Practical Private Set Intersection Protocols with Linear Computational and Bandwidth Complexity*

Emiliano De Cristofaro and Gene Tsudik
University of California, Irvine
{edecrist,gts}@ics.uci.edu

Abstract

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. 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 properties.
4. Federal tax authority wants to learn whether any suspected tax evaders have accounts with a certain foreign bank and, if so, obtain their account records. 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 airline. 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 example 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.

Paper Organization. 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

*An earlier version of this paper appeared in the Proceedings of Financial Cryptography and Data Security 2010.

techniques in Section 4. Then, we present our protocols in Section 5 and evaluate them in Section 6. We conclude in Section 7. Appendices contain proof sketches and simplified construction of the protocols.

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 (assuming that neither player aborts the protocol prematurely), 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 the terms client (C, i.e., the entity that learn the intersection) and server (S) to refer to the protocol participants.

One natural PSI extension is what we call *PSI with Data Transfer* or PSI-DT. In it, one or both parties have data associated with each element in the set, e.g., a database record. Data associated with each element in the intersection must be transferred to the client. 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 (set items) 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 (client) 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 communication overhead. Our work makes several contributions:

1. We evaluate and compare existing PSI and APSI protocols in terms of efficiency (computation and communication overhead), 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 version. Both are appreciably more efficient that current state-of-the-art.
4. We construct another, even more efficient 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. We also note that most PSI variations can be realized via general secure multi-party techniques. However, it is usually far more efficient to design special-purpose protocols, as highlighted in [19, 26]. This is the direction we pursue in this paper.

4.1 PSI Protocols

We start with PSI protocols based on *Oblivious Polynomial Evaluations (OPE-s)*:

- Freedman, Nissim, and Pinkas (FNP) [19] represent a set as a polynomial, and elements of the set as its roots. Specifically, a client C encodes 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 is C's public key for any additively homomorphic cryptosystem (such as Paillier's [28]), C encrypts the coefficients with pk_C, and sends them to server S, whose private set is $\mathcal{S} = (s_1, \cdots, s_w)$. S homomorphically evaluates f at each $s_j \in \mathcal{S}$. 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. To enable *data transfer*, the server may return $E(r_j f(s_j) + (s_j || data_j))$, for each $s_j \in \mathcal{S}$.

  Therefore, the number of server operations is related to the evaluation of client's encrypted polynomial, with v coefficients, on w points in $\mathcal{S}$. Using Paillier cryptosystem [28] and a 1024-bit modulus, this costs $O(vw)$ of 1024-bit mod 2048-bit exponentiations.[1] On the other hand, client computes $O(v + w)$ of 1024-bit mod 2048-bit exponentiations. However, note that server computation can be reduced to $O(w \log\log v)$ using (1) Horner's rule for polynomial evaluations, and (2) a hashing-to-bins method (see [19]). Note that, if one does not need *data transfer*, it is more efficient to use the Exponential ElGamal cryptosystem [16] (i.e., an ElGamal variant that provides additively homomorphism).[2] Such a cryptosystem does not provide efficient decryption, but it allows client to test whether a ciphertext is an encryption of "0", thus, to learn that the corresponding element belongs to the set intersection. As a result, efficiency is improved, since in ElGamal the computation may make use of: (1) very short random exponents (e.g., 160-bit) and (2) shorter moduli in exponentiations (1024-bit).

  The FNP [19] protocol is proven secure against Honest-but-Curious (HbC) adversaries in the standard model, and can be extended for malicious adversaries in the Random Oracle Model (ROM), at an increased cost.

