Cheaper Private Set Intersection via Differentially Private Leakage

Adam Groce[*] Peter Rindal[†] Mike Rosulek[‡]

February 27, 2019

Abstract

In this work we demonstrate that allowing differentially private leakage can significantly improve the concrete performance of secure 2-party computation (2PC) protocols. Specifically, we focus on the private set intersection (PSI) protocol of Rindal and Rosulek (CCS 2017), which is the fastest PSI protocol with security against malicious participants. We show that if differentially private leakage is allowed, the cost of the protocol can be reduced by up to 63%, depending on the desired level of differential privacy. On the technical side, we introduce a security model for differentially-private leakage in malicious-secure 2PC. We also introduce two new and improved mechanisms for "differentially private histogram overestimates," the main technical challenge for differentially-private PSI.

1 Introduction

Secure two-party computation (2PC) allows two parties to evaluate a function on private inputs, and learn only the output of the function. Standard security definitions for secure computation provide extremely strong cryptographic guarantees. In many situations it is reasonable to consider relaxing these guarantees if it results in a significantly faster protocol.

In this work, we consider relaxing security by allowing the protocol to leak some "extra" information. This raises the question, what extra information is reasonable to leak without completely undermining the idea of secure computation? A natural candidate is to leak only *differentially private* (DP) information about the private inputs. Differential privacy [DMNS06] ensures that the adversary obtains similar leakage, whether a particular individual's information was used in the computation or not.

1.1 Our Contributions

We show a sizeable tradeoff between performance and differentially private leakage in the setting of private set intersection (PSI). In PSI, two parties have input sets X and Y, and wish to learn (only) their intersection $X \cap Y$. Differential privacy is an especially good match for this problem, since many of its most notable applications involve sets of *individuals.* For example, the application of private contact discovery in social networks involves a client with a small set of email addresses (i.e., address book), and a server with a large set of user email addresses, who compute the corresponding

[*] Reed College. agroce@reed.edu. Partially supported by NSF award SaTC-1817245. Work done partially while visiting Oregon State University.

[†] Visa Research. PeterRindal@gmail.com. Work done while at Oregon State University.

[‡] Oregon State University. rosulekm@eecs.oregonstate.edu. Partially supported by NSF award 1617197 and a Google Research Award.

intersection. Differential privacy in this setting would ensure that, e.g., the server cannot learn whether a particular user Charlie was present in the client's address book.

Our starting point is the protocol of Rindal & Rosulek [RR17], which is the fastest malicious-secure PSI protocol to date. Like many PSI protocols (especially in the tradition of [FNP04]), this one works by first having the parties hash their items into bins (e.g., item x is placed in bin $h(x)$), and a smaller PSI is performed in each bin. When using hashing, dummy items must be added to fill each bin to a maximum size. This is because the number of true items in each bin is information that cannot be inferred from the intersection alone (i.e., it is leakage from a security standpoint). Unfortunately, dummy items vastly outnumber real items, and so contribute the majority of the protocol's cost.

Instead of *completely* hiding the load of the bins (by padding with dummy items to an upper bound), we can consider releasing a differentially-private estimate of the load. Furthermore, this estimate must be an *overestimate* of the true load, in order to preserve correctness of the PSI output. The challenge is to design a "differentially-private load overestimate" mechanism that is as close to the true load as possible, thereby minimizing the number of dummy items.[1]

The idea of differentially-private over-estimates of the true bin load has been used previously in the context of PSI (and private record linkage) protocols (e.g., [HMFS17]). As we show, the specific mechanism used in prior work is less effective when the load in each bin is rather small (e.g., typically 10-20), as is the case here. Hence, **we introduce two improved mechanisms (i.e., better accuracy) for load overestimation.** We consider these new mechanisms in the context of the [RR17] PSI protocol, but they can be used to give coresponding improvement in [HMFS17] and in other work unrelated to PSI (e.g., [KKNO17]). Our first mechanism improves accuracy by leveraging the fact that the load of a bin follows a known binomial distribution. Our second mechanism is a simple rounding approach (round the number of items in a bin up to either t_1 or t_2) that gives even better performance. Since this second mechanism is deterministic, it does not achieve standard differential privacy, but we show that it provides a guarantee known as *distributional differential privacy (DDP)* [BGKS13]. The DDP model applies when the adversary has sufficient uncertainty in the sensitive data. We discuss how to interpret DDP and assess its applicability for PSI in Section 5.3.

On the theoretical side, we **introduce a security model for *malicious-secure* MPC with differentially private leakage.** Previous work combining MPC and differential privacy (starting with [BNO08]) considered only the semi-honest setting, which masks some important subtleties that we discuss. We prove that a straightforward generalization of the [RR17] protocol is a secure PSI protocol against malicious adversaries in this model, when using any suitable private load-overestimate function.

Finally, we have incorporated our new techniques into the implementation of [RR17] to present a **thorough empirical analysis of the tradeoff between privacy and performance.** Allowing differentially private leakage improves the performance of the protocol by 33% (for a relatively strict $\epsilon = 0.5$ guarantee) to 63% (for a weaker $\epsilon = 4$ guarantee). Allowing $\epsilon = 0.33$ distributionally DP leakage for the sender (and $\epsilon = 1$ standard DP for the receiver) improves the performance by 52% over the fully secure variant.

1.2 Related Work

Trading leakage for performance in generic MPC. The dual execution paradigm, introduced by Mohassel & Franklin [MF06] and extended in other work [HKE12, KMRR15], relaxes the ma-

[1] One can think of the fully-secure protocol of [RR17] as a protocol that uses a very pessimistic load overestimate which leaks nothing about the true load.

licious security model by allowing the adversary to learn an arbitrary *bit* about the honest party's input. The resulting protocol is only 2× the cost of a standard semi-honest protocol. The dual execution paradigm can be used to securely evaluate any functionality, but does not guarantee that the leakage satisfies any meaningful privacy limitation (other than being 1 bit). Our approach also relaxes security by allowing some leakage, but in the specific case of PSI we ensure that the leakage is differentially private.

Beimel et al. [BNO08] explored a model of MPC where the standard privacy guarantee is replaced by the natural differentially private analog. This security model is inherently tied to the semi-honest adversarial setting, and not suitable for our malicious setting. We believe our security model more clearly separates the different intuitive security goals of MPC and makes it explicit which goals are ensured in a cryptographic or differentially private sense (e.g., we enforce correctness in the standard sense but input independence and privacy in a differentially private sense).

Trading leakage for performance in special-purpose MPC. The most closely related work to ours is that of He et al. [HMFS17]. They study the problem of private record linkage (a generalization of private set intersection) with differentially private leakage. They include several approaches that degrade the correctness of the result — we do not consider these approaches since we focus on computing the intersection exactly. Their technique that preserves exact correctness uses a similar high-level approach to ours, with "differentially private load overestimates." However, we show that their technique (also used in [KKNO17]) is less effective when applied to the PSI protocol of [RR17] and realistic privacy parameters. Our new mechanisms for the same problem have significantly better accuracy (i.e., fewer dummy items) and lead to significant performance improvement. Our mechanisms could easily be used in their protocol for a corresponding improvement.

Another notable instance of trading leakage for performance is in the context of searchable encryption, and more generally secure database search. The encrypted database systems BlindSeer [PKV$^+$14] and that of Cash et al. [CJJ$^+$13] provide standard database (DBMS) functionality, while hiding the nature of the query from the server with the exception of some leaked information. Schoppmann et al. [SGB18] perform nearest neighbor queries on a private database, while allowing differentially private leakage. As in our work, this leakage is tailored to the protocol and specific functionality.

Oblivious RAM (ORAM) [Gol87, Ost90] refers to a protocol between a client and server that allows the client to read/write its data held by the server, but hiding the client's access pattern from the server. Some relatively recent work [WCM16, MG17, CCMS17] has explored relaxing the obliviousness requirement of ORAM to allow the server to learn only differentially private information about the access pattern. Similar to our work, the goal is to compute a "plain" functionality with the standard notion of correctness — i.e., without adding any noise. In this case, the functionality corresponds to RAM read/write instructions; in our case the functionality is PSI.

In all works in this section, the security model is honest-but-curious whereas we consider the malicious security setting.

Securely Computing Differentially-Private Functions. There is a great deal of work on using secure computation to evaluate a differentially-private mechanism – e.g., [PNH17, KOV14, CGBL$^+$17, RN10, BSMD10] – including protocols for closely related functionalities like cardinality of set intersection cardinality [NH12] and union [FMJS17]. In these works, the focus is to compute a differentially-private functionality, under a *standard security notion*. In other words, the parties learn *only* a differentially private answer, while in our work the parties learn a non-differentially private function, plus some *additional* (incidental) differentially private information.

2 Differential Privacy Background

Differential privacy [DMNS06] is a condition meant to ensure that releases from large databases do not violate the privacy of any of the individuals whose data that database contains. In particular, let $X = (X_1, X_2, \ldots, X_n)$ be a database of information about n individuals, with X_i being the data corresponding to the i^{th} individual. A query is a (possibly randomized) function f that takes a database as input. Differential privacy ensures that the inclusion of individual i in the database does not allow an adversary to infer anything about them.

The strength of the definition is controlled by a parameter ϵ. Intuitively, differential privacy guarantees that whatever $f(X)$ outputs, that output would have been roughly equally likely even if any individual had instead changed their data arbitrarily. See [KS08] for a rigorous treatment.

More formally, we say that two databases X and X' are *neighboring* if they differ only in the addition or deletion of a single value. We can then define differential privacy.

Definition 1. *A query function f is **ϵ-differentially private (ϵ-DP)** if for any two neighboring databases X and X' and for any set S of possible outputs, we have*

$$\frac{\Pr[f(X) \in S]}{\Pr[f(X') \in S]} \leq e^{\epsilon}. \tag{1}$$

Differential privacy also benefits from composition. That is, if f is ϵ_1-differentially private and g is ϵ_2-differentially private, the function h that gives both their outputs (i.e., $h(X) = (f(X), g(X))$) is $(\epsilon_1 + \epsilon_2)$-differentially private. This is useful for two reasons. First, it's a useful tool for building differentially private functions, since one can build them up from smaller private components. Second, it means that the definition protects people's privacy even if many separate differentially private computations are done, including queries called by different people.

Differential privacy is also preserved by any additional post-processing. That is, if f is ϵ-differentially private, then so is $g \circ f$ for any g. This means anything that can be computed from a private output (with additional access to the database) is itself private. This, like composition, is useful both because it's an intuitively desirable property of a privacy definition and because it's useful in practice when designing private queries.

There are differentially private mechanisms that approximate a wide variety of naturally desirable functions. In the most simple query algorithms, the output is simply equal to the desired (non-private) output plus some carefully calibrated noise. In particular, say that F is some function on databases with output in $\mathbb{R}^n$. We define the *sensitivity* F to be the maximum amount of change F can see when one row of the database is changed.

Definition 2. *The **sensitivity** of a function F with output in $\mathbb{R}^n$ is $\max \|F(X) - F(X')\|$, where the maximum is taken over all neighboring databases X and X' and the norm is an ℓ_1 norm.*

Our techniques rely primarily on several uses of the Laplace mechanism. This uses random noises generated according to a *Laplace distribution*, a double-sided exponential distribution.

Definition 3. *The **Laplace distribution** Lap_c is parameterized by a scaling parameter c. We denote the probability density function at x with $\mathsf{Lap}_c(x)$. Precisely,*

$$\mathsf{Lap}_c(x) = \frac{1}{2c} e^{-|x|/c}. \tag{2}$$

Any function can be made differentially private by adding Laplace noise proportional to its sensitivity and $1/\epsilon$. (In some cases this is optimal, while in others more elaborate techniques can give greater utility.)

