Privacy-Preserving Sharing of Sensitive Information is (Really) Practical

Emiliano De Cristofaro, Yanbin Lu, Gene Tsudik
University of California, Irvine

Abstract

The need for controlled sharing of sensitive information occurs in many realistic everyday scenarios, ranging from critical (e.g., national security) to mundane (e.g., social networks). A typical scenario involves two parties, at least one of which seeks some information from the other. The latter is either willing, or compelled, to share information. This poses two challenges: (1) how to enable this type of sharing such that parties learn no (or minimal) information beyond what they are entitled to, and (2) how to do so efficiently, in real-world practical terms. The first challenge has been addressed by prior work that yielded cryptographic techniques, such as Private Set Intersection (PSI) protocols. However, such tools have only very recently become efficient enough for actual deployment.

In this paper, we show how some cryptographic privacy techniques are implemented in a real working system, called PSST: Privacy-preserving Sharing of Sensitive information Toolkit. PSST functions as a *privacy shield* that protects parties from disclosing their respective sensitive information. The design and deployment of PSST prompts a number of new and interesting practical challenges, which we address in this paper. We describe the menu of services offered by PSST and present experimental results that attest to the practicality of the attained privacy features.

1 Introduction

In today's increasingly digital world, entities (companies, agencies, and individuals) are often confronted with a dilemma regarding information privacy. On one hand, they need to keep sensitive data confidential. On the other hand, they might be either motivated or forced to share some of that data. There are several scenarios where this dilemma is impossible to resolve, unless (at least) one party sacrifices some privacy. Consider the following examples:

Airline Safety. Department of Homeland Security (DHS) checks whether any passenger on each flight from/to the United States must be denied boarding or disembarkation, based on several secret lists, including: the *No Fly List* [54], the *Terrorist Watch List* [27], and the *Secondary Security Screening Selection List* [56]. Today, airlines surrender their entire passenger manifests to DHS, together with a large amount of sensitive information, such as credit card numbers [51]. Besides its obvious privacy implications, this *modus operandi* poses liability issues with regard to (mostly) innocent passengers' data and concerns about possible data loss[1]. Ideally, DHS would obtain information pertaining *only* to passengers on one of those lists, without disclosing any information to the airlines.

Social Networking. A social network user (Alice) wants to find out whether there are any other users nearby with whom she has friends, interests or group memberships in common, without relying on a third-party provider, e.g., Facebook. Some of this information might be very sensitive, e.g., reveal Alice's medical issues or sexual orientation. Today, users like Alice would have to broadcast their information in order to discover a nearby match, thus compromising privacy. Whereas, they might be willing to disclose sensitive information only to users with a *matching* profile.

[1] See [16] for a litany of recent incidents where large amounts sensitive data were lost or mishandled by government agencies.

Healthcare. A health insurance company needs to retrieve information about an insured (client) from other parties, such as other insurance carriers or hospitals. The latter cannot provide any information on other patients and the former cannot disclose the identity of the target client.

Law Enforcement. An investigative agency (e.g., the FBI) needs to obtain electronic information about a suspect from other agencies, e.g., local police, the military, the DMV, the IRS, or the suspect's employer. In many cases, it is dangerous (or simply forbidden) for the FBI to disclose the subject of the investigation. For its part, the other party cannot disclose its entire data-set and trust the FBI to only extract desired information. Furthermore, FBI requests might need to be pre-*authorized* by some appropriate authorization authority (e.g., a federal judge). This way, the FBI can only obtain information related to authorized requests.

Other examples include recent developments in *collaborative* denial-of-service attacks identification [3] and botnet detection [43].

1.1 Roadmap & Contributions

Motivated by the above examples, this paper presents the design and implementation of the Privacy-preserving Sharing of Sensitive information Toolkit (PSST). PSST functions as a *privacy shield* that protects parties from disclosing more than the required minimum of their respective sensitive information. Our "roadmap" for the design of PSST is as follows:

1. We introduce the concept of privacy-preserving sharing of sensitive information and identify its privacy requirements and system assumptions. We then examine techniques for Private Set Intersection (PSI) and argue that they are well-suited as the basic building block of our system. We also discuss why other related cryptographic primitives do not meet efficiency, system, or privacy criteria.
2. We explore several PSI flavors, corresponding to different problems and scenarios and we motivate the need for a comprehensive toolkit that provides a *menu* of privacy-preserving operations targeted to sharing sensitive information. We include *optimal* implementations of most prominent PSI protocols, selecting those best-suited for different scenarios.
3. Next, we look at a realistic large-scale scenario that captures a general class of information sharing and mimics the operation of database querying. In the process of adapting PSI protocols, we discuss a number of encountered challenges. We then present and discuss the design of the PSST toolkit, which demonstrates, via real working code running on a common hardware/software platform, that protecting privacy in sharing sensitive information is both possible and practical.

Contributions. This paper makes several contributions. First, we show that PSI is the most appropriate and efficient building block for privacy-preserving sharing of sensitive information. We then identify a number of challenges in extending PSI to large-scale database applications, such as handling multi-sets, data pointers, high communication overhead as well as liability issues. Finally, we design an architecture that allows privacy-preserving interactions (similar to database querying), while overcoming above challenges and achieving negligible overhead compared to a standard MySQL-based database interface.

Toolkit. Our research was paralleled by an implementation effort that yielded the open-source PSST toolkit.[2] PSST provides users and application developers with several notable features. It includes *optimized* implementations of several prominent PSI protocols. Since they achieve somewhat different privacy properties (along different overhead and underlying assumptions), our toolkit allows users to select the PSI technique best-suited to the specific application. Finally, PSST provides a versatile privacy-preserving querying system, that combines efficiency, usability, and provably-secure privacy guarantees for both querier and database server.

[2] PSST toolkit source is available at `http://code.google.com/p/psst-toolkit/`.

Paper Organization. Section 2 overviews state-of-the-art PSI protocols, presents privacy definitions, and motivates choosing PSI as the main building block. Then, section 3 focuses on the efficient implementation (and benchmarking) of PSI protocols. Next, Section 4 discusses some challenges stemming from adapting PSI protocols to realistic scenarios, and presents the system architecture aimed at overcoming these challenges. Section 5 summarizes our contribution and concludes with a laundry-list of future work items. Appendix A presents the details of all PSI protocols considered in the paper.

2 Preliminaries

Since the PSST toolkit builds upon PSI protocols, we now review relevant definitions, privacy properties, and discuss several related primitives that motivate the choice of PSI as our main building block.

2.1 Private Set Intersection (PSI)

The term "Private Set Intersection" has been used to denote a family of cryptographic protocols providing different privacy flavors, rather than a single primitive. We distinguish among the following variants:

- *Plain PSI.* Two parties compute the intersection of their private sets such that nothing is learned other than the intersection itself and, perhaps, the size of the respective sets. We also call this variant *mutual* PSI.
- *One-way PSI.* Only one party (client) learns the intersection, while the other party (server) learns nothing beyond the size of client set. Assuming that no party aborts the protocol prematurely, mutual PSI can be trivially obtained via two executions of one-way PSI. In the rest of this paper, we focus on the one-way variant and hence use *PSI* to mean *one-way PSI*.
- *PSI with Data Transfer (PSI-DT).* If the server has data associated with each element in its set, this data must be transferred to the client whenever the corresponding element is in the intersection. This variant (introduced in [22]) is appropriate in many realistic scenarios, where the server holds a set of records (as opposed to a simple list of items).
- *Authorized PSI (APSI).* One assumption implicit in all PSI protocols is that parties do not manipulate their input sets.[3] In other words, initial inputs are assumed to be somehow valid. APSI [22] relaxes this assumption by ensuring that client input is *authorized* by an appropriate authorization authority (denoted as **CA**). Thus, unless the client holds an authorization (typically, in the form of a digital signature), it does not learn whether the corresponding input is in the set intersection. At the same time, the server does not learn whether client input is authorized, i.e., verification is performed obliviously.
- *Authorized PSI-DT (APSI-DT).* Similar to its PSI counterpart, APSI-DT can be defined from APSI by adding data transfer for each element in the intersection.

In the rest of the paper, we focus on APSI-DT and PSI-DT, as they are general and appeal to many realistic application settings. Looking back at Section 1, Healthcare and Airline Safety examples correspond to PSI-DT. Whereas, the Law Enforcement example is better suited for APSI-DT (i.e., the FBI requests must be pre-authorized by a court).[4]

We now formally define PSI-DT and APSI-DT. Our current notation is self-explanatory; a more formal version is in Appendix A.

Definition 1 Private Set Intersection with Data Transfer (PSI-DT) [22]: *a protocol involving Server, on input of a set of w items, each with associated data record, $\mathcal{S} = \{(s_1, data_1), \cdots, (s_w, data_w)\}$, and Client, on input of a set of v items, $\mathcal{C} = \{c_1, \cdots, c_v\}$. It results in Client outputting $\{(s_j, data_j) \in \mathcal{S} \mid \exists c_i \in \mathcal{C}\ s.t.\ c_i = s_j\}$.*

[3] Note that such an assumption occurs not only in presence of honest-but-curious but also arbitrarily malicious adversaries.
[4] However, the Social Networking example best matches mutual/plain PSI.

Definition 2 Authorized Private Set Intersection (APSI-DT) [22]: *a protocol between Server, on input of a set of w items: $\mathcal{S} = \{(s_1, data_1), \cdots, (s_w, data_w)\}$, and Client, on input of a set of v items with associated authorizations, $\mathcal{C} = \{(c_1, \sigma_i) \cdots, (c_v, \sigma_v)\}$. It results in Client outputting $\{(s_j, data_j) \in \mathcal{S} \mid \exists (c_i, \sigma_i) \in \mathcal{C}\ s.t.\ c_i = s_j \wedge \mathsf{Vrfy}_{pk}(\sigma_i, c_i) = 1\}$, (where pk is the authority's public key).*

2.2 Desired Properties

Based on our discussion in Section 2.1, PSI (in its several flavors) can address privacy issues outlined in Section 1. In Section 3, we show that recent work on PSI has yielded practical solutions, with linear computation and communication complexities and inexpensive cryptographic operations.

