbb{G}$ and generator g as in section 5.1. Choose $x \overset{R}{\leftarrow} \mathbb{Z}_q$ and compute g^x. The public key is $pk = (\mathbb{G}, g, g^x, q)$ and the private key is $sk = x$.
- **Encryption:** Choose $r \overset{R}{\leftarrow} \mathbb{Z}_q$ and output the ciphertext $c(m) = (g^r, m(g^x)^r)$.
- **Decryption:** The ciphertext is decrypted as $m(g^x)^r \cdot (g^r)^{-x} = mg^{rx-rx} = m$.

ElGamal is multiplicative homomorphic, so it is suitable in our protocol. As mentioned before we will convert the plaintext m to g^m before encryption, so that oblivious polynomial evaluation is possible using ElGamal.

5.3 Zero Knowledge Proof Protocols

PK_{poly}: Proof of Correct Construction of a Polynomial In step 1 of the PSI protocol, A has to prove to B that the polynomial is constructed correctly. Namely, A has to convince B that it knows the polynomial and the polynomial has no more than n' roots. For each coefficient d_i, the ciphertext is $E_{pk_A}(g^{d_i}) = (g^{t_i}, g^{d_i} g^{x_A t_i}) = (\alpha_{d_i}, \alpha'_{d_i})$, where t_i is a random number in $\mathbb{Z}_q$. To prove it knows the polynomial, A runs the following protocol:

$$\aleph\, d_i \in \mathbb{Z}_q : \exists t_i \in \mathbb{Z}_q : \alpha_{d_i} = g^{t_i} \wedge \alpha'_{d_i} = g^{d_i} (g^{x_A})^{t_i}$$

As the maximum degree of the polynomial is determined beforehand and can be verified by counting the number of encrypted coefficients received, then for a polynomial of degree n', the only case that it can have more than n' roots is when all coefficients are zero. To show the coefficients are not all zero, we require A to prove that $d_{n'}$ is not zero by running

$$\exists t_{n'}, t'_{n'} \in \mathbb{Z}_q : \alpha_{d_{n'}} = g^{t_{n'}} \wedge \alpha'_{d_{n'}} = (g^{x_A})^{t'_{n'}} \wedge t_{n'} \neq t'_{n'}$$

Intuitively, $t'_{n'} = t_{n'} + d_{n'}/x_A$ and therefore $t_{n'} = t'_{n'}$ iff $d_{n'} = 0$. So by verifying $t_{n'} \neq t'_{n'}$, B can be convinced that $d_{n'} \neq 0$. To prove the inequality of discrete logarithms, we can use the protocol proposed in [29].

PK_{prop}: Proof of Proper Polynomial Evaluation and Encryption In step 2 of the PSI protocol, B must prove that each $E_{pk_A}(g^{r_j \cdot Q(y_j) + r'_j + y_j})$ is a proper ciphertext for $g^{r_j \cdot Q(y_j) + r'_j + y_j}$, and also each $\mathcal{E}^L_{pk_R}(g^{r'_j})$ is a proper encryption under R's public key and the label L.

Recall that for an encrypted coefficient d_i, $E_{pk_A}(g^{d_i}) = (g^{r_i}, g^{d_i} g^{x_A r_i}) = (\alpha_{d_i}, \alpha'_{d_i})$. Then for each term $d_i y_j^i$ of the polynomial, the ciphertext computed using the homomorphic property from $E_{pk_A}(g^{d_i})$ is $E_{pk_A}(g^{d_i y_j^i}) = ((\alpha_{d_i})^{y_j^i}, (\alpha'_{d_i})^{y_j^i})$. Similarly, for each $r_j \cdot Q(y_j)$, the ciphertext is

$$E_{pk_A}(g^{r_j \cdot Q(y_j)}) = ((\prod_{i=0}^{n'} (\alpha_{d_i})^{r_j y_j^i}), (\prod_{i=0}^{n'} (\alpha'_{d_i})^{r_j y_j^i}))$$