Theorem 4. *Let F be a function with sensitivity ΔF and output in $\mathbb{R}^n$. Let f be a randomized algorithm with output $f(X) = F(X) + \mathsf{Lap}^n_{\Delta F/\epsilon}$, where $\mathsf{Lap}^n_{\Delta F/\epsilon}$ denotes a vector of n independently chosen values from the Laplace distribution with parameter $c = \Delta F/\epsilon$. Then f is ϵ-differentially private.*

3 Security model

3.1 UC-Secure 2-party Computation

We define security for 2-party computation using the Universal Composition (UC) framework of [Can01]. The formal details of this model are beyond the scope of this work, but we present a brief summary here.

Informally, security is defined in the real/ideal paradigm. A protocol is deemed secure if, for any attack against the protocol, there is an equivalent attack in an "ideal" world where the function is computed by a trusted third party.

In more detail, the UC model considers the following entities: *Functionalities* are trusted third parties who cannot be corrupted, and who carry out a specific behavior in response to commands/inputs from parties. *Parties* execute a specified protocol interacting with each other, the environment, and functionalities. *Adversaries* corrupt parties, and cause them to deviate arbitrarily from the protocol. In this work we consider static security, where the identity of corrupt parties is decided before the interaction starts, and parties cannot become corrupted during the execution of the protocol. The *environment* is an arbitrary program that chooses the inputs of honest parties, receives their outputs, and interacts arbitrarily with the adversary.

Suppose π is a protocol (i.e., π is the program that describes the behavior of the honset parties). We allow protocols to be *hybrid protocols* in the sense that π can instruct parties to send messages to each other and also interact with a trusted functionality $\mathcal{G}$. We write $\textsc{uc}(\pi, \mathcal{G}, \mathcal{A}, \mathcal{Z}, \kappa)$ to denote the output of an interaction between: honest parties running protocol π, functionality $\mathcal{G}$, adversary $\mathcal{A}$ (possibly corrupting some of the parties), and environment program $\mathcal{Z}$, all with common security parameter κ. The output is defined as the terminal output of $\mathcal{Z}$.

Definition 5. *A protocol π **UC-securely realizes** a functionality $\mathcal{F}$ if: for all adversaries $\mathcal{A}$ there exists an adversary $\mathcal{S}$ (called a simulator) who corrupts the same set of parties, such that for all environments $\mathcal{Z}$:*

$$\Pr[\textsc{uc}(\pi, \mathcal{G}, \mathcal{A}, \mathcal{Z}, \kappa) = 1] - \Pr[\textsc{uc}(\pi_{\mathsf{dummy}}, \mathcal{F}, \mathcal{S}, \mathcal{Z}, \kappa) = 1]$$

is negligible in κ. Here π_{dummy} is the so-called dummy protocol where parties simply relay inputs from the environment directly to the functionality, and relay outputs from the functionality directly to the environment.

Intuitively, the goal of $\mathcal{Z}$ (with the help of $\mathcal{A}$) is to determine whether it is controlling honest parties who run the protocol (the first, "real-world" interaction), or communicating directly with the functionality $\mathcal{F}$ (the second, "ideal-world" interaction).

The role of the simulator $\mathcal{S}$ is two-fold: (1) it must provide to $\mathcal{A}$ messages that look like protocol messages from honest parties running π, without knowing the honest parties' inputs — only seeing outputs given by $\mathcal{F}$. (2) it must observe $\mathcal{A}$'s protocol messages and "extract" meaning from them, so that it can send appropriate inputs to $\mathcal{F}$ that induce correct outputs of honest parties (since $\mathcal{Z}$ sees these outputs).

Parameters:

- Functions $f_1, f_2 : (\{0,1\}^*)^2 \to \{0,1\}^*$
- Class of leakage function pairs $\mathcal{L}$

Behavior:
If no parties are corrupt:

1. Wait for inputs x_1 from P_1 and x_2 from P_2
2. Deliver output $f_1(x_1, x_2)$ to P_1 and $f_2(x_1, x_2)$ to P_2

If any party is corrupt, let P_c denote the corrupt party and P_h denote the honest party. Then:

1. Wait for input x_h from P_h and $(\textsc{leak}, L_{\mathsf{pre}})$ from P_c, where $\mathcal{L}$ contains some pair of the form $(L_{\mathsf{pre}}, \cdot)$.
2. Give $L_{\mathsf{pre}}(x_h)$ to P_c
3. Wait for inputs x_c and $(\textsc{leak}, L_{\mathsf{post}})$ from P_c, where $(L_{\mathsf{pre}}, L_{\mathsf{post}}) \in \mathcal{L}$.
4. Give $L_{\mathsf{post}}(x_h)$ and $f_c(x_1, x_2)$ to P_c
5. Wait for input $(\textsc{deliver}, b)$ from P_c. If $b = 0$ deliver output $\perp$ to P_h; otherwise if $b = 1$ deliver output $f_h(x_1, x_2)$ to P_h.

Figure 1: Ideal functionality $\mathcal{F}_{\mathsf{leak},f,\mathcal{L}}$ for securely evaluating function $f = (f_1, f_2)$ with leakage.

3.2 UC Security with Differentially Private Leakage

Prior Work. Prior work (*e.g.*, [BNO08, HMFS17, SGB18]) introduces security definitions for secure computation with differentially private leakage. All of these works consider only security against semi-honest adversaries (i.e., adversaries who still follow the protocol), while in this work we consider security against malicious adversaries (who may deviate from the protocol). The semi-honest setting is significantly simpler from a definition standpoint.

To understand why, let's consider the security definition given in [HMFS17] as a concrete representative example of these prior works. For simplicity, consider only security against a corrupt Bob. Their definition of security is as follows: for all inputs y for Bob, and for all x, x' which are "neighbors" in the DP sense, and that satisfy $f(x, y) = f(x', y)$, consider the distribution of Bob's view when the parties run honestly with inputs (x, y) vs (x', y). Instead of requiring these view distributions to be indistinguishable (as in standard 2PC security), they require only that these two view distributions to be close in a differentially-private sense. More precisely, the randomized function that generates Bob's view, from Alice's private input, is differentially private.

Challenges of Malicious Security. Generalizing these existing definitions to the malicious setting is not trivial. In particular, there are several problems that are encountered.

First, note how in the semi-honest model, even a corrupt Bob's input y is fixed before the interaction starts. It is therefore possible to quantify over inputs x, x' for Alice such that $f(x, y) = f(x', y)$. In the malicious model, the input of a corrupt party is not well-defined in the real interaction. Even in the ideal interaction, the input is not determined until the simulator extracts it. To see one example of why this could be a problem, consider a protocol that leaks differentially-private information about x to Bob early in the protocol, before Bob becomes bound to his own input. Our protocol indeed has this property, and it allows Bob's choice of input to depend on Alice's. Intuitively, malicious-secure 2PC guarantees not only privacy (the protocol leaks no more than $f(x, y)$) but it also guarantees *input independence* (Bob must choose y independently of x). We may wish to consider protocols that degrade the input-independence guarantee — not only the privacy guarantee — in a differentially-private way! Even better, we might want a model that allows granularity in choosing to allow input-independence or only privacy to be degraded in a differentially-private way.

Second, the model should allow adversarial influence on the differentially-private leakage. Since a malicious adversary has arbitrary influence over the protocol, and the honest party responds to the adversary's protocol messages, the adversary has potential influence over the leakage. As a simple example, suppose the protocol leaks information $L(x, q)$ about Alice's input x, where q is somehow chosen by the adversary. If for *every* q, the function $L(\cdot, q)$ is differentially private, then the model should allow such leakage.

Finally, the existing semi-honest definitions are not simulation-based but indistinguishability-based. In the semi-honest setting, these notions are equivalent, but in general they are not for malicious security. In particular, a desirable property of simulation-based security is that it leads to a **composability** property: if π uses $\mathcal{G}$ to securely realizes $\mathcal{F}$, and ρ securely realizes $\mathcal{G}$, then substituting calls to $\mathcal{G}$ in π with calls to ρ results in a secure protocol for $\mathcal{F}$. Hence, a simulation-based definition is highly desirable.

Our Model. We choose to address these complications by *adding explicit leakage* to the ideal functionality in the UC security definition.

Figure 1 describes an ideal functionality for securely evaluating a function while also giving some leakage. The adversary's choice of leakage is constrained to some class $\mathcal{L}$ of allowable leakage functions (a parameter of the functionality). The functionality allows the corrupt party to obtain leakage on the honest party's input at two different times: both before and after choosing its own input. If one restricts $\mathcal{L}$ to contain only pairs of the form $(L_{\mathsf{pre}}, \perp)$ or of the form $(\perp, L_{\mathsf{post}})$, where $\perp$ is overloaded to denote the function $\perp(x) = \perp$ for all x, then one obtains a definition with just one leakage phase. In particular, if the L_{pre} leakage is $\perp$, then input independence is guaranteed in a cryptographic sense (i.e., not degraded in a differentially private manner), since the adversary must choose its input based on no information about the honest party's input.

Definition 6. *A protocol π securely realizes $f = (f_1, f_2)$* ***with $\mathcal{L}$ leakage*** *if π is a UC-secure protocol for $\mathcal{F}_{\mathsf{leak},f,\mathcal{L}}$ (Figure 1).*

The protocol realizes f ***with ϵ-DP leakage*** *if it realizes f with $\mathcal{L}$ leakage, where for every $(L_{\mathsf{pre}}, L_{\mathsf{post}}) \in \mathcal{L}$, the function $x \mapsto (L_{\mathsf{pre}}(x), L_{\mathsf{post}}(x))$ is ϵ-differentially private (under the appropriate "neighbor" relation for values of x).*

For example, in the case of PSI, the neighbor relation is addition/removal of a single item from a party's input set. Secure computation with ϵ-DP leakage ensures that an adversary learns only ϵ-DP information about the individuals in the honest party's set.

Note that in this model, correctness of the output is still guaranteed in the standard cryptographic sense (e.g., with overwhelming probability).

Interplay Between Cryptographic and DP Guarantees. Our model allows differentially-private leakage (explicitly from the functionality) in addition to negligible simulation error (as part of UC security definition). When the leakage is ϵ-DP, it is reasonable to interpret the entire view of the adversary as having (ϵ, δ)-differential privacy[2], where δ is the negligible error from the UC-security simulation. Standard security corresponds to the case of $\epsilon = 0$ and δ negligible.

In order to compare to the original, fully-secure PSI protocol of [RR17], we use the same security parameters in our work. In particular, the protocol can fail with probability 2^{-40}. The reader who prefers to think of the entire system in terms of (ϵ, δ)-DP can do so with $\delta = 2^{-40}$.

4 PSI Protocol Framework

In this section we describe our PSI protocol framework (i.e., PSI with histogram leakage). The framework is based on the PSI protocol of Rindal & Rosulek [RR17], which is the fastest known PSI protocol achieving malicious security.

4.1 Overview

We first review the Rindal-Rosulek protocol. At a high level, the protocol works by having the parties first hash their items into bins (with a random hash function). Within each bin, they perform a quadratic-cost PSI protocol on the items assigned to that bin. However, the abstraction boundary is broken slightly by an optimization that combines together some information across all of the bins.

Quadratic PSI. This step of the protocol uses an oblivious encoding functionality, which can be thought of as an *oblivious pseudorandom function* (OPRF) or a variant of random oblivious transfer over an exponentially large number of values. The functionality chooses a random function which we denote as $x \mapsto [\![x]\!]$. A receiver can learn $[\![x]\!]$ for a *single, chosen* value x, while the sender can compute $[\![x]\!]$ for *any number of* values x. Looking forward, each party will choose n such functions which for $j \in [n]$ we denote as $x \mapsto [\![x]\!]^{\mathsf{B}}_j$ when Bob is the sender and $x \mapsto [\![x]\!]^{\mathsf{A}}_j$ for Alice. Concretely, the encoding protocol uses the OT extension protocol of Orrù, Orsini & Scholl [OOS17], which is secure in the random oracle model. It achieves a slight relaxation of this functionality where a corrupt sender is allowed to choose the mapping $[\![\cdot]\!]$. This variant functionality is described formally in Figure 2, and it is sufficient for use in the PSI protocol.

