Practical Private Set Intersection Protocols

Emiliano De Cristofaro and Gene Tsudik

Computer Science Department, University of California, Irvine

Abstract. The constantly increasing dependence on anytime-anywhere availability of data and the commensurately increasing fear of losing privacy motivate the need for privacy-preserving techniques. One interesting and common problem occurs when two parties need to privately compute an intersection of their respective sets of data. In doing so, one or both parties must obtain the intersection (if one exists), while neither should learn anything about other set elements. Although prior work has yielded a number of effective and elegant Private Set Intersection (PSI) techniques, the quest for efficiency is still underway. This paper explores some PSI variations and constructs several secure protocols that are appreciably more efficient than the state-of-the-art.

1 Introduction

In today's increasingly electronic world, privacy is an elusive and precious commodity. There are many realistic modern scenarios where private data must be shared among mutually suspicious entities. Consider the following examples:

1. 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.
2. Two national law enforcement bodies (e.g., USA's FBI 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.
3. 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 houses.
4. Federal tax authority wants to learn whether any suspected tax evaders have any accounts with a certain foreign bank and, if so, obtain their account records and details. The bank's domicile forbids wholesale disclosure of account holders and the tax authority clearly can not reveal its list of suspects.
5. Department of homeland security (DHS) wants to check its list of terrorist suspects against the passenger manifest of a flight operated by a foreign air carrier. Neither party is willing to reveal its information, however, if there is a (non-empty) intersection, DHS will not give the flight permission to land.

These examples motivate the need for privacy-preserving set operations, in particular, set intersection protocols. Such protocols are especially useful whenever one or both parties who do not fully trust each other must compute an intersection of their respective sets (or some function thereof). As discussed in Section 4 below, prior work has yielded a number of interesting techniques. As usually happens in applied cryptography, the next step (and the current quest) is to improve efficiency. To this end, this paper's main goal is to consider several flavors of Private Set Intersection (PSI) and construct provably secure protocols that are more efficient than the state-of-the-art.

The rest of the paper is structured as follows. Section 2 discusses Private Set Intersection (PSI) with its several different flavors. After summarizing our work in Section 3, we overview prior PSI techniques in Section 4. Then, we present and evaluate our protocols in Section 5 and conclude in Section 6. (Appendices A and B contain some proof sketches.)

2 PSI Flavors

Generally speaking, Private Set Intersection (PSI) is a cryptographic protocol that involves two players, Alice and Bob, each with a private set. Their goal is to compute the intersection of their respective sets, such that minimal information is revealed in the process. In other words, Alice and Bob should learn the elements (if any) common to both sets and nothing (or as little as possible) else. This can be a mutual process where, ideally, neither party has any advantage over the other. Examples 1-3 above require mutual PSI. In a one-way version of PSI, Alice learns the intersection the two sets, however, Bob learns (close to) nothing. Examples 4 and 5 correspond to one-way PSI.

Since mutual PSI can be easily obtained by two instantiations of one-way PSI, in the remainder of this paper we focus on the latter. Hereafter, the term PSI denotes the one-way version and, instead of proverbial Alice and Bob, we use client (C) and server (S) to refer to the protocol participants.

One natural extension is what we call PSI with Data Transfer or PSI-DT. In this setting, one or both parties have data associated with each element in the set e.g., a database record. In PSI-DT, data associated with each element in the intersection must be transferred to one or both parties, depending whether mutual or one-way version of PSI is used. Example 4 corresponds to PSI-DT. It is also easy to see that PSI-DT is quite appealing in terms of actual database (rather than plain set) applications.

Another twist on PSI is the authorized version – APSI – where each element in the client set must be authorized (signed) by some recognized and mutually trusted authority. This requirement could be applicable to Examples 2 and 4. In the former, one or both agencies might want to make sure that names of terrorist suspects held by its counterpart are duly authorized by the country's top judiciary. In example 4, the bank could demand that each suspected tax cheat be pre-vetted by some international body, e.g., Interpol. In general, the

main difference between PSI and APSI is that, in the former, the inputs of one or both parties might be arbitrarily chosen, i.e., frivolous.

Clearly, other more interesting or more exotic variations are possible, e.g., the notion of *group PSI* with its many types of possible outputs. However, we limit the scope of this paper to the PSI flavors described above.

3 Roadmap

In contrast to prior work, we do not start with constructing PSI protocols and piling on extra features later. Instead, somewhat counter-intuitively, we begin with prior work on a specific type of protocols – called Privacy-preserving Policy-based Information Transfer (PPIT) – that provide APSI-DT (one-way authorized private set intersection with data transfer) for the case where one party has a set of size one. PPIT matches a typical database query scenario where client has a single keyword or a record identifier and server has a database.

We start by seeing how some previously-proposed PPIT protocols can be trivially extended into inefficient PSI and APSI protocols, with and without data transfer. We then construct several efficient (and less trivial) provably secure PSI and APSI protocols that incur **linear** computation and bandwidth overhead. Concretely, this work makes several contributions:

1. We evaluate and compare existing PSI and APSI protocols in terms of efficiency (computation and bandwidth), security model (random oracle vs standard) and adversary type (honest-but-curious vs malicious).
2. We investigate whether APSI protocols can yield (efficient) PSI counterparts.
3. We present an APSI protocol and its PSI dual that are more efficient than prior work.
4. We construct another PSI protocol geared for scenarios where the server can perform some pre-computation and/or the client is computationally weak.

4 Prior Work

This section overviews relevant prior results, which fall into several categories: (1) PSI protocols, (2) OPRF constructs, and (3) APSI variations. Also, we note that most PSI variations can be realized via general secure multiparty techniques. However, it is usually far more efficient to have dedicated protocols; which is the direction we pursue in this paper.

4.1 PSI Protocols

The seminal work by Freedman, et al. (FNP) [12] addressed the problem of private set intersection by means of oblivious polynomial evaluation. The approach works as follows: let pk_C be the public key of a client C for any additively homomorphic cryptosystem (e.g. Paillier [19]). C represents its private set $\mathcal{C} = (c_1, \cdots, c_v)$ as the roots of a degree v polynomial, $f = \prod_{i=1}^{v}(t - c_i) = \sum_{i=0}^{k} \alpha_i t^i$,

encrypts the coefficients with pk_C, and sends them to server S (whose private set is denoted with $\mathcal{S}$). S evaluates f at each $s_i \in \mathcal{S}$ homomorphically. Note that $f(s_i) = 0$ if and only if $s_i \in \mathcal{C} \cap \mathcal{S}$. S, for each $s_j \in \mathcal{S} = (s_1, \cdots, s_w)$ returns $u_j = E(r_j f(s_j) + s_j)$ to C (where r_j is chosen at random). If $s_j \in \mathcal{C} \cap \mathcal{S}$ then C learns s_j upon decrypting. If $s_j \notin \mathcal{C} \cap \mathcal{S}$ then u_j decrypts to a random value. Therefore, the number of server's operations is related to the evaluation of client's encrypted polynomial, with v coefficients, on w points in $\mathcal{S}$. Using Horner's rule (and assuming Paillier encryption) this would take $O(vw)$ of m-bit mod 2048-bit exponentiations, where m is the number of bits needed for representing each entry. On the other hand, the number of client operations is $O(w)$, i.e., 1024-bit exponentiations mod 2048 bits. However, certain optimizations can be applied to reduce the total number of server's exponentiations to $O(w \log(\log(v)))$.

