SpOT-Light: Lightweight Private Set Intersection from Sparse OT Extension

Benny Pinkas* Mike Rosulek† Ni Trieu† Avishay Yanai*

June 2, 2019

Abstract

We describe a novel approach for two-party private set intersection (PSI) with semi-honest security. Compared to existing PSI protocols, ours has a more favorable balance between communication and computation. Specifically, our protocol has the *lowest monetary cost* of any known PSI protocol, when run over the Internet using cloud-based computing services (taking into account current rates for CPU + data). On slow networks (e.g., 10Mbps) our protocol is actually the *fastest*.

Our novel underlying technique is a variant of oblivious transfer (OT) extension that we call *sparse OT extension*. Conceptually it can be thought of as a communication-efficient multipoint oblivious PRF evaluation. Our sparse OT technique relies heavily on manipulating high-degree polynomials over large finite fields (i.e. elements whose representation requires hundreds of bits). We introduce extensive algorithmic and engineering improvements for interpolation and multi-point evaluation of such polynomials, which we believe will be of independent interest.

Finally, we present an extensive empirical comparison of state-of-the-art PSI protocols in several application scenarios and along several dimensions of measurement: running time, communication, peak memory consumption, and — arguably the most relevant metric for practice — monetary cost.

1 Introduction

Private set intersection (PSI) allows two parties, who each hold a set of items, to learn the intersection of their sets without revealing anything else about the items. PSI has many privacy-preserving applications: e.g., private contact discovery [CLR17, RA17, DRRT18][1], DNA testing and pattern matching [TKC07], remote diagnostics [BPSW07], record linkage [HMFS17], and measuring the effectiveness of online advertising [IKN+17]. Over the last several years PSI has become truly practical with extremely fast implementations [CKT10, CGT12, DCW13, PSZ14, KKRT16, KMP+17, RR17, CLR17, HV17, RA17, GNN17a, CCS18] that can process millions of items in seconds.

In this paper we focus on **two-party** PSI with **semi-honest** security (with one variant of our protocol achieving malicious security for one party). While we describe our protocols in terms of any number of items, our evaluation focuses on the case where the two parties have **sets of the same size.** We discuss the setting of unequal set sizes in Appendix F.

1.1 What Should We Value in a PSI Protocol?

The standard ways to measure the cost of a protocol are running time and communication. Depending on which of these metrics is prioritized, a different protocol will be preferred.

*Bar-Ilan University. benny@pinkas.net, ay.yanay@gmail.com. Supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office, and by a grant from the Israel Science Foundation.

†Oregon State University. {rosulekm,trieun}@oregonstate.edu. Partially supported by NSF award 1617197, a Google faculty award, and a Visa faculty award.

[1] See also https://whispersystems.org/blog/contact-discovery/

Minimizing time. The fastest known PSI protocols are all based on efficient oblivious transfers (OT). The idea is to reduce the PSI computation to many instances of oblivious transfer. This approach is the fastest because modern OT extension protocols [Bea96, IKNP03, KK13, ALSZ13] use only a small (fixed) number of public-key operations (e.g., elliptic curve multiplications) but otherwise use only *cheap symmetric-key operations*. The approach to PSI was introduced by Pinkas et al. [PSZ14] and refined in a sequence of works [PSSZ15, KKRT16]. The state-of-the-art protocol [KKRT16] computes an intersection of million-item sets in about 4 seconds.

Minimizing communication. To the best of our knowledge, the PSI protocol with lowest communication in this setting is due to Ateniese et al. [ACT11]. This protocol requires communication that is only marginally more than a naïve and *insecure* protocol (in which one party sends just a short hash of each item), and also has the nice property of hiding the size of the input set. However, the protocol requires at least $n \log n$ RSA exponentiations (for PSI of n items). These requirements make the protocol prohibitively expensive in practice.[2]

A more popular (as well as the earliest) approach to low-communication PSI is based on the commutative property of Diffie-Hellman key agreement (DH-PSI), and appears in several works [Sha80, Mea86, HFH99]. The idea is for the parties to compute the intersection of $\{(H(x)^\alpha)^\beta) \mid x \in X\}$ and $\{(H(y)^\beta)^\alpha) \mid y \in Y\}$ in the clear, where α and β are secrets known by Alice and Bob, respectively. The DH-PSI protocol strikes a more favorable balance between communication and computation than the RSA-based protocol. It requires n exponentiations in a Diffie-Hellman group, which are considerably cheaper than RSA exponentiations but considerably more expensive than the symmetric-key operations used in OT extension. In terms of communication, it requires less than 3 group elements per item. When instantiated with compact elliptic curve groups (ECDH-PSI), the communication complexity is very small. For example, Curve25519 [Ber06] provides 128-bit security with only 256-bit group elements (around 600 bits of communication per item).

An ideal balance. Communication cost and overall running time are clearly both important, but which metric best reflects the balance between the two costs, and the true suitability of a protocol for practice? We argue that the most appropriate metric which balances the two costs is the **monetary cost to run the protocol on a cloud computing service.** First, a typical real-world application of PSI is likely to use such a service rather than in-house computing. Second, the pricing model of such services already takes into account the difference in cost to send a bit vs. perform a CPU clock cycle.

1.2 Our Contributions

We present a new PSI protocol paradigm that is secure against semi-honest adversaries under standard-model assumptions. We offer two variants of our protocol: one is optimized for low communication and the other for fast computation. The variant that is optimized for low communication is also secure against a malicious sender in the (non-programmable) random oracle model.

Better Balance of Computation and Communication. Compared to DH-PSI and RSA-based PSI [ACT11], both of our protocol variants have much faster running time, since ours are based on OT extension (i.e., dominated by cheap symmetric-key operations). The low-communication variant has smaller communication overhead than DH-PSI (even on a 256-bit elliptic curve) while the fast-computation variant has about the same communication cost as DH-PSI.

Compared to [KKRT16], both of our protocol variants require much less communication. Our protocols perform more computation in the form of finite field operations, making our protocols slower over high-bandwidth networks. However, the variant optimized for fast computation has a competitive running time and is the fastest over low-bandwidth networks (e.g., 30Mbps and less).

Extensive Cost Comparison. In Section 6 we perform an extensive benchmark of state-of-the-art PSI protocols for various set sizes and bandwidth configurations. To the best of our knowledge, our analysis is the first to assess PSI protocols in terms of their monetary costs. Our experiments show that in *all* settings

[2] We are not aware of any prior implementation of this protocol, but estimated the running time through benchmark RSA exponentiations. For the set sizes we consider in this work, the protocol would require many hours or even a day.

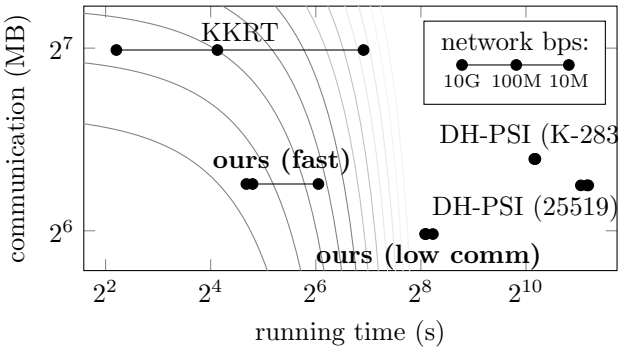


Figure 1: Communication and running time for different PSI protocols, with $n = 2^{20}$ items, on 3 network configurations. Curved lines are lines of equal monetary cost on a representative AWS configuration (see Section 6).

we considered, the fast variant of our protocol has the **least monetary cost** of all protocols — up to 40% less in some cases. A summary of the state of the art (including this work) is depicted in Figure 1.

Sparse OT extension technique. Our main technique, which we call *sparse OT extension*, is a novel twist on oblivious transfer (OT) extension. Roughly speaking, the idea allows the receiver to obliviously pick up a chosen subset of k out of N random secrets (where N may be exponential), with communication cost proportional only to k.

The concept is similar to an oblivious PRF [FIPR05] on which the receiver can evaluate k chosen points. Other PSI protocols like [PSSZ15, KKRT16] can also be expressed as a construction of OPRF from OT extension. However, these involve an OPRF that the receiver can evaluate *on only a single value*, resulting in significantly more effort to build PSI. This qualitative difference in OPRF flavor is the main source of our performance improvements.

New hashing techniques. It is common in PSI literature to assign items randomly to bins, and then perform a PSI within each bin. For security reasons, it is necessary to add dummy items to each bin. With existing techniques, dummy items account for 20-80% of the protocol cost! Our speed-optimized protocol variant is the first to use a kind of 2-choice hashing [SEK03] that requires almost no dummy items (e.g., 2.5%). This 2-choice hashing technique requires placing many items per bin, while previous PSI techniques are only efficient with 1 item per bin (due to their qualitatively different OPRF flavor). Hence, this hashing technique does not immediately benefit existing PSI protocols.

New polynomial interpolation techniques. Our communication-optimized protocol variant requires interpolation and multi-point evaluation of a polynomial, which turns out to be the main bottleneck for the following reasons: (1) The polynomial is over a large field of $\gg 2^{400}$ elements, since the polynomial encodes values related to an underlying OT-extension protocol. (2) The number of interpolation points depend on the parties' set size, which could be in the millions. (3) The best algorithms, which incur $O(n \log n)$ field operations, require a special set of interpolation points, namely, the x-values should be the roots of unity of the field or have a special algebraic structure. In contrast, in the context of our protocol the interpolation points (the x-values) are the parties PSI input items, which are arbitrary. The best algorithms with an arbitrary set of interpolation points incur $O(n \log^2 n)$ field operations [MB72].