B also encrypts $g^{r'_j+y_j}$ by itself, and the ciphertext $E_{pk_A}(g^{r'_j+y_j}) = (g^{\widehat{r}'_j}, g^{r'_j} g^{y_j} g^{x_A \widehat{r}'_j})$. The ciphertext of the whole can be obtained by multiplying the corresponding components of the two:

$$E_{pk_A}(g^{r_j \cdot Q(y_j)+r'_j+y_j}) = (\alpha, \beta) = ((\prod_{i=0}^{n'} (\alpha_{d_i})^{r_j y_j^i}) \cdot g^{\widehat{r}'_j}, (\prod_{i=0}^{n'} (\alpha'_{d_i})^{r_j y_j^i}) \cdot g^{r'_j} g^{y_j} g^{x_A \widehat{r}'_j})$$

For each $\mathcal{E}^L_{pk_R}(g^{r'_j})$, the ciphertext is $(e_{1j}, e_{2j}, e_{3j}, e_{4j})$, such that $e_{1j} = g^{z_j}, e_{2j} = h^{z_j}, e_{3j} = c^{z_j} g^{r'_j}, e_{4j} = a^{z_j} b^{z_j \sigma}$ where $z_j \overset{R}{\leftarrow} \mathbb{Z}_q$ and $\sigma = H(e_{1j}, e_{2j}, e_{3j}, L)$.

The proof has two steps. In the first step, B commits to y_j and $r_j y_j^i$ for each $y_j \in Y$ and $0 \leq i \leq n'$. We use the Pedersen Commitment Scheme [30] here. This commitment scheme is known to be perfectly hiding and computationally binding. It is a discrete logarithm based scheme, that enables us to re-use the parameters used for the encryption schemes. We use the same group $\mathbb{G}$, and parameters q, g, h as in section 5.1. To commit to v, choose a random s and create $com(v) = g^v h^s$. So we have $com(y_j) = g^{y_j} h^{\widetilde{s}_j}$, and $com(a_{j,i}) = g^{r_j y_j^i} h^{s_i}$ for each $a_{j,i} = r_j y_j^i$. Then starting from $i = 1$, B must prove that the value committed in $com(a_{j,i})$ is the product of the values committed in $com(a_{j,i-1})$ and $com(y_j)$. To do this, we use the protocol from [33] which proves a committed value in γ_i is the product of two other values committed in δ, γ_{i-1}:

$$\exists y_j, a_{j,i-1}, a_{j,i}, \widetilde{s}_j, s_{i-1}, s_i \in \mathbb{Z}_q : \gamma_i = g^{a_{j,i}} h^{s_i} \wedge \delta = g^{y_j} h^{\widetilde{s}_j} \wedge \gamma_{i-1} = g^{a_{j,i-1}} h^{s_{i-1}}$$

The protocol is correct because $a_{j,i} = a_{j,i-1} \cdot y_j$. Now A has a series of correct commitments of a geometric sequence $a_{j,i} = r_j y_j^i$ for $0 \leq i \leq n'$. In the second step, B runs the following protocol for each $0 \leq j \leq n$:

$$\begin{aligned} \aleph\, r'_j, y_j \in \mathbb{Z}_q : \exists a_{j,0}, ..., a_{j,n'}, \widehat{r}'_j, z_j \in \mathbb{Z}_q : \delta = g^{y_j} h^{\widetilde{s}_j} \bigwedge_{i=0}^{n'} \gamma_i = g^{a_{j,i}} h^{s_i} \\ \wedge \alpha = (\prod_{i=0}^{n'} (\alpha_{d_i})^{a_{j,i}}) \cdot g^{\widehat{r}'_j} \wedge \beta = (\prod_{i=0}^{n'} (\alpha'_{d_i})^{a_{j,i}}) \cdot g^{r'_j} g^{y_j} g^{x_A \widehat{r}'_j} \\ \wedge e_{1j} = g^{z_j} \wedge e_{2j} = h^{z_j} \wedge e_{3j} = c^{z_j} g^{r'_j} \wedge e_{4j} = a^{z_j} b^{z_j \sigma} \end{aligned}$$

B proves in the first two lines that it knows y_j, r'_j, also each exponent $a_{j,i}$ in α and β match the value committed in γ_i, y_j in β matches the value committed in δ, r'_j matches the value encrypted in e_{3j}, and (α, β) is a proper ciphertext of the polynomial evaluation result. In the last line, B proves that the verifiable encryption is correct.

