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 Science,University of Strathclyde,
changyu.dong@strathclyde.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 and not collude with any of the two parties. The arbiter can resolve disputes blindly 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 intelligence agencies (e.g., USA's CIA and UK's MI5) want to compare their respective databases of terrorist suspects. National privacy laws prevent them from revealing bulk data, however, by treaty, they are allowed to share information on suspects of common interest.*
- *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.*

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 first example above, as the two national intelligence agencies mutually distrust each other, the arbiter has to be an organisation that is not controlled or influenced by one of the states. Under this constraint, the two agencies might have to consider an organisation in a third country, e.g. the French intelligence agency DCRI. However choosing an overseas third party and disclosing private information about citizens to it will infringe the privacy laws. There seems no satisfactory solution to the dilemma. The same happens in the other two examples: an independent auditing service provider could be well qualified to resolve the disputes expect the privacy concerns. We can find more cases in which the two parties may trust the 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 and not collude with any parties.

The paper is organised as follows: in the next section we review the related work in PSI protocols and fairness. Section 3 briefly summaries the building blocks of our protocol. Section 4 gives a high-level overview of the protocol. Section 5 provides the details of a concrete construction. In section 6 we analyse the protocol and prove it is secure. Section 7 concludes the paper and gives an outlook of future works.

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.

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 prematurely after obtaining the output (and before the other party obtains it) [11].

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 [12] 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 [13–15]. 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 [16–22]. 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 [23] 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.

3 Building Blocks

3.1 Homomorphic Encryption

A semantically secure additively homomorphic public key encryption scheme is used as a building block in the protocol. The 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, $E_{pk}(c \cdot m_1) = E_{pk}(m_1)^c$.

3.2 The FNP protocol

Our staring 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. Without loss of generality, we assume the sizes of the two input sets are equal, i.e. $|X| = |Y| = n$. The two parties interact as follows

1. A chooses a key pair (pk, sk) for a homomorphic encryption scheme and makes the public key pk available to B.
2. A defines a polynomial $Q(y) = (y - x_1)(y - x_2)\ldots(y - x_n) = \sum_{i=0}^{n} d_i y^i$, 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. [5]
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 $PK_{(...)}$ and use the notation introduced in [24] to present the protocols in the rest of the paper:

$$PK\{(x, y, ...) : statements \text{ involving } x, y, ...\}$$

In short, the prover is proving the knowledge of $(x, y, ...)$ such that these values satisfy certain *statements*. In our protocol, the statements are knowledge of and relations between discrete logarithms. 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 $\log_g \alpha = \log_h \beta = x$ and it knows the discrete logarithm x.

$$PK\{(x) : \alpha = g^x \wedge \beta = h^x\}$$

3.4 Proxy Re-encryption

Another building block of our protocol is a semantically secure proxy re-encryption scheme. In a proxy re-encryption scheme, one party can delegate decryption to another party, with the help from a proxy. The delegator generates a re-encryption key for the delegatee and gives it to the proxy. Later the proxy can use the re-encryption key to re-encrypt a ciphertext encrypted under the delegator's public key. The re-encryption will transform the ciphertext into a ciphertext encrypted under the delegatee's public key. In the whole process, the proxy learns nothing about the private keys and the plaintext. Thus the definition of semantic security of a proxy re-encryption scheme is twofold in nature: firstly, the ciphertexts are indistinguishable to an external adversary who has access to public keys; secondly, the ciphertexts are indistinguishable also to the proxy who in addition has access to the re-encryption key. A comprehensive study on proxy cryptography and formal definition of its security can be found in [25].

In our protocol, we propose to use proxy re-encryption to allow the semi-trusted arbiter to recover the set intersection computed from the protocol without knowing the intersection or the private input sets of the two parties. To achieve this, we require the proxy re-encryption scheme to be also additively homomorphic and allow efficient zero knowledge proofs. In Section 5.1, we provide a construction of a proxy re-encryption scheme that satisfies the requirements.

[5] 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.

3.5 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 [26]. 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 secret key or a data item. The escrow is essentially an encryption of the escrowed item under the offline arbiter's public key. The escrow is verifiable so that the other party can verify, without decrypting it, the escrowed item satisfies its requirements. We follow this convention and in Section 5.2 we will show how to obtain such a verifiable encryption.

3.6 Perfectly Hiding Commitement

In our protocol, we also use a perfectly hiding commitment scheme [27]. 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 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 binding and hiding 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.

- **Setup**: A homomorphic proxy encryption scheme E, a verifiable encryption scheme $\mathcal{E}$ and a perfectly hiding commitment scheme are chosen. Public parameters are published. The offline arbiter R also generates a key pair for $\mathcal{E}$ and publishes the public key.
- **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. A and B each generates a random key pair for E and sends the public key to the other.
  1. A generates a re-encryption key $rk_{a\to b}$ for B. The re-encryption key is then encrypted using R's public key and the ciphertext is sent to B. A then runs a sub-protocol PK_{rk} to convince B that the re-encryption key is correctly generated and the ciphertext is indeed a valid encryption of the re-encryption key under pk_R.
  2. B commits to its input set using a perfect hiding commitment scheme. B sends all commitments to A.
  3. A generates a polynomial as described in Section 3.2, encrypts all the coefficients and sends the ciphertexts to B. A then runs another protocol PK_{poly} to prove that the polynomial is indeed correctly constructed.
  4. For each element $y_j \in Y$, B evaluates the polynomial using the homomorphic property, returns the result $E_{pk_A}(r_j \cdot Q(y_j) + y_j)$ to A and proves that the evaluation is consistent with regard to a value committed in step 2. A decrypts each ciphertext and checks whether there is an element in X matches $r_j \cdot Q(y_j) + y_j$, if so the element is in $X \cap Y$.