The protocol presented above is secure against an honest-but-curious adversary in the standard model. However, FNP [12] also propose protocols to address the case of either C or S being malicious, although in the random oracle model. Such protocol, using standard commitments and zero-knowledge proofs of arithmetic relationships between committed values, requires that each m-bit exponentiation on the server would have to be replaced by at least two 160-bit exponentiations as pointed out in [15].

Kissner and Song (KS) [17] propose improved protocols for several private set operations. Let R be a ring, $R[t]$ be the polynomials with coefficients in R, and D – the domain of the elements in C's set ($\mathcal{C}$) and S's set ($\mathcal{S}$), and assume $|D|/|R|$ is negligible. They consider two random polynomials s and t chosen in $R[t]$, and two polynomials f and g representing the sets $\mathcal{S}$ and $\mathcal{C}$ respectively, and all such polynomials have degree k, assuming $|\mathcal{C}| = |\mathcal{S}| = k$. As proven in [17], $sf + tg = gcd(f, g) \cdot u$, where u is a random element of $R[t]$. If the domain D of $\mathcal{S}$ and $\mathcal{C}$ is very small compared to R, there is a negligible probability that u contains an element from D as a root. Hence, the only elements of D that are roots of $sf + tg$ with non-negligible probability are those in $gcd(f, g) = \mathcal{S} \cap \mathcal{C}$. Therefore, if two parties jointly compute the encryptions of $sf + tg$, they learn the intersection upon decryption. The protocol is proven secure in the malicious model. However, it involves quadratic computation and bandwidth overhead – $O(k^2)$.

4.2 OPRF-based Protocols

Some other constructs rely on so-called Oblivious Pseudo-Random Functions (OPRFs) introduced in [11]. An OPRF is a two-party protocol (between a sender and a receiver) that securely computes a pseudorandom function $f_k(\cdot)$ on key k contributed by the sender and input x contributed by the receiver, such that the former learns nothing from the interaction and the latter learns only the value $f_k(x)$.

Hazay and Lindell [14] were the first to show how to use OPRFs to construct PSI protocols. The protocol of [14] is secure in the standard model in the presence of a malicious server and an honest-but-curious client. It has been since improved

by Jarecki and Liu [15], who came up with a protocol secure in the standard model against both malicious parties, based on the Decisional q-Diffie-Hellman Inversion assumption, in the Common Reference String (CRS) model, where a safe RSA modulus must be pre-generated by a trusted party.

OPRF-based PSI protocols work as follows: Server S holds a secret random key k. For each $s_j \in \mathcal{S}$ (of size w), S precomputes $u_j = f_k(s_j)$, and publishes (sends to client) the set $\mathcal{U} = \{u_1, \cdots, u_w\}$. Then, C and S engage in an OPRF computation of $f_k(c_i)$ for each $c_i \in \mathcal{C}$ (of size v), such that S learns nothing about $\mathcal{C}$ (except the size) and C learns $f_k(c_i)$. Finally, C learns that $c_i \in \mathcal{C} \cap \mathcal{S}$ if and only if $f_k(c_i) \in \mathcal{U}$.

We now briefly describe the OPRF approach of [15], and show how to speed up the computation based on a recent result of Belenkiy, et al. [1]. The actual OPRF is $f_k(x) = g^{\frac{1}{k+x}}$. During setup, the server picks two primes p, q (e.g., 1024 and 160 bits, respectively), where q is a large divisor of $p-1$, and runs the setup algorithm of the additively homomorphic encryption scheme of Camenisch and Shoup [7] (CS) to obtain a safe RSA modulus n (e.g., 1024 bits), a generator g of $\mathbb{Z}_n^*$, and a pair (pk, sk). Then, S picks a random key $k \in \mathbb{Z}_q$ and sends to client: $e_k = Enc_{pk}(k)$. C picks two random values $a_1 \in \mathbb{Z}_q$ and $a_2 \in 2^{q\tau}$ (where τ is the security parameter), then for each $c_i \in \mathcal{C}$, client computes (and sends) $c_i' = [Enc_{pk}(c_i \cdot e_k)^{a_1} \cdot Enc_{pk}(a_2 \cdot p) = Enc_{pk}(a_1(c_i + k) + a_2 p)]$. Then, S obtains $b_i = Dec_{sk}(c_i')$ and sends each g^{1/b_i} to C. Finally, C recovers $f_k(c_i) = (g^{1/b_i})^{a_1}$.

In terms of computational complexity, assuming the honest-but-curious model, S has to perform $O(w)$ PRF evaluations with w inputs, i.e., $O(w)$ modular exponentiations of m-bit exponents mod n^2 (2048 bits), where m is the number of bits needed to represent each entry. However, we focus on operations that cannot be precomputed, hence, OPRF evaluations of client's inputs. Also, we do not take into account more efficient operations, such as modular multiplications and exponentiations with short exponents. The client computes $O(v)$ CS encryptions (i.e., $O(v)$ m-bit exponentiations mod 2048 bits, plus $O(v)$ 1024-bit exponentiations mod 1024 bits). Whereas, S computes (online) $O(v)$ CS decryptions, (i.e. $O(v)$ 1024-bit exponentiations mod 2048 bits).

Finally, we note that, as acknowledged in [15], complexity in the malicious model grows by a factor of 2. As a result, since [15] achieves linear computation overhead, it improves the efficiency of FNP [12] in both honest-but-curious and malicious models.

4.3 APSI Protocols

We now briefly review related work in Authorized Private Set Intersection protocols.

PPIT. Recently, a new PSI-related concept was introduced, called *Privacy-preserving Policy-based Information Transfer* (PPIT) [10]. It is targeted for scenarios where a client holding an authorization (i.e., a signature by a trusted

authority) on some identifier needs to retrieve information matching that identifier from a server, such that: (1) the client only gets the information it is entitled to, and (2) the server knows that the client is duly authorized to obtain information but does not learn what information is retrieved. Besides requiring the client to be authorized, PPIT is focused on the situation where the client holds a single identifier, i..e, PPIT offers APSI where Alice (client) has a set of size one.

[10] gives three PPIT protocols, based respectively on: RSA [20], Schnorr [21], and Identity-based Encryption (IBE) [4]. Below we only discuss RSA-PPIT since it serves as a starting point for the work in this paper.

As shown in [10], it is easy to extend PPIT to support the case of the client holding multiple authorizations and thus obtain a full-blown APSI protocol. The result is also secure in the random oracle model for honest-but-curious parties. However, the complexity (both bandwidth and computation) becomes quadratic.