We develop and demonstrate new techniques, called *Slice & Stream* and *Subproduct-Tree Reuse*, to speed up the concrete efficiency of these tasks by up to $2\times$ for the special case in which the x and y-coordinates of the points are drawn from the domains $\mathcal{D}_x$ and $\mathcal{D}_y$ where $|\mathcal{D}_x| \ll |\mathcal{D}_y|$. We believe those techniques could have a general interest (even outside of the field of cryptography).

1.3 Related Work and Comparison

We compare our results to relevant related work here, focusing on qualitative differences between the protocols. A quantitative comparison is given later in Section 6.

DH-PSI Our protocol uses less communication than DH-PSI, even when the latter is instantiated with the most compact elliptic curve known. In terms of computation, our protocol uses only symmetric-key

operations (apart from a fixed number of base OTs). Its main computational bottleneck is computing polynomial interpolation, requiring either $O(n \log^2 \lambda)$ or $O(n \log^2 n)$ finite field operations (i.e., multiplications), depending on the variant, where n is the set size and λ is the statistical security parameter. The DH-PSI protocol computes $O(n)$ exponentiations (or elliptic curve multiplications, which are each computed using $\log |\mathbb{G}|$ multiplication operations in the underlying cyclic group $\mathbb{G}$). If we consider the basic unit of computation to be a multiplication in the underlying field/group, then our protocol uses at most $O(n \log^2 n)$ multiplications whereas DH-PSI uses $O(n \log |\mathbb{G}|)$ multiplications. The experiments that we describe in Section 6 demonstrate that our protocol is substantially faster than DH-PSI for all realistic set sizes and on all network configurations.

Our communication-optimized protocol variant has security against one malicious party. In contrast, DH-PSI is not easily adapted to malicious security, even against just one party.[3] In order to harden DH-PSI against malicious parties, the leading protocol of De Cristofaro et al. [CKT10] requires *both parties* to run zero-knowledge proofs involving all of their input items. Thus, even one-sided malicious security requires significant overhead to the semi-honest protocol.

While we do not formally consider security against quantum adversaries, we do point out that our protocol exclusively uses primitives that can be instantiated with post-quantum security (OT, PRFs, and hash functions). DH-PSI on the other hand is trivially broken against quantum adversaries.

Protocols based on an RSA accumulator. The protocol of [ACT11] has a very low communication overhead of roughly $\lambda + \log^2 n$ bits per item, which may even be optimal (even for an insecure protocol). On the other hand, it computes $O(n \log n)$ RSA exponentiations, and as such is slower than DH-PSI by at least an order of magnitude (due to the $\log n$ factor, and to RSA exponentiations being slower than elliptic curve multiplications). Our protocols are substantially faster than both of these protocols (see Section 6). This protocol also requires a random oracle, whereas for semi-honest security ours is in the standard model.

OT-based protocols Our protocol requires 40-50% less communication compared to [KKRT16] and is the fastest over low-bandwidth networks (30 Mbps and lower). Over high-bandwidth networks, even though our protocol is slower than [KKRT16], ours still requires less monetary cost (see Section 6).

Independently, Lambæk [Lam16] and Patra et al. [PSS17] showed how to enhance the protocols of [PSZ14, KKRT16] with a security against a *malicious receiver* with almost no additional overhead. Interestingly, our protocol naturally provides security against a *malicious sender*. In both of these protocols, if the parties have sets of very different sizes then the party with more items should play the role of sender. Providing a different flavor of one-sided malicious security is therefore potentially valuable.

Ghosh and Nilges [GN19] proposed a PSI protocol based on oblivious linear function evaluation (OLE). This protocol requires $2n$ passive OLE invocations, polynomial interpolations at 3 times (one of degree n, and two of degree $2n$), and polynomial evaluation on $2n + 1$ points at 4 times. In terms of communication, the required passive OLE instances [IPS09, GNN17b] require $8(n+1)$ elements sent from the receiver to the sender to create a noisy encoding, and the cost of doing $4n$-out-of-$8(n + 1)$ OT which incurs an overhead of at least $8(n + 1)$ on the number of Correlated OT. Hence, this OLE-based PSI protocol requires at least $8(n+1)(\kappa+2\ell)$ bits communication, where ℓ is bit-length of item. For example, when $\ell = 128$, our protocols show a factor of $4.8 - 6.3\times$ improvement in terms of communication.

Recently, Falk, Noble and Ostrovsky [FNO18] presented a protocol for PSI that achieves linear communication complexity relying on standard assumption (i.e. in the OT-hybrid model, assuming the existence of correlation robust hash and one-way functions) and in the standard model (i.e. without a random oracle). This is in contrast to previous protocols that achieve linear communication but rely on stronger assumptions (like [CKT10, CT12] that are based on the one-more RSA assumption and a random oracle); and to previous OT-based protocols that achive only super-linear communication complexity due to the stash handling. In the protocol of [FNO18], just like previous OT-based protocols, Bob maps his n items to $O(n)$ bins using a Cuckoo hashing, hence, it has at most one item in each bin. Bob also maintains a special bin for items that could not be mapped to the 'regular' bins, this special bin is called the *stash* and it contains $\omega(1)$ items. Alice maps her items to $O(n)$ bins using simple hashing, hence, she has at most $O(\log n / \log \log n)$ items in each bin with high probability. Then, Bob can obtain the intersection between items in its 'regular' bins

[3] The main challenge is that a simulator would have to extract effective inputs $\{x_1, \ldots, x_n\}$ from a corrupt party, seeing only $\{H(x_1)^\alpha, \ldots, H(x_n)^\alpha\}$.

and Alice's set using the BaRK-OPRF technique of [KKRT16] with communication complexity $O(n \cdot \kappa)$ (where κ is the computational security parameter). It remains to compare the items in Bob's stash to all Alice's items; since the stash is of size $\omega(1)$ this comparison would naively require $\omega(n \cdot \kappa)$ communication overall. However, the observation in [FNO18] is that this comparison can be performed using a separate PSI protocol that is specialized for *unbalanced* set sizes in which Alice has much more items than Bob; such a protocol can achieve communication complexity that depends only on the larger set size, therefore, the overall communication complexity of [FNO18] is $O(n \cdot \kappa)$ rather than $\omega(n \cdot \kappa)$. We note that in concurrent to their work, in this paper we achieve the same (linear) communication complexity, under the same standard assumptions and without a random oracle, using a new primitive, namely the *Sparse OT Extension*.

Other paradigms. Other approaches for PSI have been proposed, including ones based on Bloom filters [DCW13] and generic MPC [HEK12]. Pinkas et al. [PSZ14, PSSZ15] performed a comprehensive comparison of semi-honest PSI techniques and found the OT-extension paradigm to strictly dominate others in terms of performance. They found that the best Bloom-filter approach is 2x worse in runtime, 4x worse in communication; best generic-MPC-based approach is 100x worse in runtime and 10x worse in communication. For this reason, we do not include these protocol paradigms in further comparisons.

Asymmetric set sizes. Several recent PSI protocols are optimized specifically for the case of highly asymmetric set sizes. [CLR17, PSSZ15, KLS+17, RA17]. We discuss these protocols in Appendix F.

Other related work. One way of viewing our new technique is that we covertly embed some protocol messages into a polynomial. Similar ideas appear in [MPP10, CDJ16]. In particular, [CDJ16] explicitly propose to embed private equality-test protocol messages into a polynomial, to yield a PSI protocol. Their protocol is based on the DH paradigm, and therefore requires a linear number of exponentiations. They also achieve a stronger *covertness* property (participants cannot distinguish other participants from random noise, until the protocol terminates). In our case, we look inside IKNP OT extension and identify the minimal part of the protocol that needs to be covertly embedded into a polynomial, in order to achieve *standard* (semi-honest or malicious) security.

2 Technical Preliminaries

2.1 Notation

Throughout the paper we use the following notation: We let κ, λ denote the computational and statistical security parameters, respectively. We write $[m]$ to denote a set $\{1, \ldots, m\}$. The notation $d_H(\boldsymbol{x}, \boldsymbol{y},)$ denotes the Hamming distance between bit vectors (strings) $\boldsymbol{x}$ and $\boldsymbol{y}$ of the same length and $w_H(\boldsymbol{x}) = d_H(\boldsymbol{x}, \boldsymbol{0})$ denotes the Hamming weight of $\boldsymbol{x}$. For a bit vector $\boldsymbol{v}$ we let v_i denote the bit in the ith coordinate. If $\boldsymbol{a} = a_1 \| \cdots \| a_p$ and $\boldsymbol{b} = b_1 \| \cdots \| b_p$ are two vectors, the notation $\boldsymbol{a} \oplus \boldsymbol{b}$ denotes the vector $(a_1 \oplus b_1) \| \cdots \| (a_p \oplus b_p)$. Similarly, the notation $\boldsymbol{a} \cdot \boldsymbol{b}$ represents the bitwise-AND of vectors: $(a_1 \cdot b_1) \| \cdots \| (a_p \cdot b_p)$.