If Alice has n items $\{x_1, \ldots, x_n\}$ and Bob has n items $\{y_1, \ldots, y_n\}$, they can use this oblivious encoding functionality to perform PSI in the following way:

1. For each x_j, the parties perform an instance of the oblivious encoding with Alice acting as receiver, so that Alice learns $[\![x_j]\!]^{\mathsf{B}}_j$. We use $[\![\cdot]\!]^{\mathsf{B}}_j$ to signify that Bob can compute the encoding for any value.

2. Symmetrically, for each y_i, the parties perform an oblivious encoding instance in the other direction, with Bob learning $[\![y_i]\!]^{\mathsf{A}}_i$.

3. The parties define n^2 "common encodings" of the form:

$$[\![v]\!]^{*}_{i,j} \overset{\text{def}}{=} [\![v]\!]^{\mathsf{A}}_i \oplus [\![v]\!]^{\mathsf{B}}_j$$

   E.g, for each $[\![x_j]\!]^{\mathsf{B}}_j$ Alice learned in step 1, she computes the n common encodings $\{[\![x_j]\!]^{*}_{i,j} = [\![x_j]\!]^{\mathsf{A}}_i \oplus [\![x_j]\!]^{\mathsf{B}}_j \mid i \in [n]\}$.

[2] This definition demands that $\Pr[f(X) \in S] \leq e^{\epsilon} \Pr[f(X') \in S] + \delta$, for all neighboring X, X' and all S. Standard DP corresponds to $\delta = 0$.

Parameters: two parties denoted as Sender and Receiver. The input domain $\{0,1\}^\sigma$ and output domain $\{0,1\}^\ell$ for a private F.

1. **[Initialization]** Create an initially empty associative array $F : \{0,1\}^\sigma \to \{0,1\}^\ell$.
2. **[Receiver Encode]** Wait for a command $(\textsc{Encode}, \mathsf{sid}, c)$ from the Receiver, and record c. Then:
3. **[Adversarial Map Choice]** If the sender is corrupt, then send $(\textsc{RecvInput}, \mathsf{sid})$ to the adversary and wait for a response of the form $(\textsc{Deliver}, \mathsf{sid}, F_{\mathsf{adv}})$. If the sender is honest, set $F_{\mathsf{adv}} = \bot$. Then:
4. **[Receiver Output]** If $F_{\mathsf{adv}} = \bot$ then choose $F[c]$ uniformly at random; otherwise set $F[c] := F_{\mathsf{adv}}(c)$, interpreting F_{adv} as a circuit. Give $(\textsc{Output}, \mathsf{sid}, F[c])$ to the receiver and $(\textsc{Output}, \mathsf{sid})$ to the sender. Then:
5. **[Sender Encode]** Stop responding to any requests by the receiver. But for any number of commands $(\textsc{Encode}, \mathsf{sid}, c')$ from the sender, do the following:
   - If $F[c']$ doesn't exist and $F_{\mathsf{adv}} = \bot$, choose $F[c']$ uniformly at random.
   - If $F[c']$ doesn't exist and $F_{\mathsf{adv}} \neq \bot$, set $F[c'] := F_{\mathsf{adv}}(c')$.
   - Give $(\textsc{Output}, \mathsf{sid}, c', F[c'])$ to the sender.

Figure 2: The Oblivious Encoding ideal functionality $\mathcal{F}_{\mathsf{encode}}$ [OOS17]

The idea of the common encodings is that Alice can predict the value of $\llbracket v \rrbracket^{*}_{i,j}$ *if and only if* she used v as input to the encoding step involving $\llbracket \cdot \rrbracket^{\mathsf{B}}_{j}$. Bob can predict the value iff he used v as input to the encoding $\llbracket \cdot \rrbracket^{\mathsf{A}}_{i}$.

The protocol continues with Alice sending the (randomly ordered) set of encodings:

$$E = \{\llbracket x_j \rrbracket^{*}_{i,j} \mid i, j \in [n]\}$$

Bob can check this set for encodings that are known to him (i.e., all encodings of the form $\llbracket y_i \rrbracket^{*}_{i,j}$). Bob computes the protocol output as:

$$Z = \{y_i \mid \exists j : \llbracket y_i \rrbracket^{*}_{i,j} \in E\}$$

Suppose the encodings $\llbracket \cdot \rrbracket$ have length $\lambda + 2\log_2 n$ bits. Then the probability of $\llbracket x_j \rrbracket^{*}_{i,j} = \llbracket y_i \rrbracket^{*}_{i,j}$ for $x_j \neq y_i$ is bounded by $2^{-\lambda}$. Conditioned on this event not happening, the protocol is correct.

To see the security of the protocol, note that the simulator can observe the adversary's inputs when the adversary plays the role of the $\mathcal{F}_{\mathsf{encode}}$-receiver. If some x is never used as such an input to $\mathcal{F}_{\mathsf{encode}}$, then the adversary could never predict any common encoding of the form $\llbracket x \rrbracket^{*}_{i,j}$. In other words, the adversary's input (which the simulator must extract) must be a subset of this set of candidates known to the simulator. In the case of a corrupt Alice, the simulator must further test which of these candidates are represented in the encodings sent in the message E. The details are discussed in Appendix A.

Hashing and Dummy Items. This basic protocol requires only $2n$ instances of the oblivious encoding functionality (n in each direction), but requires Alice to communicate n^2 encodings.

A standard approach to reduce the complexity of a PSI protocol (dating back at least to [FNP04]) is for parties to first choose[3] a random hash function $h : \{0,1\}^* \to [m]$ and use this function to assign their items into m bins. Then the quadratic-cost protocol can be performed on the items within each bin.

As mentioned in Section 1.1, it is necessary to hide the number of items per bin — in the ideal world, it is not possible to infer the number of items that honest party has that satisfy $h(x) = b$ for a chosen bin b. To obscure this information, the parties add dummy items to each bin until each bin has exactly the worst-case number of items (such a bound can be computed using standard balls-in-bins analysis).

A typical choice of parameters is $m = O(n/\log n)$ bins. In that case, both the *expected* and *worst case* (with overwhelming probability) number of items per bin is $O(\log n)$. Then the overall cost of the protocol is $m \cdot O(\log^2 n) = O(n \log n)$.

However, further optimization is still possible. Let μ be the (padded) size of each bin. As currently described, Alice would send μ encodings for each of her items, *including her dummy items!* However, it is public information that *overall* Alice has only n items and hence only $n\mu$ encodings corresponding to them. Only the distribution of these dummies *within* the bins is secret. So the suggestion of [RR17] is to let Alice gather all $n\mu$ non-dummy encodings from all bins, shuffle them, and send them together. Now the encodings must be somewhat longer (the probability of a spurious collision in these encodings has increased), and Bob must lookup candidate encodings in a larger set, rather than in small bin-specific sets. However, the gain in communication makes this optimization an overall improvement to the protocol.

4.2 Our Generalization and Details

As mentioned in Section 1.1, our modification of the Rindal-Rosulek protocol is to release some differentially private information about the number of items in each bin. We begin by defining the properties we require:

Definition 7. *Let $h : \{0,1\}^* \to [m]$ be a hash function. For a set of items X hashed into m bins by h, the load of bin i is:*

$$\mathsf{Ld}_h(X, i) = \#\{x \in X \mid h(x) = i\},$$

The load of all bins can be written as a vector:

$$\mathsf{Ld}_h(X) = \Big(\mathsf{Ld}_h(X, 1), \ldots, \mathsf{Ld}_h(X, m)\Big)$$

Definition 8. *For two vectors $u = (u_1, \ldots, u_m)$ and $v = (v_1, \ldots, v_m)$, write $u \leq v$ if $u_i \leq v_i$ for all $i \in [m]$. Let $\widetilde{L}_h$ be a randomized function. We call $\widetilde{L}$ a* ***load overestimate*** *if for all X,*

$$\Pr[\mathsf{Ld}_h(X) \leq \widetilde{L}_h(X)] \text{ is overwhelming,}$$

where the probability is taken over the randomness of $\widetilde{L}$ and the choice of h.

One can think of the function $\widetilde{L}_h(X) = (\mu, \ldots, \mu)$ to be the load-overestimate used in [RR17], where μ is an upper bound on bin size computed using balls-in-bins analysis.

Our generalization of Rindal-Rosulek is parameterized by a load overestimate $\widetilde{L}$. Roughly speaking, after choosing the hash function h:

[3] For security reasons, h should be chosen by a secure coin-tossing protocol.

Parameters: σ is the bit-length of the parties' items. n is the size of the honest parties' sets. $n' > n$ is the allowed size of the corrupt party's set.

- On input $(\textsc{Receive}, \mathsf{sid}, Y)$ from Bob where $Y \subseteq \{0,1\}^\sigma$, ensure that $|Y| \leq n$ if Bob is honest, and that $|Y| \leq n'$ if Bob is corrupt. Give output $(\textsc{bob-input}, \mathsf{sid})$ to Alice.
- Thereafter, on input $(\textsc{Send}, \mathsf{sid}, X)$ from Alice where $X \subseteq \{0,1\}^\sigma$, likewise ensure that $|X| \leq n$ if Alice is honest, and that $|X| \leq n'$ if Alice is corrupt. Give output $(\textsc{Ouput}, \mathsf{sid}, X \cap Y)$ to Bob.

Figure 3: Ideal functionality $\mathcal{F}_{\mathsf{PSI}}$ for private set intersection (with one-sided output)

- Alice hashes her items X into bins, computes the load of each bin $(a_1, \ldots, a_n) = \mathsf{Ld}_h(X)$, and an overestimate $(\widetilde{a}_1, \ldots, \widetilde{a}_m) \leftarrow \widetilde{L}_h(X)$. She adds dummy items to each bin until the load (including dummy items) equals the overestimate.
- Likewise, Bob hashes his items Y into bins. He can compute an overestimate $(\widetilde{b}_1, \ldots, \widetilde{b}_m) \leftarrow \widetilde{L}_h(Y)$, however [RR17] security proof / simulation breaks down in the case where Alice has more items in a bin than Bob (we discuss this fact in Appendix A). We must therefore have Bob compute $\widetilde{c}_i = \max\{\widetilde{a}_i, \widetilde{b}_i\}$ and add dummy items so that the load in the bins is $(\widetilde{c}_1, \ldots, \widetilde{c}_m)$.[4]
- Within each bin i, the parties perform the standard quadratic PSI between $\widetilde{a}_i$ and $\widetilde{c}_i$ $(\geq \widetilde{a}_i)$ items.

In [RR17], it is public information that Alice will have $n\mu$ total non-dummy encodings, since each bin contains exactly μ items. In our case, Alice will have $\sum a_i \widetilde{c}_i$ non-dummy encodings. She cannot simply send the non-dummy encodings as in [RR17] since the number of these items is indeed sensitive to her true bin-load. To protect this information, we need a differentially-private overestimate of this inner product:

Definition 9. *We say that randomized function $\widetilde{I}$ is a **inner-product overestimate** if for all vectors $u = (u_1, \ldots, u_m)$ and $v = (v_1, \ldots, v_m)$, $\Pr[\langle u, v\rangle \leq \widetilde{I}(u, v)]$ is overwhelming.*

In our generalization, Alice computes a differentially-private[5] inner-product overestimate $\widetilde{e} = \widetilde{I}((a_1, \ldots, a_m), (\widetilde{c}_1, \ldots, \widetilde{c}_m))$. She includes the non-dummy encodings in a set E, adds random values until $|E| = \widetilde{e}$, and finally sends the (permuted) contents of E to Bob.