In RSA-PPIT, client's authorizations are essentially RSA signatures on a set of identifiers $\{c_i | 1 \leq \ i \leq \ v\}$, i.e. $\sigma_i = (hc_i)^d \bmod n$, where n is the CA's modulus, d is its secret key and $hc_i = H(c_i)$ for some suitable hash function $H()$. First, the client blinds and commits to its inputs by sending to the server $\mu_i = \sigma_i^2 \cdot g^{r_i} \bmod n$, where g is a generator of QR_n and r_i is a random number in $\mathbb{Z}_{N/4}$). The server, for each received μ and for each of its inputs s_j, computes and sends $h_{ij} = H'(\mu_i^{ez} \cdot H(s_j)^{-2z})$ and the value $R = g^{ez}$, where e is the CA's public exponent and z is a random value in $\mathbb{Z}_{N/4}$. The client learns whether $c_i \in \mathcal{C} \cap \mathcal{S}$ iff $h_{ij} = H(R^{r_i})$. RSA-PPIT intrinsically provides data transfer: if h_{ij} is hashed with an additional hash function and used to encrypt the data associated to s_j. The client's computation overhead is relatively low stemming mainly from the blinding operations; it amounts to $O(v)$ modular exponentiations. Whereas, the server incurs quadratic –$O(vw)$ – computation overhead. (Bandwidth overhead is quadratic as well). However, [10] shows that, the number of modular exponentiations performed by the server can be reduced to $O(v + w)$, and even to $O(v)$ by means of precomputation and randomness reuse, although the number of modular multiplications remains quadratic, as does bandwidth overhead.

Authorized Private Search. Another recent result [6] addressed a problem similar to PPIT, by means of an IBE-based technique inspired by Public-Key Encryption with Keyword Search (PEKS) [3]. It enhances PEKS by introducing a *Committed Blind Anonymous* IBE scheme. With such a scheme, the client privately obtains trapdoors from the CA, hence not revealing anything about its inputs to the CA (unlike PPIT). Nevertheless, the client commits to the inputs, so that the CA can later ask the client to prove statements on them. Although this scheme does not require the Random Oracle Model, its efficiency is much lower than PPIT. First, whereas IBE-PPIT uses Boneh-Franklin IBE [4], the underlying IBE scheme is a modification of Boyen-Waters (BW) IBE [5] which is less time and space efficient. The server has to compute $O(w)$ (BW) encryptions (each requiring 6 exponentiations and a representation of 6 group elements). Furthermore, the client has to test each $O(w)$ PEKS against its $O(v)$

trapdoors, hence performing $O(vw)$ (BW) decryptions (each requiring 5 bilinear map operations).

Certified Sets. Finally, Camenisch and Zaverucha [8] introduce the notion of *Certified Sets*. This allows a trusted third party to ensure that all protocol inputs are valid and bound to each protocol participant. The proposed protocol builds upon oblivious polynomial evaluation and achieves asymptotic computation (quadratic) and communication overhead similar to that of FNP [12] and KS [17], i.e. $O(vw)$ exponentiations. Also, protocols in [8] provide mutual set intersection and it is not clear how to convert it to one-way PSI.

5 Towards Efficient PSI and APSI Protocols

In this section, we explore the design of efficient PSI and APSI. Before proceeding to the actual protocols, we provide some definitions and assumptions.

5.1 Preliminaries

Recall that PSI involves two parties: client and server.

Definition 1. *PSI consists of two algorithms:* $\{Setup, Interaction\}$*, where: Setup: a process wherein all global/public parameters are selected. Interaction: a protocol between client and server that results in the client obtaining the intersection of two sets.*

APSI involves three parties: client, server and (off-line) CA.

Definition 2. *APSI is a tuple of three algorithms:* $\{Setup, Authorize, Interaction\}$*, where: Setup: a process wherein all global/public parameters are selected. Authorize : a protocol between client and CA resulting in client committing to its input set and CA issuing authorizations (signatures), one for each element of the set. Interaction: a protocol between client and server that results in the client obtaining the intersection of two sets.*

The following assumptions are made throughout:

1. In APSI, CA does not behave maliciously.
2. In PSI, both client and server are honest-but-curious.
3. In APSI, server is honest-but-curious, however, client might not have authorizations for all elements in its set.

5.2 Security Properties

We now informally describe security requirements for PSI and APSI.
Correctness. a PSI scheme is *correct* if, at the end of $Interaction$, client outputs the exact (possibly empty) intersection of the two respective sets.

Server Privacy. Informally, a PSI scheme is server-private if the client learns no information (except the upper bound on size) about the subset of elements on the server that are NOT in the intersection of their respective sets.
Client Privacy. Informally, client privacy (in either PSI or APSI) means that no information is leaked about client's set elements to a malicious server, except the upper bound on the client's set size.
Client Unlinkability (optional). Informally, client unlinkability means that a malicious server cannot tell if any two instances of *Interaction* are related, i.e., executed on the same inputs by the client.
Server Unlinkability (optional). Informally, server unlinkability means that a malicious client cannot tell if any two instances of *Interaction* are related, i.e., executed on the same inputs by the server.
For APSI, the **Correctness** and **Server Privacy** requirements are amended as follows:
Correctness (APSI). an APSI scheme is *correct* if, at the end of *Interaction*, client outputs the exact (possibly empty) intersection of the two respective sets and each element in that intersection has been previously authorized by CA via *Authorize*.
Server Privacy (APSI). Informally, a PSI scheme is server-private if the client learns no information (except the upper bound on size) about the subset of elements on the server that are NOT in the intersection of their respective sets (where client's set contains only authorizations obtained via *Authorize*).

5.3 Baseline: APSI from RSA-PPIT

The starting point for our design is an APSI protocol derived from RSA-PPIT [10]. This protocol is only vaguely sketched out in [10]; since our new protocols are loosely based on it, we specify it in Figure 1. Actually, the protocol in [10] is APSI-DT; however, for ease of illustration we omit the data transfer component. Also, all PSI and APSI protocols in this paper include only the *Interaction* component; *Setup* and *Authorize* (if applicable) are both intuitive and trivial. Our notation is reflected in Table 1.

$a \leftarrow A$	variable a is chosen uniformly at random from set A
τ	security parameter
n, e, d	RSA modulus, public and private exponents
g	group generator; exact group depends on context
p, q	large primes, where $q = k(p-1)$ for some integer k
$H()$	full-domain hash function
$H'()$	regular cryptographic hash function: $H' : \{0,1\}^* \rightarrow \{0,1\}^\tau$
$\mathcal{C}, \mathcal{S}$	client's and server's sets, respectively
v, w	sizes of $\mathcal{C}$ and $\mathcal{S}$, respectively
$i \in [1, v],\ j \in [1, w]$	indices of elements of $\mathcal{C}$ and $\mathcal{S}$, respectively
c_i, s_j	i-th and j-th elements of $\mathcal{C}$ and $\mathcal{S}$, respectively
hc_i, hs_j	$H(c_i)$ and $H(s_j)$, respectively
$R_{c:i}, R_{s:j}$	i-th and j-th random value generated by client and server, respectively

Table 1. Notation

It is easy to see that this protocol is correct, since: for any (σ_i, c_i) held by the client and s_j held by the server, if: (1) σ_i is a genuine CA's signature on c_i, and (2) $c_i = s_j$ (hence, $hc_i = hs_j$):

$$\begin{aligned} K_{c:i} &= (Z)^{R_{c:i}} = g^{eR_s \cdot R_{c:i}} \\ K_{s:i,j} &= (\mu_i)^{eR_s} \cdot (hs_j)^{-2R_s} = (\sigma_i^2 \cdot g^{R_{c:i}})^{eR_s} \cdot (hs_j)^{-2R_s} \\ &= ((hc_i)^{d2} \cdot g^{R_{c:i}})^{eR_s} \cdot (hs_j)^{-2R_s} = hc_i^{2R_s} \cdot g^{eR_s \cdot R_{c:i}} \cdot hs_j^{-2R_s} = g^{eR_s \cdot R_{c:i}} \end{aligned}$$

However, the protocol in Figure 1, as mentioned in Section 4.3, incurs quadratic computation overhead by the server and quadratic bandwidth. It is possible to reduce the number of on-line exponentiations on the server to $O(v)$ by precomputing all values $(hs_j)^{-2R_s}$ in Step 3. Nonetheless, the number of multiplications needed to compute all $K_{s:i,j}$ would still remain quadratic, i.e., $O(vw)$, as would the bandwidth.