2.2 Oblivious Transfer

Oblivious Transfer (OT) is a central cryptographic primitive in the area of secure computation, which was introduced by Rabin [Rab05]. 1-out-of-2 OT [EGL85] refers to the setting where a sender with two input strings $(\boldsymbol{m_0}, \boldsymbol{m_1})$ interacts with a receiver who has a input choice bit b. As the result, the receiver learns $\boldsymbol{m_b}$ without learning anything about $\boldsymbol{m_{1-b}}$, while the sender learns nothing about b. Rabin OT protocol requires expensive public-key operations. In 2003, Ishai et al. [IKNP03] proposed a construction of OT extension (refer as IKNP) that allows a large number of OTs executions at the cost of computing a small number of expensive OTs [NP99]. Later, Kolesnikov and Kumaresan [KK13] improved IKNP for short secrets. It gives $O(\log(\kappa))$ factor performance improvement in communication and computation. In the same year, [ALSZ13] presented several IKNP optimizations and several weaker variants of OT. In Random OT (ROT), the sender's OT inputs $(\boldsymbol{m_0}, \boldsymbol{m_1})$ are chosen at random, therefore, it allows the protocol itself to give him the values $(\boldsymbol{m_0}, \boldsymbol{m_1})$ randomly. With ROT, the bandwidth requirement is significantly reduced since the sender sends nothing to receiver. In our construction, we require this weaker variant, random OT, whose functionality is described in Figure 2.

PARAMETERS: Sender $\mathcal{S}$, receiver $\mathcal{R}$, length κ
FUNCTIONALITY:

- Wait for an input $b \leftarrow \{0,1\}$ from the receiver $\mathcal{R}$.
- Choose $\boldsymbol{m_0}, \boldsymbol{m_1} \leftarrow \{0,1\}^\kappa$, and give both to sender $\mathcal{S}$.
- Give $\boldsymbol{m_b}$ to receiver $\mathcal{R}$.

Figure 2: The $\mathcal{F}_{\mathsf{ROT}}^{\kappa}$ ideal functionality for Random Oblivious Transfer.

2.3 (Hamming) Correlation Robustness

Our PSI construction is proven secure under a *correlation robust* assumption which was introduced for IKNP OT extension [IKNP03] and later generalized in [KKRT16] to the version we use in this work.

Definition 2.1. *[KKRT16] Let H be a function with input length n. Then H is d-**Hamming correlation robust function** (CRF) if, for any $\boldsymbol{a_1}, \ldots, \boldsymbol{a_m}, \boldsymbol{b_1}, \ldots, \boldsymbol{b_m}$ with $\boldsymbol{a_i}, \boldsymbol{b_i} \in \{0,1\}^n$ and $w_H(\boldsymbol{b_i}) \geq d$ for all $i \in [m]$, the following distribution, induced by random sampling of $\boldsymbol{s} \leftarrow \{0,1\}^n$, is pseudorandom:*

$$H(\boldsymbol{a_1} \oplus [\boldsymbol{b_1} \cdot \boldsymbol{s}]), \ldots, H(\boldsymbol{a_m} \oplus [\boldsymbol{b_m} \cdot \boldsymbol{s}])$$

The IKNP protocol uses this assumption with $n = d = \kappa$. In that case, the only valid choice for $\boldsymbol{b_i}$ is 1^κ, and the distribution simplifies to $H(\boldsymbol{a_1} \oplus \boldsymbol{s}), \ldots, H(\boldsymbol{a_m} \oplus \boldsymbol{s})$. In our case, we use $n > d = \kappa$, so other choices for the $\boldsymbol{b_i}$ values are possible.

2.4 Private Set Intersection

PSI is a special case of secure two-party computation, and we use the standard security definitions for two-party computation in this work. The guarantees of PSI are captured in the ideal functionality $\mathcal{F}_{\mathsf{PSI}}$ defined in Figure 3. For security against malicious parties, we use the framework of universal composability (UC) [Can01].

2.5 The IKNP OT Extension: A Reminder

It is well-known that oblivious transfer cannot be obtained from scratch using only symmetric-key primitives [IR88]. OT extension [Bea96] refers to the idea that parties can perform only a small number κ of OTs (using public-key primitives), and then, *using only symmetric-key operations thereafter*, obtain $N \gg \kappa$ effective instances of OT. Modern OT extension protocols follow the overall structure of the IKNP protocol [IKNP03]. In Figure 4 we review the variant of the IKNP protocol where the sender's OT payloads are chosen uniformly.

From the correctness of the base OTs, we have that:

$$\boldsymbol{q_i} = \boldsymbol{t_i} \oplus s_i \cdot (\boldsymbol{t_i} \oplus \boldsymbol{u_i}) = \begin{cases} \boldsymbol{t_i} & \text{if } s_i = 0 \\ \boldsymbol{u_i} & \text{if } s_i = 1 \end{cases}$$

This relationship can be extended across the rows of the $N \times \kappa$ matrices to obtain: $Q(i) = T(i) \oplus \boldsymbol{s} \cdot (T(i) \oplus U(i))$, where $T(i)$ and $U(i)$ correspond to the rows of T and U. Then:

$$\begin{aligned} Q(i) \oplus \boldsymbol{s} \cdot P(i) &= \Big(T(i) \oplus \boldsymbol{s} \cdot (T(i) \oplus U(i))\Big) \oplus \boldsymbol{s} \cdot (T(i) \oplus U(i) \oplus C(i)) \\ &= T(i) \oplus \boldsymbol{s} \cdot C(i) = \begin{cases} T(i) & \text{if } r_i = 0 \\ T(i) \oplus \boldsymbol{s} & \text{if } r_i = 1 \end{cases} \end{aligned}$$

From this we can deduce that Bob's output is $\boldsymbol{m_i^*} = \boldsymbol{m_{i,r_i}}$, whereas $\boldsymbol{m_{i,1-r_i}} = H(T(i) \oplus \boldsymbol{s})$. From the correlation-robust property of H, this value is pseudorandom from Bob's perspective.

PARAMETERS: Sender $\mathcal{S}$, receiver $\mathcal{R}$, set sizes n_1, n_2.
FUNCTIONALITY:

- Wait for input $X = \{x_1, \ldots, x_{n_1}\} \subseteq \{0,1\}^*$ from sender $\mathcal{S}$.
- Wait for input $Y = \{y_1, \ldots, y_{n_2}\} \subseteq \{0,1\}^*$ from receiver $\mathcal{R}$.
- Give output $X \cap Y$ to receiver $\mathcal{R}$.

Figure 3: PSI ideal functionality $\mathcal{F}_{\mathsf{PSI}}$.

PARAMETERS:

- A PRG $G : \{0,1\}^\kappa \to \{0,1\}^N$
- A κ-Hamming CRF $H : \{0,1\}^\kappa \to \{0,1\}^\kappa$.

INPUT OF SENDER ALICE: none.
INPUT OF RECEIVER BOB: an N-bit string $\boldsymbol{r}$.

PROTOCOL:

1. Alice chooses $\boldsymbol{s} \leftarrow \{0,1\}^\kappa$ uniformly at random.
2. Alice and Bob invoke κ instances of Random OT $\mathcal{F}_{\mathsf{ROT}}^\kappa$. In the i-th instance:
   - Alice acts as *receiver* with input s_i.
   - Bob acts as *sender*, and receives outputs $\boldsymbol{t_i}, \boldsymbol{u_i} \in \{0,1\}^\kappa$.
   - Alice receives output $\boldsymbol{q_i}$.
3. Bob computes the following $N \times \kappa$ matrices:
   - T whose ith column is $G(\boldsymbol{t_i})$
   - U whose ith column is $G(\boldsymbol{u_i})$
   - C whose ith **row** is 0^κ if $r_i = 0$ and 1^κ if $r_i = 1$

   Bob sends the matrix $P = T \oplus U \oplus C$ to Alice. For each $i \in [N]$, Bob outputs $\boldsymbol{m_i^*} = H(T(i))$, where $T(i)$ denotes the ith **row** of T.
4. Alice computes an $N \times \kappa$ matrix Q whose ith column is $G(\boldsymbol{q_i})$. Now let $Q(i)$ denote the ith **row** of this matrix, and let $P(i)$ denote the ith **row** of P. For each $i \in [N]$, Alice outputs:

$$\boldsymbol{m_{i,0}} = H(Q(i) \oplus \boldsymbol{s} \cdot P(i))$$
$$\boldsymbol{m_{i,1}} = H(Q(i) \oplus \boldsymbol{s} \cdot P(i) \oplus \boldsymbol{s})$$

Figure 4: The IKNP protocol for OT extension.

3 Our Main Protocol

3.1 A Conceptual Overview: PSI From a Multi-Point OPRF

A conceptually simple way to realize PSI is with an **oblivious PRF** (OPRF) [FIPR05, HL08], which allows a sender Alice to learn a [pseudo]random function F, and allows the receiver Bob to learn $F(y_i)$ for each chosen item in his set $\{y_1, \ldots, y_n\}$. If Alice has items $\{x_1, \ldots, x_n\}$, she can send $F(x_1), \ldots, F(x_n)$ to Bob. If the output of F is sufficiently long, then except with negligible probability we have $F(x_i) = F(y_j)$ if and only if $x_i = y_j$. Hence, Bob can deduce the intersection of the two sets. The fact that F is pseudorandom ensures that for any item $x_i \notin \{y_1, \ldots, y_n\}$, the corresponding $F(x_i)$ looks random to Bob. Hence, no information about such items is leaked to Bob.