We now discuss desired privacy properties. We assume that all interactions occur in the presence of an *Honest-but-Curious* adversary. Specifically, we assume that parties always faithfully follow protocol specifications and do not misrepresent any information related to their inputs, e.g., size and content. However, during or after protocol execution, any party might attempt to infer additional information about the other party's input.

Desired properties are as follows:

- *Correctness.* At the end of the interaction, Client outputs all items in the set intersection, along with associated data records, as specified in the definitions above.
- *Client Privacy.* Server learns no information about Client input, except its size.
- *Server Privacy.* Client learns no information about any Server input that is NOT in the intersection of their respective sets, except the size of $\mathcal{S}$.
- *Server Privacy (with Authorization).* Client learns no information about any Server input that is NOT in the intersection of their respective sets, where Client set ($\mathcal{C}$) contains only items authorized by the authorization authority.
- *Forward Security (with Authorization).* Client cannot violate Server privacy for any past interactions, using any authorizations obtained at a later time.
- *Client Unlinkability.* Server cannot determine whether any two interactions with Client are related, i.e., executed on the same inputs.
- *Server Unlinkability.* Client cannot determine whether any two interactions with Server are related, i.e., executed on the same inputs.

While *privacy* requirements are intuitive, forward security and unlinkability are more subtle. Nonetheless, they are needed in several scenarios. Consider forward security in the context of authorized Client inputs. Suppose that an FBI agent engages in an interaction without being authorized for a given "suspect": Server privacy prevents the agent from accessing any information on the suspect. Later, the agent obtains authorization. Unless forward security is guaranteed, the agent can use the authorization to recover the suspect's file from the previous interaction.[5] Unlinkability, beyond keeping from noticing changes in parties' inputs, minimizes possibility of privacy leaks. Unless it is guaranteed, if inputs of one interaction are leaked, then all linked inputs would be leaked.

2.3 Related Primitives

Several cryptographic primitives provide privacy properties comparable to those in Section 2.1. Below, we discuss related primitives and motivate the choice of PSI-DT and APSI-DT as our building block.

Secure Two-Party Computation. Two parties, with respective inputs x and y, respectively can use Secure Two-Party Computation (2PC) to privately compute the value of a public function f at point (x, y).

[5] Our definition of forward security is not to be confused by forward secrecy, i.e., preventing a *third-party* adversary corrupting Server from learning data encrypted prior to corruption.

Both parties learn $f(x, y)$ and nothing else. A general procedure for 2PC of any function expressed as a Boolean circuit is due to Yao [57]. Although one could implement PSI-DT with 2PC, this technique would incur impractical computation and communication complexities – at least quadratic, as pointed out in [30, 41].

Oblivious Transfer (OT). Introduced by Rabin [48], OT involves a sender with n secret messages and a receiver with one index i. The receiver wants to retrieve the i-th among sender's messages. The sender does not learn which message is retrieved, and the receiver learns no other message. OT privacy requirements resemble those of PSI-DT. However, in PSI-DT, inputs are items (e.g., keywords), whereas, in OT, the receiver needs to know (and input) an existing index – an unrealistic assumption for the applications we have in mind.

Private Information Retrieval (PIR). PIR [18] allows a client to retrieve an item from a server (public) database without revealing which item it is retrieving, with the additional requirement that communication overhead must be strictly lower than $O(n)$ (if n is the database size). Note that, in PIR, privacy of server's database is not protected – the client may receive items/records beyond those requested. Symmetric PIR (SPIR) [32] additionally offers server privacy, thus achieving OT with communication overhead lower than $O(n)$. However, similar to OT, a symmetric PIR client needs to know and input the index of the desired item in server's database. An extension to keyword-based retrieval is Keyword-PIR (KPIR) [17]. However, it is still focused on minimizing bandwidth, rather than optimizing computation and protecting server privacy. Thus, it incurs significantly higher computational overhead, as well as multiple rounds of PIR executions. We discuss PIR further in Section 3.2.

Private Keyword Search [29]. This primitive is akin to a special case of PSI-DT, where Client input is a singleton and Server input is a multi-set. Similarly, Oblivious Keyword Search (OKS) [44] allows Client to search for v different keywords. We discuss how to handle multi-sets using PSI-DT in Section 4.

Searchable Encryption. The problem of APSI-DT could be solved using *Public-key Encryption with Keyword Search* (PEKS) [9] (or, similarly, searchable encrypted logs [55]), based on IBE [10]. A sender can use a PEKS scheme to append encryptions of keywords (items) to encrypted data records. Whereas, a receiver can "test" a keyword (and obtain associated data) only if it has a corresponding trapdoor, i.e., an authorization. Consequently, (1) the sender learns nothing about receiver's trapdoors, and (2) the receiver learns nothing about keywords not matching its searches. This is the intuition of the work in [13], which shows a modified PEKS construct: it additionally hides receiver's keywords from the authority and offers security against malicious adversaries in the standard model. Nonetheless, PEKS Test algorithm requires the receiver to test each trapdoor against each encrypted keyword it receives, thus incurring *quadratic* computational overhead. Furthermore, [13] is built atop the relatively expensive Boyen-Waters IBE scheme [11]. Finally, searchable encryption has been studied in symmetric key settings [53]: users outsource encrypted data at an untrusted server and privately search over it. This is substantially different from our goals.

3 Implementing Efficient PSI-s

In this Section, we turn to efficient instantiation of several PSI-DT and APSI-DT protocols and compare their performance via experimental results. In the process, we also attempt to identify possible obstacles to practical use and discuss possible improvements and optimizations.

Note that, for readability's sake, details of the considered protocols are deferred to Appendix A. The evaluation presented below is necessary for our goal of constructing a practical and usable toolkit. However, those not interested in performance details may wish to skip to Section 4 without much loss of continuity.

One important variable in our discussion is authorization of Client input. Recall that the main difference between PSI-DT and APSI-DT is that, in the latter, Client input must be authorized.

Another variable is Server-side *pre-distribution*, i.e., whether Server can pre-process its inputs independent from Client inputs. If so, pre-processing can be done off-line and the results can be transferred

to Client (or published, for many clients) only once. Server-side pre-distribution is also a mandatory feature of so-called *adaptive protocols* [14], as it is needed to let Server *commit* to its input.

However, pre-distribution is incompatible with Server unlinkability (between two consecutive pre-distributions), since Server input is fixed during that period. Moreover, in APSI protocols, forward security can not be guaranteed: Client that obtains authorizations *after* interacting with the Server can still extract the *previous* intersection given the transcript of prior interaction. Therefore, for scenarios where Server input changes often and/or unlinkability is desired, protocols without pre-distribution are appropriate. Whereas, if Server input is (mostly) static and bandwidth overhead is critical, protocols with pre-distribution are preferred.

Candidate Protocols. We discuss efficient implementation of several (A)PSI-DT protocols (see Appendix A for details), and compare their performance and privacy properties:

	w/o Pre-Distribution	**w/ Pre-Distribution**
PSI-DT	FNP04 ([30]), DT10-1 (Fig.3 in [22])	JL09 ([39]), JL10 ([40]), DT10-2 (Fig.4 in [22])
APSI-DT	DT10-APSI (Fig.2 in [22])	IBE-APSI (Fig.5 in [21])

Table 1: PSI-DT and APSI-DT protocols included in the PSST toolkit.

3.1 Experimental Analysis

Each protocol was implemented in C++ using GMP (ver. 5.01) [28] and PBC (ver. 0.57) [42] libraries. All benchmarks were collected on a Ubuntu 9.10 desktop platform with Intel Core i7-920 (2.66MHz and 8MB cache) and 6GB RAM.

Evaluation Methods and Assumptions. For protocols supporting data transfer, data associated with each Server element can be arbitrarily long. Also, performance of some protocols is dominated by each element's data size, rather than set size. For a fair comparison, we aim to capture the intrinsic cost of each protocol. To this end, we employ the following strategy to eliminate data size effects: First, in all protocols, we encrypt each element's data with a distinct random symmetric key and consider these keys as the new associated data. Assuming that a different key is selected at each interaction, this technique does not violate Server unlinkability. This way, our evaluation is fair for all schemes, in that the computation cost of each protocol is measured based upon the same fixed-length key, regardless of data size. We set symmetric key size to 128 bits.

Besides the basic cost (incurred by all protocols) to transfer a key, each protocol execution involves additional overhead of symmetric en-/de-cryption of records. Figure 1 estimates this overhead for variable data sizes, using RC4 [50] and AES-CBC [20] with 128-bit keys. To estimate the total cost of a protocol, we just need to combine this overhead with the basic cost of each protocol and add data transfer delay for all Server encrypted data.

We further assume that Client does not perform any pre-computation, while Server performs as much pre-computation on its input as possible. This reflects the reality where Client input is (usually) determined in real time, while Server input is pre-determined. Figure 2 shows the pre-computation overhead for each protocol.

In the following, we only consider on-line computation overhead. Figures 3 and 4 show Client online computation overhead in terms of Client and Server input sizes, respectively. Figures 5 and 6 show Server online computation overhead in terms of Client and Server input size, respectively.

Figures 7 and 8 present protocol bandwidth complexity in terms of Client and Server input sizes. For protocols with pre-distribution, bandwidth consumption (since it is performed off-line) does not include pre-distribution overhead. Note that, in these figures, we sometimes use the same marker for different protocols to indicate that these protocols share the same value. Client input size v (or Server input size w) is fixed at $5,000$ in figures where x-axis refers to Server (or Client) input size.