- Common input: $n, g, e, H(), H'()$
- Client's input: $\mathcal{C} = \{\sigma_1, \cdots, \sigma_v\}$, where: $\sigma_i = (hc_i)^d \bmod n$, and $hc_i = H(c_i)$
- Server's input: $\mathcal{S} = \{hs_1, \cdots, hs_w\}$, where: $hs_i = H(s_i)$

1. Client:
   - $\forall i, R_{c:i} \leftarrow \mathbb{Z}_{n/4}$,
   - $\forall i, \mu_i = \sigma_i^2 \cdot g^{R_{c:i}} \bmod n$
2. Client $\longrightarrow$ Server: $\boxed{\{\mu_1, .., \mu_v\}}$
3. Server:
   - $R_s \leftarrow \mathbb{Z}_{n/4}$ and $Z = g^{eR_s} \bmod n$
   - $\forall i, \forall j$, compute: $K_{s:i,j} = (\mu_i)^{eR_s} \cdot (hs_j)^{-2R_s} \bmod n$, and $t_{i,j} = H'(K_{s:i,j})$
4. Server $\longrightarrow$ Client: $\boxed{Z,\ \{t_{1,1}, .., t_{v,w}\}}$
5. Client:
   - $\forall i, K_{c:i} = (Z)^{R_{c:i}} \bmod n$, and $t'_i = H'(K_{c:i})$
   - OUTPUT: $\{t'_1, .., t'_v\} \cap \{t_{1,1}, .., t_{v,w}\}$

Fig. 1. APSI Protocol derived from RSA-PPIT

5.4 APSI with Linear Costs

Although the trivial realization of APSI obtained from RSA-PPIT is relatively inefficient, we now show how to use it to derive an efficient protocol, shown in Figure 2. This protocol incurs **linear** computation (for both parties) and bandwidth complexity. Specifically, the client performs $O(v)$ exponentiations and the server – $O(v + w)$. Communication is dominated by server's reply in Step 4 – $O(v + w)$. To see that the protocol is correct, observe that, for any (σ_i, c_i) held by the client and s_j held by the server, if: (1) σ_i is a genuine CA's signature on c_i, and (2) $c_i = s_j$, hence, $hc_i = hs_j$:

$$\begin{aligned} K_{c:i} &= y'_i \cdot Z^{R_c} \cdot Z^{-R_{c:i}} = (PCH_i^*)^{eR_s} \cdot g^{eR_sR_c} \cdot g^{-eR_sR_{c:i}} \\ &= (PCH_i)^{R_s} \cdot g^{eR_{c:i}R_s} \cdot g^{eR_cR_s} \cdot g^{-eR_sR_{c:i}} = (PCH_i)^{R_s} \cdot g^{eR_cR_s} \\ K_{s:j} &= (X^e/hs_j)^{R_s} = [(PCH^* \cdot g^{R_c})^e/hs_j]^{R_s} \\ &= (PCH/hs_j \cdot g^{eR_c})^{R_s} = (PCH_i)^{R_s} \cdot g^{eR_cR_s} \end{aligned}$$

Note that: $(PCH^*)^e = \prod_{i=1}^{v}(\sigma_i^e) = PCH$ and: $(PCH_i^*)^e = PCH_i$

- Common input: $n, g, e, H(), H'()$
- Client's input: $\mathcal{C} = \{\sigma_1, \cdots, \sigma_v\}$, where: $\sigma_i = (hc_i)^d \bmod n$, and $hc_i = H(c_i)$
- Server's input: $\mathcal{S} = \{hs_1, \cdots, hs_w\}$, where: $hs_i = H(s_i)$

1. Client:
   - $PCH = \prod_{i=1}^{v} hc_i$ **and** $PCH^* = \prod_{i=1}^{v}(\sigma_i) = \prod_{i=1}^{v}({hc_i}^d)$
   - $R_c \leftarrow \mathbb{Z}_n^*$ and $X = PCH^* \cdot g^{R_c}$
   - $\forall i$, $PCH_i^* = PCH^*/\sigma_i$, and $R_{c:i} \leftarrow \mathbb{Z}_n^*$, $y_i = PCH_i^* \cdot g^{R_{c:i}}$
2. Client $\longrightarrow$ Server: $\boxed{X, \{y_1, .., y_v\}}$
3. Server:
   - $R_s \leftarrow \mathbb{Z}_n^*$ and $Z = g^{eR_s} \bmod n$
   - $\forall\ j$, compute: $K_{s:j} = (X^e/hs_j)^{R_s}$, and $t_j = H'(K_{s:j})$
   - $\forall\ i$, compute: $y_i' = (y_i)^{eR_s}$
4. Server $\longrightarrow$ Client: $\boxed{Z,\ \{y_1', ...y_v'\},\ \{t_1, .., t_w\}}$
5. Client:
   - $\forall i, K_{c:i} = y_i' \cdot Z^{R_c} \cdot Z^{-R_{c:i}}$, and $t_i' = H'(K_{c:i})$
   - OUTPUT: $\{t_1', .., t_v'\} \cap \{t_1, .., t_w\}$

Fig. 2. APSI Protocol with linear complexity

We claim that the APSI Protocol in Fig. 2 is: (1) Server-Private (APSI), (2) Client-Private, (3) Client-Unlinkable and (4) Server-Unlinkable. Proofs of these assertions can be found in Appendix A.

5.5 Deriving Efficient PSI

We now convert the above APSI protocol into a PSI variant, shown in Figure 3. In doing so, the main change is the obviated need for the RSA setting. Instead, the protocol operates in Z_p where p is a large prime and and q is a large divisor of $p-1$. This change makes the protocol more efficient, especially, because of smaller ($|q|$-size) exponents. Nonetheless, the basic complexity remains the same: linear bandwidth – $O(v + w)$, and linear computation – $O(v + w)$ for the server and $O(v)$ for the client. However, we note that, in Step 3b, the server can precompute all values of the form: $(hs_j)^{-R_s}$. Thus, the cost of computing all $K_{s:j}$ values can be reduced to w multiplications (from w exponentiations). In fact, the same optimization applies to the protocol in Figure 2. Correctness of the protocol is self-evident, since its essential operation is very similar to that of the APSI variant.

With the exception of authorization, security and privacy features of this protocol are the same as that of its APSI counterpart described above.