Sparse OT Extension: Key Idea. We can interpret IKNP OT extension (Figure 4) as an OPRF as follows: Define the function $F(i) = \boldsymbol{m_{i,0}}$. Clearly the sender who knows the key of F can compute $F(i)$ for

any i. The receiver can set his i'th choice bit in the OT to be $r_i = 0$ if he chooses to learn $F(i)$ (in this case he learns $\boldsymbol{m_{i,0}}$), and use $r_i = 1$ if he chooses not to learn $F(i)$ (now he learns $\boldsymbol{m_{i,1}}$). To learn k OPRF outputs, the receiver includes k 0s among his choice bits. The security of OT extension implies that the receiver learns nothing about $F(i)$ whenever $r_i = 1$, and the sender learns nothing about the r_i bits.

This yields an OPRF of the form $F : [N] \to \{0,1\}^\kappa$, where N is the number of rows in the OT extension matrix. To be useful for PSI, N should be exponentially large, making this simple approach extremly inefficient. The following two key observations allow us to make the above approach efficient:

1. The parties require only *random access* to the large OT extension matrices. In the PSI application, they only read the $n \ll N$ rows indexed by their PSI inputs. While IKNP defines the matrices T, U, Q by expanding base OT values via a *PRG*, we instead expand with a PRF[4].

2. Besides the base OTs, the only communication in IKNP is when Bob sends the $N \times \kappa$ matrix P. In PSI, Bob only has knowledge of the $n \ll N$ rows of P indexed by his PSI input. Yet he must not let Alice identify the indices of these rows. Our idea is to have Bob interpolate a degree-n polynomial P where $P(y)$ is the correct "target row" of the IKNP OT extension matrix, for each y in his PSI input set. He then **sends this polynomial P instead of a huge matrix**. This change reduces Bob's communication from $O(N\kappa)$ to $O(n\kappa)$, allowing N to be exponential.

The polynomial P is distributed as a random polynomial (hiding Bob's inputs) since all rows of the IKNP matrix are pseudorandom from Alice's point of view. The more important concern is whether Bob learns too much. For example, suppose Bob interpolates P on points $\{y_1, \ldots, y_n\}$, but P happens to match the correct "IKNP target value" on some other $y^* \notin \{y_1, \ldots, y_n\}$ as well. This would allow Bob to learn whether Alice holds y^*, violating privacy. We argue that: (1) When the OT extension matrix is sufficiently wide, all relevant values $P(y^*)$ are sufficiently far in Hamming distance from their "target value". (2) When this is true, then Bob gets no information about Alice's items not in the intersection.

Comparison to other PSI paradigms. Other state-of-the-art PSI protocols (*e.g.*, [KKRT16, PSZ14]) can also be interpreted as constructing an OPRF from OT extension ([KKRT16] is explicitly described this way). These works construct an OPRF that the receiver can evaluate on *only one point,* and use various hashing tricks to reduce PSI to many independent instances of such an OPRF. In contrast, we construct a single instance of an OPRF where the receiver can evaluate *many points.* With such a multi-point OPRF it is trivial to achieve PSI, as illustrated above.

3.2 Protocol Details, Correctness, Performance

The formal details of our protocol are given in Figure 5. We use n_1 for the size of Alice's set and n_2 for the size of Bob's. We write $\mathsf{Interp}_{\mathbb{F}}(\{(x_1, y_1), \ldots, (x_d, y_d)\})$ to denote the unique polynomial P over field $\mathbb{F}$ of degree less than d where $P(x_i) = y_i$. In IKNP, the width of the matrices (and number of base OTs) is κ whereas the width in our instantiation is $\ell > \kappa$, where ℓ is determined by the security analysis.

Costs. The main computational cost is evaluating the degree-n_2 polynomial for Alice and interpolating the polynomial for Bob. In the case of $n_1 = n_2 = n$ this can be done with $O(n \log^2 n)$ field operations (details in 5.1).

In the communication costs of the protocol, we exclude the cost of the base OTs. These are fixed and don't depend on the parties' set sizes. Bob sends $n_2\ell$ bits, while Alice sends $n_1(\lambda + \log(n_1 n_2))$ bits. Generally speaking, ℓ is much larger than $\lambda + \log(n_1 n_2)$, which suggests that the party with more items should play the role of Alice. Concrete values are discussed later in Section 6.

Correctness. The idea behind the protocol is that for every row which Bob uses to interpolate the polynomial P (namely, a row corresponding to an input of Bob), Alice sends a value which is equal to the corresponding hash value that Bob computes in the last step of the protocol.

[4] In [HS13, Sec 3.2] they also use a PRF rather than PRG, but for a completely different purpose: random access to the OT extension matrix was used to parallelize OT extension and reduce memory footprint.

INPUT OF SENDER ALICE: $X = \{x_1, \ldots, x_{n_1}\} \subseteq [N]$
INPUT OF RECEIVER BOB: $Y = \{y_1, \ldots, y_{n_2}\} \subseteq [N]$
PARAMETERS:

- The size $\ell := \log_2 |\mathbb{F}|$ as defined in Table 6.
- A κ-Hamming CRF $H : \{0,1\}^{\ell} \to \{0,1\}^{\lambda + \log(n_1 n_2)}$
- A PRF $F : \{0,1\}^{\kappa} \times [N] \to \{0,1\}$

PROTOCOL:

1. Alice chooses $\boldsymbol{s} \leftarrow \{0,1\}^{\ell}$ uniformly at random.
2. Alice and Bob invoke ℓ instances of Random OT $\mathcal{F}_{\mathsf{ROT}}^{\kappa}$. In the i-th instance:
   - Alice acts as *receiver* with input s_i.
   - Bob acts as *sender*, and receives outputs $\boldsymbol{t_i}, \boldsymbol{u_i} \in \{0,1\}^{\kappa}$.
   - Alice receives output $\boldsymbol{q_i}$.
3. For $y \in Y$, Bob computes $R(y) = T(y) \oplus U(y)$, where:
$$T(y) := F(\boldsymbol{t_1}, y) \| F(\boldsymbol{t_2}, y) \| \cdots \| F(\boldsymbol{t_\ell}, y)$$
$$U(y) := F(\boldsymbol{u_1}, y) \| F(\boldsymbol{u_2}, y) \| \cdots \| F(\boldsymbol{u_\ell}, y)$$
4. Bob computes a polynomial $P := \mathsf{Interp}_{\mathbb{F}}(\{y, R(y)\}_{y \in Y})$, and sends its coefficients to Alice
5. Alice defines Q as follows:
$$Q(x) := F(\boldsymbol{q_1}, x) \| F(\boldsymbol{q_2}, x) \| \cdots \| F(\boldsymbol{q_\ell}, x)$$
and sends $\mathcal{O} = \big\{ H\big(Q(x) \oplus \boldsymbol{s} \cdot P(x)\big) \mid x \in X \big\}$ randomly permuted to Bob
6. Bob outputs $\{y \in Y \mid H(T(y)) \in \mathcal{O}\}$

Figure 5: Our PSI protocol

Namely, following the discussion of IKNP, we can see that

$$Q(x) = T(x) \oplus \boldsymbol{s} \cdot \big(T(x) \oplus U(x)\big) = T(x) \oplus \boldsymbol{s} \cdot R(x)$$

and therefore in Step 5 Alice computes:

$$Q(x) \oplus \boldsymbol{s} \cdot P(x) = T(x) \oplus \boldsymbol{s} \cdot \big(P(x) \oplus R(x)\big) \tag{1}$$

Now, consider the case that both parties have a common item x^*. Bob constructs P so that $P(x^*) = R(x^*)$. Alice computes $H(Q(x^*) \oplus \boldsymbol{s} \cdot P(x^*))$ which from Equation 1 gives Alice $H(T(x^*))$. Hence, Bob will include x^* in his output.

In case that $x \notin Y$, $P(x)$ and $R(x)$ will be different in at least κ bits with overwhelming probability (see the analysis below). Therefore, $H\big(Q(x) \oplus \boldsymbol{s} \cdot P(x)\big)$ is pseudorandom from Bob's view, under the Hamming correlation robust assumption. If σ is the output length of H, then the probability that this random value equals $H(T(y))$ for some $y \in Y$ is $n_2 2^{-\sigma}$. By a union bound over the items of $X \setminus Y$, the overall probability of Bob including an incorrect value in the output is at most $n_1 n_2 2^{-\sigma}$. Hence, choosing $\sigma = \lambda + \log_2(n_1 n_2)$ ensures that this error probability is negligible ($2^{-\lambda}$).

3.3 Properties of Polynomials

We first prove some simple lemmas about polynomials that are used in the security proof of our PSI protocol.

Hiding Bob's input. For security against a corrupt sender Alice, we simply need Bob's polynomial to hide his input:

	n_1:							
Pr[BadPoly]	2^{10}	2^{12}	2^{14}	2^{16}	2^{18}	2^{20}	2^{22}	2^{24}
2^{-40}	416	420	424	428	432	436	440	444
2^{-80}	491	495	498	502	505	509	512	515

Figure 6: Field size $\log_2 |\mathbb{F}|$ for our protocol, with $\kappa = 128$.