***$PK_{re\text{-}enc}$*: Proof of Correct Re-encryption** In step 3 of the PSI protocol and step 2 of the dispute resolution protocol, A must prove that each value sent is the correct ciphertext $E_{pK_B}(g^{r_j \cdot Q(y_j)+r'_j+y_j})$. A generates the ciphertext by first decrypting $E_{pK_A}(g^{r_j \cdot Q(y_j)+r'_j+y_j})$, and then re-encrypting the result using B's public key. The two ciphertexts are

$$E_{pk_A}(g^{r_j \cdot Q(y_j)+r'_j+y_j}) = (g^{t_j}, g^{r_j \cdot Q(y_j)+r'_j+y_j} g^{x_A t_j}) = (g^{t_j}, m_j g^{x_A t_j})$$

$$E_{pk_B}(g^{r_j \cdot Q(y_j)+r'_j+y_j}) = (g^{t'_j}, g^{r_j \cdot Q(y_j)+r'_j+y_j} g^{x_B t'_j} = (g^{t'_j}, m_j g^{x_B t'_j}))$$

where t_j, t'_j are random numbers. The protocol is then:

$$\exists x_A, t'_j \in \mathbb{Z}_q : pk_A = g^{x_A} \wedge \alpha = m_j (g^{t_j})^{x_A} \wedge \beta = g^{t'_j} \wedge \gamma = m_j (g^{x_B})^{t'_j}$$

The proof shows that the two ciphertexts are correct and encrypt the same plaintext.

PK_{dec}: Proof of Correct Decryption In step 4 of the PSI protocol, B needs to prove that each $g^{r_j'}$ is a correct decryption of $\mathcal{E}^L_{pk_R}(g^{r_j'})$. For each $\mathcal{E}^L_{pk_R}(g^{r_j'})$, the ciphertext is $(e_{1j}, e_{2j}, e_{3j}, e_{4j})$, such that $e_{1j} = g^{z_j}, e_{2j} = h^{z_j}, e_{3j} = c^{z_j} g^{r_j'}, e_{4j} = a^{z_j} b^{z_j \sigma}$ where $z_j \overset{R}{\leftarrow} \mathbb{Z}_q$ and $\sigma = H(e_{1j}, e_{2j}, e_{3j}, L)$. What B needs to show is that it knows z_j and z_j is used consistently in all ciphertext compoents.

$$\exists z_j \in \mathbb{Z}_q : e_{1j} = g^{z_j} \wedge e_{2j} = h^{z_j} \wedge e_{3j} = c^{z_j} g^{r_j'} \wedge e_{4j} = a^{z_j} (b^{\sigma})^{z_j}$$

If $g^{r_j'}$ is not the correct decryption, then B cannot find a z_j that satisfies the relation.

5.4 Complexity Analysis

Now we give an account of the complexity of the protocol. The computational and communication complexity of the zero knowledge proof protocol is linear in the number of statements to be proved, so we separate it from the main protocol. In the PSI protocol, A needs to perform $3n'$ exponentiations to encrypt the coefficients in step 1, and $3n$ exponentiations to decrypt and re-encrypt the polynomial evaluation results in step 3, B needs $2(n'n + 2n)$ exponentiations to evaluate the polynomial obliviously and $3n$ exponentiations for the verifiable encryption in step 2. The messages sent in the protocol consist of $2n' + 9n$ group elements. In the dispute resolution protocol, R needs $6n$ exponentiations to verify and decrypt the ciphertexts of the verifiable encryption. The total traffic generated includes $5n$ group elements, plus the transcript sent in step 1. In total, the computational complexity is $O(nn')$ and the communication complexity is $O(n + n')$. The complexity of the zero-knowledge proof protocols: PK_{poly} is $O(n')$, PK_{prop} is $O(nn')$, $PK_{re\text{-}enc}$ is $O(n)$, and PK_{dec} is $O(n)$. The complexity of our protocol is similar to other PSI protocols in the malicious model [2, 3].

