Malicious-Secure Private Set Intersection via Dual Execution[*]

Peter Rindal[†] Mike Rosulek[†]

August 9, 2017

Abstract

Private set intersection (PSI) allows two parties, who each hold a set of items, to compute the intersection of those sets without revealing anything about other items. Recent advances in PSI have significantly improved its performance for the case of semi-honest security, making semi-honest PSI a practical alternative to insecure methods for computing intersections. However, the semi-honest security model is not always a good fit for real-world problems.

In this work we introduce a new PSI protocol that is secure in the presence of malicious adversaries. Our protocol is based entirely on fast symmetric-key primitives and inherits important techniques from state-of-the-art protocols in the semi-honest setting. Our novel technique to strengthen the protocol for malicious adversaries is inspired by the dual execution technique of Mohassel & Franklin (PKC 2006). Our protocol is optimized for the random-oracle model, but can also be realized (with a performance penalty) in the standard model.

We demonstrate our protocol's practicality with a prototype implementation. To securely compute the intersection of two sets of size 2^{20} requires only 13 seconds with our protocol, which is $\sim 12\times$ faster than the previous best malicious-secure protocol (Rindal & Rosulek, Eurocrypt 2017), and only $3\times$ slower than the best semi-honest protocol (Kolesnikov et al., CCS 2016).

1 Introduction

Private set intersection (PSI) allows two parties with respective sets X and Y to compute the intersection $X \cap Y$, without revealing the remaining elements in X or Y. PSI and closely related protocols have numerous applications including auctions [NPS99], remote diagnostics [BPSW07], DNA searching [TKC07], social network check-ins, private contact discovery [Mar14], botnet detection, advertising [PSSZ15], and many others.

PSI is a special case of secure two-party computation. One may consider two adversarial models for secure computation: the *semi-honest* model, where the protocol is protected against adversaries who follow the protocol but try to learn as much as possible from the messages they have seen; and the *malicious* model, where the protocol is protected against adversaries who may arbitrarily deviate from the protocol. Over the last several years there has been significant progress made in efficient PSI protocols for the semi-honest model. In this work, we focus on the more demanding malicious security model.

1.1 Paradigms for PSI

To put our results into context, we review different approaches for PSI, with a special emphasis on those approaches which achieve malicious security.

[*]Full version of a paper published in ACM CCS 2017.

[†]Oregon State University, {rindalp,rosulekm}@oregonstate.edu. Supported by NSF awards 1149647, 1617197, and a Google Research Award. The first author is also supported by an ARCS foundation fellowship.

Based on Oblivious Transfer. Pinkas, Schneider & Zohner (PSZ) [PSZ14] proposed a technique for PSI that relies heavily on oblivious transfers (OT). Using modern *OT extension* protocols [IKNP03, KK13, ALSZ13, KOS15], it is possible to perform millions of OT instances per second, almost entirely from cheap symmetric-key cryptographic operations.

The PSZ approach for PSI has been improved in a series of works [PSSZ15, PSZ16, OOS16, KKRT16], with the protocol of Kolesnikov et al. [KKRT16] currently being the fastest PSI protocol against semi-honest adversaries. There have been modifications of the protocol [OOS16] that provide security against a restricted class of malicious adversaries (i.e., the protocol protects against a malicious Alice only), but so far there has been no success in leveraging this most promising PSI paradigm to provide security in the full malicious security model.

Based on Bloom Filters. Dong, Chen & Wen [DCW13] describe an approach for PSI based on representing the parties' sets as Bloom filters. This approach also relies heavily on OT extension, and is reasonably efficient (though not as fast as the PSZ paradigm above).

The authors of [DCW13] described a protocol that was claimed to have security against malicious adversaries, but that was later found to have bugs [Lam16, RR17]. In a prior work [RR17], we gave a protocol based on Bloom filter encodings that indeed has security against malicious adversaries. Previously this protocol was the fastest in the malicious security model. In [RR17] we also argued that the use of Bloom filters in this paradigm is likely to have an *inherent* dependence on the random-oracle model.

Based on Diffie-Hellman. One of the first protocols proposed for PSI is due to Meadows [Mea86] and fully described by Huberman, Franklin & Hogg [HFH99]. It uses a simple modification of the Diffie-Hellman key agreement protocol to achieve PSI in the presence of semi-honest adversaries. De Cristofaro, Kim & Tsudik [DKT10] showed how to augment the protocol to provide security against malicious adversaries.

The main benefit of this Diffie-Hellman paradigm is its extremely low communication complexity. Indeed, protocols in this paradigm have by far the smallest communication complexity. However, the Diffie-Hellman paradigm requires expensive public-key operations for each item in the parties' sets, making them much slower than the OT-based approaches that require only a constant number of public-key operations.

Other paradigms. Freedman, Nissim & Pinkas [FNP04] proposed an approach for PSI based on oblivious polynomial evaluation. This technique was later extended to the malicious setting [DMRY09, HN10, FHNP14]. The idea behind these protocols is to construct a polynomial Q whose roots are Alice's items. The coefficients of Q are homomorphically encrypted and sent to the Bob. For each of Bob's items y, he homomorphically evaluates $\hat{y} := r \cdot Q(y) + y$ for a random r. When Alice decrypts the result, she will see $\hat{y} = y$ for all y in the intersection. These protocols require expensive public-key operations for each item.

Huang, Evans & Katz [HEK12] discuss using general-purpose secure computation (garbled circuits) to perform PSI. Later improvements were suggested in [PSZ14, PSSZ15]. At the time of [HEK12], such general-purpose PSI protocols in the semi-honest setting were actually faster than other special-purpose ones. Since then, the results in OT-based PSI have made special-purpose PSI protocols significantly faster. However, we point out that using general-purpose 2PC makes it relatively straight-forward to achieve security against malicious adversaries, since there are many well-studied techniques for general-purpose malicious 2PC.

Kamara et al. [KMRS14] presented techniques for both semi-honest and malicious secure PSI in a *server-aided model.* In this model the two parties who hold data enlist the help of an untrusted third party who carries out the majority of the computation. Their protocols are extremely fast (roughly as fast as the plaintext computation) and scale to billions of input items. In our work, we focus on on the more traditional (and demanding) setting where the two parties do not enlist a third party.

1.2 Our Results

From the previous discussion, we see that the fastest PSI paradigm for semi-honest security (due to PSZ) has no fully malicious-secure variant. We fill this gap by presenting a protocol based on the PSZ paradigm that achieves malicious-secure private set intersection.

We start with the observation that in the PSZ paradigm the two parties take the roles of "sender" and "receiver," and it is relatively straight-forward to secure the protocol against a malicious receiver [OOS16]. Therefore our approach is to run the protocol in both directions, so that each party must play the role of receiver at different times in the protocol. This high-level idea is inspired by the *dual-execution* technique of Mohassel & Franklin [MF06]. In that work, the parties perform two executions of Yao's protocol in opposite directions, taking advantage of the fact that Yao's protocol is easily secured against a malicious receiver. In that setting, the resulting dual-execution protocol achieves malicious security but leaks one adversarially-chosen bit. In our setting, however, we are able to carefully combine the two PSI executions in a way that achieves the usual notion of *full* malicious security.

Because our protocol is based on the fast PSZ paradigm, it relies exclusively on cheap symmetric-key cryptography. We have implemented our protocol and compare it to the previous state of the art. We find our protocol to be 12× faster than the previous fastest malicious-secure PSI protocol of [RR17], on large datasets. Our implementation can securely compute the intersection of million-item sets in only 12.6 seconds on a single thread (2.9 seconds with many threads).

Finally, as mentioned above, the previous fastest malicious PSI protocol [RR17] appears to rely inherently on the random-oracle model. We show that our protocol can be instantiated in the standard model. Both our standard model and random-oracle optimized protocols are faster than [RR17] in the LAN setting, with our latter protocol being the fastest across all settings.

2 Preliminaries

2.1 Notation

Alice is the sender and Bob is the receiver who learns the intersection. Alice holds the set X and Bob Y, where $X, Y \subseteq \{0,1\}^\sigma$. For simplicity, we assume $|X| = |Y| = n$. The computational and statistical security parameters are denoted as κ and λ. Let $[m] := \{1, ..., m\}$.

2.2 Security for Secure Computation

We define malicious security with the standard notation for secure computation. Namely, our protocol is malicious secure in the *universal composability (UC)* framework of Canetti [Can01]. This simulation based paradigm defines security with respect to two interaction,

- Real interaction: a malicious adversary $\mathcal{A}$ attacks the honest party who runs the protocol π. An *environment* $\mathcal{Z}$ chooses the honest party's input and is forwarded their final output of π. $\mathcal{Z}$ may arbitrarily interact with $\mathcal{A}$. The protocol is in a *hybrid* world where $\mathcal{A}$ and the honest

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 1: Ideal functionality for private set intersection (with one-sided output)

party have access to the ideal $\mathcal{F}_{\mathsf{encode}}$ functionality of Figure 2. We define $\textsc{real}[\pi, \mathcal{Z}, \mathcal{A}]$ to be the output of $\mathcal{Z}$ in this interaction.

- Ideal interaction: a malicious adversary $\mathcal{S}$ and an honest party interact with the ideal $\mathcal{F}_{\mathsf{PSI}}$ functionality of Figure 1. The honest party forwards the input provided by the *environment* $\mathcal{Z}$ to the $\mathcal{F}_{\mathsf{PSI}}$ functionality and returns their output to $\mathcal{Z}$. We define $\textsc{ideal}[\mathcal{F}_{\mathsf{PSI}}, \mathcal{Z}, \mathcal{S}]$ to be the output of $\mathcal{Z}$ in this interaction.

The protocol π **UC-securely realizes** $\mathcal{F}_{\mathsf{PSI}}$ if: for all PPT adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$, such that for all PPT environments $\mathcal{Z}$:

$$\textsc{real}[\pi, \mathcal{Z}, \mathcal{A}] \approx \textsc{ideal}[\mathcal{F}_{\mathsf{PSI}}, \mathcal{Z}, \mathcal{S}]$$

where "$\approx$" denotes computational indistinguishable.

3 Overview of PSZ Paradigm

Pinkas, Schneider, and Zohner [PSZ14] (hereafter PSZ) introduced a paradigm for PSI that is secure against semi-honest adversaries. There have since been several improvements made to this general paradigm [PSSZ15, KKRT16, OOS16]. In particular, the implementation of [KKRT16] is the fastest secure PSI protocol to date. Adapting this paradigm to the malicious security model is therefore a natural direction.

In this section, we describe the PSZ paradigm, and discuss what prevents it from achieving malicious security.