5.6 More Efficient PSI

Although efficient in principle, the PSI protocol in Figure 3 is *sub-optimal* for application scenarios where the client is a resource-poor device, e.g. a PDA or

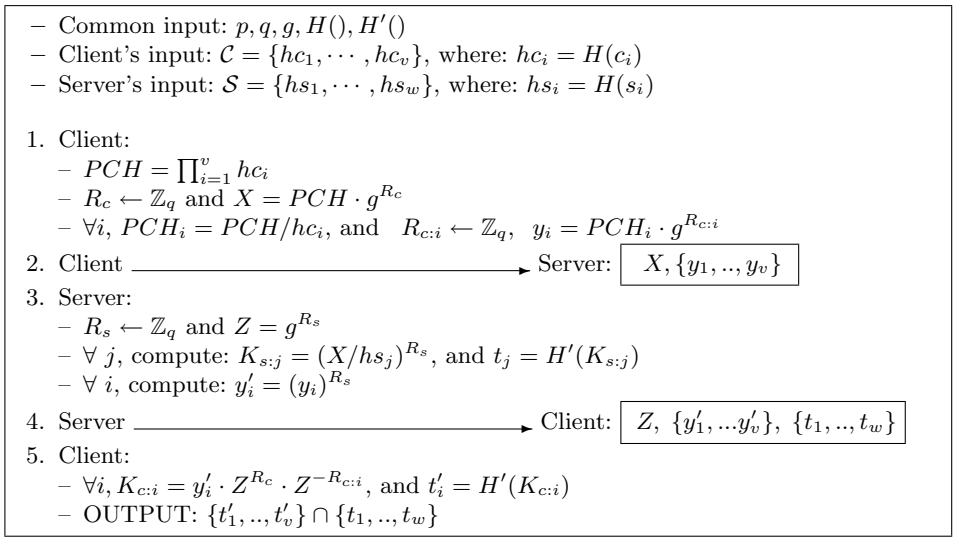

Fig. 3. PSI Protocol with linear complexity

a cell-phone. In other words, $O(v)$ exponentiations might still represent a fairly heavy burden. Also, if the server's set is very large, overhead incurred by $O(w)$ modular multiplications might be substantial.

To this end, we present an even more efficient PSI protocol (see Figure 4) where the client does not perform any modular exponentiations on-line. Instead, it only needs $O(v)$ on-line modular multiplications (Step 7). Also, server's on-line computation overhead is reduced to $O(v)$ exponentiations in Step 5. Server precomputation in Step 1 amounts to w exponentiations – RSA signatures. Client precomputation in Step 2 involves $O(v)$ multiplications, since, as is well-known that, e can be a small integer.

The main idea behind this protocol comes from the Ogata and Kurosawa's adaptive k-out-of-n Oblivious Transfer [18]. We adapt it for the PSI scenario and show that the resulting protocol in Fig. 4 is: (1) Server-Private, (2) Client-Private, (3) Client-Unlinkable (see Appendix B).

Although this protocol uses the RSA setting, RSA parameters are initialized *a priori* by the server. This is in contrast to the protocol in Figure 2 where the CA sets up RSA parameters. To see that the present protocol is correct, consider that: $K_{s:j} = (hs_j)^d$ in Step 1, and, in Step 6:

$$K_{c:i} = y'_i / R_{c:i} = (hc_i \cdot (R_{c:i})^e)^d / R_{c:i} = (hc_i)^d$$

Thus, $K_{c:i} = K_{s:j}$ iff $hc_i = hs_j$.

Drawbacks: although very efficient, this PSI protocol has some issues. First, it is unclear how to convert it into an APSI version. Second, if precomputation is somehow impossible, its performance becomes worse than that of the PSI protocol in Figure 3, since the latter uses much shorter exponents. Privacy features

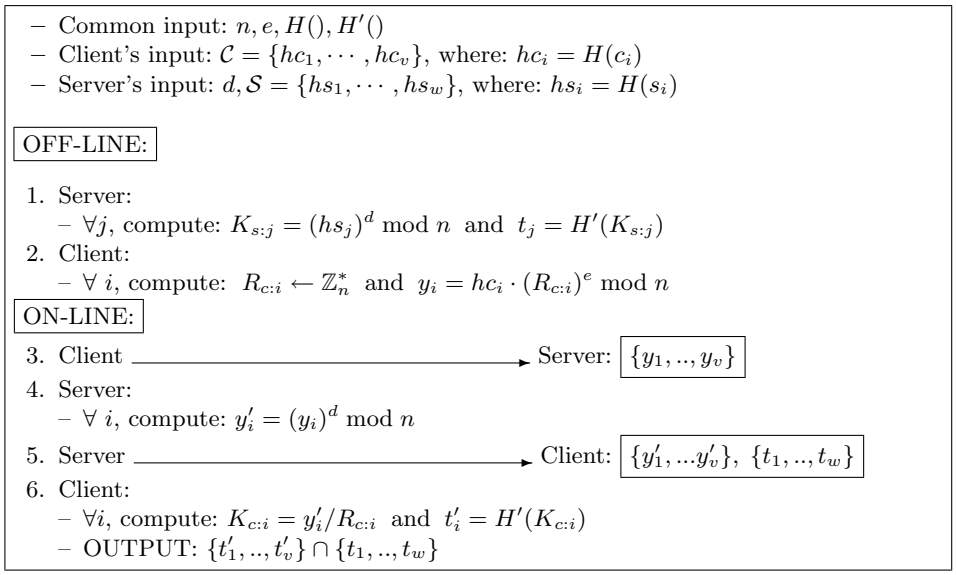

Fig. 4. Blind RSA-based PSI Protocol with linear complexity

of this protocol also differ from others discussed above. In particular, it lacks server unlinkability. (Recall that this feature is relevant only if the protocol is run multiple times.) We note that, in Step 1 the server computes tags of the form $t_j = H'(hs_j)^d$. Consequently, running the protocol twice allows the client to observe any and all changes in the server's set.

There are several ways of patching the protocol to provide this missing feature. One is for the server to select a new set of RSA parameters for each protocol instance. This would be a time-consuming extra step at the start of the protocol; albeit, with precomputation, no extra on-line work would be required from the server. On the other hand, the client would need to be informed of the new RSA public key (e, n) before Step 2, which means that, at the very least, v multiplications in Step 2 would have to be done on-line. Also, two additional initial messages would be necessary: one from the client – to "wake up" the server, and the other – from the server to the client bearing the new RSA public key and (perhaps) $\{t_1, .., t_w\}$, thus saving space in the last message. Another simple way of providing server unlinkability is to change the hash function $H()$ for the server each protocol instance. If we assume that the client and server maintain either a common protocol counter (monotonically increasing and non-wrapping) or sufficiently synchronized clocks, it is easy to select/index a distinct hash function based on such unique and common values. One advantage of this approach is that we no longer need the two extra initial messages.

5.7 From PSI (APSI) to PSI-DT (APSI-DT)

It is easy to add data transfer functionality to the protocols in Figures 1, 2, 3 and 4. Following the approach outlined in [10], we assume that an additional secure

cryptographic hash function $H'' : \{0,1\}^* \rightarrow \{0,1\}^\tau$ is chosen during setup. In all aforementioned protocols, we then use H'' to derive a symmetric key for a semantically secure symmetric cipher, such as AES [9]. For every j, server computes $k_{s:j} = H''(K_{s:j})$ and encrypts associated data using a distinct key $k_{s:j}$. For its part, the client, for every i, computes $k_{c:i} = H''(K_{c:i})$ and decrypts ciphertexts corresponding to the matching tag. (Note that $k_{s:j} = k_{c:i}$ iff $s_j = c_i$ and so $t_j = t_i$). As long as the underlying encryption scheme is semantically secure, this extension does not affect the security or privacy arguments for any protocol discussed thus far.