- Hazay and Nissim [22] recently present an improved construction of the OPE-based PSI in [19], in presence of malicious adversaries without ROM. They introduce zero-knowledge proofs allowing client to demonstrate that encrypted polynomials are correctly produced. Also, they use a technique based on a perfectly hiding commitment scheme with an Oblivious Pseudo Random Function (OPRF) evaluation protocol, to prevent server from deviating from the protocol. Their protocol incurs $O(v + w(\log\log v + m))$ computational and $O(v + w \cdot m)$ communication complexity, where m is the number of bits needed to represent a set element. Note that execution of the underlying OPRF in [22] requires m oblivious transfer invocations, and hence $O(m)$ modular exponentiations, for each set element. However, such overhead can be avoided by instantiating the protocol in ROM. This protocol can be also optimized if the size of the intersection is allowed to be leaked to the server, which would, however, violate our strict privacy definitions. Nonetheless, the resulting protocol would incur complexities of $O(v + |\mathcal{S} \cap \mathcal{C}| \cdot m)$ in communication and $O(v + w \cdot \log\log v + |\mathcal{S} \cap \mathcal{C}| \cdot m)$ in computation (still not linear).

- Kissner and Song [26] propose OPE-based protocols for *mutual* PSI (as well as for additional set operations), and may involve more than two players. Protocols are secure in the standard model against

[1] Recall that encryption and decryption in the Paillier cryptosystem [28] involve exponentiations mod n^2: if $|n| = 1024$ bits, then $|n^2| = 2048$ bits (being n the public modulus).

[2] In the Exponential ElGamal variant, encryption of message m is computed as $E_{g,y}(m) = (g^r, y^r \cdot g^m)$ instead of $(g^r, m \cdot y^r)$, for random r and public key y.

semi-honest and also malicious adversaries. The former incurs quadratic ($O(vw)$) computation (but linear communication) overhead. The latter uses (expensive) generic zero-knowledge proofs to prevent parties from deviating to the protocol. Also, note that it is not clear how to enable *data transfer* in their PSI solution.

- Dachman-Soled, et al. [13] present an improved PSI construction, based on [26]. Their construction incorporates a secret sharing of polynomial inputs. Since Shamir's secret sharing [31] implies Reed Solomon codes, they do not need generic zero-knowledge proofs. Complexity of their protocol amounts to $O(wk^2 \log^2(v))$ in communication and $O(wvk \log(v) + wk^2 \log^2(v))$ in computation, being k the security parameter.

Other techniques rely on so-called *Oblivious Pseudo-Random Functions* (OPRF-s), introduced in [18]. An OPRF is a two-party protocol 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 $f_k(x)$. Most prominent OPRF-based protocols include:

- Hazay and Lindell [21] propose the first PSI construction based on OPRF-s. The intuition is the following: 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 $f_k(c_i) \in \mathcal{U}$.

  They propose one solution with security in the presence of malicious adversaries with one-sided simulatability, and another secure in the presence of covert adversaries [2].

- Jarecki and Liu [24] improve the previous solution, proposing 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, namely the Camenisch-Shoup cryptosystem (CS) [10].

  As pointed out in [24], this solution can be further optimized, leading to the work by Belenkiy, et al. [3]. In fact, the OPRF construction could work in groups with a 160-bit prime order, unrelated to the RSA modulus, instead of the more expensive composite order groups. Assuming this improved construction, [24] incurs the following computational complexity: S needs to perform $O(w)$ PRF evaluations, specifically $O(w)$ modular exponentiations of m-bit exponents mod n^2—where m is the number of bits needed to represent each entry and n^2 is typically 2048 bits. 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. Finally, S computes *online* $O(v)$ CS decryptions—i.e., $O(v)$ 1024-bit exponentiations mod 2048 bits. As discussed in [24], complexity in the malicious model grows by a factor of 2.

  Note that one limitation is that the input domain size of the pseudorandom function should be polynomial in the security parameter. In fact, the security proof makes use of the ability to exhaustively search over the input domain.

- Finally, the work-in-progress in [25] relies on a similar concept – *Unpredictable Functions* (UPFs). A specific UPF, $f_k(x) = (H(x))^k$, is used as a basis for two-party computation (in ROM), with S contributing the key k and 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 secure computation of *Adaptive* Set Intersection, under the *One-More-Gap-DH* assumption in ROM [4]. Note that this solution is similar to prior techniques in [23] and [17]. The computational complexity of the UPF-based PSI (in presence of honest-but-curious adversaries) amounts to $O(w + v)$ exponentiations with short exponents at server side and $O(v)$ at client side (e.g., 160-bit mod 1024-bit).