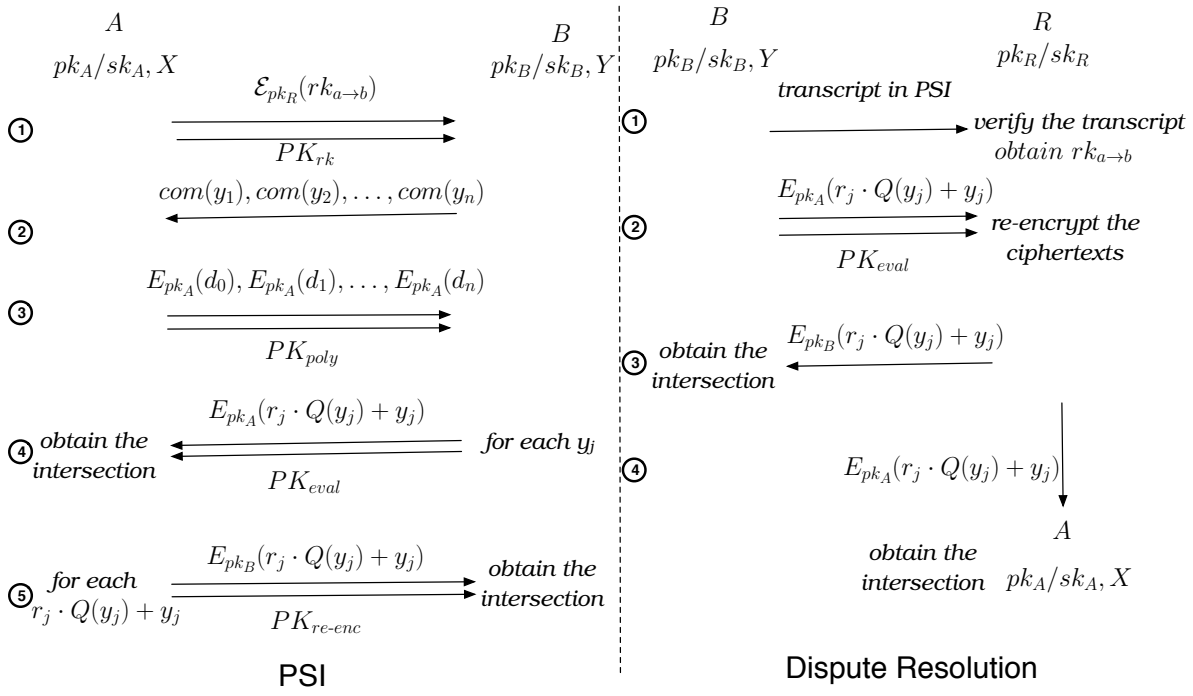

Fig. 1. Overview of the Fair PSI protocol

5. For each $E_{pk_A}(r_j \cdot Q(y_j) + y_j)$, A sends $E_{pk_B}(r_j \cdot Q(y_j) + y_j)$ back to B by performing a re-encryption of $E_{pk_A}(r_j \cdot Q(y_j) + y_j)$, and proves that the value is indeed a re-encrypted ciphertext of the corresponding $E_{pk_A}(r_j \cdot Q(y_j) + y_j)$. B decrypts each received value and checks whether it matches y_j, if so y_j is in $X \cap Y$.

– **Dispute Resolution**: B can raise a dispute with R if A aborts the PSI protocol unfairly

1. B sends the transcript of the PSI protocol execution to R. The transcript contains all messages sent and received in the PSI protocol execution. R verifies it and obtains $\mathcal{E}_{pk_R}(rk_{a\to b})$ from the transcript. R then decrypts it using its private key to obtain the re-encryption key $rk_{a\to b}$.
2. B sends each $E_{pk_A}(r_j \cdot Q(y_j) + y_j)$ to R and proves that the evaluation is consistent with regard to a value committed in step 2 of the PSI protocol.
3. R uses the re-encryption key to transform each $E_{pk_A}(r_j \cdot Q(y_j) + y_j)$ into $E_{pk_B}(r_j \cdot Q(y_j) + y_j)$ which is a ciphertext can be decrypted by B.
4. R also sends a copy of $E_{pk_A}(r_j \cdot Q(y_j) + y_j)$ to A.

Remark 1: In the first step of the PSI protocol, A creates a verifiable escrow of the re-encryption key for B. The intuition is to protect A's privacy against both B and R. In optimistic protocols, all three parties are assumed to be not collude (see [20] for a detailed account). Therefore because the re-encryption key is encrypted under R's public key, B cannot use it to decrypt A's ciphertexts. On the other hand, having the re-encryption key does not enable R to gain any additional information about the ciphertexts.

Remark 2: In the FNP paper [1], the authors proposed strategies to deal with malicious adversaries. Namely, they use cut-and-choose to force A to construct the polynomial correctly and use hash functions to force B to evaluate the polynomial correctly. Cut-and-choose can become inefficient with large input sets. The hash based solution works only in a single-output PSI protocol and will allow a

malicious B to compromise A's privacy in a mutual PSI protocol. Therefore, we replace them with two efficient zero knowledge proofs in step 3 and step 4.

Remark 3: Releasing $E_{pk_B}(r_j \cdot Q(y_j) + y_j)$ to B in the last step of the PSI protocol does not affect A's privacy because as we will see later, B cannot recover $r_j \cdot Q(y_j) + y_j$, but only a group element $Z^{r_j \cdot Q(y_j)+y_j}$. This gives no information to B regarding A's input unless $Q(y_j) = 0$.

Remark 4: To ensure timeliness of dispute resolution and all the messages in the transcript (sent by B in the conflict resolution protocol) are from the same execution of the PSI protocol, each message in the PSI protocol needs to include a session ID and be signed by the sender. We assume a standard format and semantics of the session ID have been agreed by all parities beforehand. Public keys for signature verification need to be agreed and known to all parties beforehand. To simplify presentation, we omit them in the protocol description .

Remark 5: In the last step of the dispute resolution protocol, R sends $E_{pk_A}(r_j \cdot Q(y_j) + y_j)$ to A. This is needed because from the transcript, R cannot tell whether B has sent them to A or not. It is possible that B unfairly aborts the protocol after receiving the coefficients in step 3 and then uses R to recover the result. we add this step to make sure A also receives the output in this case. Thus we do not need a dispute resolution protocol for A because in case of B being unfair, either B does not raise a dispute and then no one gets the output, or B raises a dispute and both get the output.

5 A Concrete Construction

5.1 A Homomorphic Proxy Re-encryption Scheme

At the core of our construction is a semantically secure proxy re-encryption scheme which is also additively homomorphic. This scheme is adapted from [28]. The scheme makes use of bilinear maps. We briefly review the necessary facts about bilinear maps. Let G_1, G_2 and G_T be three cyclic groups of prime order q. A bilinear map $e : G_1 \times G_2 \to G_T$ has properties such that for any $g_1 \in G_1$, $g_2 \in G_2$ and $a, b \in Z_q$, (1) $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$ is efficiently computable (2) $e(g_1, g_2) \neq 1$. To facilitate the construction of the verifiable encryption scheme, we also require the external Diffie-Hellman (XDH) assumption [29–32] to hold in the bilinear map. That is, the Decisional Diffie-Hellman (DDH) assumption must be hard in G_1. The XDH assumption is widely believed to hold in bilinear maps obtained from certain MNT curves [33]. The homomorphic proxy encryption scheme is described as follows:

- **Public parameters**: two groups G_1 and G_2 of prime order q with a bilinear map $e : G_1 \times G_2 \to G_T$, random generators $g_1 \in G_1, g_2 \in G_2$ and $Z = e(g_1, g_2) \in G_T$.
- **Key generation**: A user U's key pair is of the form $pk_u = (Z^{u_1}, g_1^{u_2})$ and $sk_u = (u_1, u_2)$.
- **Re-encryption key generation**: A user A can generate a re-encryption key from another user B's public key $rk_{a\to b} = (g_1^{b_2})^{a_1} = g_1^{a_1 b_2}$.
- **Encryption**: To encrypt a message $m \in Z_q$, A chooses a random r and computes $c = (g_2^r, Z^m Z^{a_1 r})$.
- **Decryption**: To decrypt $c = (\alpha, \beta)$, A computes $\beta / e(g_1, \alpha)^{a_1} = Z^m Z^{a_1 r} / e(g_1, g_2^r)^{a_1} = Z^m Z^{a_1 r} / Z^{a_1 r} = Z^m$ (note that m cannot be recovered, but Z^m serves our purpose).
- **Re-encryption**: $c = (\alpha, \beta)$ can be converted by computing $\alpha' = e(rk_{a\to b}, \alpha) = e(g_1^{a_1 b_2}, g_2^r) = Z^{r a_1 b_2}$ then publishing $(\alpha', \beta) = (Z^{r a_1 b_2}, Z^m Z^{r a_1}) = (Z^{r' b_2}, Z^m Z^{r'})$ where $r' = r a_1$.
- **Re-decryption**: B can decrypt the re-encrypted ciphertext (α', β): $\beta / \alpha'^{1/b_2} = Z^m Z^{r a_1} / Z^{r a_1} = Z^m$.