3.1 High-Level Overview

The PSZ paradigm works as follows. First, for simplicity suppose Alice has n items $X = \{x_1, \ldots, x_n\}$ while Bob has one item y. The goal of *private set inclusion* is for Bob to learn whether $y \in X$, and nothing more. We abstract the main step of the protocol as an **oblivious encoding** step, which is similar in spirit to an oblivious pseudorandom function [FIPR05]. The parties interact so that Alice learns a random mapping F, while Bob learns only $F(y)$. The details of this step are not relevant at the moment. Then Alice sends $F(X) = \{F(x_1), \ldots, F(x_n)\}$ to Bob. Bob can check whether his value $F(y)$ is among these values and therefore learn whether $y \in \{x_1, \ldots, x_n\}$. Since F is a random mapping, the other items in $F(X)$ leak nothing about the set X.

The protocol can be extended to a proper PSI protocol, where Bob has a set of items $Y = \{y_1, \ldots, y_n\}$. The parties simply perform n instances of the private set inclusion protocol, one for each y_i, with Alice using the same input X each time. This leads to a PSI protocol with $O(n^2)$ communication.

To reduce the communication cost, the parties can agree on a hashing scheme that assigns their items to bins. In PSZ, they propose to use a variant of Cuckoo hashing. For the sake of example, suppose Bob uses cuckoo hashing with two hash functions to assign his items to bins. In cuckoo hashing, Bob will assign item y to the bin with index either $h_1(y)$ or $h_2(y)$, so that each bin contains at most one item. Alice will assign each of her items x to *both* bins $h_1(x)$ and $h_2(x)$, so that each of her bins may contain several items. Overall, for each bin Alice has several items while Bob has (at most) one, so they can perform the private set inclusion protocol for each bin. There are of course many details to work out, but by using this main idea the communication cost of protocol can be reduced to $O(n)$.

3.2 Insecurity against Malicious Adversaries

The PSZ protocol and its followups are proven secure in the semi-honest setting, but are not secure against malicious adversaries. There are several features of the protocol that present challenges in the presence of malicious adversaries:

- Even if the “oblivious encoding” subprotocol is made secure against malicious adversaries, the set-inclusion subprotocol does not become malicious-secure. The technical challenge relates to the problem of the simulator extracting inputs from a malicious Alice. The simulator sees only the random mapping F and the items $\{F(x_1), \ldots, F(x_n)\}$ sent by Alice. For the simulator to extract Alice’s effective input, the mapping F must be invertible. However, the oblivious encoding instantiations generally do not result in an invertible F.

- In the PSZ protocol, Bob uses cuckoo hashing to assign his items to bins. Each item y may be placed in two possible locations, and the final placement of item *y depends on all of Bob’s other items.* A corrupt Alice may exploit this in the protocol to learn information about Bob’s set. In particular, Alice is supposed to place each item x in *both* possible locations $h_1(x)$ and $h_2(x)$. A corrupt Alice may place x only in $h_1(x)$. Then if x turns out to be in the intersection, Alice learns that Bob placed x in $h_1(x)$ but not $h_2(x)$. As just mentioned, whether Bob places an item according to h_1 or h_2 depends on all of Bob’s items, so it is information that cannot be simulated in the ideal world.

- In the $O(n^2)$ PSI protocol, Alice is supposed to run many instances of the simple set-inclusion protocol with the same set X each time. However, a malicious Alice may use different sets in different instances. In doing so, she can influence the output of the protocol in ways that cannot be simulated in the ideal world.

4 Oblivious Encoding

As discussed in the previous section, the PSZ paradigm uses an **oblivious encoding** step. In Figure 2 we define an ideal functionality for this task. Intuitively, the functionality chooses a random mapping F, allows the receiver to learn $F[c]$ for a single c, and allows the sender to learn $F[c]$ for an unlimited number of c’s. However, if the sender is corrupt, the functionality allows the sender to *choose* the mapping F (so that it need not be random). This reflects what our instantiations of this functionality are able to achieve.

We describe two instantiations of this functionality that are secure in the presence of malicious adversaries.

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 $F_{\mathsf{adv}} = \perp$. Then:
4. **[Receiver Output]** If $F_{\mathsf{adv}} = \perp$ 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. 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 $F_{\mathsf{adv}} = \perp$, choose $F[c']$ uniformly at random.
    - If $F[c']$ doesn't exist and $F_{\mathsf{adv}} \neq \perp$, 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 $\mathcal{F}_{\mathsf{encode}}$

In the programmable-random-oracle model. Orrù, Orsini & Scholl [OOS16] describe an efficient 1-out-of-N oblivious transfer protocol, for random OT secrets and N exponential in the security parameter. The protocol is secure against malicious adversaries. In order to model an exponential number of OT secrets, they give an ideal functionality which is identical to ours except that the adversary is never allowed to choose the mapping. Hence, their protocol also realizes our functionality as well (the simulator simply chooses $F_{\mathsf{adv}} = \perp$ so that the functionality always chooses a random mapping).

Their protocol is proven secure in the programmable random-oracle model. Concretely, the cost of a single OT/oblivious encoding in their protocol is roughly 3 times that of a single semi-honest 1-out-of-2 OT.

In the standard model. In the standard model, it is possible to use a variant of the semi-honest oblivious encoding subprotocol from PSZ. The protocol works as follows, where the receiver has input c:

- The sender chooses 2σ random κ-bit strings: $m[1,0], m[1,1], \ldots, m[\sigma,0], m[\sigma,1]$.
- The parties perform σ instances of OT, where in the ith instance the sender provides inputs $m[i,0], m[i,1]$, the receiver provides input c_i and receives $m[i,c_i]$.
- The receiver computes output $\bigoplus_i \mathsf{PRF}(m[i,c_i], c)$, where PRF is a secure pseudorandom function with ℓ bits of output.
- To obtain the encoding of any value c', the receiver can compute $\bigoplus_i \mathsf{PRF}(m[i,c'_i], c')$.

For security against a corrupt receiver, the simulator can extract c from the receiver's OT inputs. We can then argue that all other oblivious encodings look random to the receiver. Indeed, for every $c' \neq c$, there is a position i in which $c'_i \neq c_i$, so the corresponding encoding $\bigoplus_i \mathsf{PRF}(m[i, c'_i], c')$ contains a term $\mathsf{PRF}(m[i, c'_i], c')$ that is random from the receiver's point of view.

For security against a corrupt sender, the simulator can extract the $m[i, b]$ values from the sender's OT inputs. It can then hard-code these values into a circuit $F_{\mathsf{adv}}(c) = \bigoplus_i \mathsf{PRF}(m[i, c_i], c)$ and send this circuit to the ideal functionality.

The cost of this protocol is σ instances of OT per oblivious encoding. Since the protocol uses OTs with *chosen* secrets (not random secrets chosen by the functionality), it can be instantiated in the standard model.[1]

5 A Warmup: Quadratic-Cost PSI

The main technical idea for achieving malicious security is to carefully apply the dual execution paradigm of Mohassel & Franklin [MF06] to the PSZ paradigm for private set intersection. In this section we give a protocol which contains the main ideas of our approach, but which has quadratic complexity. In the next section we describe how to apply a hashing technique to reduce the cost.

5.1 Dual Execution Protocol

The main idea behind our approach is as follows (a formal description is given in Figure 3):

1. The parties perform an encoding step similar to PSZ, where Alice acts as receiver. In more detail, the parties invoke $\mathcal{F}_{\mathsf{encode}}$ once for each of Alice's items. Alice learns $[\![x_j]\!]^{\mathsf{B}}_j$, where x_j is her jth item and $[\![\cdot]\!]^{\mathsf{B}}_j$ is the encoding used in the jth instance of $\mathcal{F}_{\mathsf{encode}}$. Note that Bob can obtain $[\![v]\!]^{\mathsf{B}}_j$ for any v and any j, by appropriately querying the functionality.

2. The parties do the same thing with the roles reversed. Bob learns $[\![y_i]\!]^{\mathsf{A}}_i$, where y_i is his ith item and $[\![\cdot]\!]^{\mathsf{A}}_i$ is the encoding. As above, Alice can obtain any encoding of the form $[\![v]\!]^{\mathsf{A}}_j$.

At this point, let us define a **common encoding**:

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

The important property of this encoding is:

- If Alice knows $[\![v]\!]^{\mathsf{B}}_j$ then she can compute the common encoding $[\![v]\!]_{i,j}$ for any i.
- If Alice does not know $[\![v]\!]^{\mathsf{B}}_j$, then it is actually random from her point of view. It is therefore hard for her to predict common encoding $[\![v]\!]_{i,j}$ for any i.

A symmetric condition holds for Bob. Now the idea is for the parties to compute all of the common encodings that they can deduce from these rules. Then the intersection of these encodings will correspond to the intersection of their sets. In other words (continuing the protocol overview):

3. Alice computes a set of encodings $E = \{[\![x_j]\!]_{i,j} \mid i, j \in [n]\}$, and sends it to Bob.

[1]Modern OT extension protocols can be optimized for OT of random secrets, but it is not known how to make this special case less expensive while avoiding the programmable-random-oracle model.

4. Bob likewise computes a set of encodings and checks which of them appear in E. These encodings correspond to the intersection. More formally, Bob outputs:

$$Z = \{y_i \in Y \mid \exists j \in [n] : [\![y_i]\!]_{i,j} \in E\}$$

We note that in this protocol, only Bob receives output. In fact, it turns out to be problematic if Bob sends an analogous set of encodings to Alice. In Section 6.7 we discuss in more detail the problems associated with both parties receiving output.

5.2 Security

The protocol achieves malicious security:

Theorem 1. *The protocol in Figure 3 is a UC-secure protocol for PSI in the $\mathcal{F}_{\mathsf{encode}}$-hybrid model.*

We defer giving a formal proof for this protocol in favor of a single proof of our final protocol in the next section. Instead, we sketch the high-level idea of the simulation.