5.8 Evaluation

We now highlight the differences between existing PSI techniques and protocols proposed in this paper. We focus on performance in terms of server and client computation and bandwidth complexities.

As before, we use w and v to denote with the number of elements in the server's and client's sets, respectively. Let m be the number of bits needed to represent each element. BW denotes the Boyen-Waters IBE scheme [5]. Recall that each BW encryption requires 6 exponentiations and a representation of 6 group elements, and each decryption requires 5 bilinear map operations. We count only the number of *online* operations. The results (which also include bandwidth) are summarized in Table 2.

	Auth	**Model Adv Type**	**BW**	**Server Precomp.**	**Server Ops**	**Client Ops**
[FNP04]-1 PSI from [12]	N	Std HbC	$O(v+w)$	-	$O(vw)$ m-bit mod 2048 exps	$O(w)$ 1024-bit mod 2048 exps
[FNP04]-2 PSI from [12]	N	ROM Mal	$O(v+w)$	-	$\approx$ double "	$\approx$ same "
[CKRS09] APSI from [6]	Y	Std Mal	$O(w)$	-	$O(w)$ BW encrs	$O(vw)$ BW decrs
[JL09] PSI from [15]	N	Std Mal	$O(v+w)$	$O(w)$ 1024-bit mod 1024 exps	$O(v)$ 1024-bit mod 2048 exps	$O(v)$ m-bit mod 2048 + 1024-bit mod 1024 exps
RSA-PPIT APSI from [10] in Fig 1	Y	ROM HbC	$O(v \cdot w)$	$O(w)$ 1024-bit mod 1024 exps	$O(v)$ 1024-bit mod 1024 exps + $O(vw)$ 1024-bit mod 1024 mults	$O(v)$ 1024-bit mod 1024 exps
APSI in Fig 2	Y	ROM HbC	$O(v+w)$	$O(w)$ 1024-bit mod 1024 exps	$O(v)$ 1024-bit mod 1024 exps	$O(v)$ 1024-bit mod 1024 exps
PSI in Fig 3	N	ROM HbC	$O(v+w)$	$O(w)$ 160-bit mod 1024	$O(v)$ 160-bit mod 1024 exps	$O(v)$ 160-bit mod 1024 exps
PSI in Fig 4	N	ROM HbC	$O(v+w)$	$O(w)$ 1024-bit mod 1024 exps	$O(v)$ 1024-bit mod 1024 exps	$O(v)$1024-bit 1024-bit mults

Table 2. Performance Comparison of PSI and APSI protocols.

All protocols proposed in this paper have been implemented in ANSI C (using the well-known OpenSSL [22] library) and tested on a Dell PC with two quad-core CPUs Intel Xeon at 1.60GHz with 8GB RAM. The source code will be made available along with the final version of the paper. To confirm the claimed efficiency of our protocols, we compared on-line run-times of our protocols to

those of prior work. We omit run-times for operations that can be precomputed. We also do not measure all prior techniques discussed in Section 4 and summarized in Table 2. Instead, we pick only the two that offer the best performance: Jarecki and Liu's PSI [15] and the APSI adaptation of RSA-PPIT [10] presented in Fig 1.

Measured *online* computation overhead for the tested protocols is reflected in Table 3. As the results illustrate, among APSI protocols, the one in Figure 2 performs noticeably better than its PPIT-based counterpart from [10] when both server and client have sets of size $1,000$ (and this advantage accelerates for larger set sizes). For smaller sets sizes, the difference is minor. Looking at PSI protocols, the toss-up is between protocols in Figures 3 and 4; the choice of one or the other depends on whether client or server overhead is more important. If client is a weak device, the blind-RSA-based protocol in Figure 4 is a better bet. Otherwise, if server burden must be minimized, we opt for the protocol of Figure 3.

	Player					
	Server	Client	Server	Client	Server	Client
Set size	1	1000	1000	1	1000	1000
PSI [JL09]	5794	23726	8	30	5784	23654
APSI in Fig 1 [10]	2191	2159	8	3	5975	2164
APSI in Fig 2	2195	2277	5	8	2160	2298
PSI in Fig 3	377	1100	5	1	384	1161
PSI in Fig 4	1964	321	2	0	1966	325

Table 3. On-line computation overhead (in ms)

6 Conclusions

In this paper, we proposed efficient protocols for plain and authorized private set intersection (PSI and APSI). Proposed protocols offer appreciably better efficiency than prior results. The choice between them depends on whether there is a need for client authorization and/or server unlinkability, as well as on server's ability to engage in precomputation. Our efficiency claims are supported by experiments with prototype implementations.

7 Acknowledgment

The blind RSA-based protocol resembles in a way some work-in-progress by Jarecki and Liu at University of California, Irvine[16]. One of the Private Set Intersection protocols in their work operates with a lower computation cost for the server (but higher for the client), than the protocol in Figure 4. Its security is shown in ROM under the One-More-Gap-DH assumption; whereas, we use

ROM under the One-More-RSA assumption. The structure of the server privacy proofs in Appendix B and C is inspired by the one by Jarecki and Liu.

References

1. M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Randomizable Proofs and Delegatable Anonymous Credentials. In *Proceedings of CRYPTO'09*, 2009.
2. M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The one-more-RSA-inversion problems and the security of Chaum's blind signature scheme. *Journal of Cryptology*, 16(3):185–215, 2008.
3. D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key Encryption with Keyword Search. In *Eurocrypt'04*, pages 506–522, 2004.
4. D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.
5. X. Boyen and B. Waters. Anonymous Hierarchical Identity-Based Encryption (Without Random Oracles). In *Crypto'06*, pages 290–307, 2006.
6. J. Camenisch, M. Kohlweiss, A. Rial, and C. Sheedy. Blind and Anonymous Identity-Based Encryption and Authorised Private Searches on Public Key Encrypted Data. In *Proceedings of PKC'09*, pages 196–214, 2009.
7. J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *Proceedings of CRYPTO'03*, pages 126–144, 2003.
8. J. Camenisch and G. Zaverucha. Private intersection of certified sets. In *Proceedings of Financial Cryptography and Data Security*, 2009.
9. J. Daeman and V. Rijmen. AES proposal: Rijndael. 1999.
10. E. De Cristofaro, S. Jarecki, J. Kim, and G. Tsudik. Privacy-Preserving Policy-Based Information Transfer. In *Proceedings of PETS'09*, pages 164–183, 2009.
11. M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Proceedings of TCC'05*, pages 303–324, 2005.
12. M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Eurocrypt'04*, pages 1–19, 2004.
13. D. Freeman. Pairing-based identification schemes. *Arxiv preprint cs/0509056*, 2005.
14. C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC'08*, pages 155–175, 2008.
15. S. Jarecki and X. Liu. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *TCC*, pages 577–594, 2009.
16. S. Jarecki and X. Liu. Fast Secure Computation of Set Intersection. Manuscript available from the authors, 2009.
17. L. Kissner and D. Song. Privacy-preserving set operations. In *Proceedings of CRYPTO'05*, pages 241–257, 2005.
18. W. Ogata and K. Kurosawa. Oblivious keyword search. *Journal of Complexity*, 20(2-3):356–371, 2004.
19. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of Eurocrypt'99*, pages 223–238, 1999.
20. R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

21. C. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.
22. E. Young and T. Hudson. OpenSSL: The Open Source toolkit for SSL/TLS. `http://www.openssl.org`.