The formal details of the protocol are given in Figure 4. Unsurprisingly, the security proof is extremely similar to that of [RR17]. For the sake of completeness, we present the self-contained proof in Appendix A. Roughly speaking, the only differences from [RR17] are that (1) the number of items per bin is changed to be input-dependent; (2) the number of common encodings sent by Alice is input-dependent. However, if the simulator is given these values (e.g., as leakage from the functionality), then simulation proceeds just as in [RR17].

Theorem 10. *The protocol, in the $\mathcal{F}_{encode}$ hybrid, securely realizes the leaky PSI functionality*

[4] Note that Alice's privacy depends only on the load overestimate that she computes; it does not depend on any property of Bob's load overestimate (and vice-versa). An adversary hurts only itself by reporting a different overestimate.

[5] Specifically, for all second arguments, $\widetilde{I}$ should be a differentially private function of its first input.

Parameters: X is Alice's input, Y is Bob's input, where $X, Y \subseteq \{0,1\}^\sigma$. m is the number of bins. The protocol uses instances of $\mathcal{F}_{\mathsf{encode}}$ with input length σ, and output length $\lambda + 2\log(n\mu)$, where λ is the security parameter and μ is an upper bound on the number of items in a bin.
Finally, a load-overestimate function $\widetilde{L}$ and an inner-product overestimate function $\widetilde{I}$.

1. **[Parameters]** Parties agree on a random hash function $h : \{0,1\}^\sigma \to [m]$ using a coin tossing protocol.
2. **[Hashing]**
   (a) For $x \in X$, Alice adds x to bin $\mathcal{B}_X[h(x)]$. She computes load overestimate $(\widetilde{a}_1, \dots, \widetilde{a}_m) \leftarrow \widetilde{L}_h(X)$ and announces this value to Bob. She adds dummy items to the bins so that each bin i has exactly $\widetilde{a}_i$ items (she aborts in the event that there are already more than $\widetilde{a}_i$ items in bin i). The items in each bin $\mathcal{B}_X[i]$ are randomly permuted.
   (b) For $y \in Y$, Bob adds y to bin $\mathcal{B}_Y[h(y)]$. He computes an overestimate $(\widetilde{b}_1, \dots, \widetilde{b}_m) \leftarrow \widetilde{L}_h(X)$ and computes $(\widetilde{c}_1, \dots, \widetilde{c}_m) = (\max\{\widetilde{a}_1, \widetilde{b}_1\}, \dots, \max\{\widetilde{a}_m, \widetilde{b}_m\})$ and announces it to Alice. He adds dummy items until each bin i contains exactly $\widetilde{c}_i$ items.
3. **[Encoding]** For bin index $d \in [m]$:
   (a) For $p \in [\widetilde{a}_d]$, let x be the pth item $\mathcal{B}_X[d]$. Alice sends $(\textsc{Encode}, (\mathsf{sid}, \mathsf{B}, d, p), x)$ to the $\mathcal{F}_{\mathsf{encode}}$ functionality which responds with $(\textsc{Output}, (\mathsf{sid}, \mathsf{B}, d, p), \llbracket x \rrbracket^{\mathsf{B}}_{d,p})$. Bob receives $(\textsc{Output}, (\mathsf{sid}, \mathsf{B}, d, p))$ from $\mathcal{F}_{\mathsf{encode}}$.
   (b) For $p \in [\widetilde{c}_d]$, let y be the pth item $\mathcal{B}_Y[d]$. Bob sends $(\textsc{Encode}, (\mathsf{sid}, \mathsf{A}, d, p), y)$ to the $\mathcal{F}_{\mathsf{encode}}$ functionality which responds with $(\textsc{Output}, (\mathsf{sid}, \mathsf{A}, d, p), \llbracket y \rrbracket^{\mathsf{A}}_{d,p})$. Alice receives $(\textsc{Output}, (\mathsf{sid}, \mathsf{A}, d, p))$ from $\mathcal{F}_{\mathsf{encode}}$.
4. **[Output]**
   (a) **[Alice's Common Mask]** For each $x \in X$, in random order, let d, p be the bin index and position that x was placed in during Step 2a. For $j \in [\widetilde{c}_d]$, Alice sends $(\textsc{Encode}, (\mathsf{sid}, \mathsf{A}, d, j), x)$ to $\mathcal{F}_{\mathsf{encode}}$ and receives $(\textsc{Output}, (\mathsf{sid}, \mathsf{A}, d, j), \llbracket x \rrbracket^{\mathsf{A}}_{d,j})$ in response. Alice adds $\llbracket x \rrbracket^{\mathsf{A}}_{d,j} \oplus \llbracket x \rrbracket^{\mathsf{B}}_{d,p}$ to a set E.
   (b) Now E contains $\mathsf{Ld}_h(X) \cdot (\widetilde{c}_1, \dots, \widetilde{c}_m)$ items. Alice computes inner-product overestimate $\widetilde{e} \leftarrow \widetilde{I}(\mathsf{Ld}_h(X), (\widetilde{c}_1, \dots, \widetilde{c}_m))$ and adds randomly chosen values to E until $|E| = \widetilde{e}$. She sends E (randomly permuted) to Bob.
   (c) **[Bob's Common Mask]** Similarly, for $y \in Y$, let d, p be the bin index and position that y was placed in during Step 2b. For $j \in [\widetilde{a}_d]$, Bob sends $(\textsc{Encode}, (\mathsf{sid}, \mathsf{B}, d, j), y)$ to $\mathcal{F}_{\mathsf{encode}}$ and receives $(\textsc{Output}, (\mathsf{sid}, \mathsf{B}, d, j), \llbracket y \rrbracket^{\mathsf{B}}_{d,j})$ in response. Bob outputs

$$\left\{ y \in Y \;\middle|\; \exists j : \llbracket y \rrbracket^{\mathsf{A}}_{d,p} \oplus \llbracket y \rrbracket^{\mathsf{B}}_{d,j} \in E \right\}$$

Figure 4: Our malicious-secure PSI protocol (with leakage).

$\mathcal{F}_{\mathsf{leak,PSI},\mathcal{L}}$, for leakage $\mathcal{L}$ where every $(L_{\mathsf{pre}}, L_{\mathsf{post}}) \in \mathcal{L}$ has the form:

$$L_{\mathsf{pre}}(S) = \widetilde{L}_h(S)$$
$$L_{\mathsf{post}}(S) = \begin{cases} \widetilde{I}(\mathsf{Ld}_h(S), (\widetilde{c}_1, \dots, \widetilde{c}_m)) & \text{if Bob corrupt} \\ \perp & \text{if Bob honest} \end{cases}$$

(i.e., for some $\widetilde{c}_1, \dots, \widetilde{c}_m$).

Recall that in our security model, the adversary is allowed to choose the leakage function from a class of allowable functions. In the case of a corrupt Bob, that choice includes the choice of $\widetilde{c}_i$'s.

Using the efficient $\mathcal{F}_{encode}$ protocol of [OOS17], the entire protocol can be instantiated in the random oracle model assuming UC-secure oblivious transfer. [RR17] also present a less efficient standard-model instantiation that is secure assuming UC-secure oblivious transfer and a secure PRG. Our implementation uses the random-oracle instantiation.

Adversary's set size.. The adversary may "under-report" the number of items it actually has. For example, we expect $\widetilde{a}_i$ and $\widetilde{c}_i$ to be significant over-estimates of the true number of items, but a corrupt Alice could use $\widetilde{a}_i$ different non-dummy items in the calls to $\mathcal{F}_{\mathsf{encode}}$ for bin $\#i$ (similarly, Bob could use $\widetilde{c}_i$ distinct items).

This fact is reflected in the ideal functionality (Figure 3), where honest parties agree to use sets of size n, but a corrupt party is allowed to actually provide a set of $n' > n$ items for some bound n'. We stress that the original protocol of [RR17] has the same property, and the same analysis of the n' bound in [RR17] applies here: briefly, with overwhelming probability we have $n' = O(n)$ with a small hidden constant factor, e.g. 3.

5 Choosing Appropriate Leakage

In this section we introduce private load overestimates and inner-product overestimates that can be used in the PSI protocol to provide different flavors of differential privacy. We begin with standard differential privacy and then consider a variant called distributional differential privacy.

5.1 Differentially private load overestimate

Recall that Alice & Bob assign each item z to a bin $h(z)$ specified by a random hash function. The load of the bins $\mathsf{Ld}_h(X)$ leaks information about a party's input set X, hence our protocol instructs the parties to reveal a load overestimate $\widetilde{L}_h(X)$ instead. We introduce more accurate and differentially-private load overestimates. The more accurate $\widetilde{L}_h$ is, the fewer dummy items need to be added, and the cheaper the protocol becomes.

As a reference point, one can think of the fully malicious-secure protocol of [RR17] as using the overestimate $\widetilde{L}_h(X) = (\mu, \mu, \cdots \mu)$, where μ is computed such that:

$$\Pr_{\beta \leftarrow \mathsf{Binom}_{n,1/m}}[\beta > \mu] < 1/m2^{\lambda}.$$

The fact that this is an overestimate follows from the union bound. With m bins, the probability of some bin having more than μ items is at most $2^{-\lambda}$. This overestimate mechanism is independent of the input, and hence leaks nothing.

Simple Laplacian mechanism. A simple approach is to add Laplacian noise and also add a fixed "safety buffer" z, setting $\widetilde{a}_i = a_i + \mathsf{Lap}_{1/\epsilon_1} + z$. The buffer z is chosen so that $z + \mathsf{Lap}_{1/\epsilon_1}$ is positive (i.e., $\widetilde{a}_i \geq a_i$) with high probability – a condition that does not depend on a_i. The mechanism is described formally in Figure 5.

Theorem 11. *The load overestimate in Figure 5 is ϵ_1-DP and is an overestimate of the true load with probability at least $1 - 2^{-\lambda}$.*

Proof. Differential privacy follows from the privacy of the Laplacian mechanism, and the fact that adding z to each component is input-independent post-processing.

z is chosen so that $\Pr[\widetilde{a}_i \geq a_i] < 1/m2^{\lambda}$ for each i. By a union bound over the m bins, we have $\Pr[\exists i : \widetilde{a}_i \geq a_i] < 1/2^{\lambda}$. □

This is the mechanism used in [HMFS17] for private record linkage, and also in other contexts unrelated to PSI (e.g., by Kellaris et al. [KKNO17]).

Our experiments show that this overestimate mechanism only gives a modest improvement to the PSI protocol. For example, with $n = 2^{20}$ items, $m = 2^{20}/12$ bins, $\epsilon = 1$, and $\lambda = 40$, the safety buffer is $z = 38.4$. To put that into perspective, 98% of bins have less than 20 items. (See Figure 7 for a comparison to our new mechanism for these parameters.)

In [HMFS17], only relatively large bins (i.e., with many items) are considered. Since the effect of the safety buffer is additive, it has less of an effect for such large bins. However, in the RR17 PSI protocol, bins are much smaller in order to optimize the tradeoff between number of bins and the quadratic costs within each bin.

Recall that the RR17 protocol computes a worst-case bin load μ and then treats all bins as if they have μ items ($\mu = 51$ for the example parameters above). For low ϵ and small expected load, the safety buffer mechanism can lead to many bins whose reported size exceeds μ. However, it is safe to truncate all overestimates at μ, since the computation of μ is input-independent. In doing so, the bins are never larger than they are in the fully-secure RR17 protocol.

Improved Bayesian-update mechanism. We refine the Laplacian mechanism by taking into account the *prior knowledge* Alice and Bob have about the distribution of the true load. In particular, they know that that items are mapped to bins uniformly, so the load of each bin follows a binomial distribution parameterized by n items and success probability $p = 1/m$.

We want Alice and Bob to use the differentially private bin size estimates not to **replace** their binomial distribution priors, but rather to **update** them in the standard Bayesian sense.