When Alice is corrupt, the simulator plays the role of $\mathcal{F}_{\mathsf{encode}}$ and therefore observes Alice's inputs to the functionality during Step 2. Let x_j denote Alice's jth input to $\mathcal{F}_{\mathsf{encode}}$, in which she learns $[\![x_j]\!]^{\mathsf{B}}_j$. Let $\tilde{X} = \{x_1, \ldots, x_n\}$. We can make the following observations:

- Suppose Bob has an item $y \notin \tilde{X}$. In the protocol, Alice will send a set of encodings E, and Bob will search this set for encodings $[\![y]\!]_{i,j}$, for certain i, j values. But by the definition of $\tilde{X}$, Alice does not know any encoding of the form $[\![y]\!]^{\mathsf{B}}_j$, and so with high probability cannot guess any encoding which will cause Bob to include y in the output. In other words, we can argue that **Alice's effective input is a subset of $\tilde{X}$.**

- Suppose for simplicity Bob's input happens to be $\tilde{X}$. This turns out to be the most interesting case for the proof. Bob will randomly permute these items and obtain an encoding of each one. Let π be the permutation such that Bob learns $[\![x_j]\!]^{\mathsf{A}}_{\pi(j)}$. Now Bob will be looking in the set E for common encodings of the form $[\![x_j]\!]_{\pi(j),*}$. Note that from the definition of $\tilde{X}$, Alice can only produce valid encodings of the form $[\![x_j]\!]_{*,j}$. It follows that **Bob will include a value x_j in his output if and only if Alice includes encoding $[\![x_j]\!]_{\pi(j),j} \in E$.**

Since the distribution of π is random, the simulator can simulate the effect. More precisely, the simulator chooses a random π and sets $X^* = \{x_j \mid [\![x_j]\!]_{\pi(j),j} \in E\}$. It is this X^* that the simulator finally sends to the ideal functionality. In the above, we were considering a special case where Bob's input happens to be $\tilde{X}$. However, this simulation approach works in general.

The simulation for a malicious Bob is simpler, and it relies on the fact that common encodings look random, for values not in the intersection.

The protocol is correct as long as there are no spurious collisions among common encodings. That is, we do not have any $x_j \in X$ and $y_i \in Y \setminus X$ for which $[\![x_j]\!]_{i,j} = [\![y_i]\!]_{i,j}$ (which would cause Bob to erroneously place y_i in the intersection). The probability of this happening for a fixed x_j, y_i is $2^{-\ell}$, if the encodings have length ℓ. By a union bound, the total probability of such an event is $n^2 2^{-\ell}$. We set $\ell = \lambda + 2\log n$ to ensure this error probability is at most $2^{-\lambda}$.

5.3 Encode-Commit Protocol

In addition to the approach described above, we present an alternative protocol based on $\mathcal{F}_{\mathsf{encode}}$ and commitments that offers communication/computation trade-offs. Fundamentally, the dual execution protocol above first restricts Alice to her set by requiring her to encode it as $\{[\![x_1]\!]^{\mathsf{B}}_1, ..., [\![x_n]\!]^{\mathsf{B}}_n\}$.

Parameters: $\mathcal{F}_{\mathsf{encode}}$ is the Oblivious Encoding functionality with input domain $\{0,1\}^\sigma$ output bit length $\lambda + 2\log n$.

On Input $(\textsc{Send}, \mathsf{sid}, X)$ from Alice and $(\textsc{Receive}, \mathsf{sid}, Y)$ from Bob, where $X, Y \subseteq \{0,1\}^\sigma$ and $|X| = |Y| = n$. Each party randomly permutes their set.

1. **[A Encoding]** For $i \in [n]$, Bob sends $(\textsc{Encode}, (\mathsf{sid}, \mathsf{A}, i), y_i)$ to $\mathcal{F}_{\mathsf{encode}}$ who sends $(\textsc{Output}, (\mathsf{sid}, \mathsf{A}, i), [\![y_i]\!]_i^{\mathsf{A}})$ to Bob and $(\textsc{Output}, (\mathsf{sid}, \mathsf{A}, i))$ to Alice.

   For $j \in [n]$, Alice sends $(\textsc{Encode}, (\mathsf{sid}, \mathsf{A}, i),\ x_j)$ to $\mathcal{F}_{\mathsf{encode}}$ and receives $(\textsc{Output}, (\mathsf{sid}, \mathsf{A}, i),\ [\![x_j]\!]_i^{\mathsf{A}})$ in response.

2. **[B Encoding]** For $i \in [n]$, Alice sends $(\textsc{Encode}, (\mathsf{sid}, \mathsf{B}, i), x_i)$ to $\mathcal{F}_{\mathsf{encode}}$ who sends $(\textsc{Output}, (\mathsf{sid}, \mathsf{B}, i), [\![x_i]\!]_i^{\mathsf{B}})$ to Alice and $(\textsc{Output}, (\mathsf{sid}, \mathsf{B}, i))$ to Bob.

   For $j \in [n]$, Bob sends $(\textsc{Encode}, (\mathsf{sid}, \mathsf{B}, i),\ y_j)$ to $\mathcal{F}_{\mathsf{encode}}$ and receives $(\textsc{Output}, (\mathsf{sid}, \mathsf{B}, i),\ [\![y_j]\!]_i^{\mathsf{B}})$ in response.

3. **[Output]** Alice sends the common encodings

$$E = \{[\![x_j]\!]_i^{\mathsf{A}} \oplus [\![x_j]\!]_j^{\mathsf{B}} \mid i, j \in [n]\}$$

   to Bob who outputs

$$\{y_i \mid \exists j : [\![y_i]\!]_i^{\mathsf{A}} \oplus [\![y_i]\!]_j^{\mathsf{B}} \in E\}$$

Figure 3: Malicious-secure n^2 PSI.

In some sense this encoding operation can be viewed as Alice committing to her inputs. The property that we need from the B encoding are: 1) $[\![*]\!]^{\mathsf{B}}$ must allow the simulator to extract the set of candidate x_j values; 2) provides a binding proof to the value x_j. Continuing to view $[\![*]\!]^{\mathsf{B}}$ as a commitment, the dual execution protocol instructs Alice to then decommit (prove she was bound to x_j) to these values by sending all $[\![x_j]\!]_j^{\mathsf{B}}$ encodings to Bob, but masked under $[\![x_j]\!]_i^{\mathsf{A}}$ so that the commitment can only be "decommitted" if Bob knows *one* of these encodings of x_j.

Taking this idea to its conclusion, we can formulate a new protocol where Alice simply commits to her inputs by sending $\textsc{Comm}(x_1; r_1), ..., \textsc{Comm}(x_n; r_n)$ to Bob in lieu of Figure 3 Step 2, where $\textsc{Comm}$ is a standard (non-interactive) commitment scheme. The final step of the protocol is for her to send the decommitment r_j masked under the encodings of x_j

$$E = \left\{[\![x_j]\!]_i^{\mathsf{A}} \oplus r_j \mid i, j \in [n]\right\}$$

In the event that Bob knows $[\![x_j]\!]_i^{\mathsf{A}}$, i.e. his input contains $y_i = x_j$, he will be able to recover the decommitment value r_j and decommit $\textsc{Comm}(x_j; r_j)$, thereby inferring that x_j is in the intersection.

The security proof of this protocol follows the same structure as before. For the more interesting case of a malicious Alice, we require an extractable commitment scheme. The simulator is able to extract the set $\tilde{X} = \{x_1, ..., x_n\}$ from the commitments $\textsc{Comm}(x_1; r_1), ..., \textsc{Comm}(x_n; r_n)$ and sends $X^* = \{x_j \mid [\![x_j]\!]_{\pi(j)}^{\mathsf{A}} \oplus r_j \in E\}$ to the functionality. The correctness of this simulation strategy follows from the sketch in the previous section by viewing $\textsc{Comm}(x_j; r_j)$ as equivalent to the encoding $[\![*]\!]_j^{\mathsf{B}}$ and r_j as equivalent to $[\![x_j]\!]_j^{\mathsf{B}}$.

The communication and computation complexity for both of these protocols is $O(n^2)$. However, we will later show that the concrete communication/computation overheads of these two approaches result in interesting performance trade-offs. Most notable is that the commitment based approach requires less computation at the expense of additional communication, making it more efficient in the LAN setting.

6 Our Full Protocol

After constructing a quadratic-cost PSI protocol, the PSZ paradigm is for the parties to use a hashing scheme to assign their items into *bins*, and then perform the quadratic-cost PSI on each bin. We review this approach here, and discuss challenges specific to the malicious setting.

6.1 Hashing

Cuckoo hashing and its drawbacks. The most efficient hashing scheme in PSZ is Cuckoo hashing. In this approach, the parties agree on two (or more) random functions h_1 and h_2. Alice uses Cuckoo hashing to map her items into bins. As a result, each item x is placed in either bin $\mathcal{B}[h_1(x)]$ or $\mathcal{B}[h_2(x)]$ such that each bin has at most one item. Bob conceptually places each of his items y into *both* bins $\mathcal{B}[h_1(y)]$ or $\mathcal{B}[h_2(y)]$. Then the parties perform a PSI for the items in each bin. Since Alice has only one item per bin, these PSIs are quite efficient.

Unfortunately, this general hashing approach does not immediately work in the malicious security setting. Roughly speaking, the problem is that Bob may place an item y into bin $\mathcal{B}[h_1(y)]$ but *not* in $\mathcal{B}[h_2(y)]$. Suppose Alice also has item y, then y will appear in the output if and only if Alice's cuckoo hashing has chosen to place it in $\mathcal{B}[h_1(y)]$ and not $\mathcal{B}[h_2(y)]$. Because of the nature of Cuckoo hashing, whether an item is placed according to h_1 or h_2 event depends in a subtle way on *all other items* in Alice's set. As a result, the effect of Bob's misbehavior cannot be simulated in the ideal world.

Simple hashing. While Cuckoo hashing is problematic for malicious security, we can still use a *simple hashing* approach. The parties agree on a random function $h : \{0,1\}^* \to [m]$ and assign item x to bin $\mathcal{B}[h(x)]$. Then parties can perform a PSI for each bin. Note that under this hashing scheme, the hashed location of each item does not depend on other items in the set. Each item has only one "correct" location.

Note that the load (number of items assigned) of any bin leaks some information about a party's input set. Therefore, all bins must be padded to some maximum possible size. A standard balls-and-bins argument shows that the maximum load among the $m = O(n/\log n)$ bins is $O(\log n)$ with very high probability.

Phasing. In the standard-model variant of our protocol, the oblivious encoding step scales linearly with the length of the items being encoded. Our random-oracle protocol also has a weak dependence on the representation length of the items which is affected by the size of the sets. Hence, it is desirable to reduce the length of these items as much as possible.

Pinkas et al. [PSSZ15] described how to use a hashing technique of Arbitman et al. [ANS10] called **phasing** (permutation-based hashing) to reduce the length of items in each bin. The idea is as follows. Suppose we are considering PSI on strings of length σ bits. Let h be a random function with output range $\{0,1\}^d$, where the number of bins is 2^d. To assign an item x to a bin, we write $x = x_L \| x_R$, with $|x_L| = d$. We assign this item to bin $h(x_R) \oplus x_L$, and store it in that bin with x_R as its representation. Arbitman et al. [ANS10] show that this method of assigning items to bins results in maximum load $O(\log n)$ with high probability.