A: APSI protocol in Fig. 2

We now consider security and privacy properties of the protocol in Figure 2.
Client Privacy. Recall that APSI is client-private if no information is leaked to the server about client's private inputs. It is easy to show that client's inputs are polynomially indistinguishable from a random distribution. This is because, in Step 1, the client selects all values uniformly and at random, i.e., $[R_c, \{R_{c:1}, ..., R_{c:v}\}] \leftarrow \mathbb{Z}_n^*$. Thus, $X = PCH \cdot g^{R_c}$ and $\{y_i = PCH_i^* \cdot g^{R_{c:i}}\}$ form a random sequence.
Server privacy. To claim server privacy, we need to show that no efficient $\mathcal{A}$ has a *non-negligible* advantage over 1/2 against a challenger Ch in the following game. Our proof works in the random oracle model (ROM) under the RSA assumption.

1. Ch executes $(PK, SK) \leftarrow \mathsf{Setup}(1^\tau)$ and gives PK to $\mathcal{A}$.
2. $\mathcal{A}$ invokes *Authorize* on c_i of its choice and obtains the corresponding signature σ_i.
3. $\mathcal{A}$ generates elements c_0^*, c_1^* different from every c_i mentioned above.
4. $\mathcal{A}$ participates in the protocol as the client with messages X^* and y_0^*, y_1^*.
5. Ch picks one record pair by selecting a random bit b and executes the server's part of the interaction on public input PK and private input (c_b^*) with message (Z, y', t) as described in the protocol.
6. $\mathcal{A}$ outputs b' and wins if $b = b'$.

Let HQuery be an event that $\mathcal{A}$ ever queried H' on input K^*, where K^* is defined (as the combination of message X^* sent by $\mathcal{A}$ and message Z sent by Ch), as follows: $K^* = (X^*)^{eR_s} \cdot (h^*)^{-R_s} \bmod N$, where $Z = (g)^{eR_s}$ and $h^* = H(c^*)$. In other words, HQuery is an event that $\mathcal{A}$ computes (and invoked hash function H' on input of) the key-material K^* for the challenging protocol.

Unless HQuery happens, $\mathcal{A}$'s view of interaction with Ch on bit $b = 0$ is indistinguishable from $\mathcal{A}$'s view of the interaction with Ch on bit $b = 1$.

Since the distribution of $Z = g^{eR_s}$ is independent from (c_b), it reveals no information about which c_b is related in the protocol. Also, since y_0^*, y_1^* are not related to $H(c_0)^d$ nor $H(c_1)^d$, $y' = (y_b)^{eRs}$ reveals no information about which c_b is related in the protocol (y' is similar to an RSA encryption). Finally, assuming that H' is modeled as a random oracle, the distribution with $b = 0$ is indistinguishable from that with $b = 1$, unless $\mathcal{A}$ computes $k^* = H'(K^*)$, in the random oracle model, by querying H', i.e., HQuery happens.

If event HQuery happens with non-negligible probability, then $\mathcal{A}$ can be used to violate the RSA assumption.

We construct a reduction algorithm called RCh using a modified challenger algorithm. Given the RSA challenge (N, e, α), RCh simulates signatures on each c_i by assigning $H(c_i)$ as $\sigma_i^e \bmod N$ for some random value σ_i. This way, RCh can present the authorization on c_i as σ_i. RCh embeds α to each H query, by setting $H(c_i) = \alpha(a_i)^e$ for random $a_i \in \mathbb{Z}_N$. Note that, given $(H(c_i))^d$ for any c_i, the simulator can extract $\alpha^d = (H(c_i))^d/a_i$.

RCh responds to $\mathcal{A}$ and computes $(H(c_i))^d$(for some c_i) as follows: On $\mathcal{A}$'s input message X^*, y_0^*, y_1^*, RCh picks a random $m \leftarrow \mathbb{Z}_N$, computes $Z = g^{(1+em)}$, and sends Z and $y' = (y_b)^{em}$. We see that $g^{1+em} = g^{e(d+m)}$. On the HQuery event, RCh gets $K^* = (X^*)^{e(d+m)}(h^*)^{-(d+m)}$ from $\mathcal{A}$. Since RCh knows X^*, h^*, e, and m, it can compute $(h^*)^d$.

B: PSI protocol in Fig. 3

We now consider privacy properties of the protocol in Figure 3.

Client Privacy. Recall that a PSI protocol is client-private if no information is leaked to the server about client's private inputs. It is easy to show that client's inputs are polynomially indistinguishable from a random distribution. This is because, in Step 1, the client selects all values uniformly and at random, i.e., $[R_c, \{R_{c:1}, ..., R_{c:v}\}] \leftarrow \mathbb{Z}_q$. Thus, $X = PCH \cdot\ g^{R_c}$ and $\{y_i = PCH_i \cdot g^{R_{c:i}}\}$ form a random sequence.

Server Privacy. We present a concise construction of an ideal (*adaptive*) world SIM_c from a honest-but-curious real-world client C^*, and show that the views of C^* in the real game with the real world server and in the interaction with SIM_c are indistinguishable, under the *One-More Gap Diffie-Hellman* assumption in the random oracle model.

First, SIM_c picks a random $R_s \in \mathbb{Z}_q$ and prepares a set of random values $T = \{t_1, \cdots, t_w\}$. SIM_c models the hash function H and H' as random oracles. A query to H is recorded as $(q, h = H(q))$, a query to H' is recorded as $(k, h' = H'(k))$, where q and h' are random values. We describe below the details on SIM_c's answers to H' queries. Finally, SIM_c creates two empty sets A, B.

During the interaction, SIM_c stores the incoming value X, and, for every $y_i \in \{y_1, \cdots, y_v\}$ received from C^*, SIM_c answers with $y_i' = (y_i)^{R_s}$.

We now describe how SIM_c answers to queries to H'. On query k to H', SIM_c checks if it had recorded a value h, s.t. $k = (X/h)^{R_s}$:

- If not, SIM_c answers a random value h' and record (k, h') as mentioned above.
- If yes, SIM_c can recover the q s.t. $h = H(q)$ and $k = (X/h)^{R_s}$
  Then, SIM_c checks if it had been previously queried on the value k:
  - If yes, check if $q \in A$.
    - If $q \notin A$, it means that C^* queried q to H (which returned h), and also made an independent query k to H' s.t. $k = (X/h)^{R_s}$. In this case SIM_c aborts the protocol. However, it easy to see that this happens with negligible probability.
    - If $q \in A$, SIM_c returns the value h' previously stored for k.

- If not, this means that SIM_c is learning one of C^*'s outputs. Hence, $A = A \cup \{q\}$.
  Then, SIM_c checks if $|A| > v$.
  - If $|A| <= v$, then SIM_c checks if $q \in \mathcal{C} \cap \mathcal{S}$ by playing the role of the client with the real world server.
    - If $q \in \mathcal{C} \cap \mathcal{S}$, SIM_c answers to the query on k with a value $t_j \in T \backslash B$, records the answer (k, t_j) and sets $B = B \cup \{t_j\}$.
    - If $q \notin \mathcal{C} \cap \mathcal{S}$, SIM_c answers with a random value h' and records the answer.
  - If $|A| > v$, then a reduction *Red* that breaks the *One-More-DH* assumption can be constructed.