The scheme [28] is semantically secure under the Decisional Bilinear Diffie-Hellman (DBDH) assumption [34]. The additive homomorphic property is easy to see: (1) for $E_{pk}(m_1) = (\alpha_1, \beta_1)$ and $E_{pk}(m_2) = (\alpha_2, \beta_2)$, $E_{pk}(m_1 + m_2) = (\alpha_1 \cdot \alpha_2, \beta_1 \cdot \beta_2)$ (2) for $E_{pk}(m_1) = (\alpha_1, \beta_1)$ and a constant c, $E_{pk}(c \cdot m_1) = (\alpha_1^c, \beta_1^c)$. We also summarise the changes we made to the original scheme:

- Originally the scheme uses a symmetric bilinear map where $G_1 = G_2$. We use an asymmetric bilinear map and require XDH assumption to hold so that a verifiable encryption scheme can be designed and integrated easily. These changes do not affect the security of the scheme. The security proof of the original scheme still holds with the changes.
- Originally the encryption algorithm encrypts the message m directly as $(g_2^r, m \cdot Z^{a_1 r})$. To make it additively homomorphic, we encrypt m as $(g_2^r, Z^m Z^{a_1 r})$. This makes it impossible to fully decrypt m. But it is sufficient in our protocol because we only need to check that given m', whether $Z^{m'} = Z^m$. For convenience, we still call the relevant algorithms "decryption" although they do not really decrypt to the plaintext m.

5.2 Verifiable Encryption of a Re-encryption Key

In the first step of the PSI protocol, A must encrypt a re-encryption key for B. B must verify that the re-encryption key is generated correctly and the ciphertext is a proper ciphertext of the re-encryption key encrypted under R's public key. The tricky part is that B has to verify the ciphertext without knowing the plaintext, i.e. the re-encryption key.

The encryption scheme used by R is the ElGamal encryption scheme [35] linked to the bilinear groups used in Section 5.1. Namely, R re-uses the group G_1 and the generator g_1, chooses a random secret key $sk_R = k \in Z_q$ and publishes the public key $pk_R = g_1^k$. As we require XDH assumption to hold, then DDH assumption is hard in G_1 and consequently ElGamal in G_1 is semantically secure.

We also designed a zero knowledge proof protocol PK_{rk} to allow B to efficiently verify the ciphertext without knowing the re-encryption key. Recall that the re-encryption key is of format $rk_{a \to b} = g_1^{a_1 b_2}$. After being encrypted using R's public key, the ciphertext $\mathcal{E}_{pk_R}(rk_{a \to b}) = (g_1^r, g_1^{a_1 b_2} g_1^{r \cdot k})$ where r is a random value. Now let $E_{pk_R}(rk_{a \to b}) = (\alpha_1, \alpha_2)$, PK_{rk} is defined as follows:

$$PK\{(a_1, r) : \alpha_1 = g_1^r \wedge \alpha_2 = (g_1^{b_2})^{a_1} (g_1^k)^r \wedge \alpha_3 = Z^{a_1}\}$$

where (a_1, r) is known to A, $g_1^{b_2}$ is part of B's public key, g_1^k is R's public key and Z^{a_1} is part of A's public key. Although the proof involves two groups G_1 and G_T, we don't need to add a range proof as in [36] because $|G_1| = |G_T| = q$. The correctness of the protocol is easy to see: the re-encryption key is correctly constructed because the same exponent a_1 is used in the first part of α_2 and in α_3, the ciphertext is correctly encrypted because the same exponent r is used in the second part of α_2 and in α_1.

5.3 Perfectly Hiding Commitment

We use the Pedersen Commitment Scheme [27] in step 2 of the PSI protocol. The Commitment Scheme is known to be perfectly hiding and computationally binding. The underlying group we use is G_T. For an element in B's input set $y_i \in Y$, B chooses a random $s_i \in Z_q$, and computes $com(y_i) = Z^{y_i} H^{s_i}$, where $H \in G_T$ is random and B does not know $log_Z H$. For efficiency, here we do not require B to prove that the commitments are correctly constructed, this will be done later in PK_{eval}.

5.4 PK_{poly}: Proof of Correct Construction of a Polynomial

In step 3 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 an encrypted coefficient d_i, the ciphertext is $E_{pk_A}(d_i) = (g_2^{r_i}, Z^{d_i} Z^{a_1 r_i}) = (\alpha_{d_i}, \alpha'_{d_i})$. To prove it knows the polynomial, A runs the following protocol for each encrypted coefficient:

$$PK\{(r_i, d_i) : \alpha_{d_i} = g_2^{r_i} \wedge \alpha'_{d_i} = Z^{d_i}(Z^{a_1})^{r_i}\}$$

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. We require A to prove that at least one coefficient is not zero by running

$$PK\{(r_i, r'_i) : \alpha_{d_i} = g_2^{r_i} \wedge \alpha'_{d_i} = (Z^{a_1})^{r'_i} \wedge r_i \neq r'_i\}$$

Intuitively, $r'_i = r_i + d_i/a_1$ and therefore $r_i = r'_i$ iff $d_i = 0$. So by verifying $r_i \neq r'_i$, B can be convinced that $d_i \neq 0$. To prove the inequality of discrete logarithms, we can use the protocol proposed in [26].

5.5 PK_{eval}: Proof of Correct Evaluation of the Polynomial

In step 4 of the PSI protocol and step 2 of the dispute resolution protocol, B must prove that each value sent by it is indeed $E_{pk_A}(r_j \cdot Q(y_j) + y_j)$. Recall that for an encrypted coefficient d_i, the ciphertext is $E_{pk_A}(d_i) = (g_2^r, Z^{d_i} Z^{a_1 r}) = (\alpha_{d_i}, \alpha'_{d_i})$. Then for each term $d_i y_j^i$ of the polynomial, the ciphertext computed using the homomorphic property from $E_{pk_A}(d_i)$ is $E_{pk_A}(d_i y_j^i) = ((\alpha_{d_i})^{y_j^i}, (\alpha'_{d_i})^{y_j^i})$. Similarly, for each $r_j \cdot Q(y_j)$, the ciphertext is

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

B also encrypts y_j by itself, and the ciphertext $E_{pk_A}(y_j) = (g_2^{r'_j}, Z^{y_j} Z^{a_1 r'_j})$. The ciphertext of the whole can be obtained by multiplying the corresponding components of the two:

$$E_{pk_A}(r_j \cdot Q(y_j) + y_j) = ((\prod_{i=0}^{n} (\alpha_{d_i})^{r_j y_j^i}) \cdot g_2^{r'_j}, (\prod_{i=0}^{n} (\alpha'_{d_i})^{r_j y_j^i}) \cdot Z^{y_j} Z^{a_1 r'_j}) = (\alpha, \beta)$$

The proof has two steps. In the first step, B commits to $r_j y_j^i$ for $0 \leq i \leq m$: $com(r_j y_j^i) = Z^{r_j y_j^i} H^{\tilde{s}_i}$. Then starting from $i = 1$, B must prove that the value committed in $com(r_j y_j^i)$ is the product of the values committed in $com(r_j y_j^{i-1})$ and $com(y_j)$, where $com(y_j)$ is committed in step 2 of the PSI protocol. To do this, we use the protocol proposed in [37]:

$$PK\{(\hat{a}, \hat{b}, \hat{r}_1, \hat{r}_2, \hat{r}_3) : \hat{\alpha} = \hat{g}^{\hat{a}} \hat{h}^{\hat{r}_1} \wedge \hat{\beta} = \hat{g}^{\hat{b}} \hat{h}^{\hat{r}_2} \wedge \hat{\gamma} = \hat{g}^{\hat{a}\hat{b}} \hat{h}^{\hat{r}_3}\}$$

which proves a committed value is the product of two other committed values. Now A (or R) has a series of correct commitments of a geometric sequence $com(r_j y_j^i)$ for $0 \leq i \leq m$. In the second step,