Note that the representations in each bin are $\sigma - d$ bits long — shorter by d bits. Importantly, shrinking these representations does not introduce any collisions. This is because the mapping $\mathsf{phase}(x_L \| x_R) = (h(x_R) \oplus x_L, x_R)$ is a Feistel function and therefore invertible. So distinct items will either be mapped to distinct bins, or, in the case that they are mapped to the same bin, they *must* be assigned different representations. Hence the PSI subprotocol in each bin can be performed on the shorter representations.

The idea can be extended as follows, when the number m of bins is not a power of two (here h is taken to be a function with range $[m]$):

$$\mathsf{phase}_{h,m}(x) = \Big(h(\lfloor x/m \rfloor) + x \bmod m, \;\; \lfloor x/m \rfloor \Big)$$
$$\mathsf{phase}^{-1}_{h,m}(b, z) = zm + [h(z) + b \bmod m]$$

We show that phasing is a secure way to reduce the length of items, in the presence of malicious adversaries.

6.2 Aggregating Masks Across Bins

Suppose we apply the simple hashing technique to our quadratic PSI protocol. The resulting protocol would work as follows.

1. First, the parties hash their n items into $m = O(n/\log n)$ bins. With high probability each bin has at most $\mu = O(\log n)$ items. Bins are artificially padded with dummy items to a fixed size of μ items.

2. For each bin the parties perform the quadratic-cost PSI protocol from Section 5. Each party acts as $\mathcal{F}_{\mathsf{encode}}$ sender and receiver, and computes common encodings of the items. For each bin, Alice sends all $\mu^2 = O(\log^2 n)$ encodings to Bob, who computes the intersection.

The total cost of this protocol is therefore $m\mu^2 = O(n \log n)$, a significant improvement over the quadratic protocol.

We present an additional optimization which reduces the cost by a significant constant factor. Our primary observation is that in order to hide the number of items in each bin, the parties must pad the bins out to the maximum size μ. However, this results in their bins containing mostly dummy items (in our range of parameters, around 75% are dummy items).

When Alice sends her common encodings in the final step of the protocol, she knows that the encodings for dummy items cannot contribute to the final result. If she had a way to avoid sending these dummy encodings, it would reduce the number of encodings sent by roughly a factor of 4.

Hence, we suggest an optimization in which Alice aggregates her encodings *across all the bins*, and send only the *non-dummy* encodings to Bob, as a unified collection. Similarly, Bob need not check Alice's set of encodings for one of his dummy encodings. So Bob computes common encodings only for his actual input items.

To show the security of this change, we need only consider Bob's view which has been slightly altered. Suppose Alice chooses a random value d to be a "universal" dummy item in each bin. Since this item is chosen randomly, it is negligibly likely that Bob would have used it as input to any instance of $\mathcal{F}_{\mathsf{encode}}$ where he was the receiver. Hence, the common encodings of dummy values look random from Bob's perspective. Intuitively, the only common encodings we removed from the protocol are ones that looked random from Bob's perspective (and hence, had no effect on his output, with overwhelming probability).

Note that it is not secure to eliminate dummy encodings within a *single* quadratic-PSI. This would leak how many items Alice assigned to that bin. It is not secure to leak the number of items

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

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 computes $(b, x') = \mathsf{phase}_{h,m}(x)$ and adds x' to bin $\mathcal{B}_X[b]$ at a random unused position $p \in [\mu]$.
   (b) For $y \in Y$, Bob computes $(b, y') = \mathsf{phase}_{h,m}(y)$ and adds y' to bin $\mathcal{B}_Y[b]$ at a random unused position $p \in [\mu]$.

   Both parties fill unused bin positions with the zero string.
3. **[Encoding]** For bin index $b \in [m]$ and position $p \in [\mu]$:
   (a) Let x' be the value in bin $\mathcal{B}_X[b]$ at position p. Alice sends $(\textsc{Encode}, (\mathsf{sid}, \mathsf{B}, b, p), x')$ to the $\mathcal{F}_{\mathsf{encode}}$ functionality which responds with $(\textsc{Output}, (\mathsf{sid}, \mathsf{B}, b, p), [\![x']\!]^{\mathsf{B}}_{b,p})$. Bob receives $(\textsc{Output}, (\mathsf{sid}, \mathsf{B}, b, p))$ from $\mathcal{F}_{\mathsf{encode}}$.
   (b) Let y' be the value in bin $\mathcal{B}_Y[b]$ at position p. Bob sends $(\textsc{Encode}, (\mathsf{sid}, \mathsf{A}, b, p), y')$ to the $\mathcal{F}_{\mathsf{encode}}$ functionality which responds with $(\textsc{Output}, (\mathsf{sid}, \mathsf{A}, b, p), [\![y']\!]^{\mathsf{A}}_{b,p})$. Alice receives $(\textsc{Output}, (\mathsf{sid}, \mathsf{A}, b, p))$ from $\mathcal{F}_{\mathsf{encode}}$.
4. **[Output]**
   (a) **[Alice's Common Mask]** For each $x \in X$, in random order, let b, p be the bin index and position that x' was placed in during Step 2a to represent x. For $j \in [\mu]$, Alice sends $(\textsc{Encode}, (\mathsf{sid}, \mathsf{A}, b, j), x')$ to $\mathcal{F}_{\mathsf{encode}}$ and receives $(\textsc{Output}, (\mathsf{sid}, \mathsf{A}, b, j), [\![x']\!]^{\mathsf{A}}_{b,j})$ in response. Alice sends
   $$[\![x']\!]^{\mathsf{A}}_{b,j} \oplus [\![x']\!]^{\mathsf{B}}_{b,p}$$
   to Bob. Let E denote the $n\mu$ encodings that Alice sends.
   (b) **[Bob's Common Mask]** Similarly, for $y \in Y$, let b, p be the bin index and position that y' was placed in during Step 2b to represent y. For $j \in [\mu]$, Bob sends $(\textsc{Encode}, (\mathsf{sid}, \mathsf{B}, b, j), y')$ to $\mathcal{F}_{\mathsf{encode}}$ and receives $(\textsc{Output}, (\mathsf{sid}, \mathsf{B}, b, j), [\![y']\!]^{\mathsf{B}}_{b,j})$ in response. Bob outputs
   $$\left\{ y \in Y \;\middle|\; \exists j \in [\mu] : [\![y']\!]^{\mathsf{A}}_{b,p} \oplus [\![y']\!]^{\mathsf{B}}_{b,j} \in E,\ \text{where } (b, y') = \mathsf{phase}_{h,m}(y) \right\}$$

Figure 4: Our malicious-secure Dual Execution PSI protocol.

in each bin. (It is for this reason that we still must perform exactly μ oblivious encoding steps per bin.) However, it is safe to leak the fact that Alice has n items *total.* By aggregating encodings across *all* bins we are able to use this common knowledge. Bob now sees a single collection of $n\mu$ encodings, but does not know which bins they correspond to.

After making this change, Bob is comparing each of his $n\mu$ non-dummy encodings to each of Alice's $n\mu$ encodings. Without this optimization, he only compares encodings within each bin. With more comparisons made among the common encodings, the probability of spurious collisions increases. We must therefore increase the length of these encodings. A similar argument to the previous section shows that if the encodings have length $\lambda + 2\log(n\mu)$, then the overall probability of a spurious collision is $2^{-\lambda}$.

6.3 Dual Execution Protocol Details & Security

The formal details of our dual execution protocol are given in Figure 4. The protocol follows the high-level outline developed in this section. We use the following notation:

- $[\![x]\!]^{\mathsf{A}}_{b,p}$ denotes an encoding of value x, in an instance of $\mathcal{F}_{\mathsf{encode}}$ where Alice is sender, corresponding to position p in bin b. Each bin stores a maximum of μ items, so there are μ positions.

- We write $(b, x') = \mathsf{phase}_{h,m}(x)$ to denote the phasing operation (Section 6.1), where to store item x we place representative x' in bin b.

Theorem 2. *The protocol in Figure 4 is UC-secure in the $\mathcal{F}_{\mathsf{encode}}$-hybrid model. The resulting protocol has cost $O(Cn\log n)$, where $C \approx \kappa$ is the cost of one $\mathcal{F}_{\mathsf{encode}}$ call on a $\sigma - \log n$ length bit string.*

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 does nothing in Step 2 and Step 3a (steps where Bob receives no output). To extract Bob's set, the simulator observes all of Bob's $\mathcal{F}_{\mathsf{encode}}$ messages $(\textsc{Encode}, (\mathsf{sid}, \mathsf{A}, b, p), y'_{b,p})$ in Step 3b. The simulator computes $Y = \{\mathsf{phase}^{-1}_{h,m}(b, y'_{b,p}) \mid b \in [m], p \in [\mu]\}$ and sends it to the ideal $\mathcal{F}_{\mathsf{PSI}}$ functionality which responds with the intersection $Z = X \cap Y$.
>
> Set Z^* to be equal to Z along with arbitrary dummy items not in Y, so that $|Z^*| = n$. For each $z \in Z^*$, compute $(b, z') = \mathsf{phase}_{m,h}(z)$ and insert z' into a random unused position bin $\mathcal{B}_X[b]$. For $z \in Z^*$ in random order, and $j \in [\mu]$, compute $(b, z') = \mathsf{phase}_{m,h}(z)$ and send $[\![z']\!]^{\mathsf{A}}_{b,p} \oplus [\![z']\!]^{\mathsf{B}}_{b,j}$ to Bob, where these encodings are obtained by playing the role of $\mathcal{F}_{\mathsf{encode}}$.

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

Hybrid 0 The first hybrid is the real interaction as specified in Figure 3 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 $(\textsc{Encode}, (\mathsf{sid}, \mathsf{A}, b, p), y'_{b,p})$ in Step 3b. Based on these, define the set $\tilde{Y} = \{\mathsf{phase}^{-1}_{h,m}(b, y'_{b,p}) \mid b \in [m], p \in [\mu]\}$.

Hybrid 1 In this hybrid, we modify Alice to send dummy values to $\mathcal{F}_{\mathsf{encode}}$ in Step 2a. Then we further modify Alice to perform the hashing at the last possible moment in Step 4. The simulation can obtain the appropriate encodings directly from the simulated $\mathcal{F}_{\mathsf{encode}}$. 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 $[\![x']\!]^{\mathsf{A}}_{b,j} \oplus [\![x']\!]^{\mathsf{B}}_{b,p}$, for some position p, where $(b, x') = \mathsf{phase}_{h,m}(x)$. Suppose $x \notin \tilde{Y}$. By construction of $\tilde{Y}$, Bob never obtained an encoding of the form $[\![x']\!]^{\mathsf{A}}_{b,j}$. This encoding is therefore distributed independent of everything else in the simulation. In particular, the *common* encodings corresponding to this x are distributed independently of the choice of (b, x') and hence the choice of x.