4.2 Protocols related to APSI

We now briefly review prior work related to *Authorized Private Set Intersection*.

Privacy-preserving Policy-based Information Transfer (PPIT). De Cristofaro, et al. introduce the concept of PPIT [15] 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. PPIT ensures that: (1) client only gets the information it is entitled to, and (2) 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. [15] describes three PPIT protocols, based respectively on: RSA [29] and Schnorr [30] signatures, and Identity-based Encryption (IBE) [6]. A client authorization is, respectively, an RSA, Schnorr, or BLS [7] signature on a record identifier. As shown in [15], it is easy to extend RSA- and Schnorr-PPIT to support the case of the client holding multiple authorizations. Thus, one could 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 the construction from the RSA-PPIT in Section 5.3 as it serves as a starting point for our paper.

Identity-based Encryption (IBE). [15] shows that PPIT can be constructed from IBE [6]: the resulting (computational and communication) complexity is linear, even in the case of the client holding multiple authorizations. Thus, one could obtain efficient (linear) APSI from IBE-PPIT, using a tagging technique. However, such a construction would allow the client to violate server privacy with respect to past (recorded) interactions, using any authorizations obtained at a later time. This is not the case for the RSA-based construction, which is hence defined in [15] as *forward secure*.[3]

Similarly to PPIT, the problem of APSI can be solved using IBE-based *Public-key Encryption with Keyword Search* (PEKS) [5, 1] (or, similarly, searchable encrypted logs [32]). The server could use a PEKS scheme to produce encryptions of keywords (i.e., set elements). The client can then test a matching keyword only if equipped with a corresponding trapdoor—i.e., the authorization. We remark that: (1) server learns nothing about client's trapdoors, and (2) client learns nothing about keywords not matching its searches. This is the intuition of the work in [9], which shows a modified PEKS construct. It additionally hides client keyword from the authorization authority and proves security against malicious adversary in the standard model. Nonetheless, note that the PEKS Test algorithm requires the client to test each trapdoor against each encrypted keyword it receives, thus incurring a quadratic overhead. Furthermore, note that [9] is built on top of the expensive Boyen-Waters (BW) IBE scheme [8] (each BW encryption require 6 exponentiations and 6 group elements, and each BW decryption requires 5 bilinear map operations). Note that, similarly to IBE-PPIT, an APSI construction from a PEKS scheme would not guarantee *forward security*. Besides, we will present a linear APSI solution (secure under the RSA assumption) that does not use bilinear map groups nor relies on the Bilinear Diffie Hellman (BDH) assumption [6].

Certified Sets. Finally, Camenisch and Zaverucha [11] introduce the notion of *Certified Sets* (independently from PPIT [15]). This construct allows a trusted third party to ensure that all protocol inputs are valid and bound to each protocol participant. The proposed protocol is *mutual* (i.e., both parties receive the intersection) and builds upon oblivious polynomial evaluation and achieves quadratic computation and communication overhead.

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.

[3]This definition of forward security is not to be confused with forward secrecy, i.e., preventing a *third-party* adversary corrupting the server from learning data encrypted prior to corruption.

$a \leftarrow A$	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
$H()$	cryptographic hash function, codomain depends on context
$H'()$	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

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, leaving modified constructions and proofs in the malicious model as part of future work.

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, 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 [15]. This protocol is only sketched out in [15]; since our new protocols are loosely based on it, we specify it in Figure 1.