Suppose β is sampled from some known distribution (e.g., binomial) $\mathcal{D}$. For any threshold τ, we can compute $\Pr_{\beta \leftarrow \mathcal{D}}[\beta > \tau]$. After getting some partial information about the actual value of β in the form of $\hat{a} \leftarrow \beta + \mathsf{Lap}_{1/\epsilon_1}$, we can update this prior distribution to compute a more accurate probability:

$$\Pr_{\beta \leftarrow \mathcal{D}}[\beta > \tau \mid \beta + \mathsf{Lap}_{1/\epsilon_1} = \hat{a}]$$

Such probabilities can be computed via a straight-forward Bayesian calculation from the known distributions $\mathcal{D}$ and Lap. For each concrete value of $\hat{a}$, we get a different posterior distribution over β. Hence, for each value of $\hat{a}$, and for any desired cutoff probability p, it is possible to compute the *minimum* τ such that

$$\Pr_{\beta \leftarrow \mathcal{D}}[\beta > \tau \mid \beta + \mathsf{Lap}_{1/\epsilon_1} = \hat{a}] < p.$$

In practice, the mapping $\hat{a} \mapsto \tau$ can be computed as a lookup table hard-coded into the protocol. When such a threshold τ is computed for every bin (with respect to the appropriate binomial distribution), we get the load overestimate described formally in Figure 6.

Theorem 12. *The load overestimate in Figure 6 is ϵ_1-DP and is an overestimate of the true load with probability at least $1 - 2^{-\lambda}$.*

Proof. Differential privacy follows the fact that τ is computed depending only on $\hat{a}_i$ (not a_i), but the $\hat{a}_i$'s are a differentially private function of X.

The definition of $\widetilde{a}_i$ ensures that $\Pr[a_i > \widetilde{a}_i] < 1/m2^\lambda$ for each i. Again, by a union bound over the m bins, $\Pr[\exists i : \widetilde{a}_i \geq a_i] < 1/2^\lambda$. □

In Figure 7 we show a concrete example of a lookup table for load overestimates, for typical parameters encountered in the PSI protocol context. This approach does indeed lead to significantly tighter load overestimates compared to the simple approach of adding a safety buffer to the differentially-private estimate.

```
$\widetilde{L}_h(X)$:
  compute true load $(a_1, \ldots, a_m) = \mathsf{Ld}_h(X)$
  compute $z$ such that $\Pr[\mathsf{Lap}_{1/\epsilon_1} < -z] < 1/m2^\lambda$
  for $i = 1$ to $n$:
    $\widetilde{a}_i \leftarrow a_i + \mathsf{Lap}_{1/\epsilon_1} + z$
  return $(\widetilde{a}_1, \ldots, \widetilde{a}_m)$
```

Figure 5: Simple load overestimate based on Laplacian noise and safety buffer.

```
$\widetilde{L}_h(X)$:
  compute true load $(a_1, \ldots, a_m) = \mathsf{Ld}_h(X)$
  for $i = 1$ to $n$:
    $\hat{a}_i \leftarrow a_i + \mathsf{Lap}_{1/\epsilon_1}$
    compute (e.g., via a lookup table) minimum $\tau$ such that
        $\Pr_{\beta \leftarrow \mathsf{Binom}_{n,1/m}} \left[ \beta > \tau \;\middle|\; \beta + \mathsf{Lap}_{1/\epsilon_1} = \hat{a}_i \right] < 1/m2^\lambda$
    $\widetilde{a}_i := \tau$
  return $(\widetilde{a}_1, \ldots, \widetilde{a}_m)$
```

Figure 6: Load overestimate based on Bayesian updating.

Laplacian load estimate $\hat{a}_i$	Resulting output $\widetilde{a}_i$
$\hat{a}_i \leq 7$	31
$7 < \hat{a}_i \leq 10$	32
$10 < \hat{a}_i \leq 13$	33
$13 < \hat{a}_i \leq 15$	34
$15 < \hat{a}_i \leq 17$	35
$17 < \hat{a}_i \leq 19$	36
$19 < \hat{a}_i \leq 22$	37

Figure 7: Example lookup table for the Bayesian-update mechanism. We use $n = 2^{20}$ items, $m = n/12$ bins, $\epsilon = 1$, and $\lambda = 40$. Because the expected bin size is 12, the most likely Laplacian estimate is also 12, so the most common overestimate is 33. In the fully-secure RR17 protocol with same parameters, all bins are taken to have size $\mu = 51$. When using the simple Laplacian method (with same ϵ and λ), the safety buffer alone is 38.4.

5.2 Differentially private inner product overestimate

The protocol requires an inner product overestimate function $\widetilde{I}$, to upper bound the number of masks that Alice must send to Bob. The number of non-dummy masks that Alice has is $I = \sum_i a_i \widetilde{b}_i$, but this cannot be released exactly since a_i is sensitive.

In this case, the simple mechanism involving Laplacian noise and a safety buffer is good enough. Taking the $\widetilde{b}_i$ values as constant, we view I as a function of Alice's input set X:

$$I(X) = \sum_i \mathsf{Ld}_h(X, i) \cdot \widetilde{b}_i.$$

It is clear that the sensitivity of I is $\Delta I = \max_i\{\widetilde{b}_i\} - \min_i\{\widetilde{b}_i\}$. Hence, adding Laplacian noise with parameter $\Delta I/\epsilon_2$ is sufficient for ϵ_2-differential privacy. As before, we compute z such that $\Pr[\mathsf{Lap}_{\Delta I/\epsilon_2} < -z] < 1/2^\lambda$. The final overestimate function is therefore:

$$\widetilde{I}(X) = \sum_i \mathsf{Ld}_h(X, i) \cdot \widetilde{b}_i + \mathsf{Lap}_{\Delta I/\epsilon_2} + z.$$

Theorem 13. *$\widetilde{I}$ above is ϵ_2-differentially private and is an overestimate of I with probability at least $1 - 1/2^\lambda$.*

The reason this simpler mechanism is sufficient here (rather than the Bayesian-update mechanism) is due to the scale of the numbers involved. With typical parameters, the load in a single bin is roughly the same magnitude as the safety buffer z. So adding z to the load of each bin has a huge effect. However, the inner product I is, e.g., 5 orders of magnitude larger than the load of a single bin. So adding z to this value has minimal effect.

Allocating the privacy budget. We have described the load overestimate from the previous section to have privacy parameter ϵ_1, and the inner-product overestimate to have privacy parameter ϵ_2. The overall PSI protocol will leak both of these values, leading to overall differential privacy parameter $\epsilon = \epsilon_1 + \epsilon_2$ by the composition theorem for DP.

If a privacy budget of ϵ is desired for the entire PSI protocol, this budget must be divided between these two mechanisms somehow. We found it best to allocate the vast majority of the privacy budget to the load overestimates, letting ϵ_2 be quite small in comparison to ϵ_1 (e.g., $\epsilon_1 = 0.99\epsilon$ and $\epsilon_2 = 0.01\epsilon$).

5.3 Distributional-differentially-private load overestimate

Say the receiver was to take the histogram of hash values and announce for each bin whether or not it was larger than the average bin size n/m. This would allow removing most of the dummies in roughly half of the bins, but it is a deterministic function of the histogram so can't possibly be differentially private (for any ϵ). However, actually exploiting such information would require the adversary to have significant knowledge about the set of items — for example, knowing with certainty n/m items that happen to be placed in the same bin, with another target item of interest also being mapped to that bin. Then the inclusion/exclusion of the target item will change whether this bin is above/below the threshold.

If one is working in a use case where this sort of side information is unlikely, then we can use more nuanced notions of privacy to give better efficiency. In particular, we consider here *distributional differential privacy (DDP)*, introduced by Bassily et al. [BGKS13].[6] Rather than considering the worst case over all possible databases, DDP treats the database itself as a random variable, modeling adversarial uncertainty.

Definition 14. *Say Δ is a set of distributions for (X, Z), where X is the database and Z is the adversary's auxiliary information about the database. A mechanism f has $(\epsilon, \delta, \Delta)$-**distributional differential privacy** if for all distributions $\mathcal{D} \in \Delta$, all i, all (x_i, z) in the support of (X, Z), and all possible sets of output Q,*

$$\begin{aligned}&\Pr[f(X) \in Q \mid X_i = x_i, Z = z] \\ &\qquad \leq e^{\epsilon} \Pr[f(X_{-i}) \in Q \mid X_i = x_i, Z = z] + \delta\end{aligned}$$

and

$$\begin{aligned}&\Pr[f(X_{-i}) \in Q \mid X_i = x_i, Z = z] \\ &\qquad \leq e^{\epsilon} \Pr[f(X) \in Q \mid X_i = x_i, Z = z] + \delta,\end{aligned}$$

where X_i is the i^{th} item of the database and X_{-i} is the database with that row removed.[7]

Inuitively, the definition guarantees to an individual that, given the adversary's present knowledge/uncertainty, the mechanism will output roughly the same distribution of values whether or not that individual's data is included in the database. As a result, the inclusion of this data does not reveal anything about this value beyond the leakage which is parameterized by ϵ. The definition is less stringent than differential privacy, more narrowly tailored to a subset of all possible auxiliary information. But this is not without ramifications. If the adversary knows more about the database than Δ allows, then there is no guarantee of privacy. (Fortunately, if the adversary knows *less* there is no problem.) Additionally, DDP does not have automatic composition in the way that differential privacy does, so it cannot safely be combined with other private data releases without a case-by-case analysis. The nuances of this privacy definition can be quite subtle, and we refer the reader to [BGKS13] for a more detailed discussion.

Deterministic rounding mechanism.. We consider a load-overestimate mechanism in which the bin size is rounded up to either $t - 1$ or the a-priori maximum bin size – whichever is closer. Hence, for each bin we reveal whether its load is in $\{0, \ldots, t-1\}$ or $\{t, \ldots\}$. The mechanism is described formally in Figure 8.

Theorem 15. *The load overestimate in Figure 8 is an overestimate of the true load with probability at least $1 - 2^{\lambda}$.*

Proof. By construction, $\widetilde{a}_i$ cannot be an underestimate in the case that $\widetilde{a}_i = t - 1$. Hence, $\widetilde{L}$ fails to be an overestimate only in the case that some a_i exceeds μ. By the definition of μ and a standard union bound, the probability of such an a_i existing is at most $1/2^{\lambda}$. □

Understanding the concrete privacy of the mechanism is not easy. As mentioned, the analysis depends on the choice of threshold t and the characterization of the adversary's auxiliary information (Δ). A simple formula relating these parameters to the privacy of the mechanism is not known, and any kind of meaningful formula/bound is left as important open work on this mechanism.

[6] DDP is a special case of *coupled-worlds privacy*. Coupled-worlds privacy has additional parameters and allows for more flexibility in choosing the most relevant privacy definition for a particular use case. Here we use DDP because it is the most straightforward instantiation, the one we feel is most well-motivated here, and the one that is most analogous to differential privacy.

[7] The original DDP definition uses a simulator $\mathsf{Sim}(X_{-i})$ in place of $F(X_{-i})$. In all cases here we will choose $\mathsf{Sim} = F$, so we present a simplified definition.

```
$\widetilde{L}_h(X)$:
  compute true load $(a_1, \dots, a_m) = \mathsf{Ld}_h(X)$
  compute $\mu$ such that $\Pr_{\beta \leftarrow \mathsf{Binom}_{n,1/m}}[\beta > \mu] < 1/m2^\lambda$
  for $i = 1$ to $n$:
    $\widetilde{a}_i :=$ if $a_i < t$ then $t - 1$ else $\mu$
  return $(\widetilde{a}_1, \dots, \widetilde{a}_m)$
```

Figure 8: Load overestimate based on deterministic rounding.

To give some sense of the privacy guarantee, we have analyzed the mechanism's guarantees for a specific set of parameters: $n = 2^{16}$ items, $m = n/10 = 6553$ bins, and threshold $t = 13$. The analysis considers the case where the adversary's auxiliary information of X leaves $h(X)$ uniform. More formally, Δ contains all joint distributions over (X, Z) such that the conditional distribution $h(X) \mid Z$ is uniform in $\{1, \dots, m\}^n$.