We therefore modify the hybrid in the following way. Before Alice adds the items of X to her hash table in Step 4a, she replaces all items in $X \setminus \tilde{Y}$ (*i.e.*, all items not in $X \cap \tilde{Y}$) with fixed dummy values not in Y. By the above argument, the adversary's view is identically distributed in this modified hybrid.

The final hybrid works as follows. A simulator interacts with the adversary and determines a set $\tilde{Y}$, without using Alice's actual input X. Then it computes $X \cap \tilde{Y}$ and simulates Alice's message in Step 4a using only $X \cap \tilde{Y}$. Hence, this hybrid corresponds to our final simulator, where we 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 does nothing in Step 2 and Step 3b. In Step 3a, the simulator intercepts Alice's commands of the form $(\textsc{Encode}, (\mathsf{sid}, \mathsf{B}, b, p), x'_{b,p})$. The simulator computes a set of candidates $\tilde{X} = \{\mathsf{phase}^{-1}_{h,m}(b, x'_{b,p}) \mid b \in [m], p \in [\mu]\}$ and for $x \in \tilde{X}$ let $\mathsf{c}(x)$ denote the number of times that $\mathsf{phase}^{-1}_{h,m}(b, x'_{b,p}) = x$ for $b \in [m], p \in [\mu]$.
>
> The simulator computes a hash table $\mathcal{B}$ as follows. For $x \in \tilde{X}$ and $i \in \mathsf{c}(x)$, the simulator computes $(b, x') = \mathsf{phase}_{h,m}(x)$ and places x' in a random unused position in bin $\mathcal{B}[b]$. Although $|\tilde{X}|$ may be as large as $m\mu$, by construction no bin will have more than μ items. For each such 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
>
> $$X^* = \big\{x \in \tilde{X} \ \mid \exists j \in [\mu], p \in \mathsf{p}(x) : \\ [\![x']\!]^{\mathsf{A}}_{b,j} \oplus [\![x']\!]^{\mathsf{B}}_{b,p} \in E \\ \wedge (b, x') = \mathsf{phase}_{h,m}(x)\big\} \tag{1}$$
>
> 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 as specified in Figure 3 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 $(\textsc{Encode}, (\mathsf{sid}, \mathsf{B}, b, p), x'_{b,p})$ in Step 3a. Based on these, define $\tilde{X} = \{\mathsf{phase}^{-1}_{h,m}(b, x'_{b,p}) \mid b \in [m], p \in [\mu]\}$.

Hybrid 1 In this hybrid, we modify Bob to send the zero string to $\mathcal{F}_{\mathsf{encode}}$ in Step 2b. The simulation can obtain all required encodings directly from the simulated $\mathcal{F}_{\mathsf{encode}}$. We also have Bob perform his hashing not in Step 2b but at the last possible moment in Step 4b. The hybrid is indistinguishable by the properties of $\mathcal{F}_{\mathsf{encode}}$.

Hybrid 2 The hybrid computes the output as specified in Step 4b. We then modify it to immediately remove all from this output which is not in $\tilde{X}$. The hybrids differ only in the event that simulated Bob computes an output in Step 4b that includes an item $y \notin \tilde{X}$. This happens only if $[\![y']\!]^{\mathsf{A}}_{b,j} \oplus [\![y']\!]^{\mathsf{B}}_{b,p} \in E$, where $(b, y') = \mathsf{phase}_{h,m}(y)$ and Bob places y' in position p. Since $y \notin \tilde{X}$, however, the encoding $[\![y']\!]^{\mathsf{B}}_{b,p}$ 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 in Step 4b, the simulated Bob uses $\tilde{X}$ instead of his actual input Y. Each $x \in \tilde{X}$ is inserted $\mathsf{c}(x)$ times. Then he computes the protocol output as specified in Step 4b; 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 4b, 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. □

Set Size for Malicious Parties As the ideal PSI functionality in Figure 1 indicates, our protocol realized a slightly relaxed variant of traditional PSI that does not strictly enforce the size of a corrupt party's input set. The functionality allows an honest party to provide an input set of size n, but a corrupt party to provide a set of size $n' > n$. We now analyze why this is the case and what is the exact relationship between n and n'.

Let us first consider the case of a malicious Bob who learns the intersection. The simulator extracts a set based on the commands Bob gave when acting as $\mathcal{F}_{\mathsf{encode}}$ receiver. Bob is given $m\mu = O(n)$ opportunities to act as $\mathcal{F}_{\mathsf{encode}}$ receiver, and therefore the simulator extracts a set of size at most $n' = m\mu = O(n)$. Concretely, when $\lambda = 40, n = 2^{20}$ and $m = n/\log_2 n$, the optimal bin size is $\mu = 68$ and Bob's maximum set size is $n' < 4n$.

The situation for a malicious Alice is similar. As above, the simulator computes a set $\tilde{X}$ based on commands Alice gives to $\mathcal{F}_{\mathsf{encode}}$ when acting as receiver. The size of $\tilde{X}$ is therefore at most $m\mu = O(n)$. The simulator finally extracts Alice's input as X^*, a *subset* of $\tilde{X}$. Hence her input has size at most $n' = m\mu$.

However, the situation is likely slightly better than this strict upper bound. Looking closer, Alice can only send a set E of $n\mu$ (not $m\mu$) common encodings in the final step of the protocol. Each item $x \in \tilde{X}$ is associated with $\mu\mathsf{c}(x)$ common encodings, i.e. μ for each time she sends x in a $\mathcal{F}_{\mathsf{encode}}$ command as the receiver. So Alice is in the situation where if she wants more than n items to be represented in the set E, then at least one item must have one of its possible encodings excluded from E. This lowers the probability of that item being included in the final extracted input X^*.

In general, suppose for each $x \in \tilde{X}$, Alice includes $\mathsf{k}_i(x)$ encodings in her set E that are associated with the ith time she acted as $\mathcal{F}_{\mathsf{encode}}$ receiver with x. Hence $\sum_{x\in\tilde{X}} \sum_{i\in[\mathsf{c}(x)]} \mathsf{k}_i(x) \leq n\mu$. Inspecting the simulation, we see that the probability a particular $x \in \tilde{X}$ survives to be included in X^* is $\Pr[x \in \tilde{X} \Rightarrow x \in X^*] = 1 - \prod_{\in[\mathsf{c}(x)]}(1 - \mathsf{k}_i(x))/\mu$ or simply $\mathsf{k}_1(x)/\mu$ in the case $\mathsf{c}(x) = 1$ (it happens only if the simulator happens to place x in a favorable position in the hash table). Hence, the expected size of X^* is $\sum_{x\in\tilde{X}} \sum_{i\in[\mathsf{c}(x)]} \mathsf{k}_i(x)/(\mu\mathsf{c}(x)) \leq n$.

6.4 Encode-Commit Protocol

We now turn our attention to the encode-commit style PSI protocol described in Section 5.3 and outline how the optimizations of Section 6.1, 6.2 can be applied to it. Recall that the encode-commit protocol instructs Bob to encode his items as $\mathcal{F}_{\mathsf{encode}}$ receiver while Alice must send commitments of her items. The final step of this protocol is for Alice to send decommitments of her values encrypted under the corresponding $\mathcal{F}_{\mathsf{encode}}$ encodings.

It is straight forward to see that the hashing to bins technique of Section 6.1 is compatible with the encode-commit style PSI. When the optimization of aggregating masks across bins from Section 6.2 is applied, we observe that the situation becomes more complicated. Let us assume that Alice now sends the commitment to her value y together with the decommitment r encrypted under the encodings $\{[\![y']\!]^{\mathsf{A}}_{b,p} \mid p \in [\mu]\}$ where $(b, y') = \mathsf{phase}_{h,m}(y)$. That is, for a random order of $y \in Y$, Alice sends

$$\textsc{Comm}(y; r), \{[\![y']\!]^{\mathsf{A}}_{b,p} \oplus r \mid p \in [\mu]\}$$

to Bob. For each $x \in X$, Bob must trial decommit to all such $\textsc{Comm}(y; r)$ with the decommitment value $([\![y']\!]^{\mathsf{A}}_{b,\mathsf{p}(x)} \oplus r) \oplus [\![x']\!]^{\mathsf{A}}_{\mathsf{b}(x),\mathsf{p}(x)}$. This would result in Bob performing $O(n^2)$ trial decommitments, eliminating any performance benefits of hashing. This overhead can be reduced by requiring Alice to send additional information that allows Bob to quickly identify which decommitment to try. Specifically, we will use the $\mathcal{F}_{\mathsf{encode}}$ encodings to derive two values, $[\![v]\!]^{\text{TAG}}_{b,p} = \text{PRF}([\![v]\!]^{\mathsf{A}}_{b,p}, \text{TAG})$ and $[\![v]\!]^{\text{ENC}}_{b,p} = \text{PRF}([\![v]\!]^{\mathsf{A}}_{b,p}, \text{ENC})$. The important property here is that given the encoding $[\![v]\!]^{\mathsf{A}}_{b,p}$, both values can be derived, but without the encoding the two values appear pseudo-random and independent. We now have Alice send

$$\textsc{Comm}(y; r), \{[\![y']\!]^{\text{TAG}}_{b,p} \mid\mid ([\![y']\!]^{\text{ENC}}_{b,p} \oplus r) \mid p \in [\mu]\}$$

Bob can now construct a hash table mapping $[\![x']\!]^{\text{TAG}}_{b,p}$ to $([\![x']\!]^{\text{ENC}}_{b,p}, x)$. Upon receiving a commitment and the associated tagged decommitments, Bob can query each of Alice's tags in the hash table. If a match is found, Bob will add the associated x to the intersection if the associated $[\![x']\!]^{\text{ENC}}_{b,p}$ value is successfully used to decommits $\textsc{Comm}(y; r)$.

6.5 Encode-Commit Protocol Details & Security

We give a formal description of the protocol in Figure 9. The protocol requires a non-interactive commitment scheme. In Section A we discuss the security properties required of the commitment scheme. At a high level, we require an extractable commitment scheme with a standard (standalone) hiding requirement. In particular, we do not require equivocability. In the non-programmable random oracle, the standard scheme $H(x\|r)$ satisfies our required properties.