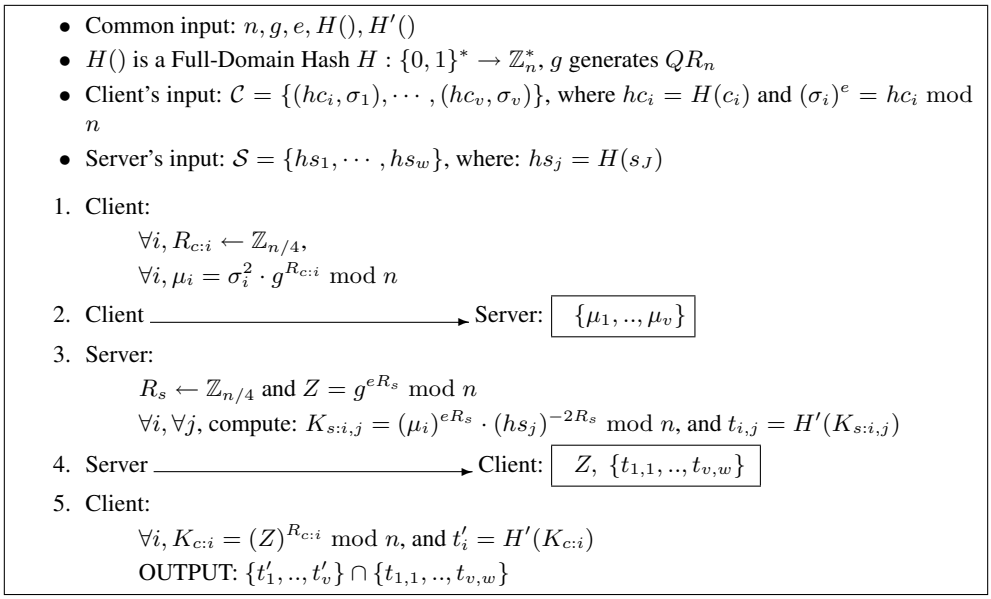

Figure 1: APSI Protocol derived from RSA-PPIT

Actually, the protocol in [15] is APSI-DT; however, for ease of illustration we omit the data transfer component at this point. 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, $hc_i = hs_j$):

$$K_{c:i} = (Z)^{R_{c:i}} = g^{eR_s \cdot R_{c:i}}$$

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

We point out that the protocol in Figure 1 incurs in quadratic computation overhead by the server and quadratic communication.

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 communication overhead.

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 communication 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) $c_i = s_j$, and so $hc_i = hs_j$, we obtain:

$$\begin{aligned} K_{s:j} = &\ (X^e/{hs_j}^2)^{R_s} = [(PCH^{*2} \cdot g^{R_c})^e/{hs_j}^2]^{R_s} = (PCH^2/{hs_j}^2 \cdot g^{eR_c})^{R_s} &= (PCH_i)^{2R_s} \cdot g^{eR_cR_s} \\ K_{c:i} = &\ y'_i \cdot Z^{R_c} \cdot Z^{-R_{c:i}} = (PCH_i^*)^{2eR_s} \cdot g^{R_{c:i}eRs} \cdot g^{eR_sR_c} \cdot g^{-eR_sR_{c:i}} &= (PCH_i)^{2R_s} \cdot g^{eR_cR_s} \end{aligned}$$

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

We claim that the APSI Protocol in Figure 2 is a: (1) Server-Private (APSI), (2) Client-Private, (3) Client-Unlinkable, and (4) Server-Unlinkable APSI. (See Appendix B.1).

- Common input: $n, g, e, H(), H'()$
- $H()$ is a Full-Domain Hash $H : \{0,1\}^* \rightarrow \mathbb{Z}_n^*$, g generates QR_n
- Client's input: $\mathcal{C} = \{(hc_i, \sigma_1), \cdots, (hc_v, \sigma_v)\}$, where: $hc_i = H(c_i)$ and $(\sigma_i)^e = hc_i \bmod n$
- Server's input: $\mathcal{S} = \{hs_1, \cdots, hs_w\}$, where: $hs_j = H(s_j)$

1. Client:
   $PCH = \prod_{i=1}^{v} hc_i$ **and** $PCH^* = \prod_{i=1}^{v}(\sigma_i) = \prod_{i=1}^{v}({hc_i}^d)$
   $R_c \leftarrow \mathbb{Z}_{n/4}$ and $X = (PCH^*)^2 \cdot (g^{R_c})$
   $\forall i, PCH_i^* = PCH^*/\sigma_i$, and $R_{c:i} \leftarrow \mathbb{Z}_{n/4}$, $y_i = (PCH_i^*)^2 \cdot (g^{R_{c:i}})$
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: $K_{s:j} = (X^e/{hs_j}^2)^{R_s}$, 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: $\{t_1', .., t_v'\} \cap \{t_1, .., t_w\}$