B runs the following protocol:

$$PK\{(r_j y_j^i, \tilde{s}_i, r'_j, y_j, s_j) : \alpha = (\prod_{i=0}^{n} (\alpha_{d_i})^{r_j y_j^i}) \cdot g_2^{r'_j}$$
$$\wedge \beta = (\prod_{i=0}^{n} (\alpha'_{d_i})^{r_j y_j^i}) \cdot Z^{y_j} (Z^{a_1})^{r'_j}$$
$$\wedge com(y_j) = Z^{y_j} H^{s_j}$$
$$\bigwedge_{i=0}^{n} com(r_j y_j^i) = Z^{r_j y_j^i} H^{\tilde{s}_i}\}$$

What B proves in the above protocol is that each exponent $r_j y_j^i$ in α and β match the value committed in $com(r_j y_j^i)$, y_j in β matches the value committed in $com(y_j)$, and (α, β) is a proper ciphertext. If B can prove these, then the polynomial evaluation must be correct. [6]

5.6 $PK_{re\text{-}enc}$: Proof of Correct Re-encryption

In the last step of the PSI protocol, A must prove that each value sent is the correct re-encryption of a ciphertext $E_{pK_A}(r_j \cdot Q(y_j) + y_j) = (g_2^{\tilde{r}}, Z^{r_j \cdot Q(y_j) + y_j} Z^{a_1 \tilde{r}}) = (\alpha, \beta)$. After re-encryption, the new ciphertext $E_{pK_B}(r_j \cdot Q(y_j) + y_j) = (Z^{a_1 b_2 \tilde{r}}, \beta) = (\alpha', \beta)$. The re-encryption only changes the first part of the ciphertext, therefore verifying β can be done by simply comparing the second part in the two ciphertexts. B and A can also compute $\gamma = Z^{b_2 \tilde{r}} = e(g_1^{b_2}, \alpha) = e(g_1^{b_2}, g_2^{\tilde{r}})$ easily from public values. The following protocol is then used to prove that α' is correctly constructed:

$$PK\{(a_1) : \alpha' = \gamma^{a_1} \wedge \delta = Z^{a_1}\}$$

where δ is a component of A's public key.

6 Security Analysis

6.1 Definitions

In this section we present the definitions of the terminologies we will use later. Let κ denote a security parameter, we have the following:

Definition 1 (Negligible). *A function f is negligible if for every polynomial $p(\cdot)$ and sufficiently large κ it holds that $f(\kappa) < \frac{1}{p(\kappa)}$.*

Definition 2 (Computationally indistinguishable). *Let $\{X_\kappa\}, \{Y_\kappa\}$ be sequences of distributions with X_κ, Y_κ ranging over $\{0,1\}^{l(\kappa)}$ for some polynomial $l(\kappa)$. $\{X_\kappa\}, \{Y_\kappa\}$ are computationally indistinguishable, written as $X_\kappa \stackrel{c}{\equiv} Y_\kappa$, if for every polynomial-time algorithm D there exist a negligible function $\mu(\cdot)$ such that for every $\kappa \in \mathbb{N}$:*

$$|Pr[D(X_\kappa) = 1] - Pr[D(Y_\kappa) = 1]| \leq \mu(\kappa)$$

[6] Although we present it in two steps, the two steps can be combined into one protocol and makes the protocol more efficient.

An n-party protocol Π is modelled as a collection of n interactive Turing machines (ITM) as in [11]. Each ITM represents the strategy run by a party. A party may be corrupted by an adversary. The adversaries are also modelled as ITMs. The adversaries have an auxiliary input string and therefore are non-uniform. The protocol realises an n-party functionality $f = (f_1, ..., f_n)$, where each f_i maps n inputs to a single output. More specifically, in this paper, the functionality we wish to compute is:

Definition 3 (Functionality $f_\cap$). *Let X and Y be subsets of a predetermined domain $\mathcal{D}$, and $|X| = |Y| = n$, the functionality $f_\cap$ is defined as:*

$$f_\cap(X, Y, \lambda) = \begin{cases} (X \cap Y, X \cap Y, \lambda) \\ (\perp, \perp, \lambda) \end{cases}$$

where $\perp$ denotes a special error symbol and λ denotes the empty string.

6.2 Security Model

We analyse the protocol in the non-colluding model proposed in [38]. The model is based on the ideal/real world paradigm and formalises secure multiparty computation in the presence of non-colluding adversaries. The reason why we use this model is that as pointed out in [20], security of optimistic fair protocols is only achievable when the participants of the protocol do not collude. For example, if R colludes with A, B will not be able to obtain the result from the protocol; or if B colludes with R, they can pool their keys to compromise A's privacy. In the past, the arbiter is modelled as a fully trusted party therefore non-colluding is implicit (we don't consider the case that A and B collude which is meaningless). In our protocol the trust on R is weakened and R is not fully trusted, therefore we need to make non-colluding explicit in the model.

The non-colluding model differs from the standard malicious model for multi-party secure computation in that: (1) in the non-colluding model, a set of *independent adversaries* can be modelled in the sense that the adversaries do not share any state. Where in the standard malicious model there is a monolithic adversary controls all corrupted parties; (2) the notion of emulation is weakened to only require that indistinguishability holds with respect to the honest parties' outputs and a single adversary's view. This partial emulation captures privacy in the presence of non-colluding adversaries: as long as the adversaries do not share information the protocol remains private.

In the non-colluding model, the participants are divided into three disjoint sets: the honest parties (denoted by $\mathcal{H}$), the corrupted but non-colluding parties (denoted by $\mathcal{I}$) and the corrupted and colluding parties (denoted by $\mathcal{C}$). Each corrupted party in $\mathcal{I}$ is controlled by an independent adversary. All corrupted party in $\mathcal{C}$ are controlled by a single adversary separated from these who each controls a corrupted party in $\mathcal{I}$. The behaviour of the adversaries in the non-colluding model can be semi-honest, malicious or non-cooperative. Semi-honest means the adversary follows the protocol but attempts to learn more information that should be kept private. A semi-honest adversary in the model is non-colluding by definition because it does not share state with others and it follows the protocol. Malicious means the adversary can deviate arbitrarily from the protocol. Because a malicious adversary can behave arbitrarily, an independent malicious adversary can collude with other adversaries by sending arbitrary messages. Non-cooperative formalises deviating but non-colluding adversaries. An adversary $\mathcal{A}_i$ is said to be non-cooperative to another adversary $\mathcal{A}_j$ if it may deviate from the protocol but does not intentionally share any useful information with $\mathcal{A}_j$. Formally,

Definition 4 (Non-cooperative adversary). *Let f be a deterministic n-party functionality and Π be an n-party protocol. Furthermore, let $\mathcal{H}$, $\mathcal{I}$ and $\mathcal{C}$ be pairwise disjoint subsets of participants and let $\mathcal{A}$*

be a set of independent PPT adversaries controlling corrupted partied in $\mathcal{I} \cup \mathcal{C}$. For any i, j such that $i \neq j$, we say that adversary $\mathcal{A}_j$ is non-cooperative with respect to $\mathcal{A}_i$ if there exists a ppt simulator $\mathcal{V}_{i,j}$ such that for all $\vec{x} \in (\{0,1\}^*)^n$ and $\vec{z} \in (\{0,1\}^*)^n$, and all $y \in Ran(f_i) \cup \{\perp\}$,

$$mathcalV_{i,j}(y, z_i) \stackrel{c}{\equiv} \{view_{i,j} | output_i = y : \{OUT_l\}_l \leftarrow REAL^{(i)}_{\Pi,\mathcal{H},\mathcal{A},\mathcal{I},\mathcal{C},\vec{z}}(\vec{x}, \kappa)\}$$

whenever $Pr[output_i = y] > 0$. Here $view_{i,j}$ denotes the messages between $\mathcal{A}_i$ and$\mathcal{A}_j$ in the real-world execution and $output_i = y$ is the event that party P_i receives output value y.