Proposition 3.1. *If $z_1, \ldots, z_d$ are uniformly distributed over $\mathbb{F}$, then for all distinct $x_1, \ldots, x_d$, the output of $\mathsf{Interp}_{\mathbb{F}}(\{(x_1, z_1), \ldots, (x_d, z_d)\})$ is uniformly distributed. In particular, the distribution does not depend on the x_i's.*

Proof. Viewing polynomial interpolation as a linear operation, we have the following, where $p_0, \ldots, p_{d-1}$ are the coefficients of the polynomial.

$$\begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_{d-1} \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{d-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{d-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_d & x_d^2 & \cdots & x_d^{d-1} \end{bmatrix}^{-1} \times \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_d \end{bmatrix}$$

Since the polynomial is computed as a nonsingular matrix times a uniform vector, the polynomial's distribution is also uniform. ■ □

Security for Alice. In our protocol, Bob generates a polynomial P such that $P(y) = R(y)$ for his input points $y \in Y$. The security of the protocol relies on the property that for *all other* points $x \notin Y$, $P(x)$ is far from $R(x)$ in Hamming distance (with very high probability).

Definition 3.2 (Bad polynomial)**.** *Let $\mathsf{BadPoly}^R_{\mathbb{F}}(X, Y)$ be the procedure defined as follows:*

1. $P := \mathsf{Interp}_{\mathbb{F}}(\{(y, R(y)) \mid y \in Y\})$
2. *Output 1 iff $\exists x \in X \setminus Y$ s.t. $d_H(P(x), R(x)) < \kappa$*

Proposition 3.3. *The probability that a polynomial interpolated over points in Y also passes "too close" to another point in X is bounded by $\frac{n_1}{|\mathbb{F}|} \sum_{i<\kappa} \binom{\log_2 |\mathbb{F}|}{i}$. Formally, for all X, Y with $|X| = n_1$,*

$$\Pr[\mathsf{BadPoly}^R_{\mathbb{F}}(X, Y) = 1] \leq \frac{n_1}{|\mathbb{F}|} \sum_{i<\kappa} \binom{\log_2 |\mathbb{F}|}{i},$$

where the probability is over choice of random function $R : \mathbb{F} \to \mathbb{F}$.

Proof. For a fixed element $v \in \mathbb{F}$, the probability of a uniformly chosen element $u \leftarrow \mathbb{F}$ being closer than Hamming distance κ to v is $\sum_{i<\kappa} \binom{\log_2 |\mathbb{F}|}{i} / |\mathbb{F}|$. This is the case when entering to the second step of the procedure in Definition 3.2, where each $P(x)$ is already fixed and $R(x)$ is uniform in $\mathbb{F}$. The claim follows by a union bound over the (at most n_1) items in $X \setminus Y$. ■ □

On the communication complexity of the protocol.. Let $\ell = \log_2 |\mathbb{F}|$. In our protocol a small ℓ leads to a bad event where two terms are close in Hamming distance. Since this bad event is a *one-time* event, it suffices to bound its probability by the statistical security parameter λ. Since the bad event involves a union bound over n, the concrete analysis involves both λ and n.

However, we could also just compute ℓ assuming the worst case $n = 2^\kappa$ (where κ is the computational security parameter), and we would get $\ell = \mathsf{poly}(\kappa)$ and a bad-event probability of $\mathsf{poly}(\mathsf{n})/2^\kappa$. For our specific protocol/analysis, $\ell = 4.3 \cdot \kappa$ appears sufficient to achieve bad event probability $n/2^\kappa$ (robust to a wide range of κ). As an analogy: in *any* OPRF-based PSI protocol, receiver learns $F(y_1), F(y_2), \ldots$ and sender sends $F(x_1), F(x_2), \ldots$. For correctness it suffices to truncate F to $\lambda + 2log(n)$ bits, but of course it is quite enough to let F have $O(\kappa)$ bits.

In summary, asymptotically $O(n \cdot \kappa)$ bits do suffice for correctness/security, but so do $O(n \cdot \ell)$ bits, where ℓ is some function of λ, κ, n. The more fine-grained analysis of ℓ leads to less concrete communication, and that is why our concrete analysis displays a dependency of ℓ on n.

Hence, given a desired κ, n_1, and $\Pr[\mathsf{BadPoly}]$ one can solve for the smallest compatible field size. A table of such field sizes is provided in Figure 6.

3.4 Semi-Honest Security

Theorem 3.4. *The protocol in Figure 5 securely realizes the PSI functionality of Figure 3 in a semi-honest setting, when F is a pseudo-random function, H is a κ-Hamming correlation robust (Definition 2.1), and the parameter ℓ is chosen according to the table in Figure 6.*

Proof. Due to space limitation we only sketch here the simulators for the two cases of corrupt Alice and corrupt Bob. The full security proof including (via hybrid arguments) is defered to Appendix A.1.

Corrupt Alice. The simulator observes Alice's inputs to the $\mathcal{F}_{\mathsf{ROT}}$ primitive and gives random $\boldsymbol{q_i}$ as OT outputs in Step 2. The only other message Alice receives is the polynomial P in Step 4. Instead of $P := \mathsf{Interp}_{\mathbb{F}}(\{y, R(y)\}_{y \in Y})$, the simulator sends a uniformly random polynomial to Alice.

Briefly, this simulation is indistinguishable for the following reasons: $R(y)$ is pseudorandom from Alice's view (by the security of the PRF which defines the conceptual OT-extension matrices). Hence, the polynomial P is distributed uniformly (from Proposition 3.1).

Corrupt Bob. The simulator for a corrupt Bob first obtains $X \cap Y$ from the ideal PSI functionality. It simulates random outputs $\boldsymbol{t_i}, \boldsymbol{q_i}$ from $\mathcal{F}_{\mathsf{ROT}}$. The only other message received by Bob is the set $\mathcal{O}$ in Step 5. To simulate this message, the simulator computes $n' = n_1 - |X \cap Y|$ and uniformly samples values $z_1, \ldots, z_{n'}$. It then simulates $\mathcal{O} = \{H(T(x)) \mid x \in X \cap Y\} \cup \{z_1, \ldots, z_{n'}\}$.

This simulation is indistinguishable because $P(x)$ and $R(x)$ will differ in at least κ bits for every $x \in X \setminus Y$ (Proposition 3.3), and as long as that is true, the corresponding outputs of H will be pseudorandom (Definition 2.1). □

3.5 Optimizations: Reducing Alice's Communication

Recall that Alice's communication consists of n_1 OPRF outputs, each of length $\lambda + \log(n_1 n_2)$. Using a trick of Tamrakar *et al.* [TLP+17], this can be reduced to roughly $\lambda + \log n_1$ bits per item. For $\lambda = 40$ and $n_1 = n_2 = 2^{20}$ this reduces communication by 25%. This improvement is even more beneficial when $n_1 \gg n_2$, since Alice's communication (despite being less *per item*) dominates the protocol overall.

For completeness, we describe the trick in Appendix B. It can be viewed as a public lossless compression of Alice's protocol message, and therefore does not affect security. We also show another approach to reduce Alice's communication to *exactly* $\lambda + \log n_1$ bits per item, inspired by polynomial encodings. However, that optimization is much slower in practice for only a marginal reduction in communication.

3.6 Security against Malicious Sender

Our protocol is secure against a malicious sender if F is modeled as a non-programmable random oracle. (In Appendix A.3 we show that our protocol is *insecure* against a malicious receiver.)

Theorem 3.5. *The protocol in Figure 5 securely realizes the PSI functionality of Figure 3 against a malicious sender Alice, when F is modeled as a (non-programmable) random oracle.*

Proof Sketch. The simulator plays the role of honest receiver Bob and the ideal $\mathcal{F}_{\mathsf{ROT}}$ functionalities in steps 1 and 2, observing Alice's $\mathcal{F}_{\mathsf{ROT}}$-input $\boldsymbol{s}$ and generating random outputs $\{\boldsymbol{q_i}\}_{i \in [\ell]}$. Throughout the protocol, the simulator also observes all of Alice's queries to the random oracle F. Without loss of generality, we can assume that whenever Alice makes a query of the form $F(\boldsymbol{q_i}, x)$ to the random oracle, where $\boldsymbol{q_i}$ is one of the $\mathcal{F}_{\mathsf{ROT}}$-outputs, it also queries $F(\boldsymbol{q_j}, x)$ for *all* $j \in [\ell]$. The simulator observes Alice's oracle queries and maintains a list

$$C = \{x \mid \text{Alice queried } F \text{ on some } F(\boldsymbol{q_i}, x)\}.$$

In step 4, the simulator sends a random polynomial P. In step 5, the simulator receives a set $\mathcal{O}$ from the corrupt Alice and computes

$$\widetilde{X} = \{x \in C \mid H(Q(x) + \boldsymbol{s} \cdot P(x)) \in \mathcal{O}\},$$

and finally sends $\widetilde{X}$ to the PSI ideal functionality.

In Appendix A.2 we use a hybrid argument to formally prove the indistinguishability of this simulator. □

4 The Fast Protocol Variant

The biggest performance bottleneck in our protocol is interpolating and evaluating extremely high-degree (e.g., $d = 2^{20}$) polynomials over large (e.g., $|\mathbb{F}| > 2^{64}$) finite fields. To reduce this computational cost, we employ a technique of hashing the items into bins, and performing PSI (involving lower-degree polynomials) within each bin. This general technique is quite common in the PSI literature, and two different types of hashing have been suggested in previous work. However, we introduce a new hashing technique that (to the best of our knowledge) has not been suggested previously for PSI. As we illustrate, previous protocols are not able to immediately benefit from this new hashing technique — only our approach enjoys the advantages of this new approach.