Finally, in all experiments, we use a 1024-bit RSA modulus, a 1024-bit cyclic-group modulus and a 160-bit subgroup order. All test results are averaged over 100 independent runs. All protocols are

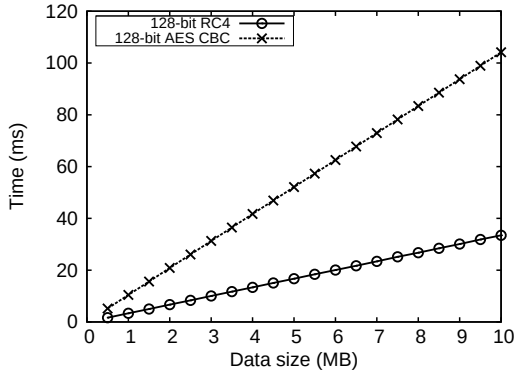


Figure 1: Symmetric key en-/de-cryption performance.

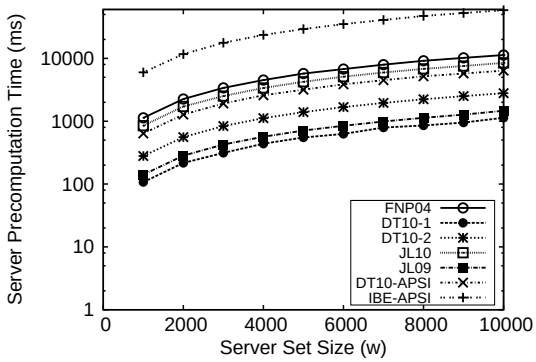


Figure 2: Server pre-computation overhead.

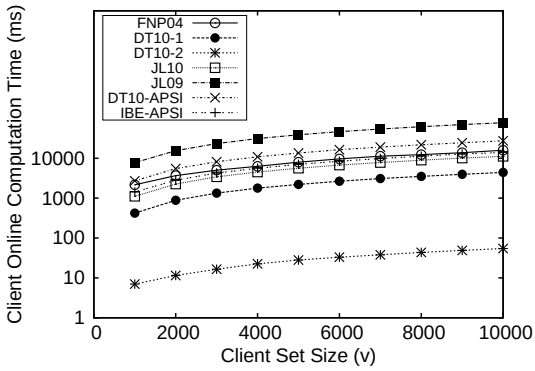


Figure 3: Client online computation w.r.t. Client set size.

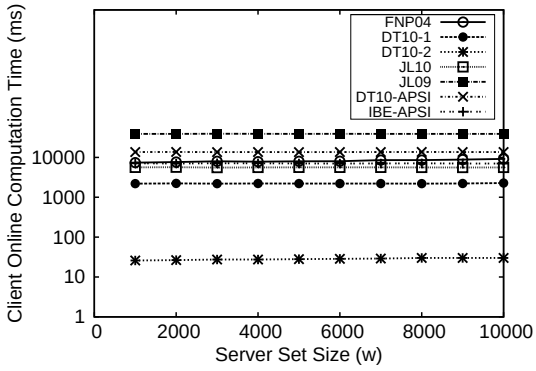


Figure 4: Client online computation w.r.t. Server set size.

instantiated under the assumption of *Honest-but-Curious* (HbC) adversaries and in the *Random Oracle Model* (ROM), as discussed in Section 2.2.

PSI-DT without pre-distribution. Here we compare FNP04 with DT10-1. Figures 3-8 show that that FNP04 is much costlier than DT10-1 in terms of Client and Server online computation as well as bandwidth consumption. For each Client set size, DT10-1 Client overhead varies from 460ms to 4400ms, while FNP04 Server overhead ranges between 1300ms and 15000ms. For each Server set size, DT10-1's Server overhead is below 1,300ms, while FNP04 Server overhead exceeds 15,000ms.

PSI-DT with pre-distribution. Next, we compare JL09, JL10 and DT10-2 as PSI-DT with pre-distribution. Recall that all protocols are instantiated in the HbC model, thus ZKPK's are not included for JL09 and JL10. Figures 3-8 show that DT10-2 has a overhead almost two orders of magnitude lower than JL09 and JL10 in terms of Client online computation. In fact, DT10-2 involves two Client multiplications for each Client item, while JL09 performs two heavy homomorphic operations and JL10 – two exponentiations. JL10 Server online computation overhead involves v 160-bit exponentiations, while DT10-2's Server overhead involves v RSA group exponentiations, that can be speeded up by using the Chinese Remainder Theorem. As a result, DT10-2 almost doubles JL10 Server online computation overhead. If we sum up Server and Client online computation overhead, DT10-2 remains the cheapest, while JL09 is the most expensive. In terms of bandwidth consumption, DT10-2 and JL10 are almost the same, while JL09 is more expensive.

APSI-DT without pre-distribution. The only protocol we evaluate for APSI-DT without data pre-distribution is DT10-APSI. Figure 3-6 illustrates that Client overhead is determined only by Client set size, whereas, Server overhead is determined by both Client and Server set sizes. Note that measurements obtained for APSI-DT naturally mirror those of DT10-1, as the former simply adds Client

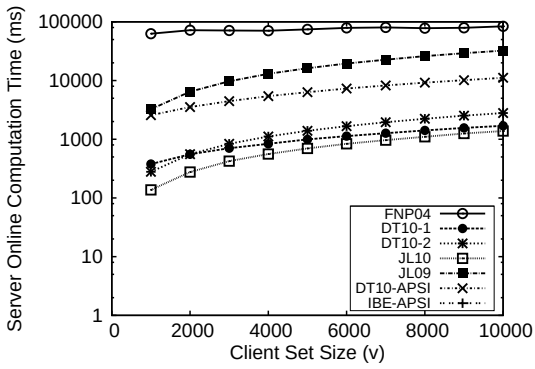


Figure 5: Server online computation w.r.t. Client set size.

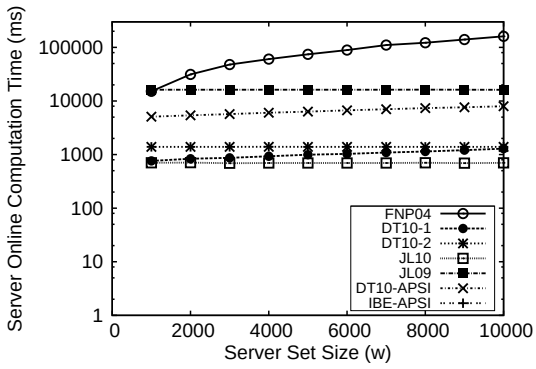


Figure 6: Server online computation w.r.t. Server set size.

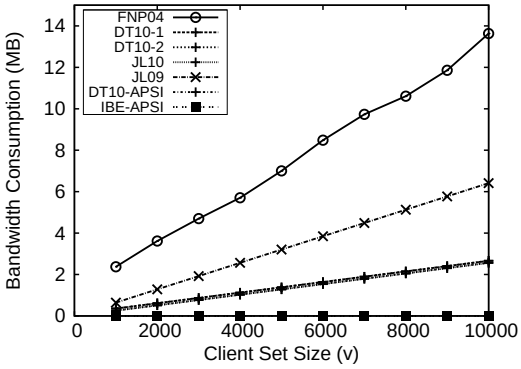


Figure 7: Bandwidth consumption w.r.t. Client set size.

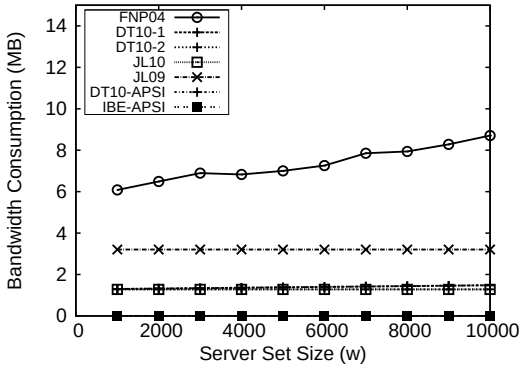


Figure 8: Bandwidth consumption w.r.t. Server set size.

authorization feature.

APSI-DT with pre-distribution. The only protocol we evaluate for APSI-DT with data pre-distribution is IBE-APSI. However, it does not provide forward security. Nonetheless, it is included in the evaluation, as we later show how to overcome this issue (see Section 4.2.1). Figure 3-4 shows that Client side overhead increases linearly with Client set size and does not depend on Server set size. In fact, Client only needs to compute v pairing operations. Recall that, in IBE-APSI, Server needs to compute pairing operations for each item, independent of Client input. Moreover, since these operations can be pre-computed, Server-side overhead and bandwidth consumption are negligible, as shown in Figures 5-8.[6]

During pre-computation, Server needs to compute w pairing and exponentiations, which makes pre-computation relatively expensive. If Server unlinkability is desired, Server would need to repeat, for every interaction, the operations otherwise performed only during pre-computation. As a result, if we sum up the overhead derived from Figures 4, 6 and 2, we see that DT10-APSI achieves better performance; also, it is the only protocol providing forward security.

One party small set case. Finally, we compare online computation costs and show the trend with small Client or Server set size. Our goal is to address scenarios where one party only has a single input. Table 2 shows Client and Server overhead for different protocols where either party's input is a singleton. We observe that the result in the table agrees with our conclusions from Figures 3-6.

Summary. Based on the above experimental results, our toolkit includes several efficient PSI-DT and APSI-DT implementations, each targeting a distinct setting. In particular, DT10-1 is best-suited for PSI-DT with unlinkability, and DT10-2 – for (linkable) PSI-DT with pre-distribution. DT10-APSI is geared for APSI-DT with unlinkability and forward security. Whereas, IBE-APSI is best for for the more efficient case APSI-DT with pre-distribution.[7]

[6] In these figures, y-values for IBE-APSI are all 0 which is out of the scope of the y-axis.

[7] This is, however, linkable and not forward secure.