The adversarial behaviours of multiple adversaries are described in terms of adversary structures. For example, a four-party protocol is secure against the following adversary structure

$$ADV = \{(\mathcal{A}_1[m], \mathcal{A}_2[h], \mathcal{A}_3[nc_4], \mathcal{A}_4[sh])\}$$

if it is secure when $\mathcal{A}_1$ is malicious, $\mathcal{A}_2$ is honest, $\mathcal{A}_3$ is non-cooperating with $\mathcal{A}_4$, and $\mathcal{A}_4$ is semi-honest.

The real world. Our 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, cryptographic keys and other cryptographic parameters to be used. A has a private input X, B has a private input Y and R has a private key. We use $\mathcal{A}_A, \mathcal{A}_B$ and $\mathcal{A}_R$ to denote the adversary that corrupts A, B and R respectively.

At the end of the execution, the honest parties output whatever prescribed in the protocol, a corrupted party has no output, and an adversary outputs its view. Let $\mathcal{C} = \emptyset$, $\mathcal{I} \subseteq \{A, B, R\}$, $\mathcal{H} = \{A, B, R\}/\mathcal{I}$ and $\mathcal{A}$ be the set of real world adversaries, the partial output with respect to $\mathcal{A}_i$ ($i \in \mathcal{I}$) is defined as

$$REAL^{(i)}_{\Pi,\mathcal{H},\mathcal{A},\mathcal{I},\mathcal{C},\vec{z}}(\vec{x}, \kappa) = \{OUT_j : j \in \mathcal{H}\} \cup OUT_i$$

which consisted of all honest parties' outputs and $\mathcal{A}_i$'s output.

The ideal process. In the ideal process, there is an incorruptible trust party T who will help A and B to compute the set intersection as follows:

- *Setup:* R generates a public/private key pair and gives the public key to A and B.
- *Inputs:* A has an input set X, B has an input set Y and R has no input.
- *Sending inputs to* T: An honest party always sends its input to T. A malicious party may either send $\perp$, its input or some other input (with the same length) to T.
- T *answers* A: If T receives $\perp$ from either A or B, T replies $\perp$ to both parties and the execution terminates here. Else T receives (X', Y') from the A and B, then T replies to A with $X' \cap Y'$.
- A *replies*: After receiving $X' \cap Y'$, A can send $\perp$ or *continue* to T.
- T *answers* B: If T receives *continue* from A, T sends $X' \cap Y'$ to B and the empty string λ to R. If T receives $\perp$, T sends $\perp$ to B. B can output $\perp$ and stop or interact with R.
- B *interacts with* R: B can send *resolve* to R at any point of the time. R can send $\perp$ or *resolve* to T. T sends $X' \cap Y'$ to B if and only if T receives *resolve* from R and has sent $X' \cap Y'$ to A. Otherwise T answers $\perp$ to B.

Similar to the real world execution, let $\mathcal{C} = \emptyset, \mathcal{I} \subseteq \{A, B, R\}$, $\mathcal{H} = \{A, B, R\}/\mathcal{I}$, the partial output of the ideal world execution with respect to an independent simulator $\mathcal{S}_i$ ($i \in \mathcal{I}$) is defined as,

$$IDEAL^{(i)}_{f_\cap,\mathcal{H},\mathcal{S},\mathcal{I},\mathcal{C},\vec{z}}(\vec{x}, \kappa) = \{OUT_j : j \in \mathcal{H}\} \cup OUT_i$$

which consisted of all honest parities output and $\mathcal{S}_i$'s output.

Partial Simulatability. The security of the protocol in the presence of non-colluding adversaries is defined in terms of partial simulatability, i.e. the indistinguishability of the partial output in the real world execution and the ideal world execution. To make it clear that each $\mathcal{S}_i$ depends only on $\mathcal{A}_i$, in the definition we require the existence of a set of transformation Sim_i such that $\mathcal{S}_i = Sim_i(\mathcal{A}_i)$.

Definition 5. *Let $\mathcal{C} = \emptyset, \mathcal{I} \subseteq \{A, B, R\}$, $\mathcal{H} = \{A, B, R\}/\mathcal{I}$, and let ADV be an adversary structure. We say that Π $(\mathcal{I}, ADV)$-securely computes $f_\cap$ if there exists a set $\{Sim_i\}_{i\in\mathcal{I}}$ of PPT transformations such that for all PPT adversaries $\{\mathcal{A}_i\}_{i\in\mathcal{I}}$ that satisfy ADV, for all $\vec{x}, \vec{z}$ and i*

$$REAL^{(i)}_{\Pi,\mathcal{H},\mathcal{A},\mathcal{I},\mathcal{C},\vec{z}}(\vec{x}, \kappa) \stackrel{c}{\equiv} IDEAL^{(i)}_{f_\cap,\mathcal{H},\mathcal{S},\mathcal{I},\mathcal{C},\vec{z}}(\vec{x}, \kappa)$$

where $\mathcal{S}_i = Sim_i(\mathcal{A}_i)$.

6.3 Security Proof

We are now ready to state and prove the security of our protocol. The protocol uses zero knowledge proof protocols $PK_{rk}, PK_{poly}, PK_{eval}, PK_{re\text{-}enc}$ as subprotocols. As they are obtained by using existing protocols which have been proved secure 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 zero knowledge proof protocols is replaced by an ideal call to a trusted party. If the zero knowledge proof protocols are secure, then by the composition theorem [39] 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. *The fair mutual private set intersection protocol securely computes $f_\cap$ with respect to the following adversary structure*

$$ADV = \{(\mathcal{A}_A[nc_B, nc_R], \mathcal{A}_B[sh], \mathcal{A}_R[sh]), (\mathcal{A}_A[sh], \mathcal{A}_B[nc_A, nc_R], \mathcal{A}_R[sh])\}$$

Proof. We consider each case in ADV separately.

Case 1: $\mathcal{A}_A$ is non-cooperative with regard to semi-honest $\mathcal{A}_B$ and $\mathcal{A}_R$.

Case 1.A: We first show partial simulatability of $\mathcal{A}_A$ by constructing a simulator $\mathcal{S}_A = Sim_A(\mathcal{A}_A)$ that uses the adversary $\mathcal{A}_A$ as a subroutine. If $\mathcal{A}_A$ aborts prematurely, $\mathcal{S}_A$ sends $\perp$ to T and outputs whatever $\mathcal{A}_A$ outputs.