Theorem 3. *The protocol in Figure 9 is UC-secure in the $\mathcal{F}_{\mathsf{encode}}$-hybrid model, when the underlying commitment scheme satisfies Definition 4. The resulting protocol has cost $O(Cn \log n)$, where $C \approx \kappa$ is the cost of one (sender) $\mathcal{F}_{\mathsf{encode}}$ call on a $\sigma - \log n$ length bit string.*

Proof. Due to the similarity to the previous proof we defer giving hybrids and simply describe the simulators. We start with the case of a **corrupt Bob**. The simulator must extract Bob's input, and simulate the messages in the protocol. The simulator is nearly the same as in the previous protocol:

> The simulator plays the role of the ideal $\mathcal{F}_{\mathsf{encode}}$ functionality. The simulator does nothing in Step 2. To extract Bob's set, the simulator observes all of Bob's $\mathcal{F}_{\mathsf{encode}}$ messages $(\textsc{Encode}, (\mathsf{sid}, \mathsf{A}, b, p), y'_{b,p})$ in Step 3. The simulator computes $Y = \{\mathsf{phase}^{-1}_{h,m}(b, y'_{b,p}) \mid b \in [m], p \in [\mu]\}$ and sends it to the ideal $\mathcal{F}_{\mathsf{PSI}}$ functionality which responds with the intersection $Z = X \cap Y$.
>
> Set Z^* to be equal to Z along with arbitrary dummy items not in Y, so that $|Z^*| = n$. For each $z \in Z^*$, compute $(b, z') = \mathsf{phase}_{m,h}(z)$ and insert z' into bin $\mathcal{B}_X[b]$ at a random unused position $p \in [\mu]$. For $z \in Z^*$ in random order, compute $(b, z') = \mathsf{phase}_{m,h}(z)$ and

send $\textsc{Comm}(z; r_z), \{[\![z']\!]^{\text{TAG}}_{b,p} \mid\mid [\![z']\!]^{\text{ENC}}_{b,p} \oplus r_z\}$ to Bob, where these encodings are obtained by playing the role of $\mathcal{F}_{\mathsf{encode}}$.

Importantly, the simulator extracts Bob's input in step 3 and thus knows the protocol output before step 4. It can therefore send appropriate commitments and use dummy commitments for those that are guaranteed not to be openable by Bob (those commitments whose decommitment values are perfectly masked by random encodings). Security follows from standard standalone hiding of the commitment scheme.

In the case of a **corrupt Alice** 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 and initializes the commitment scheme in extraction mode (i.e., fixes the coin tossing in step 1 to generate simulated parameters). The simulator does nothing in Step 2 and Step 3. In Step 4, the simulator extracts Alice's commitments of the form $\textsc{Comm}(x; r_x)$ and inserts x' in the bin $\mathcal{B}_X[b]$ at a random unused position $p \in [\mu]$, where $(b, x') = \mathsf{phase}_{m,h}(x)$. Let S denote the set of the μ associated $(tag \mid\mid decommit)$ pairs. If there exists $(T \mid\mid D) \in S$ such that $T = [\![x']\!]^{\text{TAG}}_{b,p}$ and $\textsc{Comm}(x; r_x) = \textsc{Comm}(x; D \oplus [\![x']\!]^{\text{ENC}}_{b,p})$, add x to the set X^*. The simulator sends X^* to the $\mathcal{F}_{\mathsf{PSI}}$ functionality.

We see here that the simulator extracts candidate inputs for Alice by extracting from her commitments. Thus the protocol requires an extractable commitment scheme. This protocol also benefits from restricting Alice to a set of size exactly n item, unlike the dual execution protocol which achieves n items in exception and upper bounded by roughly $n' < 4n$ items. □

6.6 Parameters

Let us now review the protocol as a whole and how to securely set the parameters. The parties first agree on hashing parameters that randomly map their sets of n items into m bins with the use of phasing. The bins are padded with dummy items to size $\mu = O(\log n)$. The parties both act as $\mathcal{F}_{\mathsf{encode}}$ receiver to encode all $m\mu$ items in their bins, including dummy items. Each bin position uses a unique $\mathcal{F}_{\mathsf{encode}}$ session. For all non-dummy encodings, both parties compute $\mu = O(\log n)$ common encodings. If an item is in the intersection, exactly one of these μ encodings will be the same for both parties. Alice then sends Bob all of these common encodings in a random order (not by bins). Bob is able to identify the matching encodings and infer the intersection.

By applying a bins into balls analysis, it can be seen that for m bins and n balls, the probability of there existing a bin with more than μ items is $\leq m \sum_{i=\mu+1}^{n} \binom{n}{i} \left(\frac{1}{m}\right)^i \left(1 - \frac{1}{m}\right)^{n-i}$. Bounding this to be negligible in the security parameter gives the required bin size for a given n, m. By setting $m = O(n/\log n)$ and minimizing the overall cost, we obtain the set of parameters specified in Figure 5 with statistical security $\lambda = 40$. We found that $m = n/10$ minimizes the communication for our choices of n at the expense of increased computation when compared to $m = n/4$. As such, we choose $m = n/10$ in the WAN setting where communication is the dominant cost and $m = n/4$ in the LAN setting where computation has increased importance.

6.7 Discussion

Challenges of Two Party Output An obvious question is whether our protocol be extended to support two party output. In the semi-honest case, this is trivial, since the party who learns the intersection first can simply report it to the other. In the malicious setting, the parties cannot be trusted to relay this information faithfully.

Set size n		2^8	2^{12}	2^{16}	2^{20}	2^{24}
LAN	μ	24	25	26	28	29
	m	64	1024	16384	262144	4194304
WAN	μ	40	43	45	47	49
	m	25	409	6553	104857	1677721

Figure 5: Hashing parameters μ, m for statistical security $\lambda = 40$.

A natural idea to solve this problem is to have Bob send all of his encodings to Alice, making the protocol completely symmetric. We briefly describe the problem with this approach. Suppose Bob behaves honestly with input set Y throughout most of the protocol. Let $y_0 \in Y$ be a distinguished element. In the last step, he sends his common encodings to Alice, but replaces all the encodings corresponding to y_0 with random values.

Now Bob will learn $X \cap Y$, but his effect on Alice will be that she learns only $X \cap (Y \setminus \{y_0\})$. More generally, a malicious Bob can always learn $X \cap Y$ but cause Alice to receive output $X \cap Y'$ for any $Y' \subseteq Y$ of Bob's choice.

7 Performance Evaluation

We have implemented several variants of out main protocol, and in this section we report on its performance. We denote our dual execution random-oracle protocol as DE-ROM and the encode-commit random-oracle protocol as EC-ROM. Only the dual execution protocol was implemented in the standard model and denoted as SM. We do not implement the encode-commit protocol in the standard model due to the communication overhead of standard model commitments such as [FJNT16], see 7.1 *Communication Cost*. All implementations are freely available at `github.com/osu-crypto/libPSI`.

We give detailed comparisons to two leading malicious-secure PSI protocols: our previous Bloom-filter-based protocol [RR17] and the Diffie-Hellman-based protocol of De Cristofaro, Kim & Tsudik [DKT10]. We utilized the implementation provided by [RR17] of that protocol and [DKT10]. All implementations were compared on the same hardware.

Implementation Details & Optimizations. We implemented our protocol in C++ and both the standard-model and random-oracle instantiation of $\mathcal{F}_{\mathsf{encode}}$, to understand the effect of the random-oracle assumption on performance.

We implement $\mathcal{F}_{\mathsf{encode}}$ by directly utilizing [OOS16] in the ROM model or with several chosen message 1-out-of-2 OTs [KOS15] in the standard model as specified by Section 4. When we instantiate $\mathcal{F}_{\mathsf{encode}}$ with [OOS16], we use the BCH-$(511, 76, 171)$ linear code. As such, the $\mathcal{F}_{\mathsf{encode}}$ input domain is $\{0,1\}^{76}$. To support PSI over arbitrary length strings in the random-oracle model, we use the *hash to smaller domain* technique of [PSZ16] in conjunction with phasing. The hashed elements are 128 bits. This enables us to handle sets of size n such that $76 \geq \lambda + \log n$, e.g. $n = 2^{36}$ with $\lambda = 40$ bits of statistical security. For larger set sizes and/or security level, a larger BCH code can be used with minimal additional overhead. In the standard model, we perform PSI over strings of length 32 and 64 bits due to hash to smaller domain requiring the random-oracle to extract.

We used SHA1 as the underlying hash function, and AES as the underlying PRF/PRG (counter mode for a PRG) where needed. The random-oracle instantiation requires the OT-extension hash function to be modeled as a random-oracle. We optimize the $\mathcal{F}_{\mathsf{encode}}$ instantiations by not hashing dummy items.

Setting	Protocol	Set size n										
		2^8		2^{12}		2^{16}		2^{20}		2^{24}		asymptotic
		Total	Online	Total	Online	Total	Online	Total	Online	Total	Online	
LAN	[KKRT16]*	0.19	0.19	0.21	0.21	0.4	0.4	3.8	3.8	59	59	$4n$ (RO)
	[DKT10]	1.6	1.6	22.4	22.4	365	365	5630	5630	–	–	$6n$ (PK)
	[RR17]	0.21	0.002	0.8	0.03	9.6	0.7	148	16	–	–	$4\kappa n$ (RO)
	Ours (EC-ROM)	0.13	0.004	0.19	0.06	0.94	0.69	12.6	11.3	239	218	$4n \log n$ (RO)
	Ours (DE-ROM)	0.13	0.006	0.23	0.08	1.3	1.0	18	16	296	261	
	Ours (SM, $\sigma = 32$)	0.15	0.018	0.48	0.19	3.5	1.8	56	31	–	–	$6\sigma n$ (CRH)
	Ours (SM, $\sigma = 64$)	0.19	0.034	0.84	0.31	8.0	3.7	134	35	–	–	
WAN	[KKRT16]*	0.56	0.56	0.59	0.59	1.3	1.3	7.5	7.5	107	106	$4n$ (RO)
	[DKT10]	1.7	1.7	23.2	23.2	367	367	5634	5634	–	–	$6n$ (PK)
	[RR17]	0.97	0.14	5.3	0.95	69	13	1080	216	–	–	$4\kappa n$ (RO)
	Ours (EC-ROM)	0.67	0.26	1.5	1.1	16	15	255	254	3208	3194	$4n \log n$ (RO)
	Ours (DE-ROM)	0.90	0.33	1.2	0.63	6.3	5.6	106	105	2647	2626	
	Ours (SM, $\sigma = 32$)	1.3	0.11	8.0	0.56	78	5.4	1322	115	–	–	$6\sigma n$ (CRH)
	Ours (SM, $\sigma = 64$)	1.9	0.14	16.8	0.74	226	82	3782	164	–	–	

Figure 6: Single-threaded running time in seconds of our protocol compared to semi-honest [KKRT16] and malicious [DKT10, RR17]. We report both the total and online running time. DE-ROM, EC-ROM respectively denotes our dual execution and encode-commit model protocols. SM denotes the standard model dual execution variant on input bit length σ. Cells with – denote trials that either ran out of memory or took longer than 24 hours. (PK) denotes public key operations, (RO) denotes random oracle operations and (CRH) denotes correlation robust hash function operations. * [KKRT16] is a Semi-Honest secure PSI protocol. We show the [KKRT16] performance numbers here for comparison purposes.