4.1 Previous Hashing Techniques

In **simple hashing**, parties choose a random hash function $h : \{0,1\}^* \to [m]$ and assign each item x to bin with index $h(x)$. Since if Alice and Bob have the same item they both map it to the same bin, then they can perform a separate PSI within each bin. The load of each bin leaks information (i.e., it cannot be simulated just given the intersection), and therefore the parties must pad each bin up to a maximum size with dummy items. For example, with n items and $m = O(n/\log n)$ bins, the expected load of each bin is $n/m = O(\log n)$ and the maximum load B is $O(\log n)$ with high probability. In practice, B may be 4 to 5 times higher than n/m, meaning that **about 80% of the items are dummies.**

In **Cuckoo hashing** (used in [PSZ14, KKRT16]), the parties choose two hash function $h_1, h_2 : \{0,1\}^* \to [m]$. The receiver Bob places his items into m bins so that x is placed in either $h_1(x)$ or $h_2(x)$, and each bin contains at most one item. Alice places each of her items x in *both* locations $h_1(x)$ and $h_2(x)$. As above, Bob must pad each bin with dummy items to contain *exactly* one item (we can avoid dummy items for Alice). The parties perform a PSI in each bin. Cuckoo hashing leads to roughly **20% dummy items** (this is for Cuckoo hashing with three hash functions; Cuckoo hashing with two hash functions has even more dummy items), not to mention extra protocol costs associated with the stash (a special bin for items that cannot find a home in the Cuckoo hashing).

4.2 Our High-Level Approach

An important feature of Cuckoo hashing is that it results in at most one item per bin for Bob. This situation is the ideal fit for the underlying OPRF primitive of [PSZ14, KKRT16], which allows the receiver (Bob in this case) to evaluate the OPRF on *a single value.* With Cuckoo hashing, the PSI performed in each bin can be achieved with such an OPRF.

But our sparse OT extension technique results in a multi-point OPRF primitive that allows the receiver to evaluate on many values. Hence we have no need to constrain the receiver Bob to have only one item per bin. We propose to use a generalization of Cuckoo hashing called **2-choice hashing**. Similar to Cuckoo hashing, there are two hash functions h_1 and h_2, and item x can be placed in either $h_1(x)$ or $h_2(x)$. Unlike Cuckoo hashing, there is no restriction on the number of items per bin.

Cuckoo hashing is also often synonymous with an *online* hashing procedure, where all the items are processed in a single pass. For the application to PSI, though, all items are known upfront. We are free to make the best assignment of items to bins, taking into account global information about all items. [5]

[5] This observation was concurrently and independently noted in [FNO18]; however, their focus is exclusively on Cuckoo hashing, with at most one item per bin. They do not consider our generalized 2-choice hashing.

These facts about 2-choice hashing indeed lead to much better performance (in terms of dummy items). The following theorem of Czumaj, Riley, and Scheideler [CRS03] shows that when the bins are allowed to contain significantly many items, *no dummy items are needed at all!*

Theorem 4.1 ([CRS03])**.** *Let $h_1, h_2 : \{0,1\}^* \to [m]$ be two random functions. Suppose there are n items and m bins, where each item x can be placed in either $h_1(x)$ or $h_2(x)$. Let $L = \lceil n/m \rceil$. If $n = \Omega(m \log m)$ then with high probability there exists an **optimal assignment**, where each bin contains no more than L items.*

The proof uses an explicit randomized algorithm to generate an optimal assignment. However, we found that the algorithm takes prohibitively long to converge. Also, its analysis of error probability is not concrete. However, if we are willing to settle for merely an "almost optimal" assignment of items to bins, the following theorem of Sanders, Egner, and Korst [SEK03] suggests that one can be found quite efficiently:

Theorem 4.2 ([SEK03])**.** *Let n, m, h_1, h_2 be as above, with $L = \lceil n/m \rceil$. There is a deterministic algorithm running in time $O(n \log n)$ that assigns at most $L+1$ items to each bin, with probability $1 - O(1/m)^L$ over the choice of h_1, h_2.*

We propose the two-pass heuristic in Algorithm 1 for assigning items to bins. This very simple, **linear time** algorithm seems to perform well. In our experience, it never fails to find a near-optimal assignment with maximum load $L + 1 = \lceil n/m \rceil + 1$, for the parameters we use. In the rare event that it *does* fail, more iterations of the final loop are likely to succeed.

Algorithm 1 FindAssignment(X, m, h_1, h_2)

1: **for** $x \in X$ **do**
2: Assign item x to bin $h_1(x)$
3: **for** $x \in X$ **do**
4: Assign item x to whichever of $h_1(x), h_2(x)$ currently has fewest items

With such a near-optimal assignment, we can see that for each of the n/m bins there is only one dummy item. In practice, we set n/m to be the statistical security parameter λ so that an assignment exists with overwhelming probability. Setting $n/m = \lambda = 40$ leads to the most dummy items one would ever consider for our protocol, but still there are **only 2.5% ($= 1/40$) dummy items.**

In the overall PSI protocol, Bob will send a polynomial of degree $\lceil n/m \rceil + 1$ for each bin. For each item of Alice $x \in X$, she considers both locations $h_1(x)$ and $h_2(x)$ and derives an OT-extension / OPRF output for both possibilities. She then sends these 2 outputs for each item.

4.3 Protocol Details

The details of the protocol are given in Figure 7. It mostly follows the outline given above, with one important exception. Most of the time, Alice computes two distinct mask values for each $x \in X$: one for $h_1(x)$ and one for $h_2(x)$. But $h_1(x) = h_2(x)$ is possible with probability $1/m$. In that case, depending on how one specifies this edge case, Alice will either send a repeated mask or send less masks overall. Either way, this event leaks to Bob that Alice holds such an item satisfying $h_1(x) = h_2(x)$. This issue is common to all PSI protocols that use Cuckoo hashing as well.

To address this issue, we let Bob append to each item y a bit $b \in \{1, 2\}$ indicating which hash function h_b was used to assign it to this bin. If $h_1(y) = h_2(y)$ we just choose b arbitrarily. Then the OT extension & polynomials are done with respect to these "extended" values. Now in the case of $h_1(y) = h_2(y)$, Bob will only learn the OT-extension output for one variant $y \| b$, but Alice (if she has such an item) will still be able to compute two distinct OT-extension outputs for the two variants.

Theorem 4.3. *The protocol in Figure 7 securely realizes the PSI functionality of Figure 3 in a semi-honest setting, with F, H as in Theorem 3.4 and ℓ according to the column indexed by $2n_1$ in Table Figure 6.*

The semi-honest security of the modified protocol follows with a very similar proof as the original protocol, therefore we omit it for the sake of space. Unlike the original protocol, this new one is *not secure* against malicious adversaries (details are given in Appendix A.4).

INPUT OF SENDER ALICE: $X = \{x_1, \ldots, x_{n_1}\} \subseteq [N]$
INPUT OF RECEIVER BOB: $Y = \{y_1, \ldots, y_{n_2}\} \subseteq [N]$
PARAMETERS: (same as Figure 5, except ℓ is chosen to be compatible with $2n_1$ rather than n_1 — see text for discussion)
PROTOCOL: (steps 1–3 are the same as Figure 5)

4. Bob sets $m = n_2/\lambda$, chooses random functions $h_1, h_2 : [N] \to [m]$, and sends them to Alice. Then Bob assigns its items using $\mathsf{FindAssignment}(Y, m, h_1, h_2)$ (from Alg. 1) and adds dummy items so that each bin has exactly $\lceil n_2/m \rceil + 1$ items. Write $y\|b \in \mathcal{B}_i$ to mean that y was assigned to bin i by hash h_b. For each bin i, Alice computes a polynomial $P_i := \mathsf{Interp}_{\mathbb{F}}(\{y\|b, R(y\|b)\}_{y\|b \in \mathcal{B}_i})$, and sends its coefficients to Alice.

5. Alice defines Q as in Figure 5 and defines the sets:

$$\mathcal{O}_1 = \left\{ H\big(Q(x\|1) \oplus s \cdot P_{h_1(x)}(x\|1)\big) \;\middle|\; x \in X \right\}$$
$$\mathcal{O}_2 = \left\{ H\big(Q(x\|2) \oplus s \cdot P_{h_2(x)}(x\|2)\big) \;\middle|\; x \in X \right\}$$

   She permutes each one randomly and sends them to Bob.

6. Bob outputs $\{y \mid y\|b \in \bigcup_i \mathcal{B}_i \text{ and } H(T(y\|b)) \in \mathcal{O}_b\}$

Figure 7: PSI protocol using 2-choice hashing optimization.