6 Security Analysis

6.1 Security Model

The basic security requirements of our protocol are correctness, privacy and fairness. Informally, correctness means an honest party is guaranteed that the output it receives is correct with regard to the actual input and the functionality realised by the protocol; privacy means no party should learn more than its prescribed output from the execution of the protocol; fairness means a dishonest party should receive its output if and only if the honest party also receives its output.

We define a security model to capture the above security requirements in terms of the simulation paradigm [14]. We model the parties A, B and R as probabilistic interactive Turing machines. A *functionality* is denoted as $f : \mathcal{X}_A \times \mathcal{X}_B \rightarrow \mathcal{Y}_A \times \mathcal{Y}_B$, In our protocol, the functionality to be computed by A and B is the set intersection. The model is similar to the one used in the optimistic fair secure computation protocol [23]. Generally speaking the protocol is executed in a real world model where the participants may be corrupted and controlled by an adversary. To show the protocol is secure, we define an ideal process which satisfies all the security requirements. In the ideal process, there is an incorruptible trusted party which helps in the computation of the functionality, e.g. in our case the set intersection. The protocol is said to be secure if for every adversary

in the real world model there is also an adversary in the ideal world model who can simulate the real world adversary.

The Real World. The protocol has three participants A, B and R. All participants have the public parameters of the protocol including the function $f_\cap$, the security parameter κ, R's public key and other cryptographic parameters to be used. A has a private input X, B has a private input Y and R has an input $\in \{\diamond, \perp\}$. The participants of the protocol can be corrupted by an adversary. The adversary can corrupt up to two parties in the protocol. We use C to denote the adversary. The adversary can behave arbitrarily, e.g. substitute local input, abort the protocol prematurely, and deviate from the protocol specification. At the end of the execution, an honest party outputs whatever prescribed in the protocol, a corrupted party has no output, and an adversary outputs its view. For a fixed adversary C, and input X, Y, the joint output of A, B, R, C is denoted by $O_{ABRC}(X, Y)$ which is the random variable consisted of all the outputs as stated.

The Ideal Process. In the ideal process, there is an incorruptible trust party T, and parties $\bar{A}, \bar{B}, \bar{R}$. $\bar{A}$ has input X, $\bar{B}$ has input Y and $\bar{R}$ has an input $\in \{\diamond, \perp\}$. The operation is as follows:

- $\bar{A}$ sends X' or $\perp$ to T, then $\bar{B}$ sends Y' or $\perp$ to T, then $\bar{R}$ sends two messages $b_A \in \mathcal{Y}_A \cup \{\diamond, \perp\}$ and $b_B \in \mathcal{Y}_B \cup \{\diamond, \perp\}$ to T. The actual input X' and Y' may be different from X and Y if the party is malicious.
- T sends private delayed output to $\bar{A}$ and $\bar{B}$. T's reply to $\bar{A}$ depends on $\bar{A}$ and $\bar{B}$'s messages and b_A. T's reply to $\bar{B}$ depends on $\bar{A}$ and $\bar{B}$'s messages and b_B.
  - T to $\bar{A}$: (1) If $b_A = \diamond$, $\bar{A}$ sends X' and $\bar{B}$ sends Y', T sends $X' \cap Y'$ to $\bar{A}$.
    (2) Else if $b_A = \diamond$, but $\bar{A}$ or $\bar{B}$ sends $\perp$, T sends $\perp$ to $\bar{A}$
    (3) Else if $b_A \neq \diamond$, T sends b_A to $\bar{A}$.
  - T to $\bar{B}$: (1) If $b_B = \diamond$, $\bar{A}$ sends X' and $\bar{B}$ sends Y', T sends $X' \cap Y'$ to $\bar{B}$.
    (2) Else if $b_B = \diamond$, but $\bar{A}$ or $\bar{B}$ sends $\perp$, T sends $\perp$ to $\bar{B}$.
    (3) Else if $b_A \neq \diamond$, T sends b_B to $\bar{B}$.

