Practical Private Set Intersection Protocols

Emiliano De Cristofaro and Gene Tsudik

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.

Such scenarios provide with interesting examples that 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, i.e. the entity receiving the intersection) 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. Due to space limitation, we keep this section at minimum, and we refer the reader to an extended version of the paper for more details [12]. Also, we note that most PSI variations can be realized via general secure multi-party techniques. However, it is usually far more efficient to have dedicated protocols; which is the direction we pursue in this paper.

PSI Protocols. The work by Freedman, et al. (FNP) [15] addressed the problem of private set intersection by means of Oblivious Polynomial Evaluation (OPE). In [15], the idea is to represent of a set as a polynomial, and the elements of the set as its roots. Specifically, a client C represents elements in its private set, $\mathcal{C} = (c_1, \cdots, c_v)$, as the roots of a v-degree polynomial over a ring R, i.e. $f = \prod_{i=1}^{v}(t - c_i) = \sum_{i=0}^{k} \alpha_i t^i$. Then, assuming pk_C to be C's public key of any additively homomorphic cryptosystem (such as Paillier [22]), C encrypts the coefficients with pk_C, and sends them to server S. S's private set is denoted with $\mathcal{S} = (s_1, \cdots, s_w)$. S evaluates f at each $s_j \in \mathcal{S}$ homomorphically. Note that $f(s_j) = 0$ if and only if $s_j \in \mathcal{C} \cap \mathcal{S}$. Hence 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 $O(v + w)$, i.e., 1024-bit exponentiations mod 2048 bits. However, certain optimizations can be applied to reduce the total number of server's exponentiations to $O(w \log(\log(v)))$. Such protocol is proved secure against an Honest-but-Curious (HbC) adversary in the standard model, and can be extended for malicious adversaries in the Random Oracle Model (ROM).

Subsequently, the work by Kissner and Song (KS) [20] has proposed OPE-based protocols that apply to several set operations (e.g. union, intersection, element reduction) and may involve more than two players. [20] contributes constructions secure in the standard model against HbC (with similar complexity to [15]) and also malicious adversaries. The latter incurs into quadratic computation overhead, i.e. $O(wv)$, and involves expensive zero-knowledge proofs, whereas a more efficient construction has been recently proposed by [9].

Protocols based on Oblivious Pseudo Random Functions. Other constructs rely on so-called Oblivious Pseudo-Random Functions (OPRFs), introduced in [14]. 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 $f_k(x)$.

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 $f_k(c_i) \in \mathcal{U}$.

The idea of using OPRFs for PSI protocols is due to Hazay and Lindell [16]. Their protocol is secure in the standard model in the presence of a malicious server and an HbC client. It has been since improved by Jarecki and Liu [18], 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. Encryption operations are performed using an additively homomorphic encryption scheme, such as the one presented by Camenisch and Shoup, CS for short [7].

As pointed out in [18], such solution can be further optimized, inspiring to the concurrent work by Belenkiy, et al. [1]. In fact, the OPRF construction could work on groups with a 160-bit prime order unrelated to the RSA modulus, instead of the more expensive composite order groups. Assuming such improved construction, [18] incurs in the following computational complexity. The server S needs to perform $O(w)$ PRF evaluations with w inputs, more precisely $O(w)$ modular exponentiations of m-bit exponents (where m is the number of bits needed to represent each entry) mod n^2 (e.g. 2048 bits). Moreover, the client C needs to compute $O(v)$ 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). As acknowledged in [18], complexity in the malicious model grows by a factor of 2.

Finally, the work-in-progress in [19] leverages an idea similar to the OPRF, namely the *Unpredictable Function* (UPF) $f_k(x) = (H(x))^k$ in the Random Oracle Model.

Authors construct a two-party computation of this function, with a server S contributing the key k and a client C the argument x: C picks a random exponent α and sends $y = (H(x))^{\alpha}$ to S, that replies with $z = y^{k}$, so that C recovers $f_k(x) = z^{1/\alpha}$. Note that random exponents, given that the hash functions are carefully chosen, can be taken from a subgroup (e.g. they can be 160-bits long). Similarly to OPRF-based solutions, the UPF can then be used to implement the secure computation of *Adaptive* Set Intersection, under the *One-More-Gap-DH* assumption in ROM [2]. We remark that this solution is essentially almost the same as the ones given before by [17] and [13]. However, in such works, security is only superficially analyzed and no proof is provided, whereas [19] provides security also against a malicious server. The computational complexity of the UPF-based PSI amounts to $O(w + v)$ (resp. $O(v)$) 160-bit mod 1024-bits exponentiations for the server (resp. the client).