Protocols	Online Computation Overhead (ms)					
	v=1, w=10,000		v=10,000, w=1		v=1, w=1	
	Client	Server	Client	Server	Client	Server
FNP04	1,556.3	19,450.4	12,627.1	65.1	1.2	2.3
DT10-1	0.3	22.7	3,140.8	1,376.6	0.3	0.1
DT10-2	0	0.3	52.6	2,787.7	0	0.3
JL09	7.6	3.3	77,622.6	32,373.4	7.6	3.2
JL10	1.1	0.2	11,270.9	1,415.7	1.1	0.2
IBE-APSI	1.4	0	14,142.3	0	1.4	0
DT10-APSI	1.9	26.8	18,646.5	9,162.3	2.2	2.1

Table 2: On-line computation overhead (in ms)

We claim that careful and optimized implementations of (A)PSI protocols included in the PSST toolkit represent an independent contribution. This is because, prior to our work, it has been generally believed that PSI protocols are impractical. For example, [3] reports the measurement of 213 **seconds** to run FNP04 [30] PSI over 100-item sets on a 3GHz Intel machine. In contrast, our implementation completes the same task well below 100 seconds even for 1000-item sets (on a similar machine).

3.2 Comparison with PIR

In the experiments above, we benchmarked different (A)PSI-DT protocols measuring the total time needed by Client to identify the elements in the intersection and the associated symmetric key (later used to decrypt corresponding data records).[8] Additionally, however, one should also take into account the time needed by Server to transmit *all* encrypted data records to Client. For this reason, one common criticism to PSI-s has often been related to such a communication overhead, in contrast to PIR protocols, which are known for minimizing bandwidth complexity. Recall, however, that: (1) only *Symmetric* PIR-s would also provide Server privacy beyond hiding Client interests, and (2) within PIR, Client would need to know and input the index of requested items, whereas, in PSI-DT, Client inputs can be arbitrary set elements. Despite these distinctions, we nonetheless argue that the *computational* overhead introduced by PIR-s is high enough that, in reality, we are better off transferring the entire encrypted database.

To support our claim, we benchmark Gentry and Ramzan PIR protocol [31], that (to the best of our knowledge) achieves the best communication complexity, i.e., $O(\log n)$ for a database with n records. Figure 9 compares the computation time needed by the PIR implementation (i.e., we do not include data transmission time) and the *total* time needed by each (A)PSI-DT protocol to execute both off- and on-line operations, plus the time to transmit the entire Server database over a gigabit link. Comparison is made at different Server set sizes. Each element's data record is set to be 1MB; this represents a reasonably average record size. Our results show that even a "fast" PIR is appreciably more expensive than transferring the entire "encrypted" database, plus computation time needed by any PSI protocol discussed above. Indeed, this concurs with recent results in [52].

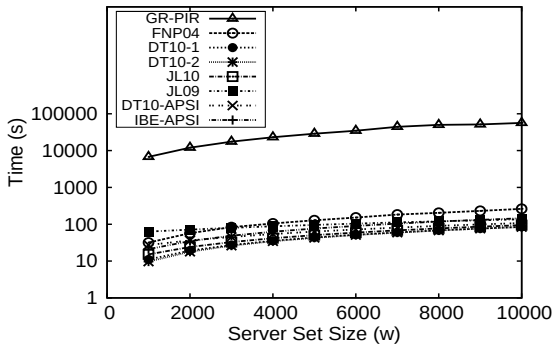


Figure 9: Comparison to GR-PIR.

[8] Recall that *Server privacy* guarantees that Client can only decrypt records associated to items in the set intersection.

4 Large-Scale Privacy-preserving Sharing of Sensitive Information

We now turn to a more realistic scenario that mimics simple database querying and captures a more general class of information sharing. Specifically, we assume that Server maintains a large database – DB that contains w records with m attributes $(attr_1, \cdots, attr_m)$. We denote $DB = \{(R_j)\}_{j=1}^{w}$. Each record $R_j = \{val_{j,l}\}_{l=1}^{m}$, where $val_{j,l}$ is R_j's value for attribute $attr_l$. Client inputs are assumed to be in the form of simple SQL queries, such as Eq. 1.

$$\text{“SELECT * FROM DB WHERE } attr^* = val^*\text{”} \tag{1}$$

After the interaction, Client gets all records in DB, where $attr^* = val^*$ and nothing else. Whereas, Server learns nothing about $attr^*$ or val^*. We assume that the database schema (format) is known to Client. Furthermore, without loss of generality, we assume that all record attributes are searchable. Later on, we discuss a simple extension to disjunctive queries.

Since security and privacy requirements are basically unaltered from those stated in Section 2.2, we do not repeat them here.

In the rest of this section, we identify and discuss some challenges and issues brought about by realizing (A)PSI-DT protocols in the context of large-scale database applications mentioned above. We then present a general design that overcomes these problems and achieves both efficiency and practicality without sacrificing privacy. Specifically, our approach is aimed at arbitrarily large Server-side databases and arbitrarily frequent Client queries. We also demonstrate, via real running code, that our system achieves performance comparable to an optimized MySQL DBMS, even with privacy-preserving indexing enabled on each attribute.

4.1 Strawman Approach: Problems and Challenges

One naïve way implementing privacy-preserving sharing of sensitive information in the aforementioned large-scale database-like setting is to extend (A)PSI-DT implementations according to the following *strawman* approach:

For each record, consider the hash of every (attribute,value) pair $(attr_l, val_{j,l})$ as a set element, and R_j – as associated data. Server "set" then becomes:

$$\mathcal{S} = \{(H(attr_l, val_{j,l}), R_j)\}_{1 \le l \le m, 1 \le j \le w}$$

Client poses an SQL query (similar to that in Eq. 1). Thus, Client "set" is actually a singleton: $\mathcal{C} = \{H(attr^*, val^*)\}$ optionally accompanied by a signature σ over $H(attr^*, val^*)$ in case authorized inputs are enforced. Then, Client and Server engage in a PSI-DT (APSI-DT) interaction and, at the end, Client obtains all records matching its query. However, this simple approach has a number of privacy and performance issues, as can be seen from prior work, e.g., [2].

Challenge 1: Multi-Sets. By our definitions, PSI-DT and APSI-DT do not support multi-sets. However, most realistic database settings include duplicate values. Although some PSI protocols (e.g., [41]) support multi-sets, their performance is not promising as they involve quadratic overhead. However, in other more efficient protocols (e.g., DT10-1), tags computed over the same $(attr_l, val_{j,l})$ are identical. Since the entire encrypted database (including all tags) is transferred to Client, the latter learns all patterns and distribution frequencies. This is problematic, since actual values can be often inferred from their frequencies.[9]

Challenge 2: Data Pointers. To enable querying by any attribute, each R_j must be copied (and separately encrypted) m times, once for each attribute. For protocols with pre-distribution, high Client-side storage overhead would result. For protocols without pre-distribution, high bandwidth overhead would

[9] For example, consider a large database where one attribute reflects "employee blood type". Since blood type frequencies are well-known for general population, tag distribution for this attribute would essentially reveal the plaintext. This is equivalent to deterministic encryption.

be incurred. This issue can be addressed by encrypting each R_j with a unique symmetric key k_j and then using k_j (instead of R_j) as data associated with $H(attr_l, val_{j,l})$. Although this would reduce storage/bandwidth overhead, it would also prompts an additional privacy issue: In order to use the key – instead of the actual record – as "data", a pointer to the encrypted record (on disk or in memory) would have to be stored alongside each tag. This would let Client determine what tags correspond to different attributes of the same record. This (potential) privacy leak is aggravated by the previous issue (multi-sets), since, given two encrypted records, Client can establish their similarity based on the number of tags they have in common. For example, a malicious Client could test how many records share exactly two attributes.

Challenge 3: Bandwidth. If Server database is large and/or records contain large bulk data, bandwidth overhead may become prohibitively high. For all protocols discussed above (and despite their seemingly low linear bandwidth complexity), the entire encrypted database must be transferred to Client. For protocols without pre-distribution, this has to be done *for each query*. For protocols with pre-distribution, the bulk transfer can be done once, at the start (as long as the database is static). Thus, if the database is very large, the delay experienced by Client would be commensurately high for the first query (with pre-distribution) or for each query (without it). Furthermore, in some scenarios, bandwidth is very limited or costly, e.g., Client running on mobile devices.

Challenge 4: Liability. Transfer of the entire encrypted database to Client also prompts the problem of long-term data safety and associated liability. An encryption scheme considered strong today might gradually weaken in the long term. Consequently, it is not too far-fetched to imagine that Client might be able to decrypt the database, e.g., 10 or 20 years later. However, data sensitivity might not dissipate over time. For example, suppose that a low-level DoD employee is only allowed to access to unclassified data. By gaining access to the encrypted database containing top secret data and patiently waiting for the encryption scheme to "age", the employee might obtain still-classified sensitive information.

Aforementioned challenges do not arise in all settings discussed in Section 1. Recall the Social Networking and Airline safety examples: they do not exhibit problems with bandwidth consumption or long-term liability: the former – because a list of friends (or a passenger manifest) is unlikely to be very long, and the latter – because, years into the future, neither friends' names nor airline passengers represent sensitive information. In contrast, the Healthcare and Law Enforcement examples prompt both long-term liability and bandwidth consumption issues. (A database of Client/patient or police records can be very large and data privacy is a long-term concern).

4.2 PSST Design

We now discuss the design of the PSST toolkit. First, we describe our system architecture with additional privacy requirements. Next, we show a methodology for adapting (A)PSI-DT protocols to support database encryption and query lookup. Then, we discuss challenges and achieved privacy. Finally, we compare the performance of PSST to that of base-line MySQL. Our notation is reflected in Table 3.