Figure 2: APSI Protocol with linear complexity

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 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 communication overhead – $O(v+w)$, 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 APSI protocol in Figure 2). Our linear PSI construct is given in Figure 3:

- Common input: $p, q, g, H(), H'()$, where $q|p-1$ but q^2 does not divide $p-1$
- g generates a subgroup in $\mathbb{Z}_p^*$ of order q, $H : \{0,1\}^* \rightarrow \mathbb{Z}_q^*$
- 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: $hs_j = H(s_j)$

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 $\longrightarrow$ 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 $\longrightarrow$ 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: $\{t_1', .., t_v'\} \cap \{t_1, .., t_w\}$

Figure 3: PSI Protocol with linear complexity

To see that the protocol is correct, observe that, for any c_i held by the client and s_j held by the server, if $c_i = s_j$, and so $hc_i = hs_j$, we obtain:

$$K_{s:j} = (X/hs_j)^{R_s} = [(PCH \cdot g^{R_c})/hs_j]^{R_s} = (PCH_i)^{R_s} \cdot g^{eR_cR_s}$$
$$K_{c:i} = y'_i \cdot Z^{R_c} \cdot Z^{-R_{c:i}} = (PCH_i)^{R_s} \cdot g^{R_{c:i}Rs} \cdot g^{R_sR_c} \cdot g^{-R_sR_{c:i}} = (PCH_i)^{R_s} \cdot g^{eR_cR_s}$$

The resulting protocol (in Figure 3) is a: (1) Server-Private, (2) Client-Private, (3) Client-Unlinkable, and (4) Server-Unlinkable PSI (see Appendix B.2).

Note. Our work-in-progress proofs against fully malicious players for protocols in Figures 2 and 3 seem to depend on the product of hashes, i.e., the PCH structure. However, for HbC adversaries these protocols can be described in a simplified version reported in Appendix C.1 and C.2, for sake of completeness,

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 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, in Figure 4, we present an even more efficient PSI protocol, where the client does not perform any modular exponentiations on-line but only $O(v)$ on-line modular multiplications (Step 7).

- Common input: $n, e, H(), H'()$
- $H()$ is a Full-Domain Hash $H : \{0,1\}^* \rightarrow \mathbb{Z}_n^*$
- 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: $hs_j = H(s_j)$

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 $y_i = hc_i \cdot (R_{c:i})^e \bmod n$

ON-LINE:

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

Figure 4: Blind RSA-based PSI Protocol with linear complexity

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 Oblivious Keyword Search [27]. However, we adapt it for the PSI scenario: instead of encrypting a string of 0's the server reveals the key as the hash of the signature for all elements in her set. The resulting protocol (in Figure 4) is a: (1) Server-Private, (2) Client-Private, and (3) Client-Unlinkable PSI (see Appendix B.3).

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:

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

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 at the server side. 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 (using $e = 3$), 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 Figure 1, 2, 3 and 4, and provide APSI-DT and PSI-DT. Following the approach outlined in [15], 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 CPA-secure symmetric cipher, such as AES [14] used in the appropriate operation mode. 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.

6 Evaluation

In this section, we highlight the differences between existing PSI techniques and protocols proposed in this paper. We focus on asymptotic complexities as for (1) involved communication overhead and (2) server and client computation (in terms of most expensive operations).

Let w and v 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 and k – a security parameter. We only count the number of *online* operations. Techniques are compared in Table 2, choosing parameters that achieve similar degrees of security. The first four rows refer to APSI protocols, whereas, the last six – to PSI. For sake of completeness, the table also includes communication overhead.