The reduction *Red* can be constructed as follows. *Red* answers to C^*'s queries to H with the inverse of the One-More-DH challenges $(1/g_1, \cdots, 1/g_{ch})$. During interaction, on C^*'s messages $y_i \in \{y_1, \cdots, y_v\}$, *Red* answers $y'_i = (y_i)^z$ by querying the DH_z oracle. When SIM_c (on query k to H') checks if there exists a recorded value h, s.t. $k = (X/h)^z$, *Red* queries the $\mathsf{DL}_z(\cdot, \cdot)$ oracle on $(X/h, k)$, hence the need for the *Gap* DH assumption. Finally, if the case depicted above happens, it means that at the end of the protocol the set B will contain at least $(v+1)$ elements (where v is the number of One-More-DH challenges), that are in the form $(h = 1/g, k = (X/h)^z)$. Thus, it is possible to extract at least $v+1$ pairs (g, g^z) thus breaking the *One-More-DH assumption* in the weaker assumption that the *Red* is given access to an additional DH_z oracle query to query X and obtain X^z. How to improve the above proof to avoid the additional oracle query is an ongoing research effort.

As a result, we have shown that the views of C^* in the real game with the real world server and in the interaction with SIM_c are indistinguishable.

C: PSI Protocol in Fig. 4

We now consider privacy properties of the protocol in Figure 4.
Client Privacy. As in Appendix A, we claim it is easy to show that client's inputs to the protocol are statistically close to random distribution.
Server Privacy. We present a concise construction of an ideal (*adaptive*) world SIM_c from a honest-but-curious real-world client C^*, and show that the views of C^* in the real game with the real world server and in the interaction with SIM_c are indistinguishable, under the *One-More-RSA* assumption in the random oracle model.

First, SIM_c runs $(N, e, d) \leftarrow$ RSA-Keygen(τ) and gives (N, e) to C^*. SIM_c models the hash function H and H' as random oracles. A query to H is recorded as $(q, h = H(q))$, a query to H' as $(k, h' = H'(k))$, where q and h' are random values. Finally, SIM_c creates two empty sets A, B.

During interaction, SIM_c publishes the set $T = \{t_1, \cdots, t_w\}$, where t_j is taken at random. Also, for every $y_i \in \{y_1, \cdots, y_v\}$ received from C^* (recall that $y_i = H(c_i) \cdot (R_{c:i})^e$), SIM_c answers according to the protocol with $(y_i)^d$.

We now describe how SIM_c answers to queries to H''. On query k to H', SIM_c checks if it had recorded a value h, s.t. $h = k^e$ (i.e. $h^d = k$):

- If not, SIM_c answers a random value h' and record (k, h') as mentioned above.
- If yes, SIM_c can recover the q s.t. $h = H(q)$ and $h = k^e$
  Then, SIM_c checks if it had been previously queried on the value k:
  - If yes, check if $q \in A$.
    - If $q \notin A$, it means that C^* queried q to H (which returned h), and also made an independent query k to H' s.t. $h = k^e$. In this case SIM_c aborts the protocol. However, it easy to see that this happens with negligible probability.
    - If $q \in A$, SIM_c returns the value h' previously stored for k.
  - If not, this means that SIM_c is learning one of C^*'s outputs. Hence, $A = A \cup \{q\}$.
    Then, SIM_c checks if $|A| > v$.
    - If $|A| <= v$, then SIM_c checks if $q \in \mathcal{C} \cap \mathcal{S}$ by playing the role of the client with the real world server.
      - If $q \in \mathcal{C} \cap \mathcal{S}$, SIM_c answers to the query on k with a value $t_j \in T \backslash B$, records the answer (k, t_j) and sets $B = B \cup \{t_j\}$.
      - If $q \notin \mathcal{C} \cap \mathcal{S}$, SIM_c answers with a random value h' and records the answer.
    - If $|A| > v$, then a reduction *Red* that breaks the *One-More-RSA* assumption can be constructed.

The reduction *Red* can be constructed as follows. *Red* answers to C^*'s queries to H with RSA challenges $(\alpha_1, \cdots, \alpha_{ch})$. During interaction, on C^*'s messages $y_i \in \{y_1, \cdots, y_v\}$, *Red* answers $(y_i)^d$ by querying the RSA Oracle. Finally, if the case depicted above happens, it means that at the end of the protocol the set B will contain at least $(v + 1)$ elements, where v is the number of RSA challenges, thus breaking the *One-More-RSA assumption.*

As a result, we have shown that the views of C^* in the real game with the real world server and in the interaction with SIM_c are indistinguishable.

D: Cryptographic Assumptions

RSA assumption. Let $RSASetup(\tau)$ be an algorithm that outputs so-called RSA instances, i.e. pairs (N, e) where $N = pq$, e is a small prime that satisfies $gcd(e, \phi(N)) = 1$, and p, q are randomly generated τ-bit primes. We say that the RSA problem is (τ, t)-hard on τ-bit RSA moduli, if for every algorithm $\mathcal{A}$ that runs in time t we have:

$$Pr[(N, e) \leftarrow RSASetup(\tau), \alpha \leftarrow \mathbb{Z}_N^* : \mathcal{A}(n, e, \alpha) = \beta \text{ s.t. } \beta^e = \alpha \pmod N] \leq \tau$$

One-More-Gap-DH assumption. Informally, the One-More-Gap-DH assumption [13] indicates that the DH problem is hard even if the adversary is given access to a DH oracle, while DDH problem is easy. Formally, let $(\mathbb{G}, q, g) \leftarrow KeyGen(\tau)$ the Key-Generation algorithm outputting a multiplicative group of order q and assume $z \leftarrow \mathbb{Z}_q$. We say that the One-More-Gap-DH problem is (τ, t)-hard if for every algorithm $\mathcal{A}$ that runs in time t we have

$Pr\left[\{(g_i, (g_i)^z)\}_{i=1,\cdots v+1} \leftarrow \mathcal{A}^{DH_z(\cdot), DDH(\cdot,\cdot,\cdot,\cdot)}(g_1, \cdots, g_{ch})\right] \leq \tau$

where $\mathcal{A}$ made at most v queries to the $DH_z(\cdot)$ oracle.

One-More-RSA assumption. Informally, the One-More-RSA assumption [2] indicates that the RSA problem is hard even if the adversary is given access to an RSA oracle. Formally, let $(N, e, d) \leftarrow KeyGen(\tau)$ the RSA Key-Generation algorithm, and let $\alpha_j \leftarrow \mathbb{Z}_N^*$ (for $j = 1, \cdots, ch$), we say that the One-More-RSA problem is (τ, t)-hard on τ-bit RSA moduli, if for every algorithm $\mathcal{A}$ that runs in time t we have

$$Pr\left[\{(\alpha_i, (\alpha_i)^d)\}_{i=1,\cdots v+1} \leftarrow \mathcal{A}^{(\cdot)^{d \bmod N}}(N, e, \tau, \alpha_1, \cdots, \alpha_{ch})\right] \leq \tau$$

where $\mathcal{A}$ made at most v queries to the RSA oracle $(\cdot)^{d \bmod N}$.