4.3.1 Efficiency.

Theorem 4.2 suggests that a near-optimal assignment of items to bins exists with probability at least $1 - 2^{-n_2/m}$. Hence, we must have $n_2/m \geq \lambda$, the statistical security parameter, to ensure that Bob's hashing step succeeds with overwhelming probability. Setting $m = n_2/\lambda$, the cost of all interpolations is now $m \cdot O(\lambda \log^2 \lambda) = O(n_2 \log^2 \lambda)$ field operations if using the asymptotically efficient algorithm, or $m \cdot O(\lambda^2) = O(n_2 \lambda)$ using the simpler quadratic interpolation algorithm (which is indeed faster in practice for such small polynomials). In either case, this is a **significant** improvement over $O(n_2 \log^2 n_2)$ of the basic protocol (not to mention that distinct bins allow for easy parallelization). The cost of Alice's polynomial evaluation is similarly improved.

No matter what m we choose (assuming n_2/m is an integer), there will always be exactly m dummy items for Bob. The percentage of dummy items is m/n_2, so Alice's communication will increase by a multiplicative factor of $(1 + m/n_2)$. We suggest $m = n_2/\lambda$, so Alice's communication increases by a $(1 + 1/\lambda)$ factor. As mentioned above, for $\lambda = 40$, this increase is only 2.5%.

Recall from Section 3.3 that the parameter ℓ (width of OT extension matrix) depends on the number of rows of the OT extension matrix that Alice accesses. With this new optimization, she accesses twice as many rows (rows $x\|1$ and $x\|2$ for every $x \in X$). This leads to a slight increase in ℓ. For the concrete parameters we consider (see Figure 6), ℓ must increase by only 2 bits.

5 Optimizations for High-Degree Polynomials

Despite using fast polynomial algorithms, having one party (the *interpolating party*) interpolating the huge-degree polynomial leads to a long idle time by the other party (*evaluating party*), which implies a serious computational bottleneck. In this section we show that in case that the x and y coordinates of the interpolation points are drawn from the domains $\mathcal{D}_x$ and $\mathcal{D}_y$, respectively, such that $\mathcal{D}_x \ll \mathcal{D}_y$,the idle time can be significantly shrinked. To this end, we developed new techniques, namely, *slice & stream* and *sub-product tree reuse* that allow a significant reduction of the overall time of the protocol. The former technique means that we "slice" the interpolation points into several parts, then we can interpolated each part over a smaller field and hence faster; when a slice is ready it is sent immediately to the other party for evaluation (i.e. streaming of polynomials). The latter technique is based on our observation that one sub-algorithm that constructs a sub-product tree (which is used both in interpolation and evaluation) depends only on the x-values of the

interpolation points. Since all polynomial slices use the same x-values and differ only on their y-values we can reuse the same sub-product tree for all slices! We believe our techniques are valuable for other applications that require an implementation of high-degree polynomial algorithms over large fields. As demonstrated in Section 5.2, our techniques reduce the overall interpolation and evaluation time by up to 60%.

In Section 5.1 we give an overview on known polynomial algorithms and in Section 5.2 we introduce our techniques in detail.

5.1 Background: Interpolation and Multi-Point Evaluation

Trivial implementations of polynomial interpolation and multi-point evaluation of *arbitrary* points adopt the $O(n^2)$ algorithms as they are sufficient for the typical use cases of low-degree polynomials. However, in our case the degree is in the millions, so the $O(n^2)$ algorithms are completely impractical. Faster algorithms, by Moenck and Borodin from 1972 [MB72], achieve computational complexity of $O(n \log^2 n)$. In the following we present a high level overview on the algorithms, while a detailed description is given in Appendix C.

Let $X = \{x_1, \ldots, x_n\} \subset \{0,1\}^\alpha$ and $Y = \{y_1, \ldots, y_n\} \subset \{0,1\}^\beta$.

- Given X and Y, the problem of *polynomial interpolation* is to find the unique $(n-1)$-degree polynomial P that passes through the points $\{(x_i, y_i)\}_{i \in [n]}$.
- Given X and an $(n-1)$-degree polynomial Q, the problem of *multi-point evaluation* is to compute $Q(X) = \{Q(x_i)\}_{i \in [n]}$.

Algorithms for both problems follow the divide-and-conquer approach such that in every iteration the problem is reduced to two half-size problems. Combining the solutions of the half-size problems to a solution of the full-size problem has a computational complexity of $O(n \log n)$. Formally, let $T(n)$ be the time to solve the interpolation and multi-point evaluation problems for $|X| = |Y| = n$, then the recurrence relation is: $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + O(n \log n) = O(n \log^2 n)$ where the second equality follows from the Master theorem [CLRS09, Ch. 4].

The evaluation and interpolation algorithms are separated to two and four sub-procedures, respectively, as follows.

5.1.1 Evaluation.

Algorithm MultipointEvaluate(Q, X) invokes $M \leftarrow$ BuildTree(X) and outputs $Y \leftarrow$ Evaluate(Q, M).

- BuildTree(X) constructs and outputs a binary tree of polynomials, denoted M. Its leaves are the 1-degree polynomials $\{(x-a)\}_{a \in X}$ and each node is the multiplication of its two children. Thus, if the degrees of the childs are d_1 and d_2 then the node's degree is $d_1 \cdot d_2$. If n is a power of 2 then the degree of M's root is n.
- Evaluate(Q, M) evaluates the polynomial Q on X, note that X is implicitly "encoded" within M. The idea is that for every node $m \in M$ (recall that m is a polynomial), if $(x-a)$ divides m then $Q(a) = R(a)$ where $R = Q \mod m$ (i.e. it is the remainder of the division of Q by m). To obtain $Q(a)$ we replace each node m with (Parent$(m) \mod m$) and finally output the result on that leaf. The remainder is computed in $O(n \log n)$ arithmetic operations in the underlying field.

5.1.2 Interpolation.

Algorithm Interpolate(X, Y) invokes $M \leftarrow$ BuildTree(X) as described above. Let M_0 be M's root, it computes M_0's derivative by $M_0' \leftarrow$ Derivative(M_0) and then evaluates M_0' over X by $A \leftarrow$ MultipointEvaluate(M_0', X). Finally it invokes $P \leftarrow$ InternalInterpolate(M, A) and outputs P. The purpose of the sub-algorithms is to enable the division of a n-size problem to two $\frac{n}{2}$-size problems. Note that within MultipointEvaluate there is a construction of the same sub-product tree as in BuildTree, therefore we can skip this and construct M only once. The time of the algorithm is the sum of the times of these four sub-algorithms, $T_{\text{Interpolate}}(n) = T_{\text{BuildTree}}(n) + T_{\text{Derivative}}(n) + T_{\text{MultipointEvaluate}}(n) + T_{\text{InternalInterpolate}}(n) = O(n \log^2 n) + O(n) + O(n \log^2 n) + O(n \log^2 n) = O(n \log^2 n)$.

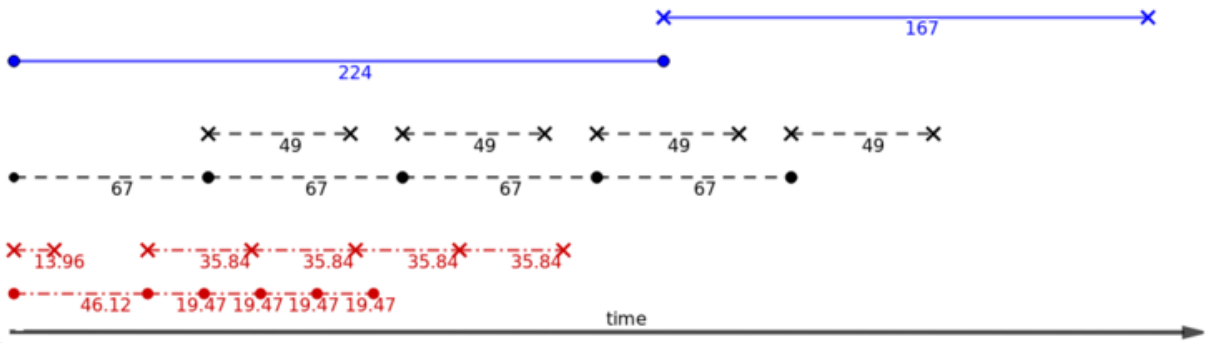


Figure 8: Illustrating the slicing technique. The lines between •'s represent the interpolating party and the lines between the ×'s represent the evaluating party. Solid (blue) lines illustrate the trivial implementation (overall 400 seconds), dashed (black) lines illustrate the initial slicing technique (overall 317 seconds) and dotted-dashed (red) lines illustrate the final optimization (overall 189 seconds).

5.1.3 Concrete times.

Times for each of the above sub-algorithms were measured separately and are presented in Figure 13.

5.2 Polynomial Slicing and Streaming

Let $x_1, \ldots, x_n \in \{0,1\}^\alpha$ and $y_1, \ldots, y_n \in \{0,1\}^\beta$ (where $\beta > \alpha$) then we interpolate the polynomial P using points $\{(x_i, y_i)\}_{i\in[n]}$ over a field $\mathbb{F}$ where $|\mathbb{F}| = 2^\beta$. For the sake of exposition suppose that α divides β and let $\rho = \frac{\beta}{\alpha}$. For each i we define y_i^j for $j \in [\rho]$ such that $|y_i^j| = \alpha$ and $y_i = y_i^1 || \ldots || y_i^\rho$. We can "cut" P into ρ slices $P_1, \ldots, P_\rho$ such that for every x_i it holds that $P(x_i) = P_1(x_i) || \ldots || P_\rho(x_i)$. This is done by interpolating the polynomial P_j (for $j \in [\rho]$) using the points $\{(x_i, y_i^j)\}_{i\in[n]}$. This requires a smaller field, i.e. we need that $|\mathbb{F}| = 2^\alpha$, hence P_j is produced in a shorter time.