When examining Table 2, observe the following facts:

- We compare (A)PSI techniques providing *data transfer*, when possible. Recall from Section 4.1 that the PSI in [19] may use shorter exponents and moduli if data transfer is not enabled.
- Solutions in [11] and [26] provide, respectively, *mutual* APSI and PSI, *without* data transfer, whereas our solutions propose one-way (A)PSI techniques with data transfer.
- [22] and [26] provide solutions secure either in presence of HbC or malicious adversaries. We report security against malicious parties, but we annotate complexities of solutions for HbC parties.
- Each encryption in [8] (i.e., Boyen-Waters IBE) requires 6 exponentiations and a representation of 6 group elements, and each decryption requires 5 bilinear map operations.
- The complexity of PSI protocols secure in presence of malicious adversaries given in [24] and [25] actually grows by a factor of (at least) 2.

- Exponentiations in the APSI of Fig.2 can actually be performed using much shorter exponents (e.g., 512-bit), although a formal proof of security equivalence is part of our future work.

Protocol	Model	Adv	Com	Server Prec	Server Ops	Client Ops	Mod
[9]	Std	Mal	$O(w)$	-	$O(w)$ enc's in [8]	$O(vw)$ dec's in [8]	
[11]	Std	Mal	$O(vw)$	-	$O(wv)$ 160-bit exps	$O(wv)$ 160-bit exps	1024
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 in [19]	Std	HbC	$O(v+w)$	-	$O(w \log\log v)$ m-bit exps	$O(v+w)$ 1024-bit exps	2048
PSI in [22]	Std	Mal	$O(v+w)$	-	$O(v+w(\log\log v))$ 160-bit exps	$O(v+v)$ 160-bit exps	1024
PSI in [26]	Std	Mal	$O(v+w)$	-	$O(wv)$ m-bit exps	$O(wv)$ m-bit exps	2048
PSI in [24]	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 in [25]	ROM	Mal	$O(v+w)$	$O(w)$ 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	$O(v)$ 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

Table 2: Performance Comparison of PSI and APSI protocols.

6.1 Experimental Analysis

All protocols proposed in this paper have been implemented in ANSI C (using the well-known OpenSSL library [33]) and tested on a Dell Precision PC on a 2.33GHz CPU and 8GB RAM. The prototype's code is available upon request.[4]

To confirm 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 variant for the former. We omit run-times for operations that can be precomputed. We do not measure all prior techniques discussed in Section 4: we only compare our solutions to those available offering linear complexities – the APSI adaptation from RSA-PPIT showed in Figure 1, and PSI-s from [24] and [25]. Remark that, since the efficiency of [24] is influenced by records' length, we assume a conservative stance and choose items to be 160-bit long, similar to the output of a hash function.

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 [24]	5	24	27,654	118,676	27,862	118,947
PSI [25]	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

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

Measured *online* computation overhead for 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 [15] (showed in Figure 1), when both server and client have sets of size $5,000$. (This advantage increases for larger set sizes). Looking at PSI protocols, the toss-up is between protocols in Figure 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.

[4]Interested readers may contact Emiliano De Cristofaro at edecrist@ics.uci.edu.

7 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 malicious parties, as well as extensions to a group setting.

Acknowledgements. This research was supported by the U.S. Intelligence Advanced Research Projects Activity (IARPA) under grant #: FA8750-09-2-0071. We would also like to thank Nikita Borisov, Stanislaw Jarecki, Xiaomin Liu, Markulf Kohlweiss, and Jihye Kim for the helpful discussion in the early stage of protocol design.

References

[1] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi. Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions. *Journal of Cryptology*, 21(3):350–391, 2008.

[2] Y. Aumann and Y. Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. In *TCC'07*, pages 137–156, 2007.

[3] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Randomizable Proofs and Delegatable Anonymous Credentials. In *CRYPTO'09*, 2009.

[4] 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.

[5] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key Encryption with Keyword Search. In *Eurocrypt'04*, pages 506–522, 2004.

[6] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.

[7] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.

[8] X. Boyen and B. Waters. Anonymous Hierarchical Identity-Based Encryption (Without Random Oracles). In *Crypto'06*, pages 290–307, 2006.

[9] 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.

[10] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO'03*, pages 126–144, 2003.

[11] J. Camenisch and G. Zaverucha. Private intersection of certified sets. In *Financial Cryptography and Data Security'09*, 2009.

[12] D. Chaum. Blind signatures for untraceable payments. In *Advances in Cryptology: Proceedings of Crypto*, volume 82, pages 199–203, 1983.