$attr_l$	the lth attribute in the database schema
R_j	the jth record in the database table
$val_{j,l}$	the value in R_j corresponding to $attr_l$
k_j	the key used to encrypt R_j
er_j	encryption of R_j
$tk_{j,l}$	a token evaluated over $attr_l, val_{j,l}$
$ctr_{j,l}$	the number of times where $val_{j',l} = val_{j,l}, \forall j' <= j$
$tag_{j,l}$	a tag for $attr_l, val_{j,l}$
$k'_{j,l}$	key used to encrypt k_j
$k''_{j,l}$	key used to encrypt index j
$ek_{j,l}$	encryption of key k_j
$eind_{j,l}$	encryption of index j

Table 3: Notation

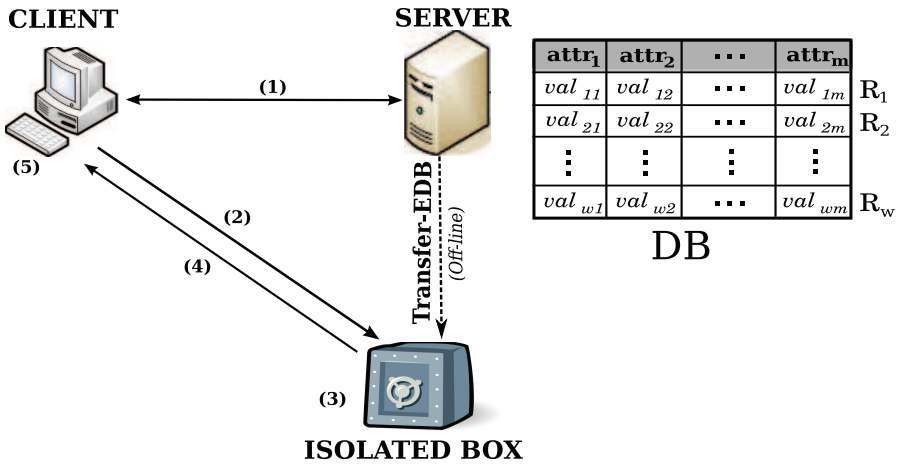


Figure 10: New system architecture with the introduction of Isolated Box.

4.2.1 Architecture

As shown in Figure 10, our system architecture involves three parties: Client, Server and a stand-alone third party, called *Isolated Box* (IB). IB is a stand-alone entity in possession of the encrypted database and a lookup table (denoted by LTable), transferred from Server. In order to pose an SQL query, Client first interacts with Server to obtain a *token* while not revealing the content of the query. From this token, it derives a set of tags and sends these tags to IB, that returns all matching records. (Again, this does not disclose the query target). Note that Server, instead of transferring the entire encrypted database to Client, transfers it (off-line) to IB.

Our IB-powered system relies on (A)PSI-DT protocols *with* pre-distribution. Indeed, all four challenges listed in Section 4.1 can be solved using such protocols within the IB architecture. Furthermore, such protocols achieve better on-line communication and computation overhead.

PSST addresses privacy concerns mainly using a novel encryption mechanism, while it requires minimal trust in IB. Note that the use of stand-alone semi-trusted parties to enhance privacy dates back to Beaver's initial intuition of *Commodity-based Cryptography* [4]. Other examples are the solutions in [12] and [23]. The former introduces an oblivious third party to let a server obliviously compare two numbers (e.g., to solve the millionaire's problem [57]), whereas, the latter uses the same intuition to solve the scalar product problem. More recent techniques, using semi-trusted parties in the context of privacy-preserving database applications, are [49, 19], discussed later in Section 4.3.

Trust Assumptions. We assume that IB does not collude with either Server or Client. (However, we discuss the consequences of collusions in Section 4.2.6). Also, we assume the existence of a private and authentic channel between Client and Server, as well as between Client and IB, e.g., using SSL/TLS. Finally, note that IB can be realized as a piece of secure hardware installed on Server premises, as long as Server does not learn what IB reads from storage (i.e., what records) and transfers to Client. (Architecture for this kind of secure hardware has been proposed, e.g., in [1]).

4.2.2 Database Encryption with counters

The procedure used for database encryption is presented in Alg. 1. It is composed of two phases: (1) *record-level* and (2) *lookup-table* encryption. Record-level encryption is relatively trivial; it is shown in lines 1–6. First, Server shuffles record locations in the database. Then, it pads each R_j up to the maximum size of all records, picks a random symmetric key k_j and encrypts R_j as $er_j = Enc_{k_j}(R_j)$. Lookup-table (LTable) encryption, shown in lines 8–14, refers to encryption of attribute name and value pairs. It enables efficient lookup and record decryption.

We use (A)PSI-DT protocols with pre-distribution (discussed above) to "obliviously" compute a function, whose input is a Client set element – specifically, the concatenation of attribute and value. Here

Algorithm 1: Database Encryption

1 Shuffle $\{R_j\}_{1\leq j\leq w}$;
2 $maxlen \leftarrow$ max length among all R_j;
3 **for** $1 \leq j \leq w$ **do**
4 Pad R_j to $maxlen$;
5 $k_j \leftarrow \{0,1\}^{128}$;
6 $er_j \leftarrow Enc_{k_j}(R_j)$;
7 **for** $1 \leq l \leq m$ **do**
8 $tk_{j,l} \leftarrow$Token$(attr_l, val_{j,l})$;
9 $tag_{j,l} \leftarrow H_1(tk_{j,l}||ctr_{j,l})$;
10 $k'_{j,l} \leftarrow H_2(tk_{j,l}||ctr_{j,l})$;
11 $k''_{j,l} \leftarrow H_3(tk_{j,l}||ctr_{j,l})$;
12 $ek_{j,l} \leftarrow Enc_{k'_{j,l}}(k_j)$;
13 $eind_{j,l} \leftarrow Enc_{k''_{j,l}}(j)$;
14 $\mathbf{LTable}_{j,l} \leftarrow (tag_{j,l}, ek_{j,l}, eind_{j,l})$;
15 **end**
16 **end**
17 Shuffle **LTable** with respect to j and l;
18 $\mathbf{EDB} \leftarrow \{\mathbf{LTable}, \{er_j\}_{1\leq j\leq w}\}$;
19 Off-line transfer $\mathbf{EDB}$ to IB.

we use **Token** to denote the result of this function computation. When creating LTable, Server computes $tk_{j,l} =$ Token$(attr_l, val_{j,l})$ for all combinations of j and l (line 8). For example, in encrypting database in DT10-2, Server computes an RSA signature $tk_{j,l} = H(attr_l, val_{j,l})^d$ and, in IBE-based APSI-DT, it computes $tk_{j,l} = e(Q, H(attr_l, val_{j,l}))^z$. (See Appendix A for protocol details.)

We use $ctr_{j,l}$ to denote the index of duplicate value for the l-th attribute. In other words, $ctr_{j,l}$ is the counter of times where $val_{j',l} = val_{j,l}, \forall j' <= j$. For example, the third occurrence of value "Smith" for attribute "Last Name" will have $ctr_{j,l} = 3$.

Next, Server computes $tag_{j,l} = H_1(tk_{j,l}||ctr_{j,l})$, $k'_{j,l} = H_2(tk_{j,l}||ctr_{j,l})$ and $k''_{j,l} = H_3(tk_{j,l}||ctr_{j,l})$, where H_1, H_2, H_3 are different hash functions (lines 9-11). In practice, we can implement $H_1(m)$, $H_2(m)$, $H_3(m)$ using SHA-1 [24] as: SHA-1$(1||m)$, SHA-1$(2||m)$, SHA-1$(3||m)$, and so on.

Note that $tag_{j,l}$ is later used as lookup tag during Client query. $k'_{j,l}$ is used for encrypting symmetric key k_j. $k''_{j,l}$ is used for encrypting the index of R_j.

Next, Server computes $ek_{j,l} = Enc_{k'_{j,l}}(k_j)$ and $eind_{j,l} = Enc_{k''_{j,l}}(j)$ (lines 12-13). It then inserts each $tag_{j,l}$, $ek_{j,l}$ and $eind_{j,l}$ into LTable (line 14), which is $\{tag_{j,l}, ek_{j,l}, eind_{j,l}\}_{1\leq j\leq w, 1\leq l\leq m}$. Next, Server shuffles LTable (line 17). The resulting encrypted database, EDB, is made up of LTable and $\{er_j\}_{j=1}^{w}$ (line 18). Finally, Server transfers EDB (off-line) to IB.

4.2.3 Query lookup

Query lookup procedure is described in Alg. 2 (see also Figure 10). Client produces a simple SQL query, i.e. "SELECT * FROM DB WHERE $attr^* = val^*$ LIMIT t". For ease of exposition, we assume that Client only wants to retrieve first t matching records. (In the next section, we describe how to cope with the case when t is omitted from Client query.) In step (1), Client runs any (A)PSI-DT with pre-distribution over a singleton set with $\{(attr^*, val^*)\}$ as its input and obliviously evaluates $tk^* =$ Token$(attr^*, val^*)$ with Server. In step (2), Client sets a counter i from 1 to t, and computes a set of tags $\{tag_i^* = H_1(tk^*||i)\}_{1\leq i\leq t}$ and a set of index decrypting keys $\{k''_i = H_3(tk^*||i)\}_{1\leq i\leq t}$. Next, Client sends $\{tag_i^*, k''_i\}_{1\leq i\leq t}$ to IB. For each $i \in [1,t]$, IB searches for tag_i^* in LTable in step (3). If there is no result, IB puts $\perp$ in response set. If a tuple $(tag_{j,l}, ek_{j,l}, eind_{j,l})$ is found where $tag_{j,l} = tag_i^*$, IB decrypts $eind_{j,l}$ and recovers index j' by running $Dec_{k''}(ek_{j,l})$. IB then puts $er_{j'}$ and $ek_{j,l}$, which equal to er_i^* and ek_i^*, to the response set. In step (4), Server returns the response set $\{ek_i^*, er_i^*\}_{1\leq i\leq t}$ to Client. In step (5), Client computes a set of decrypting keys $\{k'_i = H_2(tk^*||i)\}_{1\leq i\leq t}$. For each $i \in [1,t]$, it obtains decryption key $k_i = Dec_{k'}(ek_i^*)$, and decrypts er_i^* by $R_i = Dec_{k_i}(er_i^*)$.