Honest parties in the ideal process behave as follows: $\bar{A}$ and $\bar{B}$ send their input to T and $\bar{R}$ sends $b_a = \diamond$ and $b_B = \diamond$. The ideal process adversary $\bar{C}$ controls the behaviours of corrupted parties. It gets the input of a corrupted party and may substitute them. It also gets T's answer to corrupted parties. For a fixed adversary $\bar{C}$, and input X, Y, the joint output of $\bar{A}, \bar{B}, \bar{R}, \bar{C}$ in the ideal process is denoted by $O_{\bar{A}\bar{B}\bar{R}\bar{C}}(X, Y)$.

Simulatability. The security definition is in terms of simulatability:

Definition 1. *Let $f_\cap$ be the set intersection functionality. We say a protocol Π securely computes $f_\cap$ if for every real-world adversary C, there exists an adversary $\bar{C}$ in the ideal process such that for all $X \in \mathcal{X}_A$, for all $Y \in \mathcal{X}_B$, the joint distribution of all outputs of the ideal process is computationally indistinguishable from the outputs in the real world, i.e.,*

$$O_{\bar{A},\bar{B},\bar{R},\bar{C}}(X, Y) \stackrel{c}{\approx} O_{ABRC}(X, Y)$$

The design of the ideal process captures the security we want to achieve from the real protocol. Our assumption is that in real world, we can find a semi-trusted arbiter that can be trusted for fairly resolving disputes, but not for privacy. Then by incorporating

such an arbiter in a two-party private set intersection protocol, we can achieve fairness, correctness and privacy. In the ideal process, if $\bar{R}$ follows the protocol and does not collude with $\bar{A}$ or $\bar{B}$ then all security properties are guaranteed. In this case, $\bar{A}$ and $\bar{B}$ will always get the correct intersection with regard to the actual input to the protocol, and know nothing more than that. On the other hand, if $\bar{R}$ is corrupted and colludes with $\bar{A}$ or $\bar{B}$, then fairness is not guaranteed. However, even in this case privacy is guaranteed. That is, the corrupted parties will not get more information about the honest party's set other than the intersection.

6.2 Security Proof

We are now ready to state and prove the security of our protocol. The protocol uses zero knowledge proof protocols as subprotocols. As they are obtained by using existing secure protocols and standard composition techniques, they are consequently secure and we omit the security proofs of them. To prove the main theorem below, we work in a *hybrid model* in which the real protocol is replaced with a hybrid protocol such that every invocation of the subprotocols is replaced by a call to an ideal functionality computed by a trusted party. In our case we need ideal functionalities for zero knowledge proofs and certification authority. If the subprotocols are secure, then by the composition theorem [34] the output distribution of the hybrid execution is computationally indistinguishable from the output distribution of the real execution. Thus, it suffices to show that the ideal execution is indistinguishable from the hybrid execution.

Theorem 1. *If the encryption E and $\mathcal{E}$ are semantically secure, and the associated proof protocols are zero knowledge proof, the optimistic fair mutual private set intersection protocol securely computes $f_{\cap}$.*

Because of limited space, below we only sketch the proof. The detailed proof will appear in the full version.

Proof. Let's first consider the cases that the adversary C corrupts two parties.

Case 1: C corrupts and controls A and B. This is a trivial case because C has full knowledge on X, Y and if the encryption scheme used by R is semantically secure, a simulator can always be constructed.

Case 2: C corrupts and controls A and R. We construct a simulator S in the ideal process that corrupts and controls $\bar{A}$ and $\bar{R}$. It uses the adversary C as a subroutine and we will show the simulatability holds in this case.

1. S is given A and R's inputs, S invokes an ideal functionality CA to obtain R's key pair, then invokes C and plays the role of B.
2. S generates a public/private key pair pk_B/sk_B and gives the public key to C.
3. S receives the encrypted coefficients $E_{pk_A}(d_i)$ from C. S also receives $d_i, 0 \leq i \leq n'$ for the ideal computation of PK_{poly}, where d_i is a coefficient of the polynomial. If the polynomial is not correctly constructed, then S instructs $\bar{A}$ to send $\perp$ to T and terminates the execution. If the polynomial is correct, S extracts input X' from the coefficients, instructs $\bar{A}$ to send X' to T and instructs $\bar{R}$ to send $b_A = \diamond$ to T. S then receives the intersection $X' \cap Y$ from T.