[13] D. Dachman-Soled, T. Malkin, M. Raykova, and M. Yung. Efficient Robust Private Set Intersection. In *ACNS'09*, pages 125–142. Springer, 2009.

[14] J. Daeman and V. Rijmen. AES proposal: Rijndael. 1999.

[15] E. De Cristofaro, S. Jarecki, J. Kim, and G. Tsudik. Privacy-Preserving Policy-Based Information Transfer. In *PETS'09*, pages 164–183, 2009.

[16] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on Information Theory*, 31(4):469–472, 1985.

[17] A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *PODS'03*, pages 211–222, 2003.

[18] M. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *TCC'05*, pages 303–324, 2005.

[19] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Eurocrypt'04*, pages 1–19, 2004.

[20] D. Freeman. Pairing-based identification schemes. *Arxiv preprint cs/0509056*, 2005.

[21] 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.

[22] C. Hazay and K. Nissim. Efficient Set Operations in the Presence of Malicious Adversaries. In *PKC'10*, 2010.

[23] B. Huberman, M. Franklin, and T. Hogg. Enhancing privacy and trust in electronic communities. In *ACM Conference on Electronic Commerce*, pages 78–86, 1999.

[24] 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.

[25] S. Jarecki and X. Liu. Fast Secure Computation of Set Intersection. Manuscript available from the authors, 2009.

[26] L. Kissner and D. Song. Privacy-preserving set operations. In *CRYPTO'05*, pages 241–257, 2005.

[27] W. Ogata and K. Kurosawa. Oblivious keyword search. *Journal of Complexity*, 20(2-3):356–371, 2004.

[28] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt'99*, pages 223–238, 1999.

[29] 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.

[30] C. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[31] A. Shamir. Identity-based cryptosystems and signature schemes. In *Crypto'84*, pages 47–53, 1984.

[32] B. Waters, D. Balfanz, G. Durfee, and D. Smetters. Building an encrypted and searchable audit log. In *NDSS'04*, 2004.

[33] E. Young and T. Hudson. OpenSSL: The Open Source toolkit for SSL/TLS. `http://www.openssl.org`.

APPENDIX

A 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 [20] 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

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

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

One-More-RSA assumption. Informally, the One-More-RSA assumption [4] 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

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

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

B Proofs

B.1 Proofs of APSI protocol in Figure 2

We now consider security and privacy properties of the protocol in Figure 2.

Client and Sever Unlinkability. We argue that the use of different randomness across multiple interactions (R_s at the server and Rc and Rc_i's at client) yields server and client unlinkability. We defer the formal proofs of unlinkability to a future extension of the paper.

Client Privacy. Recall that APSI is client-private if no information is leaked to the server about client's private inputs. In the following description, we use $U \approx_S V$ to denote that distribution U is statistically close to V in the sense that the difference between these distributions is at most $O(2^\tau)$. It is easy to show that, given that, in Step 1, the client selects all values uniformly and at random, i.e., $[R_c, \{R_{c:1}, ..., R_{c:v}\}] \leftarrow \mathbb{Z}_{n/4}$, g generates QR_n, and $\mathbb{Z}_{n/4} \approx_S \mathbb{Z}_{p'q'}$, it holds that $\{X = (PCH^*)^2 \cdot g^{R_c}\} \approx_S QR_n$ and $\{y_i = (PCH_i^*)^2 \cdot g^{R_{c:i}}\} \approx QR_n$, for all $i = 1, \ldots, v$.

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 (presented in Appendix A).

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: $K^* = (X^*)^{eR_s} \cdot (h^*)^{-2R_s} \bmod N$, 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 $m \leftarrow \mathbb{Z}_{n/4}$, 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 $K^* = (X^*)^{e(d+m)}(h^*)^{-2(d+m)}$ from $\mathcal{A}$. Since RCh knows X^*, h^*, e, and m, it can compute $(h^*)^d$.

B.2 Proofs of PSI protocol in Figure 3

Client and Sever Unlinkability. We argue that the use of different randomness across multiple interactions (R_s at the server and Rc and Rc_i's at client) yields server and client unlinkability. We defer the formal proofs of unlinkability to a future extension of the paper.