Algorithm 2: Query Lookup

Step 1: Client anonymously evaluates $tk^* = \textbf{Token}(attr^*, val^*)$;
Step 2: Client sends $\{tag_i^* = H_1(tk^*|||i), k_i'' = H_3(tk^*|||i)\}_{1 \le i \le t}$ to IB;
Step 3: IB computes:
for $1 \le i \le t$ **do**
find $\mathbf{LTable}_{j,l}$
such that $tag_{j,l} = tag_i^*$
$ek_i^* \leftarrow ek_{j,l}$
$j' = Dec_{k_i''}(eind_{j,l})$
$er_i^* \leftarrow er_{j'}$
end
Step 4: IB transfers $\{ek_i^*, er_i^*\}_{1 \le i \le t}$ to Client.
Step 5: Client computes:
for $1 \le i \le t$ **do**
$k_i' = H_2(tk^*||i)$
$k_i = Dec_{k_i'}(ek_i^*)$
$R_i = Dec_{k_i}(er_i^*)$
end

4.2.4 Optimizations

If t is too large (i.e., there are fewer than t matching records) or it is simply omitted from the query, computing all the tag_i^* and k_i'' at once in step 3 might be time consuming or impossible. Note that Client can retrieve records one by one from IB by gradually incrementing counter i in each round. Thus, a possible solution is to let Client compute only one tag_i^* and k_i'' each time and pipe-line computation of tag_{i+1}^* and k_{i+1}'' with the retrieval of ek_i^* and er_i^* (step 4–5). The query terminates when either t responses or $\perp$ are (is) received. This way, overhead incurred in step 3 amounts to computation of only one tag and one key. Furthermore, Client does not need to estimate how many tags and keys to compute in step 3.

We can further optimize the computation of $ek_{j,l}$ and $eind_{j,l}$ (steps 12–13 in Alg. 1). Since we use a counter as part of the input to compute $k'_{j,l}$ (respectively, $k''_{j,l}$), each $k'_{j,l}$ (respectively, $k''_{j,l}$) is distinct for any j, l. Both $k'_{j,l}$ and $k''_{j,l}$ are 160-bit values, while k_j is 128 bits and j is clearly smaller. Hence, we can use one-time-pad-like encryption (i.e. $ek_{j,l} = k'_{j,l} \oplus k_j$ and $eind_{j,l} = k''_{j,l} \oplus j$) to speed up computation. In Alg. 2, $Dec_{k_i''}(eind_{j,l})$ becomes $k_i'' \oplus eind_{j,l}$ and $Dec_{k_i'}(ek_i^*)$ changes to $k_i' \oplus ek_i^*$.

4.2.5 Challenges Revisited

We now show how the proposed architecture addresses challenges discussed in Section 4.1.

Multi-sets. The counter used during database encryption makes each $tag_{j,l}$ (resp. $ek_{j,l}$, $eind_{j,l}$) distinct in $\mathbf{LTable}$, thus hiding plaintext patterns.

Data Pointers. Storing $eind_{j,l}$, instead of j, in LTable prevents Server from revealing the relationship between an entry $\mathbf{LTable}_{j,l}$ and its associated record R_j.

Bandwidth Overhead. Once Server transfers its database (off-line) to IB, the latter sends only those records that match the query back to Client, instead of the entire encrypted database.

Liability. Since IB assumes the role of the custodian of the encrypted database, Client only obtains the result of its queries.

4.2.6 Addressing Privacy

The introduction of IB and the use of counter mode in database encryption provides additional privacy properties. We use the term *transaction* to mean a complete query procedure from the time a SQL query is issued, until the last response from IB is received. A *retrieval* is defined as the receipt of a single response record during a transaction. We claim that, if Client performs only one query transaction, as in Alg. 2, IB can link all tag_i^*s values in step 3 to the same $(attr, val)$ pair. This poses the

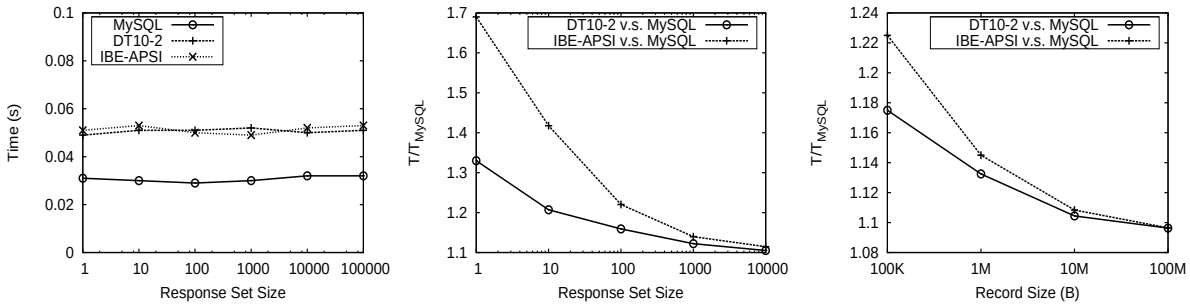
same risk as discussed in the "multi-set" challenge. However, as mentioned in Section 4.2.5, the counter allows Client to retrieve matching records one by one. Therefore, Client can choose to add a random delay between two subsequent retrievals in a single transaction. If the distribution of additional delays is indistinguishable from time gaps between two transactions, IB can not tell the difference between two continuous retrievals within one transaction from two distinct transactions. As a result, IB cannot infer whether two continuously retrieved records share the same $(attr, val)$ pair and the distribution of the attribute value remains hidden.

We also note that the introduction of IB does not violate Client or Server privacy. Client privacy is preserved because Client obliviously computes a token which is not learned by Server. IB does not learn Client interest, since Client input to IB (tag) is statistically indistinguishable from a random value. Server privacy is preserved because Client does not gain any extra information by interacting with IB. Finally, IB only has the encrypted database and learns no plaintext.

Limitations. We acknowledge that PSST has certain limitations. Over time, as it serves many queries, IB gradually learns the relationship between tags and encrypted records through pointers associated with each tag. This issue can be mitigated by letting Server periodically re-encrypt the database. Next, if Server and IB collude, Client privacy is lost, since IB learns tag that Client seeks, and Server learns an $(attr, val)$ pair each tag is related to. On the other hand, if Client and IB collude, the liability of encrypted database possession by Client becomes a problem once again. Finally, Server unlinkability is guaranteed only as far as Client. Server unlinkability as far as IB is not guaranteed, since IB learns about all changes in Server database.

Finally, PSST currently supports only equality and disjunctive SQL queries. In fact, the latter are implemented by treating each equality condition inside an "OR" clause as a separate query and removing duplicate responses. Whereas, supporting conjunctive queries would require treating all combinations of $(attr, val)$ pairs as Server set elements. Thus, set size would become exponential in terms of the number of attributes. This remains a challenge for future work.

4.2.7 Comparison to MySQL



(a) Index lookup speed comparison.

(b) Comparison to MySQL w.r.t. response set size.

(c) Comparison to MySQL w.r.t. record size.

Figure 11: Comparison to MySql

In Section 3.1, we have shown that DT10-2 and IBE-APSI are the most efficient PSI-DT and APSI-DT protocols with pre-distribution, respectively. Therefore, we implemented both protocols as the building blocks of PSST. We run both IB and Server on an Intel Harpertown Server with two Xeon E5420 CPUs (2.5 GHz) and $12GB$ RAM. Client runs on a laptop with Intel Core 2 Duo CPU (2.2 GHz) and $4GB$ RAM. Client is connected to IB and Server via Gigabit ethernet. The database has 45 searchable attributes and $100,000$ records. All records have the same size, which we vary during experiments. We compare our results against a MySQL setup for the same database with indexing enabled for each attribute. MySQL is running on the same machine as Server. Note that each result is averaged over ten independent tests.

First, we compare *index lookup time*, defined as the time between SQL query issuance and the receipt of the first response from IB. We select a set of SQL queries that return 0, 1, 10, 100, 1000, 10000 ($\pm$10%) responses, respectively and fix each record size at 500KB. Figure 11(a) shows index lookup time for DT10-2, IBE-APSI and MySQL in terms of the response set size. Both DT10-2 and IBE-APSI incur almost the same overhead for and are 1.5 times more expensive than MySQL. We also measures index lookup time in terms of general record size. Since the results are similar to the previous experiment, we omit them here.

Next, we test the impact of response set size on *total query time*, defined as the time between SQL query issuance and the arrival of the last response from IB. Figure 11(b) shows the time for Client to complete a query for a specific response set size divided by the time taken by MySQL. Results gradually converge to 1.1 for increasing response set sizes. This is because of the extra delay incurred by cryptographic operations (as part of oblivious function evaluation) being amortized by subsequent data lookup and decryption.

Last, we test the impact of record size on total query time. We fix response set size at 100 and vary each record size betwwen 100KB and 100MB. Figure 11(c) shows the ratio between DT10-2 and MySQL, IBE-APSI and MySQL, respectively. Again, results gradually converge to 1.1 with increasing of record size which occurs because the overhead of symmetric record decryption becomes dominant with growing record size.

In summary, both index lookup time and total query time of our implementation are strictly less than double their respective counterparts in MySQL.

4.3 Related Privacy-Preserving Database Systems

Privacy-preserving database querying has been considered in prior work. Although prior results support more complex query types (not just equality and disjunctive queries), they exhibit certain limitations, such as: (1) high computation overhead and no protection of database data, e.g., [45][10], (2) lack of provable privacy guarantees, e.g., [33, 37, 8], or (3) requirement for *several* independent trusted parties, e.g., [49, 19].[11]

Some recent work focused on automating and speeding up secure two-party computation tools [7, 36, 47]. On one hand, such solutions meet more theoretically-sound security requirements (e.g., do not require the random oracle model or assume arbitrarily malicious adversaries). On the other hand, they result into much lower efficiency (several orders of magnitude). Since these solutions address different problems and scenarios (and because of space limitations) we consider a detailed discussion and comparison with our toolkit to be out of the scope of this paper.