Theorem 16. *With Δ as above, and $n = 2^{16}$, $m = n/10$, $t = 13$, the mechanism from Figure 8 gives $(\Delta, \epsilon = 0.33, \delta = 5 \times 10^{-5})$-DDP.*

The calculations are quite tedious and given in Appendix B. As we discuss there, increasing n (keeping $m = n/10$) can only decrease ϵ and δ (i.e., improve privacy), as the relevant binomial distributions converge more closely to a normal distribution.

Relaxing the assumption on adversarial knowledge. The analysis assumes that every $h(x)$ of the honest party is random given the adversary's auxiliary information. When the adversary does in fact have some information about some of the items in the set, the DDP privacy of the mechanism is not completely compromised. Information about items that are placed outside the bin in question (the bin into which the target item is placed) is similar in nature to decreasing n by that amount. Information about items that are placed in the same bin as the target item is similar to reducing t by that amount. That means privacy will degrade rapidly when the adversary knows (or has significant certainty about) several values that would all hash to the same bin under the random function h. The chances of this are low unless the adversary knows a large amount about the input set. However, while we are confident of these general statements, quantifying the privacy guarantees with a specific choice of adversarily knowledge would (given the techniques discussed here) require re-running the simulation in a slightly more complex way. Finding a clear and simple characterization of how privacy degrades as adversarily knowledge increases remains an open question.

6 Experiments

We now explore the concrete tradeoff between differentially private leakage and performance (computation and communication) in the Rindal-Rosulek PSI protocol. We will see that our protocol offers a smooth trade-off, with performance improving by as much as 60% for some leakage settings.

The baseline for comparison was the Rindal-Rosulek protocol as-is. We obtained the implementation from the authors and modified it to support our variations on reported bin sizes and number of encodings in the final message, as outlined in Section 4. One can think of the unmodified Rindal-Rosulek protocol as the special case of our generalization, with $\epsilon = 0$ (i.e., no leakage at all). The $\mathcal{F}_{\mathsf{encode}}$ functionality is instantiated with the construction of [OOS17].

With ran all protocols on a single server similar to that used by [RR17] with the difference that we only consider single-threaded performance. We stress that the only difference in experiments is

n	Protocol	ϵ	LAN Setting				WAN Setting			
			n/m	Total Time (ms)	Online Time (ms)	Comm. (MB)	n/m	Total Time (ms)	Online Time (ms)	Comm. (MB)
2^{16}	This (Bayesian)	4	12	525	464	23.0	12	4341	3806	23.60
		2	12	589	541	30.4	12	5472	4951	30.4
		1	12	714	670	39.6	12	6829	6310	39.6
		0.5	12	850	812	47.4	12	8054	7549	47.4
		0.1	12	1137	1093	56.3	12	10848	10387	56.3
		0.05	12	1246	1198	58.6	12	11395	10868	58.6
		0.01	12	1375	1308	59.9	12	11742	10994	59.9
		0.005	12	1419	1350	60.6	12	11839	11021	60.6
	This (simple)	4	12	744	671	27.2	12	7036	6407	27.2
		2	12	977	906	38.3	12	9433	8806	38.3
		1	12	1433	1361	59.2	12	13879	13248	59.2
		0.5	12	1438	1368	61.7	12	14407	13779	61.7
		0.05	12	1444	1373	61.7	12	14419	13789	61.7
	RR17	(0)	4	1318	1005	69.1	10	10651	9932	54.1
2^{20}	This (Bayesian)	4	12	6780	6719	398.3	12	60579	60033	398.3
		2	12	8662	8614	530.4	12	80486	79942	530.4
		1	12	10873	10777	689.7	12	103517	102982	689.7
		0.5	12	12393	12167	838.9	12	124531	123986	838.9
		0.1	12	16288	16290	998.5	12	172089	174612	998.5
		0.05	12	17547	17485	1021.3	12	184206	183672	1021.3
		0.01	12	20774	20701	1033.7	12	186201	182962	1033.7
		0.005	12	21602	21533	1075.4	12	193092	190121	1075.4
	This (simple)	4	12	7963	7897	468.4	12	71137	70590	468.4
		2	12	10632	10570	672.1	12	96454	95918	672.1
		1	12	16472	16412	1043.8	12	155900	155370	1043.8
		0.5	12	16849	16795	1075.0	12	160304	159764	1075.0
		0.05	12	21121	21031	1075.4	12	219289	218651	1075.4
	RR17	(0)	4	18478	16295	1302.1	10	191709	189792	1071.7

Figure 9: The effect of different differential privacy parameters on our protocol performance, compared to the baseline (fully-secure) RR17 protocol which has no leakage. (Simple) refers to Simple Laplacian mechanism (Figure 5) while (Bayesian) refers to our new overestimate mechanism (Figure 6).

the bin size and number of final-round encodings, resulting from our differentially private leakage. The benchmark machine has 256GB of RAM and 36 physical cores at 2.6 GHz where a single core is assigned to each party. Two network settings were considered, 1) the LAN setting where latency is less than a millisecond and throughput is 10Gbps of which about 1% was actually utilized. 2) the WAN setting with a 80ms round trip latency and 40Mbps throughput. Both parties run on the same physical machine, communicating via a simulated LAN achieved using a Linux kernel loop-back device and the parameters listed above. Performance numbers are reported using the wall-clock time and include network latency. For each setting of parameters, we report the average of 5 runs.

Parameters. In Figure 9 we report the performance of our protocol for various choices of differential privacy parameter ϵ. Recall that there are two pieces of differentially-private leakage: the overestimate of the bin loads, and the overestimate of the number of encodings (inner product). Given an overall privacy budget of ϵ we found it advantageous to allocate the lion's share to the histogram mechanism. The row in the table corresponding to privacy budget ϵ corresponds to 0.99ϵ-DP load overestimate and 0.01ϵ-DP inner-product overestimate for the sender. The receiver uses the entire privacy budget for the bin-load overestimate. We consider both the simple Laplacian

				LAN Setting			WAN Setting			
n	Protocol	Sender ϵ	Receiver ϵ	n/m	Total Time (ms)	Online Time (ms)	n/m	Total Time (ms)	Online Time (ms)	Comm. (MB)
2^{16}	This	DDP 0.33	DP 4	10	531	451	10	4901	4368	25.52
		DDP 0.33	DP 2	10	496	452	10	5726	5207	30.31
		DDP 0.33	DP 1	10	611	568	10	6636	6112	36.48
		DDP 0.33	DP 0.33	10	697	651	10	7334	6816	41.64
	RR17	(0)	(0)	4	1318	1005	10	10651	9932	54.04
2^{20}	This	DDP 0.33	DP 4	10	6793	6727	10	66139	65593	436.92
		DDP 0.33	DP 2	10	7224	7155	10	79135	78598	527.24
		DDP 0.33	DP 1	10	8833	8770	10	95958	95424	634.10
		DDP 0.33	DP 0.33	10	10108	10035	10	110983	110449	725.52
	RR17	(0)	(0)	4	18478	16295	10	191709	189792	1071.65

Figure 10: The effect of different *distributional* differential privacy parameters on our protocol performance, compared to the baseline RR17 protocol which has no leakage. All of our DDP protocol executions use the mechanism with threshold $t = 13$.

overestimate mechanism and our improved Bayesian-update mechanism.

The main parameters that effects performance, apart from ϵ, is the expected bin load n/m. This parameter controls the relative number of dummy items that occupy each bin. More items per bin leads to fewer dummy items proportionally, but more communication because of the quadratic number of encodings sent in the last protocol message. These trade-offs must be balanced. In [RR17], the preferred value of n/m was found to be 4. In our setting, we have fewer dummy items thanks to differential privacy, and we found n/m between 10 and 16 minimizes latency. The ideal parameter n/m also depends on the network latency (i.e., WAN vs LAN setting). In both our experiments and RR17, the ideal value of n/m was determined empirically. For vastly different network settings, a different n/m value may be preferable.

Smooth privacy-performance tradeoff. Focusing on the case of $n = 2^{20}$ items, in the LAN setting $\epsilon = 4$ results in a speed reduction of 63%; $\epsilon = 2$ gives 53% reduction; $\epsilon = 1$ gives 41% reduction; $\epsilon = 0.5$ gives 33% reduction. In the WAN setting, where communication is more of a bottleneck, $\epsilon = 4$ reduces communication by 62%, $\epsilon = 2$ by 50%, $\epsilon = 1$ by 35%, and $\epsilon = 0.5$ by 22%.

We attempted to determine a value of ϵ beyond which our protocol gives only marginal improvement over the fully-secure protocol. Experimentally, we found that the improvement drops off for $\epsilon \leq 0.05$, where the performance is within 5% of the fully secure protocol. The difference in cost between the DP and fully-secure protocols depends strongly ϵ on n/m (expected load) and only minimally on n. The preferred choice of n/m parameter is quite stable for all reasonable values of n that could be considered for PSI. We therefore believe that the threshold of $\epsilon = 0.05$ is representative of our limit for all n.

Improvement over simple Laplacian mechanism. For the case of $n = 2^{20}$ we also implemented the simple Laplacian load overestimate that involves a large safety buffer. For higher $\epsilon \in \{2, 4\}$, our Bayesian method improves over the basic approach by 14-18%. But for more demanding $\epsilon \in \{0.5, 1\}$, our approach gives an improvement of roughly 25%.

Distributional Differential Privacy. We also explored allowing DDP leakage, as explained in Section 5.3. In this setting, the sender can set a threshold t, and for each bin announce whether it contains t or more items. Then each bin is treated as if it has either $t - 1$ items or the worst-case number of items (i.e., whatever the fully-secure protocol would do).

Because the analysis of this mechanism depends heavily on the adversary's auxiliary information about the inputs, we only propose using it for the sender. The receiver eventually learns the intersection, which becomes side information that cannot be predicted in advance (i.e., when the privacy parameters are being determined). In these experimends, the sender uses the deterministic-rounding mechanism and the receiver uses our Bayesian-updating mechanism.

Due to the difficulty in obtaining concrete quantitative bounds on the privacy level, we consider only one privacy setting for the DDP case. Specifically, for $n \in \{2^{16}, 2^{20}\}$ items, and $m = n/10$ bins, we let the sender announce whether each bin has less than $t = 13$ items. Following the analysis in Section 5.3, we conservatively estimate that this mechanism provides $(\epsilon = 0.33, \delta = 0.00005)$-DDP.

The performance of the DDP mechanism is reported in Figure 10. We observe that this result in decreased communication/running time in all cases except when compared to our standard DP protocol with $\epsilon = 4$. For instance, with $n = 2^{20}$ and a $\epsilon = 0.33$ our DDP protocol is 18% faster than the 0.5-DP protocol. The decrease in communication is of particular note for the WAN setting where communication is the main bottleneck. However, even in the LAN setting we observe a considerable decrease in running time.

7 Future Work

Looking more closely at our protocol, the differentially private leakage is on a **random histogram** — i.e., a random function applied to the private input set. Indeed, the nature of this leakage motivates the choice of distributional DP — rather than arguing that an attacker has limited information about some of the private items, it is enough to say that the attacker's auxiliary information about the items is sufficiently limited to make most of the hashes look random.

An interesting future direction is to extend differential privacy definitions and mechanisms to give better accuracy in such a setting — i.e., where we are trying to sanitize information that is already somewhat "random information" about the private inputs.

The most challenging part of our analysis is obtaining good concrete bounds for the DDP privacy analysis. We obtain estimates using intensive computations, but analytical bounds would be more convenient.

There is also an issue that the DDP mechanism can only be applied to one party's histogram. This stems from the fact that DDP does not automatically compose with auxiliary information — yet, the PSI output itself will become auxiliary information but is not known at the time that DDP parameters are chosen. A better understanding of this composition problem would be helpful, since our DDP mechanism gives better performance.