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, given that, in Step 1, the client selects all values uniformly and at random, i.e., $[R_c, \{R_{c:1}, ..., R_{c:v}\}] \leftarrow \mathbb{Z}_q^*$, it holds that $\{X = PCH \cdot g^{R_c}\}$ and $\{y_i = PCH_i \cdot g^{R_{c:i}}\}$ (for all $i = 1, \ldots, v$) are uniformly random in $\mathbb{Z}_q^*$, hence no information is leaked about client's inputs.

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 (presented in Appendix A) 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-Gap-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.

B.3 Proofs of PSI Protocol in Figure 4

Client Unlinkability. We argue that the use of different randomness across multiple interactions (Rc_i's at client) yields client unlinkability. We defer the formal proofs of unlinkability to a future extension of the paper.

Client Privacy. We claim it is easy to show that client's inputs to the protocol are statistically close to random distribution. We defer formal proof for future extension of the paper, as it directly derives from the security argument of blind RSA signatures [12].

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 A) 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 resembles the one of UPF-based protocol, secure under the One-More-Gap-DH assumption from [25], as well as the notion of *adaptiveness* for PSI. Adaptiveness allows the client to adaptively make queries, i.e., she does not need to specify all her inputs at once. In fact, the signing

algorithm of unique signature is indeed an unpredictable function, hence its hash in the random oracle model yields a PRF.

C Simplified Descriptions

C.1 APSI in Figure 2

- Common input: $n, g, e, H(), H'()$
- $H()$ is a Full-Domain Hash $H : \{0,1\}^* \to \mathbb{Z}_n^*$, g generates QR_n
- Client's input: $\mathcal{C} = \{(hc_i, \sigma_1), \cdots, (hc_v, \sigma_v)\}$, where: $(\sigma_i)^e = hc_i \bmod n$, and $hc_i = H(c_i)$
- Server's input: $\mathcal{S} = \{hs_1, \cdots, hs_w\}$, where: $hs_j = H(s_j)$

1. Client:
   $R_c \leftarrow \mathbb{Z}_{n/4}$ and $X = g^{R_c}$
   $\forall i$, $R_{c:i} \leftarrow \mathbb{Z}_{n/4}$, $y_i = {\sigma_i}^2 \cdot g^{R_{c:i}}$
2. Client $\longrightarrow$ Server: $\boxed{X, \{y_1, .., y_v\}}$
3. Server:
   $R_s \leftarrow \mathbb{Z}_q^*$ and $Z = g^{eR_s}$
   $\forall\, j$, compute: $K_{s:j} = (X^e \cdot {hs_j}^2)^{R_s}$, 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: $\{t'_1, .., t'_v\} \cap \{t_1, .., t_w\}$

Figure 5: Simplified APSI Protocol with linear complexity

C.2 PSI in Figure 3

- Common input: $p, q, g, H(), H'()$, where $q|p-1$ but q^2 does not divide $p-1$
- g generates a subgroup in $\mathbb{Z}_p^*$ of order q, $H : \{0,1\}^* \to \mathbb{Z}_q^*$
- 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: $hs_j = H(s_j)$

1. Client:
   $R_c \leftarrow \mathbb{Z}_q^*$ and $X = g^{R_c}$
   $\forall i$, $R_{c:i} \leftarrow \mathbb{Z}_q^*$, $y_i = hc_i \cdot g^{R_{c:i}}$
2. Client $\longrightarrow$ Server: $\boxed{X, \{y_1, .., y_v\}}$
3. Server:
   $R_s \leftarrow \mathbb{Z}_q^*$ and $Z = g^{R_s}$
   $\forall\, j$, compute: $K_{s:j} = (X \cdot hs_j)^{R_s}$, and $t_j = H'(K_{s:j})$
   $\forall\, i$, compute: $y'_i = (y_i)^{R_s}$
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: $\{t'_1, .., t'_v\} \cap \{t_1, .., t_w\}$

Figure 6: Simplified PSI Protocol with linear complexity