4. S then constructs Y' from the intersection received in last step by adding random dummy elements until $|Y'| = n$. Then It generates a set of random blinding factors $r'_1, r'_2, ..., r'_n$, computes $E_{PK_A}(g^{r_j \cdot Q(y'_j)+r'_j+y'_j})$ and encrypts all blinding factors using R's public key. It also generates commitments for each y'_j and $r_j y'^i_j$. S sends all commitments and ciphertexts to C and also emulates the ideal computation of PK_{prop} by sending "accept" to C. Depends on C's reply, executes step 5, 6 or 7. In the next three steps, S will send an instruction to T when it is ready to output, then T sends the delayed output to $\bar{B}$
5. If C instructs both A and R to abort, then S instructs $\bar{R}$ to send $b_B = \perp$ to T, then outputs whatever C outputs and terminates.
6. If C instructs A to abort and instructs R to send n ciphertexts, S decrypts them using B's private key, constructs a set by testing whether any elements in Y' match the decryption results. Then S collects all matching elements, put them in a set and instructs $\bar{R}$ to send the set as b_B. Then S outputs whatever C outputs and terminates.
7. If C instructs A to send n ciphertexts, then S extracts a set of elements from the reply and engages in the ideal computation of PK_{re-enc}. If the reply is correct, S instructs $\bar{R}$ to send $b_B = \diamond$ to T and sends $g^{r'_1}, ..., g^{r'_n}$ to C. If the reply is not correct and C instructs R to abort, S instructs $\bar{R}$ to send $b_B = \perp$ to T. If the reply is not correct and C instructs R to send n ciphertexts, S extracts a set of elements from the cipehrtexts and instructs $\bar{R}$ to send the set as b_B to T. Then it outputs whatever C outputs and terminates.

In the joint output, the honest parties' outputs are always the same. All we need to check is whether the view of the simulator is indistinguishable from the view of an adversary in the hybrid execution. The difference between a simulation and a hybrid execution is that in the simulation S uses Y' which is not the same as Y. However, this does not affect the distribution of the views. From how Y' is constructed we can see that Y' contains the correct intersection ($Y \cap X' \subseteq Y'$). For those elements in the intersection, they produce the same distributions in the simulation (using Y') and the hybrid execution (using Y). For any elements $y'_j \in Y'$ and $y_j \in Y$ not in the intersection, the commitments produced should be indistinguishable because the commitment scheme is perfectly hiding. Also $g^{r_j \cdot Q(y_j)+r'_i+y'_j}$ and $g^{r_j \cdot Q(y'_j)+r'_i+y_j}$ are uniformly random because $Q(y_j)$ and $Q(y'_j)$ are both non-zero, and so are the ciphertexts of them. The blinding factors and their ciphertexts are uniformly random in both the simulation and the hybrid execution. Therefore the two views are indistinguishable.

Case 3: C corrupts and controls B and R. We construct a simulator S in the ideal process that corrupts and controls $\bar{B}$ and $\bar{R}$. It uses the adversary C as a subroutine.

1. S is given B and R's inputs, S invokes an ideal functionality CA to obtain R's key pair, then invokes C and plays the role of A.
2. S generates a key pair pk_A/sk_A and gives the public key to C.
3. S generates a random set X' such that $|X'| = n'$, then constructs a polynomial using elements in X'. S encrypts the coefficients, sends them to C and simulates the ideal computation of PK_{poly} by sending "accept" to C.