Threads	Protocol	Set size n				
		2^8	2^{12}	2^{16}	2^{20}	2^{24}
4	[DKT10]	0.61	6.9	95	1539	24948
	[RR17]	0.15	0.52	5.8	84	–
	Ours (EC-ROM)	0.14	0.15	0.4	4.4	72
	Ours (DE-ROM)	0.14	0.17	0.6	7.0	93
	Ours (SM, $\sigma = 32$)	0.14	0.24	1.3	17	–
	Ours (SM, $\sigma = 64$)	0.15	0.37	2.7	40	–
16	[DKT10]	0.33	2.2	29	458	7265
	[RR17]	0.15	0.44	4.3	68	–
	Ours (EC-ROM)	0.14	0.16	0.4	3.0	42
	Ours (DE-ROM)	0.14	0.17	0.4	3.5	34
	Ours (SM, $\sigma = 32$)	0.14	0.18	0.6	7.5	–
	Ours (SM, $\sigma = 64$)	0.15	0.25	1.1	14.7	–
64	[DKT10]	0.11	1.2	19	315	5021
	[RR17]	0.14	0.34	2.1	32	–
	Ours (EC-ROM)	0.14	0.15	0.4	3.0	42
	Ours (DE-ROM)	0.14	0.17	0.4	2.9	25
	Ours (SM, $\sigma = 32$)	0.14	0.18	0.5	6.0	–
	Ours (SM, $\sigma = 64$)	0.15	0.21	1.0	14	–

Figure 7: Total running times in seconds of our protocol compared to [DKT10, RR17] in the multi-threaded setting. Cells with – denote trials that ran out of memory.

The implementation of [DKT10] uses the Miracl elliptic curve library using Curve 25519 achieving 128 bit computational security. It is in the random-oracle model and is optimized with the Fiat-Shamir sigma proofs. This implementation also takes advantage of the Comb method for fast exponentiation (point multiplication) with the use of precomputed tables. The [DKT10] protocol requires two rounds of communication over which $5n$ exponentiations and $2n$ zero knowledge proofs are performed. To increase performance on large set sizes, all operations are performed in a

	set size n					asymptotic	
	2^8	2^{12}	2^{16}	2^{20}	2^{24}	Offline	Online
[KKRT16]*	0.04	0.53	8	127	1956	$2\kappa^2$	$3n(\beta+\kappa)$
[DKT10]	0.05	0.8	14	213	2356	0	$6n\phi + 6\phi + n\beta$
[RR17]	1.9	23	324	4970	–	$2\kappa^2 + 2n\kappa^2$	$2n\kappa \log_2(2n\kappa) + n\beta$
Ours (EC-ROM)	0.29	4.8	79	1322	22038	$2\kappa^2$	$3\kappa n + n(C + D\log n + \log^2 n)$
Ours (DE-ROM)	0.25	3.5	61	1092	17875	$2\kappa^2$	$6\kappa n + \beta n \log n$
Ours (SM, $\sigma = 32$)	2.3	40	451	7708	–	$2\kappa^2 + 6\sigma\kappa n$	$\sigma n + \beta n \log n$
Ours (SM, $\sigma = 64$)	5.3	92	1317	22183	–		

Figure 8: The empirical communication cost for both parties when configured for the WAN setting, listed in megabytes. Asymptotic costs are in bits. $\phi = 283$ is the size of the elliptic curve elements. $\beta \approx \lambda + 2\log n - 1$ bits is the size of the final masks that each protocol sends. $C \approx 2\kappa$ bits is the communication of performing one commitment and $D \approx \kappa$ is the size of a non-interactive decommitment.

streaming manner, where data is sent as soon as it is ready.

The [RR17] implementation is also highly optimized including techniques such as hashing OTs on demand and aggregating several steps in their cut and choose. To ensure a fair comparison, we borrow many of their primitives such as SHA1 and AES.

Experimental Setup. Benchmarks were performed on a server equipped with 2 multi-core Intel Xeon processors and 256GB of RAM. The protocol was executed with both parties running on the same server, communicating through the loopback device. Using the Linux `tc` command we simulated two network settings: a LAN setting with 10 Gbps and less than a millisecond latency; and a WAN setting with 40 Mbps throughput and 80ms round-trip latency.

All evaluations were performed with computational security parameter $\kappa = 128$ and statistical security $\lambda = 40$. We consider the sets of size $n \in \{2^8, 2^{12}, 2^{16}, 2^{20}, 2^{24}\}$. The times reported are the average of 10 trials. Where appropriate, all implementations utilize the hardware accelerated AES-NI instruction set.

7.1 Results & Discussion

Execution time, single-threaded. Figure 6 shows the running time of our protocol compared with [DKT10] and [RR17] when performed with a single thread per party. We report both the total running time and the *online time*, which is defined as the portion of the running time that is input-dependent (i.e., the portion of the protocol that cannot be pre-computed).

Our experiments show that our ROM protocols' total running times are significantly less than the prior works, requiring 12.6 seconds to perform a set intersection for $n = 2^{20}$ elements in the LAN setting. A 11.7× improvement in running time compared to [RR17] and a 447× improvement over [DKT10]. Increasing the set size to $n = 2^{24}$, we find that our best protocol takes 239 seconds, whereas [RR17] runs out of memory, and [DKT10] requires over 24 hours. When considering the smallest set size of $n = 2^8$, our protocol remains the fastest with a running time of 0.13 seconds compared to 0.21 and 1.6 for [RR17] and [DKT10] respectively. Our standard model dual execution protocol is also faster than prior works when evaluated in the LAN setting, with a running time 2.6× faster than [RR17] for $\sigma = 32$ and 1.1× faster for $\sigma = 64$.

Our ROM protocol also scales very well in the WAN setting where bandwidth and latency are constrained. For set size $n = 2^8$, all protocols require roughly 1 second with ours being slightly faster at 0.9 seconds. When increasing to a set size of $n = 2^{20}$ the difference becomes much more significant. Our DE-ROM protocol requires 106 seconds compared to 1080 for [RR17], a 10× improvement. Our standard model protocol also has a fast online phase in the WAN setting due to the implementation moving a larger portion of the work to the offline as compared to the ROM protocol.

Multi-threaded performance. Figure 7 shows the total running times in the multi-threaded LAN setting. We see that our protocol parallelizes well, due to the fact that items are hashed into bins which can be processed more or less independently. By contrast [RR17] uses a global Bloom filter representation for all items, which is less amenable to parallelization. For inputs of size $n = 2^{20}$ and 16 threads, our protocol is 23$\times$ faster than [RR17] and 153$\times$ that of [DKT10]. Increasing the number of threads from 1 to 16 speeds up our protocol by a factor of 5$\times$, but theirs by a factor of only 2$\times$.

While the Diffie-Hellman-based protocol of [DKT10] is easily the most amenable to parallelization (16 threads speeding up the protocol by a factor of 12.3$\times$ for $n = 2^{20}$), its reliance on expensive public-key computations leaves it still much slower than ours.

Communication cost. Figure 8 reports both the empirical and asymptotic communication overhead of the protocols. The most efficient protocol with respect to communication overhead is [DKT10]. The dominant term in their communication is to have each party send $3n$ field elements. The next most efficient is our DE-ROM protocol, requiring each party to send $O(n)$ encodings from $\mathcal{F}_{\mathsf{encode}}$. Concretely, for a set size of $n = 2^{20}$, our protocol requires 1.1 GB of communication, roughly 5$\times$ greater than [DKT10]. However, on a modest connection of 40 Mbps, we find our protocol to remain the fastest even when [DKT10] utilizes many threads. In addition, our protocol requires almost 5$\times$ less communication than [RR17] (4.9GB).

When comparing our two ROM protocols, it can be seen that the dual execution technique requires less communication and is therefore faster in the WAN setting. The main overhead of the encode-commit protocol is the $O(n \log n)$ $\quad tag||decommitment$ values that must be sent. This is of particular concern in the standard model where commitments are typically several times larger than their ROM counterparts. In contrast, the dual execution protocol sends $O(n \log n)$ encodings which can be less than half the size of a ROM decommitment.

One aspect of the protocols that is *not* reflected in the tables is how the communication cost is shared between the parties. In our DE-COM protocol, a large portion of the communication is in the encoding steps, which are entirely symmetric between the two parties. In [RR17] the majority of the communication is done by the receiver (in the OT extension phase). Although the total communication cost of [RR17] is roughly 5$\times$ that of our protocol, the communication cost to the receiver is $\sim 10\times$ ours.

Comparison with [RR17]. We provide a more specific comparison to the protocol of Rindal & Rosulek [RR17]. Both protocols are secure against malicious adversaries; both rely heavily on efficient oblivious transfers; neither protocol strictly enforces the size of a malicious party's input set (so both protocols realize the slightly relaxed PSI functionality of Figure 1).

We now focus on our random-oracle-optimized protocol, which uses the random-oracle instantiation of $\mathcal{F}_{\mathsf{encode}}$. As has been shown, this protocol is significantly faster than that of [RR17]. We give a rough idea of why this should be the case. In [RR17], the bulk of the cost is that the parties perform an OT for each bit of a Bloom filter. With n items, the size of the required Bloom filter is $\sim kn$, where k is the security parameter of the Bloom filter. For technical reasons, k in [RR17] must be the computational security parameter of the protocol (*e.g.*, 128 in the implementation). Overall, roughly $\sim nk$ oblivious transfers are required.

The bulk of the cost in our protocol is performing the instances of $\mathcal{F}_{\mathsf{encode}}$. In our random-oracle instantiation, we realize $\mathcal{F}_{\mathsf{encode}}$ with the OT-extension protocol of [OOS16]. Each instance of $\mathcal{F}_{\mathsf{encode}}$ has cost roughly comparable to a plain OT. Our protocol requires $m\mu = O(n)$ such instances. It is this difference in the number of OT primitives that contributes the largest factor to the difference in performance between these two protocols.

We also observe that our standard model protocol is faster than [RR17] in the LAN setting for $\sigma = 32$ and $\sigma = 64$. While it is true that [RR17] only weakly depends on σ, it is still informative that our protocol remains competitive with the previous fastest protocol while eliminating the random-oracle assumption. When considering the WAN setting, the communication overhead of $\sigma m\mu = O(\sigma n)$ OTs limits our performance, resulting in $\sigma = 32$ being slightly slower than [RR17].