1. $\mathcal{S}_A$ is given A's input X, an auxiliary input and invokes $\mathcal{A}_A$ on these inputs, and plays the role of B
2. $\mathcal{S}_A$ generates a random public/private key pair pk_B/sk_B and gives the public key to $\mathcal{A}_A$
3. $\mathcal{S}_A$ receives the encrypted re-encryption key $\mathcal{E}_{pk_R}(rk_{a\to b})$.
4. $\mathcal{S}_A$ receives (a_1, r) from $\mathcal{A}_A$ for the ideal computation of PK_{rk}. If $\mathcal{E}_{pk_R}(rk_{a\to b})$ and (a_1, r) do not satisfy the condition, $\mathcal{S}_A$ sends $\perp$ to T and terminates the execution.
5. $\mathcal{S}_A$ generates a random set Y' such that $|Y'| = n$, and for each $y'_i \in Y'$, generates a commitment $com(y'_i)$. The commitments are sent to $\mathcal{A}_A$.
6. $\mathcal{S}_A$ receives the encrypted coefficients $E_{pk_A}(d_i)$ from $\mathcal{A}_A$. $\mathcal{S}_A$ also receives for all $0 \leq i \leq n, (r_i, d_i)$ for the ideal computation of PK_{poly}, where d_i is a coefficient of the polynomial. If the condition is not satisfied, then $\mathcal{S}_A$ sends $\perp$ to T and terminates the execution.

7. $\mathcal{S}_A$ reconstructs the polynomial and finds the adversary $\mathcal{A}_A$'s input X'. $\mathcal{S}_A$ sends X' to T.
8. $\mathcal{S}_A$ receives $X' \cap Y$ from T. $\mathcal{S}_A$ then constructs Y'' which contains $X' \cap Y$ and some random elements and $|Y''| = n$.
9. For each $y''_j \in Y''$, $\mathcal{S}_A$ computes $E_{pk_A}(r_j \cdot Q(y''_j) + y''_j)$ and sends the ciphertext to $\mathcal{A}_A$. $\mathcal{S}_A$ also emulates the ideal computation of PK_{eval} by sending *accept* to $\mathcal{A}_A$.
10. $\mathcal{S}_A$ receives $E_{pk_B}(r_j \cdot Q(y''_j) + y''_j)$ from $\mathcal{A}_A$. It also receives a_1 for the ideal computation of $PK_{re\text{-}enc}$. If the condition is not satisfied, then $\mathcal{S}_A$ sends $\perp$ to T otherwise it sends $continue$ to T.
11. $\mathcal{S}_A$ outputs whatever $\mathcal{A}_A$ outputs.

We claim that the simulator's output is computationally indistinguishable from the adversary's output in the hybrid execution through a sequence of games:

- $Game_0$: The simulation described above.
- $Game_1$: We construct a simulator $\mathcal{S}_A^1$ that uses Y instead of using Y' in step 5. The only difference between the outputs of $\mathcal{S}_A$ and $\mathcal{S}_A^1$ is the commitments, namely $com(y'_i)$ in $Game_0$ and $com(y_i)$ in this game. Since the commitment scheme is perfectly hiding, the distributions of the two outputs are indistinguishable.
- $Game_2$: We construct a simulator $\mathcal{S}_A^2$ that differs from $\mathcal{S}_A^1$ only in that it uses Y instead of using Y'' in step 8 and 9. Recall that Y'' contains $X' \cap Y$. For each $y''_j \in X' \cap Y$, there is a y_i such that $y_i \in X' \cap Y$ and $y''_j = y_i$. The two ciphertexts $E_{pk_A}(r_j \cdot Q(y''_j) + y''_j)$ and $E_{pk_A}(r_i \cdot Q(y_j) + y_j)$are identical. For all other $y''_j \notin X' \cap Y$ and $y_i \notin X' \cap Y$, $r_j \cdot Q(y''_j) + y''_j$ and $r_i \cdot Q(y_i) + y_i$ are uniformly random and convey no information about y''_j and y_i, so the distributions of the ciphertexts are identical. Therefore the distributions of the two outputs are indistinguishable.
- $Game_3$: The hybrid execution. It differs from $Game_2$ only in that now the adversary interacts with B. However, $\mathcal{S}_A^2$ behaves exactly like B when interacts with the adversary. Therefore the distributions of the two outputs are indistinguishable.

Case 1.B: We then show partial simulatability of the semi-honest adversary $\mathcal{A}_B$ by constructing a simulator $\mathcal{S}_B = Sim_B(\mathcal{A}_B)$ that uses the adversary $\mathcal{A}_B$ as a subroutine.

1. $\mathcal{S}_B$ is given B's input Y, an auxiliary input and invokes $\mathcal{A}_B$ on these inputs.
2. $\mathcal{S}_B$ generates a random key pair pk_A/sk_A and gives the public key to $\mathcal{A}_B$.
3. $\mathcal{S}_B$ receives the public key pk_B from $\mathcal{A}_B$, generates the re-encryption key and encrypts it. The ciphertext is sent to $\mathcal{A}_B$. $\mathcal{S}_B$ also emulates the ideal computation of PK_{rk} by sending *accept* to $\mathcal{A}_B$.
4. $\mathcal{S}_B$ receives the commitments from $\mathcal{A}_B$.
5. $\mathcal{S}_B$ sends Y to T,
   (a) it may receive $X' \cap Y$ from T, in this case
      i. $\mathcal{S}_B$ constructs a set X'' which contains $X' \cap Y$ and some random elements and $|X''| = n$.
      ii. $\mathcal{S}_B$ constructs a polynomial using X''. The coefficients are encrypted and the ciphertexts are sent to $\mathcal{A}_B$. $\mathcal{S}_B$ emulates the ideal computation of PK_{poly} by sending *accept* to $\mathcal{A}_B$.
      iii. $\mathcal{S}_B$ receives $E_{pk_A}(r_j \cdot Q(y_j) + y_j)$ for every $y_j \in Y$. $\mathcal{S}_B$ also receives the input for the ideal computation of PK_{eval}.
      iv. $\mathcal{S}_B$ transforms $E_{pk_A}(r_j \cdot Q(y_j) + y_j)$ into $E_{pk_B}(r_j \cdot Q(y_j) + y_j)$. $\mathcal{S}_B$ also emulates the ideal computation of $PK_{re\text{-}enc}$ by sending *accept* to $\mathcal{A}_B$.
      v. $\mathcal{S}_B$ outputs whatever $\mathcal{A}_B$ outputs.

(b) it may receive $\perp$ from T, it then sends $resolve$ to R and finally receives $X' \cap Y$. In this case:
   i. $\mathcal{S}_B$ constructs a set X'' which contains $X' \cap Y$ and some random elements and $|X''| = n$.
   ii. $\mathcal{S}_B$ constructs a polynomial using X''. The coefficients are encrypted and the ciphertexts are sent to $\mathcal{A}_B$. $\mathcal{S}_B$ emulates the ideal computation of PK_{poly} by sending *accept* to $\mathcal{A}_B$.
   iii. $\mathcal{S}_B$ receives $E_{pk_A}(r_j \cdot Q(y_j) + y_j)$ for every $y_j \in Y$. $\mathcal{S}_B$ also receives the input for the ideal computation of PK_{eval}. $\mathcal{S}_B$ stops the PSI protocol here and engages with $\mathcal{A}_B$ in the dispute resolution protocol.
   iv. $\mathcal{S}_B$ receives the transcript from $\mathcal{A}_B$ and verify it.
   v. $\mathcal{S}_B$ receives each $E_{pk_A}(r_j \cdot Q(y_j) + y_j)$ and input for the ideal computation of PK_{eval}.
   vi. $\mathcal{S}_B$ transforms $E_{pk_A}(r_j \cdot Q(y_j) + y_j)$ into $E_{pk_B}(r_j \cdot Q(y_j) + y_j)$ and sends to $\mathcal{A}_B$.
   vii. $\mathcal{S}_B$ outputs whatever $\mathcal{A}_B$ outputs.