4. S receives the commitments and ciphertexts from C, then receives inputs to the ideal computation of PK_{prop}, including $(y_j, r'_j), 0 \leq j \leq n$. If the ciphertexts are not properly produced, S instructs $\bar{B}$ to send $\perp$ to T, otherwise S extract Y' and instructs $\bar{B}$ to send Y' to T and instructs $\bar{R}$ to send $b_B = \diamond$ to T, and receives $X \cap Y'$ from T.
5. S constructs another set X'' such that $X \cap Y' \subseteq X''$ and $|X''| = n'$. S then constructs another polynomial Q'', and evaluates the polynomial using $(y_j, g^{r'_j})$ to construct $E_{pk_B}(g^{r_j \cdot Q''(y_j) + r'_j + y_j})$. The ciphertexts are sent to C, S also simulates the ideal computation of PK_{re-enc} by sending "accept" to C. Depends on C's reply, executes step 6,7 or 8. In the next three steps, S will send an instruction to T when it is ready to output, then T sends the delayed output to $\bar{A}$
6. If C instructs B to send the blinding factors, then S instructs $\bar{R}$ to send $b_A = \diamond$, outputs whatever C outputs and terminates.
7. If C instructs both B and R to abort, then S instructs $\bar{R}$ to send $b_A = \perp$, outputs whatever C outputs and terminates.
8. If C instructs B to abort and R to send n blinding factors, use the blinding factors to extract a set, and then instructs $\bar{R}$ to send the extracted set as b_A to T. S then outputs whatever C outputs and terminates.

The difference between a simulation and a hybrid execution is that the simulator uses X' and X'' rather than the honest party's input X. Using X' does not affect the distribution of the view if E is semantically secure, because the ciphertexts generated using A's public key are indistinguishable. Using X'' also does not affect the distribution of the view. For the two sets X'' and X, two polynomials are constructed from them Q'' and Q. We also know $X'' \cap Y' = X \cap Y'$, so $Q''(y_j) = 0$ iff $Q(y_j) = 0$ for any $y_j \in Y'$. For each $g^{r_j \cdot Q''(y_j) + r'_j + y_j}$ and $g^{r_j \cdot Q(y_j) + r'_j + y_j}$, if $Q''(y_j) = 0$ then $Q(y_j) = 0$ so the distribution of the two depends only on y_j and r'_j, if $Q''(y_j) \neq 0$ then $Q(y_j) \neq 0$ and both $Q''(y_j)$ and $Q''(y_j)$ are uniformly random, so $g^{r_j \cdot Q''(y_j) + r'_j + y_j}$ and $g^{r_j \cdot Q(y_j) + r'_j + y_j}$ are also uniformly random. Therefore the distributions of the views are indistinguishable.

For cases that C corrupts only one party, proofs can be constructed similarly. In the case that R is corrupted, R is not involved in the protocol because A and B are honest, so it is trivial to construct a simulator. In the case that A or B is corrupted, the simulator can be constructed as in case 2 step 1 – 4 or case 3 step 1 – 5, except now $\bar{R}$ is honest and always sends $\diamond$ to T. The view from the simulation is still indistinguishable.

7 Conclusion and Future Work

In this paper, we have presented a fair mutual PSI protocol which allows both parties to obtain the output. The protocol is optimistic which means fairness is obtained by using an offline third party arbiter. To address the possible privacy concerns raised by introducing a third party, the protocol is designed to enable the arbiter to resolve dispute blindly without knowing any private information from the two parties. We have analysed and shown that the protocol is secure.

The communication and computation complexity of our protocol are both $O(nn')$. The main overhead comes from the oblivious polynomial evaluation and the large accompanying zero knowledge proof. We would like to investigate PSI protocols based on

other primitives, e.g. [6, 7], to see whether efficiency can be improved. Another area we would like to investigate is whether the protocol structure that we use to obtain fairness can be made general so that it can be applied to other secure computation protocols.

Acknowledgements. We would like to thank the anonymous reviewers. Changyu Dong is supported by a Science Faculty Starter Grant from the University of Strathclyde.