APSI Protocols. We now briefly review related work in Authorized Private Set Intersection protocols. Recently, a new PSI-related concept was introduced, called *Privacy-preserving Policy-based Information Transfer* (**PPIT**) [11]. 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. [11] gives three PPIT protocols, based respectively on: RSA [23], Schnorr [24], and Identity-based Encryption (IBE) [4]. In this paper, we only discuss RSA-PPIT since it serves as a starting point for the work in this paper. In RSA-PPIT, client's authorizations are essentially RSA signatures on a set of record identifiers. As shown in [11], 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 ROM for honest-but-curious parties. However, the complexity (both communication and computation) becomes quadratic. We will review such construction in Section 5.3.

Another recent result [6] has 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).

Finally, Camenisch and Zaverucha [8] have introduced 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 [15] and KS [20].

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:* $\{Setup, Interaction\}$*.* $Setup$*: 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:* $\{Setup, Authorize, Interaction\}$*.* $Setup$*: 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. In APSI, we assume that CA does not behave maliciously. Also, server is honest-but-curious, however, client might not have authorizations for all elements in its set. Finally, in PSI we assume that both client and server are honest-but-curious.

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:

Table 1. Notation

$a \leftarrow A$	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

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, an APSI 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 [11]. This protocol is only vaguely sketched out in [11]; since our new protocols are loosely based on it, we specify it in Fig.1. Actually, the protocol in [11] 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.

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, $hc_i = hs_j$):

- 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 $\xrightarrow{\qquad\qquad}$ 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 $\xrightarrow{\qquad\qquad}$ 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

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

We point out that the protocol in Fig.1 incurs in 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.

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 Fig.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 $\xrightarrow{\hspace{6cm}}$ 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 $\xrightarrow{\hspace{6cm}}$ 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. (See Appendix A).

5.5 Deriving Efficient PSI

We now convert the above APSI protocol into a PSI variant, shown in Fig.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 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 $O(w)$ multiplications (from $O(w)$ exponentiations). In fact, the same optimization applies to the protocol in Fig.2. Correctness of the protocol is self-evident, since its essential operation is very similar to that of the APSI variant.

We defer formal proofs for the above PSI protocol to [12]. They basically mirror the proofs of the protocol constructed in the next section (see Appendix B). Simulations of the two schemes follow the same approach, except that, while proofs in Appendix B rely on the One-More-RSA assumption, privacy of protocol in Fig. 2 is based on the One-More-Gap-DH assumption [2].

- Common input: $p, q, g, H(), H'()$
- Client's input: $\mathcal{C} = \{hc_1, \cdots, hc_v\}$, where: $hc_i = H(c_i)$
- Server's input: $\mathcal{S} = \{hs_1, \cdots, hs_w\}$, where: $hs_i = H(s_i)$

1. Client:
   - $PCH = \prod_{i=1}^{v} hc_i$
   - $R_c \leftarrow \mathbb{Z}_q$ and $X = PCH \cdot g^{R_c}$
   - $\forall i, PCH_i = PCH/hc_i$, and $R_{c:i} \leftarrow \mathbb{Z}_q, \;\; y_i = PCH_i \cdot g^{R_{c:i}}$
2. Client $\xrightarrow{\hspace{4cm}}$ Server: $\boxed{X, \{y_1, .., y_v\}}$
3. Server:
   - $R_s \leftarrow \mathbb{Z}_q$ and $Z = g^{R_s}$
   - $\forall\, j$, compute: $K_{s:j} = (X/hs_j)^{R_s}$, and $t_j = H'(K_{s:j})$
   - $\forall\, i$, compute: $y'_i = (y_i)^{R_s}$
4. Server $\xrightarrow{\hspace{4cm}}$ 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. 3. PSI Protocol with linear complexity

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 Fig.3 is *sub-optimal* for application scenarios where the client is a resource-poor device, e.g. a PDA or 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 Fig. 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 [21]. 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 Fig.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:

$$K_{c:i} = y'_i/R_{c:i} = (hc_i \cdot (R_{c:i})^e)^d/R_{c:i} = (hc_i)^d \Longrightarrow K_{c:i} = K_{s:j} \text{ iff } hc_i = hs_j$$