References

[BGKS13] Raef Bassily, Adam Groce, Jonathan Katz, and Adam Smith. Coupled-worlds privacy: Exploiting adversarial uncertainty in statistical data privacy. In *54th FOCS*, pages 439–448. IEEE Computer Society Press, October 2013.

[BNO08] Amos Beimel, Kobbi Nissim, and Eran Omri. Distributed private data analysis: Simultaneously solving how and what. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 451–468. Springer, Heidelberg, August 2008.

[BSMD10] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. Sepia: Privacy-preserving aggregation of multi-domain network events and statistics. *Network*, 1(101101), 2010.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CCMS17] T-H. Hubert Chan, Kai-Min Chung, Bruce Maggs, and Elaine Shi. Foundations of differentially oblivious algorithms. Cryptology ePrint Archive, Report 2017/1033, 2017. http://eprint.iacr.org/2017/1033.

[CGBL$^+$17] Melissa Chase, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, and Peter Rindal. Private collaborative neural network learning. Cryptology ePrint Archive, Report 2017/762, 2017. https://eprint.iacr.org/2017/762.

[CJJ$^+$13] David Cash, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for Boolean queries. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 353–373. Springer, Heidelberg, August 2013.

[DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 265–284. Springer, Heidelberg, March 2006.

[FMJS17] Ellis Fenske, Akshaya Mani, Aaron Johnson, and Micah Sherr. Distributed measurement with private set-union cardinality. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 2295–2312. ACM Press, October / November 2017.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19. Springer, Heidelberg, May 2004.

[Gol87] Oded Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In Alfred Aho, editor, *19th ACM STOC*, pages 182–194. ACM Press, May 1987.

[HKE12] Yan Huang, Jonathan Katz, and David Evans. Quid-Pro-Quo-tocols: Strengthening semi-honest protocols with dual execution. In *2012 IEEE Symposium on Security and Privacy*, pages 272–284. IEEE Computer Society Press, May 2012.

[HMFS17] Xi He, Ashwin Machanavajjhala, Cheryl J. Flynn, and Divesh Srivastava. Composing differential privacy and secure computation: A case study on scaling private record linkage. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 1389–1406. ACM Press, October / November 2017.

[KKNO17] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Accessing data while preserving privacy. *CoRR*, abs/1706.01552, 2017.

[KMRR15] Vladimir Kolesnikov, Payman Mohassel, Ben Riva, and Mike Rosulek. Richer efficiency/security trade-offs in 2PC. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 229–259. Springer, Heidelberg, March 2015.

[KOV14] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. Differentially private multi-party computation: Optimality of non-interactive randomized response. *arXiv preprint arXiv:1407.1546*, 2014.

[KS08] Shiva Prasad Kasiviswanathan and Adam Smith. A note on differential privacy: Defining resistance to arbitrary side information. Cryptology ePrint Archive, Report 2008/144, 2008. http://eprint.iacr.org/2008/144.

[MF06] Payman Mohassel and Matthew Franklin. Efficiency tradeoffs for malicious two-party computation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 458–473. Springer, Heidelberg, April 2006.

[MG17] Sahar Mazloom and S. Dov Gordon. Differentially private access patterns in secure computation. Cryptology ePrint Archive, Report 2017/1016, 2017. http://eprint.iacr.org/2017/1016.

[NH12] Arjun Narayan and Andreas Haeberlen. Djoin: Differentially private join queries over distributed databases. In Chandu Thekkath and Amin Vahdat, editors, *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, pages 149–162. USENIX Association, 2012.

[OOS17] Michele Orrù, Emmanuela Orsini, and Peter Scholl. Actively secure 1-out-of-N OT extension with application to private set intersection. In Helena Handschuh, editor, *CT-RSA 2017*, volume 10159 of *LNCS*, pages 381–396. Springer, Heidelberg, February 2017.

[Ost90] Rafail Ostrovsky. Efficient computation on oblivious RAMs. In *22nd ACM STOC*, pages 514–523. ACM Press, May 1990.

[PKV$^+$14] Vasilis Pappas, Fernando Krell, Binh Vo, Vladimir Kolesnikov, Tal Malkin, Seung Geol Choi, Wesley George, Angelos D. Keromytis, and Steve Bellovin. Blind seer: A scalable private DBMS. In *2014 IEEE Symposium on Security and Privacy*, pages 359–374. IEEE Computer Society Press, May 2014.

[PNH17] Antonis Papadimitriou, Arjun Narayan, and Andreas Haeberlen. Dstress: Efficient differentially private computations on distributed data. In *Proceedings of the Twelfth European Conference on Computer Systems*, pages 560–574. ACM, 2017.

[RN10] Vibhor Rastogi and Suman Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 735–746. ACM, 2010.

[RR17] Peter Rindal and Mike Rosulek. Malicious-secure private set intersection via dual execution. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 1229–1242. ACM Press, October / November 2017.

[SGB18] Phillipp Schoppmann, Adrià Gascón, and Borja Balle. Private nearest neighbors classification in federated databases. Cryptology ePrint Archive, Report 2018/289, 2018. https://eprint.iacr.org/2018/289.

[WCM16] Sameer Wagh, Paul Cuff, and Prateek Mittal. Root oram: A tunable differentially private oblivious ram. *arXiv preprint arXiv:1601.03378*, 2016.

A Security Proof

In this section we prove the security of the (leaky) PSI protocol (Figure 4). The proof is essentially the same as the one in [RR17], and is presented here with many aspects identical to [RR17]. The modifications have to do with the leakage handling in the simulator, which is highlighted below.

Proof. We start with the case of a **corrupt Bob**. The simulator must extract Bob's input, and simulate the messages in the protocol. We first describe the simulator:

> The simulator plays the role of the ideal $\mathcal{F}_{\mathsf{encode}}$ functionality. The simulator obtains leakage $(\widetilde{a}_1, \dots, \widetilde{a}_m) \leftarrow L_{\mathsf{pre}}(X) = \widetilde{L}_h(X)$ from the functionality and simulates this as the message from Alice in Step 2a. It receives $(\widetilde{c}_1, \dots, \widetilde{c}_m)$ from corrupt Bob in Step 2b. To extract Bob's set, the simulator observes all of Bob's $\mathcal{F}_{\mathsf{encode}}$ messages $(\textsc{Encode}, (\mathsf{sid}, \mathsf{A}, d, p), y_{d,p})$ in Step 3b. The simulator computes $Y = \{y_{d,p}\}$ and sends it to the ideal $\mathcal{F}_{\mathsf{PSI}}$ functionality which responds with the intersection $Z = X \cap Y$.
>
> For each $z \in Z$, compute bin index $d = h(z)$ and place z into a random unused position $p \in [\widetilde{a}_d]$ in bin $\mathcal{B}_X[z]$. For $j \in [\widetilde{c}_d]$, add value $[\![z]\!]^{\mathsf{A}}_{d,p} \oplus [\![z]\!]^{\mathsf{B}}_{d,j}$ to a set E. Then obtain leakage $\widetilde{e} \leftarrow L_{\mathsf{post}}(X) = \widetilde{I}(\mathsf{Ld}_h(X), (\widetilde{c}_1, \dots, \widetilde{c}_m))$ from the functionality and pad E with random values until it has size $\widetilde{e}$. The simulator then sends E (randomly permuted) to the adversary.

To show that this is a valid simulation, we consider a series of hybrids.

Hybrid 0 The first hybrid is the real interaction where Alice honestly uses her input X, and $\mathcal{F}_{\mathsf{encode}}$ is implemented honestly.

Observe Bob's commands to $\mathcal{F}_{\mathsf{encode}}$ of the form $(\textsc{Encode}, (\mathsf{sid}, \mathsf{A}, d, p), y_{d,p})$ in Step 3b. Based on these, we can define (but not use) the set $\tilde{Y} = \{y_{d,p}\}$.

Hybrid 1 In this hybrid, we modify Alice to send dummy values to $\mathcal{F}_{\mathsf{encode}}$ in Step 2a (rather than her actual inputs). The hybrid is indistinguishable by the properties of $\mathcal{F}_{\mathsf{encode}}$.

Hybrid 2 In Step 4a, for each $x \in X$ the simulated Alice sends common encodings of the form $[\![x]\!]^{\mathsf{A}}_{d,j} \oplus [\![x]\!]^{\mathsf{B}}_{d,p}$, for some position p, where $d = h(x)$. Suppose $x \notin \tilde{Y}$. By construction of $\tilde{Y}$, Bob never obtained an encoding of the form $[\![x]\!]^{\mathsf{A}}_{d,j}$. This encoding is therefore distributed independent of everything else in the simulation. In particular, the *common* encoding of the form $[\![x]\!]^{\mathsf{A}}_{d,j} \oplus [\![x]\!]^{\mathsf{B}}_{d,p}$, which is added to the set E, is uniform.

We therefore modify the hybrid in the following way. In Step 4a, simulated Alice generates encodings for the set E only for items in the intersection $Z = X \cap \tilde{Y}$, instead of X as before. She still pads the set E to contain $\widetilde{e}$ items, as before. By the above argument, the adversary's view is identically distributed in this modified hybrid.

We can see that the final hybrid uses the contents X only in the following way: It uses the bin-load overestimate $\widetilde{L}_h(X)$; it uses the result of $X \cap \tilde{Y}$ for some set $\tilde{Y}$ that it computes; it uses the inner-product overestimate $\widetilde{I}(\mathsf{Ld}_h(X), (\widetilde{c}_1, \dots, \widetilde{c}_m))$. Hence, this hybrid corresponds to our final simulator, where we obtain leakage from the functionality, send $\tilde{Y}$ to the ideal $\mathcal{F}_{\mathsf{PSI}}$ functionality and receive $X \cap \tilde{Y}$ in response.

We now turn our attention to a **corrupt Alice**. In this case the simulator must simply extract Alice's effective input (Alice receives no output from $\mathcal{F}_{\mathsf{PSI}}$). The simulator is defined as follows:

The simulator plays the role of the ideal $\mathcal{F}_{\mathsf{encode}}$ functionality. The simulator obtains $(\widetilde{a}_1, \ldots, \widetilde{a}_m)$ from the adversary in Step 2a and obtains leakage $(\widetilde{b}_1, \ldots, \widetilde{b}_m) \leftarrow L_{\mathsf{pre}}(Y) = \widetilde{L}_h(Y)$ from the functionality. It computes $(\widetilde{c}_1, \ldots, \widetilde{c}_m)$ as in the protocol and simulates this as the message from Bob in Step 2b. In Step 3a, the simulator intercepts Alice's commands of the form $(\textsc{Encode}, (\mathsf{sid}, \mathsf{B}, d, p), x_{d,p})$. The simulator computes a set of candidates $\tilde{X} = \{x_{d,p}\}$ and for $x \in \tilde{X}$ let $\mathsf{c}(x)$ denote the multiplicity of x in its bin; i.e., the number of values p for which $x = x_{h(x),p}$.

The simulator computes a hash table $\mathcal{B}$ as follows. For $x \in \tilde{X}$ the simulator places $\mathsf{c}(x)$ copies of x in bin $\mathcal{B}[h(x)]$. Note that we place $\sum_{x:h(x)=d} \mathsf{c}(x)$ items in bin index d. By construction $\sum_{x:h(x)=d} \mathsf{c}(x) \leq \widetilde{a}_d \leq \widetilde{c}_d$, and hence simulated Bob has enough space in bin d. Let the items in each bin be randomly permuted, and for each x, let $\mathsf{p}(x)$ denote the set of positions of x in its bin.