5 Conclusions & Future Work

In this paper, our goal was to drag some useful and efficient privacy-preserving tools out of their cryptographic "closet" and demonstrate that privacy-preserving sharing of sensitive information is practical *today*. Indeed, we have shown that some recent Private Set Intersection techniques are efficient enough to serve as a basis for real-world large-scale systems. We have implemented a simple database-like system that allows Client, armed with one or more (optionally authorized) search elements or keywords, to search over Server database, such that no extra information is learned by either party. In the course of designing and implementing our toolkit, we encountered (and addressed) a number of interesting issues that led to specific architectural choices. As confirmed by experimental evaluation, our

[10] Olumofin and Goldberg [45] construct a privacy mechanism for Keyword-based PIR (KPIR [17]) and provide a transition from block-based PIR to SQL-enabled PIR. However, similar to KPIR-s, it incurs high (several order of magnitudes) computational complexity, and allows clients to obtain other bits of database data (beyond the query result).

[11] Chow, et al. [19] propose a two-party query computation model over distributed databases that involves three entities: a randomizer, a computing engine, and a query front-end. Local answers to queries are randomized by each database and aggregate results are de-randomized at the front-end.

PSST toolkit offers a *menu* of privacy-preserving services at reasonable additional cost over the base-line MySQL implementation.

Much remains to be done: First, PSST does not currently support conjunctive (AND) queries across multiple attributes. This is not difficult to do naïvely, however, the overhead is likely to be quite high (exponential in the number of attributes). Second, some on-going work yielded PSI protocols that hide the size of Client input (set). We intend to incorporate them into PSST. Finally, we plan to explore ways to support "fuzzy" querying, i.e., where Client input represents non-normalized data.

References

[1] R. Agrawal, D. Asonov, M. Kantarcioglu, and Y. Li. Sovereign Joins. In *ICDE'06*, 2009.

[2] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *SIGMOD'03*, 2003.

[3] B. Applebaum, H. Ringberg, M. Freedman, M. Caesar, and J. Rexford. Collaborative, privacy-preserving data aggregation at scale. In *PETS'10*.

[4] D. Beaver. Commodity-based cryptography. In *STOC'97*, pages 446–455, 1997.

[5] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham. Randomizable proofs and delegatable anonymous credentials. In *Crypto'09*, 2009.

[6] 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.

[7] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: a system for secure multi-party computation. In *CCS'08*, 2008.

[8] E. Bertino, J. Byun, and N. Li. Privacy-preserving database systems. *Foundations of Security Analysis and Design*, 2005.

[9] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Eurocrypt'04*, 2004.

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

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

[12] C. Cachin. Efficient private bidding and auctions with an oblivious third party. In *CCS'99*, 1999.

[13] 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*, 2009.

[14] J. Camenisch, G. Neven, and A. Shelat. Simulatable adaptive oblivious transfer. In *Eurocrypt'07*, pages 573–590, 2007.

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

[16] Caslon Analytics. Consumer Data Losses. `http://www.caslon.com.au/datalossnote.htm`.

[17] B. Chor, N. Gilboa, and M. Naor. Private information retrieval by keywords. *Manuscript*, 1998.

[18] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[19] S. Chow, J. Lee, and L. Subramanian. Two-party computation model for privacy-preserving queries over distributed databases. *NDSS'09*, 2009.

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

[21] E. De Cristofaro, S. Jarecki, J. Kim, and G. Tsudik. Privacy-preserving policy-based information transfer. In *PETS'09*, pages 164–184, 2009.

[22] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography and Data Security'10*, pages 143–159, 2010.

[23] W. Du and Z. Zhan. A practical approach to solve secure multi-party computation problems. In *NSPW'02*, 2002.

[24] D. Eastlake and P. Jones. US Secure Hash Algorithm 1 (SHA1), 2002.

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

[26] A. Evfimievski, J. Gehrke, and R. Srikant. Limiting privacy breaches in privacy preserving data mining. In *PODS'03*.

[27] Federal Bureau of Investigation. Terrorist Screening Center. `http://www.fbi.gov/terrorinfo/counterrorism/tsc.htm`.

[28] Free Software Foundation. The GNU MP Bignum Library. `http://gmplib.org/`.

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

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

[31] C. Gentry and Z. Ramzan. Single-database private information retrieval with constant communication rate. In *ICALP'05*.

[32] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. In *STOC'98*, 1998.

[33] H. Hacigümüş, B. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *SIGMOD'02*, 2002.

[34] 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.

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

[36] W. Henecka, S. Kgl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: Tool for Automating Secure Two-partY computations. In *CCS'10 (to appear)*, 2010.

[37] B. Hore, S. Mehrotra, and G. Tsudik. A privacy-preserving index for range queries. In *VLDB'04*, pages 720–731, 2004.

[38] B. Huberman, M. Franklin, and T. Hogg. Enhancing privacy and trust in electronic communities. In *ECC'09*, 1999.

[39] S. Jarecki and X. Liu. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *TCC'09*, pages 577–594, 2009.

[40] S. Jarecki and X. Liu. Fast secure computation of set intersection. In *SCN'10 (to appear)*, 2010.

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

[42] B. Lynn. PBC: The Pairing-Based Cryptography Library. `http://crypto.stanford.edu/pbc/`.

[43] S. Nagaraja, P. Mittal, C. Hong, M. Caesar, and N. Borisov. BotGrep: Finding Bots with Structured Graph Analysis. In *Usenix Security'10*, 2010.

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

[45] F. Olumofin and I. Goldberg. Privacy-preserving queries over relational databases. In *PETS'10*, 2010.

[46] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Eurocrypt'99*, 1999.

[47] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In *Asiacrypt'09*, 2009.

[48] M. Rabin. How to exchange secrets by oblivious transfer. TR-81, Harvard Aiken Computation Lab, 1981.

[49] M. Raykova, B. Vo, S. Bellovin, and T. Malkin. Secure anonymous database search. In *CCSW*, pages 115–126, 2009.

[50] R. Rivest. The RC4 Encryption Algorithm. *RSA Data Security Inc.*, 1992.

[51] Sherri Davidoff. What Does DHS Know About You? `http://philosecurity.org/2009/09/07/what-does-dhs-know-about-you`.

[52] R. Sion. On the computational practicality of private information retrieval. In *NDSS'07*, 2007.

[53] D. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *S&P'00*, 2000.

[54] Transport Security Administration. Secure Flight: The "No-Fly" list. `http://www.tsa.gov/approach/secure_flight.shtm`.

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

[56] Wikipedia, The Free Encyclopedia. Secondary Security Screening Selection. `http://en.wikipedia.org/wiki/Secondary_Security_Screening_Selection`.

[57] A. Yao. Protocols for secure computations. In *FOCS'82*, pages 160–164, 1982.

Appendix

A State-of-the-art PSI-s

In the following, we thoroughly review state-of-the-art PSI protocols. We discuss both the protocols implemented in the PSST toolkit, and those not included. During our discussion, we also motivate our choices of PSI protocols providing the best combination of privacy guarantees and efficiency. We focus on constructs providing *data transfer* (see Section 2.1). We assume honest-but-curious adversaries.

Notation. We now introduce some terminology. We use $\mathsf{Enc}_k(\cdot)$ and $\mathsf{Dec}_k(\cdot)$ to denote symmetric key encryption and decryption (under key k), respectively. Public key encryption and decryption – under keys pk and sk – are denoted by $E_{pk}(\cdot)$ and $E_{sk}(\cdot)^{-1}$, respectively. Then, $\sigma = \mathsf{Sign}_{sk}(M)$ denotes a digital signature computed over a message M using the secret key sk. Operation $\mathsf{Vrfy}_{pk}(\sigma, M)$ returns 1 or 0 indicating whether σ is a valid signature on M. Z_N^* refers to a composite-order RSA group, where N is the RSA modulus. We use Z_p^* to denote a cyclic group with a subgroup of order q, where p and q are large primes, and $q|p-1$. We use $H(\cdot), H_1(\cdot), H_2(\cdot), H_3(\cdot)$ to denote different hash functions. Finally, we assume Client and Server set sizes are v and w, respectively.

A.1 PSI-DT without Pre-distribution

FNP04. Freedman et al. [30] use *oblivious polynomial evaluation* (OPE) to implement PSI. Their approach can be slightly modified to support PSI-DT. The modified protocol – denoted with FNP04 – works as follows: Client first setups an additively homomorphic encryption scheme, such as Paillier [46], with key pair (pk_c, sk_c). Client defines a polynomial $f(y) = \prod_{i=1}^{v}(y - c_i) = \sum_{i=0}^{v} a_i y^i$ whose roots are its inputs. It encrypts each coefficient a_i under its public key pk_c and sends encrypted coefficients $\{Enc_{pk_c}(a_i)\}_{i=0}^{k}$ to Server. Since the encryption is homomorphic, Server can evaluate $Enc(f(s_j))$ for each $s_j \in \mathcal{S}$ without the support from Client. Then Server returns $\{(Enc(r_j \cdot f(s_j) + s_j), Enc(r'_j \cdot f(s_j) + data_j))\}_{j=0}^{n}$ to the Client where r_j and r'_j are fresh random numbers for each input in $\mathcal{S}$. Client, for each returned pair (e_l, e_r), decrypts e_l by computing $c' = Dec_{sk_c}(e_l)$. Then if $c' \in \mathcal{C}$, the Client continues to decrypt e_r and gets the associated data. Otherwise, Client only gets some random value and moves onto the next returned pair. In order to speed up the performance, FNP04 can use modified ElGamal encryption [25] instead of Paillier. Specifically, Client uses g^{a_i} instead of a_i as the input to the ElGamal encryption. And when it decrypts e_l, it recovers $g^{c'}$. Client can still decide whether $c' \in \mathcal{C}$ by comparing $g^{c'}$ to $g^{c_i}, \forall c_i \in \mathcal{C}$. In terms of data, Server can choose a random key g^{k_j} and uses it to symmetrically encrypt $data_j$. Then Server sends $\{(Enc(r_j \cdot f(s_j) + s_j), Enc(r'_j \cdot f(s_j) + k_j), Enc_{g^{k_j}}(data_j))\}_{j=0}^{w}$

to Client. If Client can recover g^{k_j}, it can also decrypt $data_j$. Using balanced bucket allocation to speed up operations, Client overhead is dominated by $O(v + w)$ $|q|$-bit mod p exponentiations (in ElGamal). Whereas, Server overhead is dominated by $O(w \log\log v)$ $|q|$-bit mod p exponentiations. Also, note that a recent result in [35] presents an improved construction of FNP without ROM in the malicious model, with similar asymptotic overhead but more expensive underlying cryptographic operations.