- Common input: $n, e, H(), H'()$
- Client's input: $\mathcal{C} = \{hc_1, \cdots, hc_v\}$, where: $hc_i = H(c_i)$
- Server's input: $d, \mathcal{S} = \{hs_1, \cdots, hs_w\}$, where: $hs_i = H(s_i)$

$\boxed{\text{OFF-LINE:}}$

1. Server:
   – $\forall j$, compute: $K_{s:j} = (hs_j)^d \bmod n$ and $t_j = H'(K_{s:j})$
2. Client:
   – $\forall\, i$, compute: $R_{c:i} \leftarrow \mathbb{Z}_n^*$ and $y_i = hc_i \cdot (R_{c:i})^e \bmod n$

$\boxed{\text{ON-LINE:}}$

3. Client $\xrightarrow{\hspace{6cm}}$ Server: $\boxed{\{y_1, .., y_v\}}$
4. Server:
   – $\forall\, i$, compute: $y'_i = (y_i)^d \bmod n$
5. Server $\xrightarrow{\hspace{6cm}}$ Client: $\boxed{\{y'_1, ...y'_v\},\ \{t_1, .., t_w\}}$
6. Client:
   – $\forall i$, compute: $K_{c:i} = y'_i/R_{c:i}$ and $t'_i = H'(K_{c:i})$
   – OUTPUT: $\{t'_1, .., t'_v\} \cap \{t_1, .., t_w\}$

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

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 Fig.3, since the latter uses much shorter exponents. Privacy features 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 [11], we assume that an additional secure cryptographic hash function $H'' : \{0,1\}^* \to \{0,1\}^\tau$ 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 [10]. 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. We use w and v to denote 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. We count only the number of *online* operations. The results are summarized in Table 2 and also includes bandwidth, for the sake of completeness.
We remark that: (1), each encryption in [5] requires 6 exponentiations and a representation of 6 group elements, and each decryption requires 5 bilinear map operations, and (2), the complexity for the PSI solution against Malicious model in [18] and [19] grows by a factor of 2. All protocols proposed in this paper have been implemented in ANSI C (using the well-known OpenSSL library) and tested on a Dell Precision PC on a 2.33GHz CPU and 8GB RAM. The prototype's 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. In case a solution provides security both against HbC and malicious adversary, we implement the former. We omit run-times for operations that can be precomputed. We also do not measure

Table 2. Performance Comparison of PSI and APSI protocols.

Protocol	**Model**	**Adv**	**Bandw.**	**Server Prec**	**Server Ops**	**Client Ops**	**Mod Bits**
APSI [6]	Std	Mal	$O(w)$	-	$O(w)$ encrs in [5]	$O(vw)$ decrs in [5]	
APSI Fig.1	ROM	HbC	$O(vw)$	$O(w)$ 1024-bit exps	$O(v)$ 1024-bit exps $O(vw)$ mults	$O(v)$ 1024-bit exps	1024
APSI Fig.2	ROM	HbC	$O(v{+}w)$	$O(w)$ 1024-bit exps	$O(v)$ 1024-bit exps	$O(v)$ 1024-bit exps	1024
PSI [15]	Std	HbC	$O(v{+}w)$	-	$O(vw)$ m-bit exps	$O(v+w)$ 1024-bit exps	2048
PSI [18]	Std	HbC	$O(v{+}w)$	$O(w)$ 1024-bit exps mod 1024 exps	$O(v)$ 1024-bit exps mod 2048 exps	$O(v)$ 1024 mod 1024-bit m-bit mod 2048-bit exps	1024/ 2048
PSI [19]	ROM	HbC	$O(v{+}w)$	$O(w)$ 160-bit exps	$O(v)$ 160-bit exps	$O(v)$ 160-bit exps	1024
PSI Fig.3	ROM	HbC	$O(v{+}w)$	$O(w)$ 160-bit exps $O(w)$ mults	$O(v)$ 160-bit exps	$O(v)$ 160-bit exps	1024
PSI Fig.4	ROM	HbC	$O(v{+}w)$	$O(w)$ 1024-bit exps	$O(v)$ 1024-bit exps	$O(v)$1024-bit mults	1024

all prior techniques discussed in Section 4, whereas we pick only the three that offer the best performance: the APSI adaptation of RSA-PPIT [11] in Fig.1, and PSI's from [18] and [19]. We remark that since efficiency of [18] is influenced by records' length, we assume a conservative stance and we choose items to be 160-bits long, similar to the output of a hash function.