In either case, the simulator's output is computationally indistinguishable from the adversary's output in the hybrid execution if the encryption schemes are semantically secure.
Case 1.R: Next, We show partial simulatability of $\mathcal{A}_R$ by constructing a simulator $\mathcal{S}_R = Sim_R(\mathcal{A}_R)$ that uses the adversary $\mathcal{A}_R$ as a subroutine.

1. $\mathcal{S}_R$ is given an auxiliary input and generates a public/private key pair. $\mathcal{S}_R$ invokes $\mathcal{A}_R$ on the auxiliary input and private key.
2. $\mathcal{S}_R$ generates a transcript with two random sets by playing both A's and B's role internally.
3. $\mathcal{S}_R$ engages with $\mathcal{A}_R$ in the dispute resolution protocol and outputs whatever $\mathcal{A}_R$ outputs.

The simulator's output is computationally indistinguishable from the adversary's output in the hybrid execution if the proxy encryption schemes is semantically secure.
Case 2: $\mathcal{A}_B$ is non-cooperative with regard to semi-honest $\mathcal{A}_A$ and $\mathcal{A}_R$.
Case 2.A: Now $\mathcal{A}_A$ is semi-honest, the simulator is constructed as follows:

1. $\mathcal{S}_A$ is given A's input X, an auxiliary input and invokes $\mathcal{A}_A$ on these inputs, and plays the role of B
2. $\mathcal{S}_A$ generates a random public/private key pair pk_B/sk_B and gives the public key to $\mathcal{A}_A$
3. $\mathcal{S}_A$ receives the encrypted re-encryption key $\mathcal{E}_{pk_R}(rk_{a\to b})$.
4. $\mathcal{S}_A$ receives (a_1, r) from $\mathcal{A}_A$ for the ideal computation of PK_{rk}.
5. $\mathcal{S}_A$ sends X to T and receives $X \cap Y'$ from T
6. $\mathcal{S}_A$ generates a set Y'' such that Y'' contains $X \cap Y'$ and $|Y''| = n$, and for each $y''_i \in Y''$, generates a commitment $com(y''_i)$. The commitments are sent to $\mathcal{A}_A$.
7. $\mathcal{S}_A$ receives the encrypted coefficients $E_{pk_A}(d_i)$ from $\mathcal{A}_A$. $\mathcal{S}_A$ also receives $\forall 0 \le i \le n, (r_i, d_i)$ for the ideal computation of PK_{poly}.
8. For each $y''_j \in Y''$, $\mathcal{S}_A$ computes $E_{pk_A}(r_j \cdot Q(y''_j) + y''_j)$ and sends the ciphertext to $\mathcal{A}_A$. $\mathcal{S}_A$ also emulates the ideal computation of PK_{eval} by sending *accept* to $\mathcal{A}_A$.
9. $\mathcal{S}_A$ receives $E_{pk_B}(r_j \cdot Q(y''_j) + y''_j)$ from $\mathcal{A}_A$.
10. $\mathcal{S}_A$ outputs whatever $\mathcal{A}_A$ outputs.

The simulator's output is computationally indistinguishable from the adversary's output in the hybrid execution because the commitment scheme is perfectly hiding and the encryption schemes are semantically secure.
Case 2.B: The simulator $\mathcal{S}_B$ is constructed on non-cooperative adversary $\mathcal{A}_B$ as follows :

1. $\mathcal{S}_B$ is given B's input Y, an auxiliary input and invokes $\mathcal{A}_B$ on these inputs.

2. $\mathcal{S}_B$ generates a random public/private key pair pk_A/sk_A and gives the public key to $\mathcal{A}_B$.
3. $\mathcal{S}_B$ receives the public key pk_B from $\mathcal{A}_B$, generates the re-encryption key and encrypts it. The ciphertext is sent to $\mathcal{A}_B$. $\mathcal{S}_B$ also emulates the ideal computation of PK_{rk} by sending *accept* to $\mathcal{A}_B$.
4. $\mathcal{S}_B$ receives the commitments from $\mathcal{A}_B$.
5. $\mathcal{S}_B$ constructs a polynomial with every coefficient being 0. The coefficients are encrypted and the ciphertexts are sent to $\mathcal{A}_B$. $\mathcal{S}_B$ emulates the ideal computation of PK_{poly} by sending *accept* to $\mathcal{A}_B$.
6. $\mathcal{S}_B$ receives $E_{pk_A}(r_j \cdot 0 + y'_j)$ for every $y'_j \in Y'$. $\mathcal{S}_B$ also receives the input for the ideal computation of PK_{eval}. If the condition is not satisfied, $\mathcal{S}_B$ sends $\perp$ to T and terminates the execution.
7. $\mathcal{S}_B$ sends $\mathcal{A}_B$'s input Y' extracted in the last step to T and receives $X \cap Y'$ from T.
8. For each $y'_j \in X \cap Y'$, $\mathcal{S}_B$ transforms $E_{pk_A}(r_j \cdot 0 + y'_j)$ into $E_{pk_B}(y'_j)$. For each $y'_j \notin X \cap Y'$, $\mathcal{S}_B$ chooses a random r and transforms $E_{pk_A}(r_j \cdot 0 + y'_j) = (g_2^{\tilde{r}}, Z^{y'_j} Z^{a_1 \tilde{r}})$ into $(Z^{a_1 b_2 \tilde{r} r}, Z^{y'_j} Z^{a_1 \tilde{r}})$ which is effectively $E_{pk_B}(\hat{r}_j)$ where $\hat{r}_j$ is an unknown random element. The ciphertexts are sent to $\mathcal{A}_B$. $\mathcal{S}_B$ also emulates the ideal computation of $PK_{re\text{-}enc}$ by sending *accept* to $\mathcal{A}_B$.
9. $\mathcal{S}_B$ outputs whatever $\mathcal{A}_B$ outputs.

The differences in the simulation and the hybrid execution are that (1) $\mathcal{S}_B$ uses a zero polynomial instead of $Q(\cdot)$ from step 5; (2) the ciphertext encrypts a random element for $y' \notin X \cap Y'$ in step 8. We claim the outputs in the simulation and the hybrid execution are indistinguishable:

- The encrypted coefficients in step 5 are indistinguishable in the two outputs since E is semantically secure. And so are the ciphertexts in step 6.
- In both the simulation and the hybrid execution, for each ciphertext (α, β) received in step 6, a re-encryption in the form of (α', β) is returned. If $y'_j \in X \cap Y'$, it is also true that $y'_j \in X$, therefore it is a root of $Q(\cdot)$ which means $Q(y'_j) = 0$. The ciphertexts $E_{pk_B}(r_j \cdot 0 + y'_j) = E_{pk_B}(r_j \cdot Q(y'_j) + y'_j) = E_{pk_B}(y'_j)$. Therefore the distribution of two corresponding ciphertexts in the two outputs are identical and contain no information of elements which are not in $X \cap Y'$. if $y'_j \notin X \cap Y'$, then in the hybrid execution, $Q(y'_j)$ is random, therefore the encrypted value $Z^{r_j Q(y'_j) + y'_j}$ is random. In the simulation the ciphertext encrypts $Z^{\hat{r}_j}$ which is also random. So the two corresponding ciphertexts in the two outputs are indistinguishable.

Therefore the distributions of the two outputs are indistinguishablel.