KS05. Kissner and Song [41] also use oblivious polynomial evaluation to construct a variety of set operations. However, their solution is designed for mutual intersection over *multi-set* that may contain duplicate elements, and it is unclear how to adapt it to transfer associated data. Also, their technique incurs quadratic ($O(vw)$) computation (but linear communication) overhead. As we will use a different method to handle multi-sets (see Section 4.2.1) and we only consider one-way PSI, we prefer FNP04 as a PSI-DT solution based on OPE.

DT10-1. De Cristofaro and Tsudik present an unlinkable PSI-DT protocol (Fig. 3 in [22]) with linear computation and communication complexities. This protocol, denoted as DT10-1, operates as follows: The setup phase yields primes p and q, s.t. $q|p-1$, and a generator g of the subgroup. In the following, we assume computation done mod p. First, Client sends to Server $X = [(\prod_{i=1}^{v} H(c_i)) \cdot g^{R_c}]$ where R_c is randomly selected from $\mathbb{Z}_q$. Also, for each i, Client sends $y_i = [(\prod_{l \neq i} H(c_l)) \cdot g^{R_{c:i}}]$, where the $R_{c:i}$'s are random in $\mathbb{Z}_q$. Server picks a random R_s in $\mathbb{Z}_q$ and replies with $Z = g^{R_s}$ and $y'_i = y_i^{R_s}$ (for every y_i it received). Also, for each item s_j, it computes $K_{s:j} = (X/H(s_j))^{R_s}$, and sends the *tag* $t_j = H_1(K_{s_j})$ with the associated data record encrypted under $k_j = H_2(K_{s_j})$. Client, for each of its elements, computes $K_{c:i} = y'_i \cdot Z^{R_c} \cdot Z^{-R_{c:i}}$ and the tag $t'_i = H_1(K_{c:i})$. Only if c_i is in the intersection (i.e., there exists an element $s_j = c_i$), Client finds a pair of matching tags (t'_i, t_j). Besides learning the elements intersection, Client can decrypt associated data records by key $H_2(K_{c:i})$. Client overhead amounts to $O(v)$ $|q|$-bit modulo p exponentiations and multiplications and Server overhead is $O(v + w)$ $|q|$-bit modulo p exponentiations. Communication complexity is linear – $O(v + w)$.

As a result, we consider FNP and DT10-1 protocols in the context of (non-authorized) PSI-DT *without* pre-distribution. (We discuss pre-distribution optimizations below). Note that these protocols also provide the optional requirements of Server and Client unlinkability.

A.2 PSI-DT with Pre-distribution

JL09. Jarecki and Liu [39] (following the idea in [34]) give a PSI based on Oblivious PRF (OPRF) [29]. We denote this protocol as JL09. Recall that an OPRF [29] is a two-party protocol that securely computes a pseudorandom function $f_k(\cdot)$, on key k contributed by the Server and input x contributed by Client, such that Server learns nothing about x, while Client learns $f_k(x)$. The main idea is the following: First, for every item $s_j \in \mathcal{S}$, the Server publishes $u_j = f_k(s_j)$. Then, Client, for every item $c_i \in \mathcal{C}$, obtains $u'_i = f_k(c_i)$ by running the secure computation of the OPRF with Server, so that Server does not learn Client input, and Client learns that $c_i \in \mathcal{C} \cap \mathcal{S}$ if there exists u_j s.t. $u'_i = u_j$. JL09 incurs $O(w + v)$ Server exponentiations, and $O(v)$ Client exponentiations. Note that in our implementations we use the improved OPRF construction presented in [5].

JL10. Another recent work by Jarecki and Liu [40] (denoted as JL10) leverages an idea similar to the OPRF, namely the newly-introduced *Parallel Oblivious Unpredictable Function* (POUF), $f_k(x) = (H(x)^k \bmod p)$, in the Random Oracle Model secure under the *One-More-Gap-DH* assumption in ROM [6]. During pre-distribution, Server publishes $u_j = H_1(f_k(s_j)) = H_1(H(x)^k)$, for each $s_j \in \mathcal{S}$. Associated data is encrypted under keys obtained as $H_2(f_k(s_j))$. Then, Client obtains $f_k(c_i)$ as follows. Client picks a random exponent α and sends $y_j = H(c_j)^\alpha$ to Server, that replies with $z = y^k$, so that Client recovers $f_k(x) = z^{1/\alpha}$. Being $u'_i = H_1(f_k(c_i))$, Client learns that $c_i \in \mathcal{C} \cap \mathcal{S}$ if there exists u_j s.t. $u'_i = u_j$ and may also decrypt the corresponding data. Note that this solution resembles the set intersection protocols previously given by [26, 38]. The computational complexity of this protocol amounts to $O(v)$ on-line exponentiations for both Server and Client, as Server can pre-process (off-line) its $O(w)$ exponentiations. Note that random exponents can all be taken from a subgroup, e.g., they can 160-bit long, similar to DT10-1.

DT10-2. In Fig. 4 of [22], De Cristofaro and Tsudik present a PSI-DT (secure under the One-More-RSA assumption [6]), based on blind-RSA signatures. We denote this protocol with DT10-2. It achieves the same computational complexity as JL10, but (1) Server computes RSA signatures (e.g., 1024-bit exps), and (2) Client workload is reduced to only multiplications if the RSA public key, e, is chosen short enough (e.g., $e = 3$).

In summary, we consider JL09, JL10 and DT10-2 in the context of PSI-DT *with* pre-distribution. Note that, although faster than protocols without pre-distribution, these protocols do not achieve Server unlinkability.

A.3 APSI-DT without Pre-distribution

DT10-APSI. In Fig. 2 of [22], De Cristofaro and Tsudik also present an APSI-DT technique mirroring the PSI-DT discussed above (i.e., DT10-1). We denote this protocol as DT10-APSI. It works as follows: Client first obtains authorization from the court for its element c_i, where an authorization corresponds to an RSA-signature: $\sigma_i = H(c_i)^d$. Then, Client sends Server $X = [(\prod_{i=1}^{v} \sigma_i) \cdot g^{R_c}]$ for a random R_c. Then, for each element c_i, it sends $y_i = [(\prod_{l \neq i} \sigma_l) \cdot g^{R_{c:i}}]$, where the $R_{c:i}$'s are additional random values. Server picks a random value, R_s, and replies with $Z = g^{eR_s}$, $y'_i = y_i^{eR_s}$ (for each received y_i). Also, for each element s_j, she computes $K_{s:j} = (X^e / H(s_j))^{R_s}$, and sends the *tag* $t_j = H'(K_{s:j})$ and the associated data record encrypted under the key $k_j = H''(K_{s:j})$. Client, for each of its elements, computes $K_{c:i} = y'_i \cdot Z^{R_c} \cdot Z^{-R_{c:i}}$ and the tag $t'_i = H'(K_{c:i})$. Client can find a pair of matching tag (t'_i, t_j) only if c_i is in the intersection and σ_i is a valid signature on c_i, Besides learning the elements in the intersection, Client can decrypt associated data records. The computation overhead is $O(v)$ exponentiations for Client while $O(v + w)$ for Server.

CZ09. Camenisch and Zaverucha [15] provide mutual set intersection with certification on both parties' input. The proposed protocol builds upon oblivious polynomial evaluation and has quadratic computation and communication overhead. Also, it does not provide data transfer.

As a result, we consider the DT10-APSI protocol in the context of APSI-DT *without* pre-distribution. Note that DT10-APSI also provides Server and Client unlinkability, as well as forward security.

A.4 APSI-DT with Pre-distribution

IBE-APSI. The protocol in Fig. 3 of [21] presents a protocol based on Boneh-Franklin Identity-based Encryption [10], which can be adapted to APSI-DT. We denote this protocol as IBE-APSI. Note that such a construct is described in the context of a different primitive – Privacy-Preserving Information Transfer (PPIT). However, it can be trivially converted to APSI-DT (see [22]). First, the authorization authority (acting as the IBE PKG) generates a prime q, two groups $\mathbb{G}_1, \mathbb{G}_2$ of order q, a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. A random $s \in \mathbb{Z}_q$ is selected as a secret master key. Then, a random generator $P \in \mathbb{G}_1$ is chosen, and Q is set such that $Q = s \cdot P$. (P, Q) are public parameters. Client obtains authorization for an element c_i as an IBE secret key, $\sigma_i = s \cdot H(c_i)$. In the pre-distribution phase, Server first selects a random $z \in \mathbb{G}_1$ and then, for each $(s_j, data_j)$, publishes (t_j, e_j) where $t_j = H_1(e(Q, H(s_j))^z)$ and e_j is the IBE encryption of $data_j$ under identifier s_j. Then, Server gives Client $R = zP$ and Client computes $t'_i = H_1(e(R, \sigma_i))$. For any t'_i, s.t. $t'_i = t_j$, Client can decrypt e_j. The protocol can be speeded up by encrypting e_j under symmetric key $H_2(e(Q, H(s_j))^z)$. Remark that IBE-APSI has two drawbacks compared to APSI-DT: it provides neither Server unlinkability nor forward security.