Comparison with OPE protocols. Our protocol is orders of magnitude faster than blind-RSA based protocol of [DKT10], due to [DKT10] performing $O(n)$ exponentiations. Traditional OPE-based PSI also require $O(n)$ exponentiations and their running time would be similarly high. There are very recent OPE protocols based on OT but they still require $O(n)$ OTs plus $O(n/\kappa)$ relatively expensive interpolations of degree-$O(k)$ polynomials, totaling $O(n \log \kappa)$ operations. In contrast our protocol requires $O(n)$ OTs to be communicated and $O(n \log n)$ local OT computations.

Comparison with semi-honest PSI. An interesting point of comparison is to the state-of-the-art semi-honest secure protocol of Kolesnikov et al. [KKRT16] which follows the same PSZ paradigm. Figure 6 shows the running time of our protocol compared to theirs. For sets sizes up to $n = 2^{12}$ our protocol is actually faster than [KKRT16] in the LAN setting which we attribute to a more optimized implementation. Increasing the set size to $n = 2^{20}$ we see that our protocol require 12.6 seconds compared to 3.8 by [KKRT16], a 3.3× difference. For the largest set size of $n = 2^{24}$ we see the difference increase further to a 4× overhead in the LAN setting. In the WAN setting we see a greater difference of 25× which we attribute to the $\log n$ factor more communication/computation that our protocol requires.

References

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In Sadeghi et al. [SGY13], pages 535–548.

[ANS10] Yuriy Arbitman, Moni Naor, and Gil Segev. Backyard Cuckoo hashing: Constant worst-case operations with a succinct representation. In *51st FOCS*, pages 787–796. IEEE Computer Society Press, October 2010.

[BPSW07] Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 498–507, 2007.

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

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In Sadeghi et al. [SGY13], pages 789–800.

[DKT10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 213–231. Springer, Heidelberg, December 2010.

[DMRY09] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS 09*, volume 5536 of *LNCS*, pages 125–142. Springer, Heidelberg, June 2009.

[FHNP14] Michael J. Freedman, Carmit Hazay, Kobbi Nissim, and Benny Pinkas. Efficient set-intersection with simulation-based security. In *In Journal of Cryptology*, 2014.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 303–324. Springer, Heidelberg, February 2005.

[FJNT16] Tore Kasper Frederiksen, Thomas P. Jakobsen, Jesper Buus Nielsen, and Roberto Trifiletti. On the complexity of additively homomorphic UC commitments. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 542–565. Springer, Heidelberg, January 2016.

[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.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols?, 2012.

[HFH99] Bernardo A. Huberman, Matt Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *In Proc. of the 1st ACM Conference on Electronic Commerce*, pages 78–86. ACM Press, 1999.

[HN10] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. In *In IACR PKC*, 2010.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003.

[KK13] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 54–70. Springer, Heidelberg, August 2013.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 818–829. ACM, 2016. http://doi.acm.org/10.1145/2976749.2978381.

[KMRS14] Seny Kamara, Payman Mohassel, Mariana Raykova, and Seyed Saeed Sadeghian. Scaling private set intersection to billion-element sets. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 195–215. Springer, Heidelberg, March 2014.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 724–741. Springer, Heidelberg, August 2015.

[Lam16] Mikkel Lambæk. Breaking and fixing private set intersection protocols. Master's thesis, Aarhus University, 2016. https://eprint.iacr.org/2016/665.

[Mar14] Moxie Marlinspike. The difficulty of private contact discovery, 2014. Blog post, whispersystems.org/blog/contact-discovery.

[Mea86] C. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *1986 IEEE Symposium on Security and Privacy*, pages 134–134, April 1986.

[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.

[NPS99] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *EC*, pages 129–139, 1999.

[OOS16] Michele Orrù, Emmanuela Orsini, and Peter Scholl. Actively secure 1-out-of-n OT extension with application to private set intersection. In *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, pages 381–396, 2016. http://eprint.iacr.org/2016/933.

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Security Symposium, USENIX Security 15*, pages 515–530. USENIX Association, 2015. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/pinkas.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on ot extension. In *Proceedings of the 23rd USENIX Conference on Security Symposium*, SEC'14, pages 797–812, Berkeley, CA, USA, 2014. USENIX Association. https://www.usenix.org/node/184446.

[PSZ16] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. Cryptology ePrint Archive, Report 2016/930, 2016. http://eprint.iacr.org/2016/930.

[RR17] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017,* *Proceedings, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 235–259, 2017.

[SGY13] Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors. *ACM CCS 13*. ACM Press, November 2013.

[TKC07] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Utku Celik. Privacy preserving error resilient dna searching through oblivious automata. In *Proceedings of the 2007 ACM Conference on Computer and Communications Security, CCS 2007, Alexandria, Virginia, USA, October 28-31, 2007*, pages 519–528, 2007.

A Commitment Properties

The encode-commit variant of our protocol requires a non-interactive commitment scheme. The syntax is as follows:

- $\textsc{Setup}(1^\kappa)$: samples a random reference string crs.
- $\textsc{Comm}(crs, x, r)$: generates a commitment to x with randomness r. Note that in the main body, we omit the global argument crs.
- $\textsc{SimSetup}(1^\kappa)$: samples a reference string crs along with a trapdoor τ.
- $\textsc{Extract}(crs, \tau, c)$: extracts the committed plaintext value from a commitment c.

We require the scheme to satisfy the following security properties:

Definition 4. *A commitment scheme is secure if the following are true:*

1. *(Extraction:) Define the following game:*

```
ExtractionGame(1^κ, A):
    $(crs, \tau) \leftarrow \textsc{SimSetup}(1^\kappa)$
    $(c, x', r') \leftarrow \mathcal{A}(crs)$
    if $c = \textsc{Comm}(crs, x', r')$ and $x' \neq \textsc{Extract}(crs, \tau, c)$:
        return 1
    else: return 0
```

The scheme has straight-line extraction if for every PPT $\mathcal{A}$, $\mathsf{ExtractionGame}(1^\kappa, \mathcal{A})$ outputs 1 with negligible probability.

2. *(Hiding:) Define the following game:*

```
HidingGame(1^κ, A, b):
    $crs \leftarrow \textsc{Setup}(1^\kappa)$
    $(x_0, x_1) \leftarrow \mathcal{A}(crs)$
    $r \leftarrow \{0,1\}^\kappa$
    return $\textsc{Comm}(crs, x_b, r)$
```

The scheme is hiding if, for all PPT $\mathcal{A}$, the distributions $\mathsf{HidingGame}(1^\kappa, \mathcal{A}, 0)$ and $\mathsf{HidingGame}(1^\kappa, \mathcal{A}, 1)$ are indistinguishable.

The definitions are each written in terms of a single commitment, but they apply simultaneously to many commitments using a simple hybrid argument.

In the non-programmable random oracle model, the classical commitment scheme $\textsc{Comm}(x, r) = H(x\|r)$ satisfies these definitions. In the standard model, one can use any UC-secure non-interactive commitment scheme, e.g., the efficient scheme of [FJNT16].

B Formal Encode-Commit Protocol

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

1. **[Parameters]** Parties agree on a random hash function $h : \{0,1\}^\sigma \to [m]$ and global parameters for the commitment scheme, using a coin tossing protocol.
2. **[Hashing]**
   (a) For $x \in X$, Alice computes $(b, x') = \mathsf{phase}_{h,m}(x)$ and adds x' to bin $\mathcal{B}_X[b]$ at a random unused position $p \in [\mu]$.
   (b) For $y \in Y$, Bob computes $(b, y') = \mathsf{phase}_{h,m}(y)$ and adds y' to bin $\mathcal{B}_Y[b]$ at a random unused position $p \in [\mu]$.

   Both parties fill unused bin positions with the zero string.
3. **[Encoding]** For bin index $b \in [m]$ and position $p \in [\mu]$: Let y' be the value in bin $\mathcal{B}_Y[b]$ at position p. Bob sends $(\textsc{Encode}, (\mathsf{sid}, \mathsf{A}, b, p), y')$ to the $\mathcal{F}_{\mathsf{encode}}$ functionality which responds with $(\textsc{Output}, (\mathsf{sid}, \mathsf{A}, b, p), [\![y']\!]^{\mathsf{A}}_{b,p})$. Alice receives $(\textsc{Output}, (\mathsf{sid}, \mathsf{A}, b, p))$ from $\mathcal{F}_{\mathsf{encode}}$. Bob computes

   $$[\![y']\!]^{\text{TAG}}_{b,p} = \text{PRF}([\![y']\!]^{\mathsf{A}}_{b,p}, \text{TAG})$$
   $$[\![y']\!]^{\text{ENC}}_{b,p} = \text{PRF}([\![y']\!]^{\mathsf{A}}_{b,p}, \text{ENC})$$

   and constructs a hash table H mapping $[\![y']\!]^{\text{TAG}}_{b,p}$ to $([\![y']\!]^{\text{ENC}}_{b,p}, y)$.
4. **[Output]** For each $x \in X$, in random order, let b, p be the bin index and position that x' was placed in during Step 2a to represent x. For $j \in [\mu]$, Alice sends $(\textsc{Encode}, (\mathsf{sid}, \mathsf{A}, b, j), x')$ to $\mathcal{F}_{\mathsf{encode}}$ and receives $(\textsc{Output}, (\mathsf{sid}, \mathsf{A}, b, j), [\![x']\!]^{\mathsf{A}}_{b,j})$ in response. For each response Alice computes $[\![x']\!]^{\text{TAG}}_{b,j} = \text{PRF}([\![x']\!]^{\mathsf{A}}_{b,j}, \text{TAG})$ and $[\![x']\!]^{\text{ENC}}_{b,j} = \text{PRF}([\![x']\!]^{\mathsf{A}}_{b,j}, \text{ENC})$.

   For each x Alice sends the tuple

   $$\textsc{Comm}(x; r_x), \quad \{[\![x']\!]^{\text{TAG}}_{b,j} \mid\mid [\![x']\!]^{\text{ENC}}_{b,j} \oplus r_x \mid j \in [\mu]\}$$

   to Bob who outputs the union of all y such that $\exists j : [\![x']\!]^{\text{TAG}}_{b,j} \in H.keys$ and $\textsc{Comm}(x, r_x) = \textsc{Comm}(y; ([\![x']\!]^{\text{ENC}}_{b,j} \oplus r_x) \oplus [\![y']\!]^{\text{ENC}}_{*})$ where $([\![y']\!]^{\text{ENC}}_{*}, y) := H[[\![x']\!]^{\text{TAG}}_{b,j}]$.

Figure 9: Our Encode-Commit PSI protocol.