Let E denote the set of values sent by Alice in Step 4a. The simulator computes

$$X^* = \big\{x \in \tilde{X} \mid \exists j \in [\widetilde{c}_{h(x)}], p \in \mathsf{p}(x) : \\ [\![x]\!]^{\mathsf{A}}_{h(x),j} \oplus [\![x]\!]^{\mathsf{B}}_{h(x),p} \in E\big\} \tag{3}$$

where the encodings are obtained by playing the role of $\mathcal{F}_{\mathsf{encode}}$. The simulator sends X^* to the $\mathcal{F}_{\mathsf{PSI}}$ functionality.

Hybrid 0 The first hybrid is the real interaction where Bob honestly uses his input X, and $\mathcal{F}_{\mathsf{encode}}$ is implemented honestly.

Observe Alice's commands to $\mathcal{F}_{\mathsf{encode}}$ of the form $(\textsc{Encode}, (\mathsf{sid}, \mathsf{B}, d, p), x_{d,p})$ in Step 3a. Based on these, define $\tilde{X} = \{x_{d,p}\}$.

Hybrid 1 In this hybrid, we modify Bob to send dummy inputs to $\mathcal{F}_{\mathsf{encode}}$ in Step 2b (rather than his actual items). The hybrid is indistinguishable by the properties of $\mathcal{F}_{\mathsf{encode}}$.

Hybrid 2 Note that in this hybrid, Bob's output is computed as specified in Step 4c. We then modify the interaction so that Bob removes all output items which are not in $\tilde{X}$. The hybrids differ only in the event that simulated Bob computes an output in Step 4c that includes an item $y \notin \tilde{X}$. This happens only if $[\![y]\!]^{\mathsf{A}}_{d,j} \oplus [\![y]\!]^{\mathsf{B}}_{d,p} \in E$, where Bob places y in position p of bin $d = h(y)$. Since $y \notin \tilde{X}$, however, the encoding $[\![y]\!]^{\mathsf{B}}_{d,p}$ is distributed uniformly. The length of encodings is chosen so that the overall probability of this event (across all choices of $y \notin \tilde{X}$) is at most $2^{-\lambda}$. Hence the modification is indistinguishable.

Hybrid 3 We modify the hybrid in the following way. When building the hash table, the simulated Bob uses $\tilde{X}$ instead of his actual input Y. Each $x \in \tilde{X}$ is inserted with multiplicity $\mathsf{c}(x)$. Then he computes the protocol output as specified in Step 4c; call it X^*. This is not what the simulator gives as output — rather, it gives $X^* \cap Y$ as output instead.

The hashing process is different only in the fact that items of $Y \setminus \tilde{X}$ are excluded and replaced in the hash table with items of $\tilde{X} \setminus Y$ (*i.e.*, items in $Y \cap \tilde{X}$ are treated exactly the same way). Note that the definition of $\tilde{X}$ ensures that the hash table can hold all of these items without overflowing. Also, this change is local to Step 4c, where the only thing that happens is Bob computing his output. However, by the restriction added in *Hybrid 2* , items in $Y \setminus \tilde{X}$ can never be included in X^*. Similarly, by the step added in this hybrid, items in $\tilde{X} \setminus Y$ can never be included in the simulator's output. So this change has no effect on the adversary's view (which includes this final output).

The final hybrid works as follows. A simulator interacts with the adversary and at some point computes a set X^*, without the use of Y. Then the simulated Bob's output is computed as $X^* \cap Y$. Hence, this hybrid corresponds to our final simulator, where we send X^* to the ideal $\mathcal{F}_{\mathsf{PSI}}$ functionality, which sends output $X^* \cap Y$ to ideal Bob. $\square$

B Calculations for DDP Leakage

We assume (for now) that the sender knows nothing at all about the hashes of the receiver's inputs.[8] Note that this does not mean that the receiver's inputs are completely unknown, just that they have sufficient uncertainty that their hashes under the random hashing function appear uniform. Note also that this is an assumption we can make regarding *only the sender's knowledge* of the receiver's input, because the PSI protocol itself gives output to the receiver, which is auxiliary information about the sender's input that violates this assumption.

We now wish to compute the privacy parameters ϵ and δ for the deterministic rounding mechanism with threshold t. We first want to fix a given output S and consider the following ratio:

$$\rho = \frac{\Pr[q(X) = S \mid h(X_i) = j]}{\Pr[q(X_{-i}) = S \mid h(X_i) = j]} \tag{4}$$

We note first of all that the calculation will be the same for all choices of i and j, so we assume without loss of generality that $i = n$ and $j = m$. We also leave the conditioning implicit, since from this point forward all probabilities are conditioned on $h(X_n) = m$. We now want to examine the same ratio, with the following simplified notation:

$$\rho = \frac{\Pr[q(X) = S]}{\Pr[q(X_{-n}) = S]} \tag{5}$$

We first consider the case where $m \in S$, i.e. the case where $\rho \geq 1$. Let $S_{-m} = S \setminus \{m\}$ and let C_m be a random variable equal to the size of bin m. We can then rewrite the numerator.

$$\begin{aligned} \Pr[q(X) = S] =& \Pr[q(X_{-n}) = S] \\ &+ \Pr[q(X_{-n}) = S_{-m} \wedge C_m = t-1] \\ = \Pr[q(X_{-n}) = S]& \\ + \Pr[q(X_{-n}) = S_{-m}& \mid C_m = t-1] \Pr[C_m = t-1] \end{aligned} \tag{6}$$

We now define a random variable $H_{n,m}$ equal to the set of indices of bins of size t or greater when n balls are thrown into m bins. For example, $q(X_{-n}) = H_{n-1,m}$ and once we condition on $C_m = t-1$, we have $q(X_{-n}) = H_{n-t,m-1}$. (To see this, note that it's clear when we condition on a particular set of $t-1$ values ending up in bin m, and the overall probability is the average of the value when conditioned on each specific set.) We also note that $\Pr[C_m = t-1] = \mathsf{Binom}_{n-1,1/m}(t-1)$. As a result, we have

$$\begin{aligned} \Pr[q(X) = S] =& \Pr[H_{n-1,m} = S] \\ &+ \Pr[H_{n-t,m-1} = S_{-m}] \cdot \mathsf{Binom}_{n-1,1/m}(t-1). \end{aligned}$$

[8] Formally, Δ consists of all distributions on (X, Z) such that conditioned on any value z in the support of Z, the distribution on X is such that with high probability of the choice of h, each $h(X_i)$ is a uniform, independent value.

Note that the denominator in Equation 5 can also be written this way, with $\Pr[q(X_{-n}) = S] = \Pr[H_{n-1,m} = S]$. Putting these two simplifications together, we get

$$\rho = 1 + \mathsf{Binom}_{n-1,1/m}(t-1)\frac{\Pr[H_{n-t,m-1} = S_{-m}]}{\Pr[H_{n-1,m} = S]}. \tag{7}$$

We then note that all sets S of a given size are equally likely values of H (for any choice of subscripts), so setting $s = |S|$ we can rewrite the fraction in the above equation as

$$\frac{\Pr[H_{n-t,m-1} = S_{-m}]}{\Pr[H_{n-1,m} = S]} = \frac{\Pr[|H_{n-t,m-1}| = s-1]/\binom{m-1}{s-1}}{\Pr[|H_{n-1,m}| = s]/\binom{m-1}{s}}.$$

The binomial expressions then simplify:

$$\begin{aligned}\frac{\binom{m-1}{s}}{\binom{m-1}{s-1}} &= \frac{(m-1)!/[s!(m-s-1)!]}{(m-1)!/[(s-1)!(m-s)!]} \\ &= \frac{(s-1)!(m-s)!}{s!(m-s-1)!} = (m-s)/s\end{aligned}$$

That leaves us with the following expression for the ratio ρ:

$$\rho = 1 + \mathsf{Binom}_{n-1,1/m}(t-1) \cdot \frac{m-s}{s} \cdot \frac{\Pr[|H_{n-t,m-1}| = s-1]}{\Pr[|H_{n-1,m}| = s]}$$

We must then also consider the case where $m \notin S$ and $\rho < 1$. The analysis of the denominator is unchanged, but the numerator must be handled separately in this case. Luckily it proceeds quite similarly. We have

$$\begin{aligned}\Pr[q(X) = S] = &\Pr[q(X_{-n}) = S] \\ &- \Pr[q(X_{-n}) = S \wedge C_m = t-1].\end{aligned}$$

This is almost the same as in Equation 6, except that the addition has changed to subtraction and in the second term we have S instead of S_{-m}. Doing the same manipulation gives us

$$\rho = 1 - \mathsf{Binom}_{n-1,1/m}(t-1)\frac{\Pr[H_{n-t,m-1} = S]}{\Pr[H_{n-1,m} = S]}. \tag{8}$$

As before we use the symmetry of $\Pr[H_{n-t,m-1} = S]$ across all sets S of the same size. Again using $|S| = s$ we have

$$\frac{\Pr[H_{n-t,m-1} = S]}{\Pr[H_{n-1,m} = S]} = \frac{\Pr[|H_{n-t,m-1}| = s]/\binom{m-1}{s}}{\Pr[|H_{n-1}| = s]/\binom{m-1}{s}}.$$

Unlike in the previous case, these binomial coefficients cancel and we are left with

$$\rho = 1 - \mathsf{Binom}_{n-1,1/m}(t-1) \cdot \frac{\Pr[|H_{n-t,m-1}| = s]}{\Pr[|H_{n-1,m}| = s]}.$$

Estimating concrete parameters. If we knew how to compute an exact probability distribution for $|H_{n,m}|$ we would now be finished. The distribution of $|H_{n,m}|$ approximates a normal distribution. The value of ρ is worst when s is far from its expected value, so we could simply define an interval $[\alpha, \beta]$ and compute the values of ρ with $s = \alpha$ and $s = \beta$ and set $\epsilon = \max|\ln(\rho)|$. That would bound the ratio for all values in $[\alpha, \beta]$ and we could then compute the probability that $s \notin [\alpha, \beta]$ and set δ to that value. By adjusting α and β we could control a tradeoff between ϵ and δ.

Unfortunately we cannot find an efficiently computable closed-form expression for the distribution of $|H_{n,m}|$, and the normal approximation, while pretty good, is unreliable for very precise estimates of the extreme tails needed for δ calculations. So we instead resort to experimental estimation of these values. We run a simulation of throwing $n - t$ (resp. $n - 1$) balls into $m - 1$ (resp. m) bins many times, each time noting the number of bins ending up with at least t balls.

Concretely, computational limitations meant we could only run a large simulation for a single choice of parameters, and not for the largest values of n. We choose $n = 2^{16}$, $m = n/10 = 6553$, and $t = 13$ (meaning that each bucket is said to be either in $[0, 12]$ or $[12, \infty)$). We were able to run a simulation with 20 million data points. On average 1366 bins will be over the threshold, and our data gave us confidence to say that by choosing $[\alpha, \beta] = [1257, 1473]$ we achieve delta of approximately $1/500,000$. In principle ϵ should be worst near the extreme tails, but we found that as we moved to the extreme tails we hit a point where our data was too sparse to make accurate estimates before we hit a point where ϵ was increasing. In the entire range where we have sufficient data, variation in ϵ is minimal and the maximum value it takes is 0.33. This is a lower value that we were using in the differentially private protocol, so we should ideally accept a worse ϵ value in exchange for an even better δ value, but our data is insufficient to quantify the tradeoff beyond this point.

We note also that while simulations with larger n will take even more computational power, it is clear that the privacy parameters will only improve, as the distribution of $|H_{n,m}|$ is still approximately normal but now with a larger standard deviation and therefore a better ratio between adjacent values. We therefore consider the protocol with $m = n/10$ and $t = 13$ to be $(0.33, 5 \times 10^{-5})$-DDP in the $n = 2^{20}$ case as well. In reality, the parameters are probably significantly better than this.

Because we saw no sign of increasing ϵ in the range of values we could estimate, we believe these estimates are in fact much worse than what is really achieved. A successful theoretical analysis of these parameters, rather than relying on empirical estimation, would be a highly desirable result of future work.