If $\mathcal{A}_B$ aborts the protocol prematurely and then invokes the dispute resolution protocol. As we can see, step 2 and step 3 in the dispute resolution protocol are almost identical to step 4 and step 5 in the PSI protocol. In this case, the simulator is constructed as follows:

1. $\mathcal{S}_B$ behaves exactly as described in case 1 until $\mathcal{A}_B$ aborts.
2. $\mathcal{S}_B$ plays R's role and receives the transcript from $\mathcal{A}_B$. If the transcript is not consistent or ends before the end of step 3 of the PSI protocol, it sends $\perp$ to T and terminates the execution.
3. $\mathcal{S}_B$ receives $E_{pk_A}(r_j \cdot 0 + y'_j)$ for every $y'_j \in Y'$. $\mathcal{S}_B$ also receives the input for the ideal computation of PK_{eval}. If the condition is not satisfied, $\mathcal{S}_B$ sends $\perp$ to T and terminates the execution.
4. $\mathcal{S}_B$ sends $\mathcal{A}_B$'s input Y' extracted in the last step to T and receives $X \cap Y'$ from T.
5. For each $y'_j \in X \cap Y'$, $\mathcal{S}_B$ transforms $E_{pk_A}(r_j \cdot 0 + y'_j)$ into $E_{pk_B}(y'_j)$. For each $y'_j \notin X \cap Y'$, $\mathcal{S}_B$ chooses a random r and transforms $E_{pk_A}(r_j \cdot 0 + y'_j) = (g_2^{\tilde{r}}, Z^{y'_j} Z^{a_1 \tilde{r}})$ into $(Z^{a_1 b_2 \tilde{r} r}, Z^{y'_j} Z^{a_1 \tilde{r}})$ which is effectively $E_{pk_B}(\hat{r}_j)$ where $\hat{r}_j$ is an unknown random element. The ciphertexts are sent to $\mathcal{A}_B$.

6. $\mathcal{S}_B$ outputs whatever $\mathcal{A}_B$ outputs.

The analysis is similar to that above, therefore is omitted.

Case 2.R: The simulator works analogously to the one in Case 1.R.

Now the theorem follows from the fact that in both adversarial settings, a partial emulation that is computationally indistinguishable from the hybrid execution can be produced by each simulator.

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 in the presence of non-colluding adversaries.

The protocol presented in this paper makes use of a verifiable escrow of a proxy re-encryption key to allow a third party to resolve disputes. This approach is largely independent of the specific function computed by a secure protocol. A future direction of this work will be to generalise this approach and apply it to other secure computation protocols. We will seek to build a framework which would facilitate protocol design and allow easily plug fairness into appropriate single output protocols.

References

1. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: EUROCRYPT. (2004) 1–19
2. Dachman-Soled, D., Malkin, T., Raykova, M., Yung, M.: Efficient robust private set intersection. In: ACNS. (2009) 125–142
3. Hazay, C., Nissim, K.: Efficient set operations in the presence of malicious adversaries. In: Public Key Cryptography. (2010) 312–331
4. Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In: TCC. (2008) 155–175
5. Jarecki, S., Liu, X.: Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In: TCC. (2009) 577–594
6. Cristofaro, E.D., Tsudik, G.: Practical private set intersection protocols with linear complexity. In: Financial Cryptography. (2010) 143–159
7. Cristofaro, E.D., Kim, J., Tsudik, G.: Linear-complexity private set intersection protocols secure in malicious model. In: ASIACRYPT. (2010) 213–231
8. Kissner, L., Song, D.X.: Privacy-preserving set operations. In: CRYPTO. (2005) 241–257
9. Camenisch, J., Zaverucha, G.M.: Private intersection of certified sets. In: Financial Cryptography. (2009) 108–127
10. Kim, M., Lee, H.T., Cheon, J.H.: Mutual private set intersection with linear complexity. IACR Cryptology ePrint Archive **2011** (2011) 267
11. Goldreich, O.: Foundations of Cryptography: Volume II Basic Applications. Cambridge University Press (2004)
12. Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: STOC. (1986) 364–369
13. Blum, M.: How to exchange (secret) keys. ACM Trans. Comput. Syst. **1**(2) (1983) 175–193
14. Ben-Or, M., Goldreich, O., Micali, S., Rivest, R.L.: A fair protocol for signing contracts. IEEE Transactions on Information Theory **36**(1) (1990) 40–46
15. Pinkas, B.: Fair secure two-party computation. In: EUROCRYPT. (2003) 87–105
16. Asokan, N., Schunter, M., Waidner, M.: Optimistic protocols for fair exchange. In: ACM Conference on Computer and Communications Security. (1997) 7–17
17. Asokan, N., Shoup, V., Waidner, M.: Optimistic fair exchange of digital signatures (extended abstract). In: EUROCRYPT. (1998) 591–606

18. Bao, F., Deng, R.H., Mao, W.: Efficient and practical fair exchange protocols with off-line ttp. In: IEEE Symposium on Security and Privacy. (1998) 77–85
19. Ateniese, G.: Efficient verifiable encryption (and fair exchange) of digital signatures. In: ACM Conference on Computer and Communications Security. (1999) 138–146
20. Cachin, C., Camenisch, J.: Optimistic fair secure computation. In: CRYPTO. (2000) 93–111
21. Micali, S.: Simple and fast optimistic protocols for fair electronic exchange. In: PODC. (2003) 12–19
22. Dodis, Y., Lee, P.J., Yum, D.H.: Optimistic fair exchange in a multi-user setting. In: Public Key Cryptography. (2007) 118–133
23. Chen, L., Kudla, C., Paterson, K.G.: Concurrent signatures. In: EUROCRYPT. (2004) 287–305
24. Camenisch, J., Stadler, M.: Proof systems for general statements about discrete logarithms. Technical Report 260, Institute for Theoretical Computer Science, ETH Zurich (March 1997)
25. Ivan, A.A., Dodis, Y.: Proxy cryptography revisited. In: NDSS. (2003)
26. Camenisch, J., Shoup, V.: Practical verifiable encryption and decryption of discrete logarithms. In: CRYPTO. (2003) 126–144
27. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO. (1991) 129–140
28. Ateniese, G., Fu, K., Green, M., Hohenberger, S.: Improved proxy re-encryption schemes with applications to secure distributed storage. ACM Trans. Inf. Syst. Secur. **9**(1) (2006) 1–30
29. Ballard, L., Green, M., de Medeiros, B., Monrose, F.: Correlation-resistant storage via keyword-searchable encryption. IACR Cryptology ePrint Archive **2005** (2005) 417
30. Boneh, D., Boyen, X., Shacham, H.: Short group signatures. IACR Cryptology ePrint Archive **2004** (2004) 174
31. Camenisch, J., Hohenberger, S., Lysyanskaya, A.: Compact e-cash. In: EUROCRYPT. (2005) 302–321
32. Ateniese, G., Camenisch, J., de Medeiros, B.: Untraceable rfid tags via insubvertible encryption. In: ACM Conference on Computer and Communications Security. (2005) 92–101
33. Miyaji, Nakabayashi, Takano: New explicit conditions of elliptic curve traces for FR-reduction. TIEICE: IEICE Transactions on Communications/Electronics/Information and Systems (2001)
34. Boneh, D., Franklin, M.: Identity-based encryption from the Weil pairing. SIAM J. of Computing **32**(3) (2003) 586–615 extended abstract in Crypto'01.
35. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: CRYPTO. (1984) 10–18
36. Camenisch, J., Stadler, M.: Efficient group signature schemes for large groups (extended abstract). In: CRYPTO. (1997) 410–424
37. Gennaro, R., Rabin, M.O., Rabin, T.: Simplified vss and fact-track multiparty computations with applications to threshold cryptography. In: PODC. (1998) 101–111
38. Kamara, S., Mohassel, P., Raykova, M.: Outsourcing multi-party computation. IACR Cryptology ePrint Archive **2011** (2011) 272
39. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptology **13**(1) (2000) 143–202