References

1. 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)
2. Dachman-Soled, D., Malkin, T., Raykova, M., Yung, M.: Efficient robust private set intersection. In: Abdalla, M., Pointcheval, D., Fouque, P.-A., Vergnaud, D. (eds.) ACNS 2009. LNCS, vol. 5536, pp. 125–142. Springer, Heidelberg (2009)
3. Hazay, C., Nissim, K.: Efficient set operations in the presence of malicious adversaries. In: Public Key Cryptography, pp. 312–331 (2010)
4. 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)
5. Jarecki, S., Liu, X.: Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 577–594. Springer, Heidelberg (2009)
6. De Cristofaro, E., Tsudik, G.: Practical private set intersection protocols with linear complexity. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 143–159. Springer, Heidelberg (2010)
7. De Cristofaro, E., Kim, J., Tsudik, G.: Linear-complexity private set intersection protocols secure in malicious model. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 213–231. Springer, Heidelberg (2010)
8. Kissner, L., Song, D.: Privacy-preserving set operations. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 241–257. Springer, Heidelberg (2005)
9. Camenisch, J., Zaverucha, G.M.: Private intersection of certified sets. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 108–127. Springer, Heidelberg (2009)
10. Kim, M., Lee, H.T., Cheon, J.H.: Mutual private set intersection with linear complexity. IACR Cryptology ePrint Archive 2011, 267 (2011)
11. Huang, Y., Evans, D., Katz, J.: Private set intersection: Are garbled circuits better than custom protocols? In: 19th Network and Distributed Security Symposium (2012)
12. Agrawal, R., Evfimievski, A.V., Srikant, R.: Information sharing across private databases. In: SIGMOD Conference, pp. 86–97 (2003)
13. Vaidya, J., Clifton, C.: Secure set intersection cardinality with application to association rule mining. Journal of Computer Security 13(4), 593–622 (2005)
14. Goldreich, O.: Foundations of Cryptography: Volume II Basic Applications. Cambridge University Press (2004)
15. Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: STOC, pp. 364–369 (1986)
16. Blum, M.: How to exchange (secret) keys. ACM Trans. Comput. Syst. 1(2), 175–193 (1983)
17. Ben-Or, M., Goldreich, O., Micali, S., Rivest, R.L.: A fair protocol for signing contracts. IEEE Transactions on Information Theory 36(1), 40–46 (1990)

18. Pinkas, B.: Fair secure two-party computation. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 87–105. Springer, Heidelberg (2003)
19. Asokan, N., Schunter, M., Waidner, M.: Optimistic protocols for fair exchange. In: ACM Conference on Computer and Communications Security, pp. 7–17 (1997)
20. Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures (extended abstract). In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 591–606. Springer, Heidelberg (1998)
21. Bao, F., Deng, R.H., Mao, W.: Efficient and practical fair exchange protocols with off-line ttp. In: IEEE Symposium on Security and Privacy, pp. 77–85 (1998)
22. Ateniese, G.: Efficient verifiable encryption (and fair exchange) of digital signatures. In: ACM Conference on Computer and Communications Security, pp. 138–146 (1999)
23. Cachin, C., Camenisch, J.: Optimistic fair secure computation. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 93–111. Springer, Heidelberg (2000)
24. Micali, S.: Simple and fast optimistic protocols for fair electronic exchange. In: PODC, pp. 12–19 (2003)
25. Dodis, Y., Lee, P.J., Yum, D.H.: Optimistic fair exchange in a multi-user setting. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 118–133. Springer, Heidelberg (2007)
26. Chen, L., Kudla, C., Paterson, K.G.: Concurrent signatures. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 287–305. Springer, Heidelberg (2004)
27. Kamara, S., Mohassel, P., Riva, B.: Salus: A system for efficient server-aided multi-party computation. In: ACM Conference on Computer and Communications Security (CCS 2012) (2012)
28. Camenisch, J., Krenn, S., Shoup, V.: A framework for practical universally composable zero-knowledge protocols. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 449–467. Springer, Heidelberg (2011)
29. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 126–144. Springer, Heidelberg (2003)
30. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992)
31. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998)
32. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 103–118. Springer, Heidelberg (1997)
33. Gennaro, R., Rabin, M.O., Rabin, T.: Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In: PODC, pp. 101–111 (1998)
34. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptology 13(1), 143–202 (2000)