Measured *online* computation overhead for the tested protocols is reflected in Table 3. As the results illustrate, among APSI protocols, the one in Fig.2 performs noticeably better than its PPIT-based counterpart from [11] when both server and client have sets of size $5,000$ (and this advantage accelerates for larger set sizes). 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 Fig.4 is a better bet. Otherwise, if server burden must be minimized, we opt for the protocol of Fig.3.

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

Player	**Server**	**Client**	**Server**	**Client**	**Server**	**Client**
Set size	5,000	1	1	5,000	5,000	5,000
APSI Fig.1	20	5	12,710	24,407	99,118	24,159
APSI Fig.2	23	10	12,228	25,769	12,037	25,959
PSI [18]	5	24	27,654	118,676	27,862	118,947
PSI [19]	0	1	2,029	4,227	2,108	4,249
PSI Fig.3	19	1	2,145	5,502	2,072	5,344
PSI Fig.4	1	0	4,651	1,407	4,662	1,422

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. Future work includes analysis of our protocols against a malicious server, as well as extensions to a group setting.

References

1. M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Randomizable Proofs and Delegatable Anonymous Credentials. In *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 *PKC'09*, pages 196–214, 2009.
7. J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO'03*, pages 126–144, 2003.
8. J. Camenisch and G. Zaverucha. Private intersection of certified sets. In *Financial Cryptography and Data Security'09*, 2009.
9. D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Efficient Robust Private Set Intersection. In *ACNS'09*, pages 125–142. Springer, 2009.
10. J. Daeman and V. Rijmen. AES proposal: Rijndael. 1999.
11. E. De Cristofaro, S. Jarecki, J. Kim, and G. Tsudik. Privacy-Preserving Policy-Based Information Transfer. In *PETS'09*, pages 164–183, 2009.
12. E. De Cristofaro and G. Tsudik. Practical Private Set Intersection Protocols. Cryptology ePrint Archive, `http://eprint.iacr.org/2009/491.pdf`, 2009.
13. A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *PODS'03*, pages 211–222, 2003.
14. M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *TCC'05*, pages 303–324, 2005.
15. M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Eurocrypt'04*, pages 1–19, 2004.
16. 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.
17. B. Huberman, M. Franklin, and T. Hogg. Enhancing privacy and trust in electronic communities. In *ACM Conference on Electronic Commerce*, pages 78–86, 1999.
18. 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.
19. S. Jarecki and X. Liu. Fast Secure Computation of Set Intersection. Manuscript available from the authors, 2009.
20. L. Kissner and D. Song. Privacy-preserving set operations. In *CRYPTO'05*, pages 241–257, 2005.
21. W. Ogata and K. Kurosawa. Oblivious keyword search. *Journal of Complexity*, 20(2-3):356–371, 2004.
22. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt'99*, pages 223–238, 1999.
23. 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.
24. C. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

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 $y' = (y_b)^{1+em}$. 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. 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 (presented in Appendix C) 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 whether it has recorded a value h s.t. $h = k^e$ (i.e. $h^d = k$).
If $!\exists h$ s.t. $h = k^e$, SIM_c answers a random value h' and record (k, h') as mentioned above.
If $\exists h$ s.t. $h = k^e$, SIM_c can recover the q s.t. $h = H(q)$ and $h = k^e$. Then, it checks whether it has previously been queried on the value k.

If $\exists k$ s.t. k has already been queried, then SIM_c checks whether $q \in A$. 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. Instead, if $q \in A$, SIM_c returns the value h' previously stored for k.

If $!\exists k$ s.t. k has already been queried, 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 we can construct a reduction Red breaking the *One-More-RSA* assumption. 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.

We remark that the structure of the above proof has been inspired from the one based on One-More-Gap-DH assumption from [19], as well as the notion of *adaptiveness*. In other words, the client is allowed to adaptively make queries, i.e., she does not need to specify all her inputs at once.

C: 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-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 $(\cdot)^{d \bmod N}$.