To demonstrate the above let us fix some parameters. Assume that the parties' only task is to interpolate P using $n = 2^{20}$ points and then perform a multi-point evaluation of n points; also assume an ideal network with zero latency. Consider first performing this task directly to a "single-slice" polynomial over a field of size 2^β where $\beta = 512$. Interpolation and multi-point evaluation take $233 + 167 = 400$ seconds (detailed measurements are given in Table 13. We ignore milliseconds here and in the following). Utilizing the slicing technique with $\alpha = 128$ we have $\rho = \frac{\beta}{\alpha} = \frac{512}{128} = 4$ slices. This means that the interpolating party produces the sliced polynomials one after the other and sends them immediately (i.e. without waiting until for all polynomials to be ready) and the evaluating party evaluates them one by one upon reception. This leads to $67 \cdot 4 + 49 = 317$ seconds which is 81% of the trivial implementation.

5.2.1 Further utilizing the slicing technique.

As shown above, the slicing and streaming technique leads to an improvement over the trivial implementation. The following observation significantly pushes forward the slicing technique: Building the polynomials tree M in the evaluation process depends only on $x_1, \ldots, x_n$, which means this can be performed *only once for all slices*. Similarly, in the interpolation algorithm the tasks of building the polynomials tree, calculating the derivative and evaluating it depends only on $x_1, \ldots, x_n$ and can be performed once and for all slices. Thus, taking $\beta = 512$, $\alpha = 128$ and $n = 2^{20}$ the one-time tasks of building the sub-product tree, calculating the derivative and evaluating it takes $12889 + 86 + 33144 = 46119$ ms. The one-time task of the evaluating party (building the sub-product tree) takes 13959 ms and can surely be done simultaneously. Then the interpolating party produces 4 polynomial slices, each takes 19471 ms, and the evaluating party evaluates them upon reception. Since the evaluation task is more expensive than the interpolating task (the part being performed for each slice) the total running time is $46119 + 4 \cdot 35835 = 189459$ ms. This is less than 60% of the initial slicing technique and 48% of the trivial implementation. Both of our optimizations, together with the trivial implementation are illustrated in Figure 8.

Protocol	Communication	$n = n_1 = n_2$		
		2^{16}	2^{20}	2^{24}
KKRT	$(3+s)(\lambda + \log(n_1 n_2))n_1 + 1.2\ell n_2$	$1042n$	$1018n$	$978n$
DH-PSI	$\phi n_1 + (\phi + \lambda + \log(n_1 n_2))n_2$	$584n$	$592n$	$600n$
spot-low	$1.02(\lambda + \log_2(n_2) + 2)n_1 + \ell n_2$	$488n$	$500n$	$512n$
spot-fast	$2(\lambda + \log(n_1 n_2))n_1 + \ell(1 + 1/\lambda)n_2$	$583n$	$609n$	$634n$

Table 1: Theoretical communication costs of PSI protocols (in bits), calculated using computational security $\kappa = 128$ and statistical security $\lambda = 40$. Ignores cost of base OTs (in our protocol and KKRT) which are independent of input size. ϕ is the size of elliptic curve group elements (256 is used here). ℓ is width of OT extension matrix (depends on n_1 and protocol).

5.2.2 Communication.

Observe that this technique *does not increase* the communication complexity of the protocol. This is due to the fact that instead of sending 2^n coefficients of P, each of size β, we send 2^n coefficients of P_j, each of size α, for every j. This leads to exactly same communication size of $2^n \cdot \alpha \cdot \rho = 2^n \cdot \beta$.

6 Implementation and Performance Comparison

Recall that we have presented two variants of our protocol. In this section we will refer to them as:

spot-low: the communication-optimized variant presented in Figure 5, in which Bob sends one large polynomial and Alice sends one OPRF output per item.

spot-fast: the speed-optimized variant presented in Figure 7, in which Bob uses 2-choice hashing and Alice sends two OPRF outputs per item.

We also compare our protocols to the following:

KKRT: the leading OT-extension-based protocol from [KKRT16].

DH-PSI: Diffie-Hellman-based PSI, instantiated with either Koblitz-283 (K283) or Curve25519 (25519) elliptic curves.

Our focus in this section is on the case where $n_1 = n_2$, i.e., the parties have sets of equal size. We report some findings also for the case of unequal set sizes in Appendix F. Our complete implementation is available on GitHub: `https://github.com/osu-crypto/SpOT-PSI`.

6.1 Theoretical Analysis of Communication

We first compare the *theoretical* communication complexity of protocols (Table 1). This measures how much communication the protocols require on an idealized network where we do not care about protocol metadata, realistic encodings, byte alignment, etc. In practice, data is split up into multiples of bytes (or CPU words), and different data is encoded with headers, etc. — empirical measurements of such real-world costs are given later in Table 2and Table 9.

For set sizes in the range 2^{16} to 2^{24}, our spot-low variant has the least communication of any of the protocols we consider: ∼15% less than DH-PSI and ∼50% less than KKRT. Our spot-fast variant uses up to ∼5% more communication than DH-PSI but 35-43% less than KKRT.

We note that KKRT uses a parameter ℓ similar to ours (corresponding to the width of the OT extension matrix), but their parameter is always slightly larger. This is because (as in our protocol) ℓ depends on how many rows of the OT matrix the sender accesses, which is more than in ours ($(3+s)n_1$ in KKRT).

The communication optimization (described in Section 3.5) can indeed be applied to other protocols as well (DH-PSI, KKRT, and spot-fast). For example, when $n = 2^{20}$ it saves 16 bits per item (only 2.6MB in total), so the effect does not have significant impact on any comparisons. However, the optimization would be much more expensive or cumbersome to implement since it requires all OPRF outputs to be computed and sorted, but without this optimization they can be sent as they are computed.

6.2 Experimental Comparison

We now present a comparison based on implementations of all protocols.

6.2.1 Implementation Details.

We used the implementation of KKRT provided by the authors. We implemented DH-PSI using the Miracl library implementations of Koblitz K-283 and Curve25519 elliptic curves.

For our own protocols, we implemented the polynomial interpolation and evaluation algorithms using a field of prime order p, where p is the smallest prime greater than 2^ℓ and ℓ is bit length of the output of our sparse-OT extension (the ℓ in Figure 6). We discuss this choice in Appendix D. The polynomial operations are implemented using the NTL library v10.5.0.

Note that both KKRT and our protocols require the same underlying primitives: a Hamming correlation-robust function H, a pseudorandom funtion F, and base OTs for OT extension. We instantiated these primitives exactly as KKRT: both H and F instantiated using AES, and base OTs instantiated using Naor-Pinkas [NP01]. We use the implementation of base OTs from the libOTe library[6].

All protocols use a computational security $\kappa = 128$ bits and a statistical security $\lambda = 40$ bits.

6.2.2 Experimental setup: AWS benchmark.

We performed a series of benchmarks on the Amazon web services (AWS) EC2 cloud computing service. We use the M5.large machine class, which is classified as the current state-of-the-art "general purpose" instance. These machines have 2 vCPU (2.5GHz Intel Xeon) and 8 GB RAM. We considered other kinds of instances, but ultimately rejected them. The cheaper T2 class ("burstable") was found to be too unstable for our workloads, while the more expensive C5 class ("compute-optimized") resulted in more monetary cost than M5 in all cases.

		1	2	3	4	5	6
Virginia	1	9.6	0.17	1.08	0.063	0.068	0.084
Oregon	2			0.18	0.053	0.072	0.058
Ohio	3				0.058	0.069	0.078
Mumbai	4					0.050	0.034
Sidney	5						0.031
Sao-paolo	6						

Figure 9: Gbps between AWS sites.

Based on the geographic region of the two parties, we can realize different network speeds, as illustrated in Table 9. The network speeds given in the table were measured using the iperf3 command.[7] This collection of AWS sites was chosen to give a large range of bandwidth performance.

6.2.3 Experimental setup: local benchmark.

The AWS benchmarks use a real network connection which is sometimes unpredictable. For a highly controlled experimental network, we benchmarked protocols on a single machine: Intel Xeon 2.30 GHz, 256GB RAM, 36 physical cores (note that all implementations are single-threaded unless otherwise indicated). We simulated a network connection using the Linux tc command, communicating via localhost network. We simulated a LAN setting with 10 Gbps network bandwidth and 0.2ms round-trip latency, and various WAN settings with 100 Mpbs, 10 Mpbs, 1 Mpbs and 80ms round-trip latency.

6.2.4 AWS Pricing Scheme.

Part of our motivation for evaluating protocols on AWS is to report and compare their real-world monetary costs. Hence we describe now the pricing scheme for AWS at the time of our comparison.[8] Costs are associated with both *running time* and *data transfer*, and both depend on the data center (geographic location) at which the instance runs.

The running-time cost per hour (in USD) for our instance type M5.large is 0.096 (USA), 0.101 (Mumbai), 0.12 (Sydney), 0.153 (Sao Paolo).

[6] https://github.com/osu-crypto/libOTe

[7] See https://iperf.fr/iperf-download.php.

[8] The pricing can be found in https://aws.amazon.com/ec2/pricing/on-demand/.

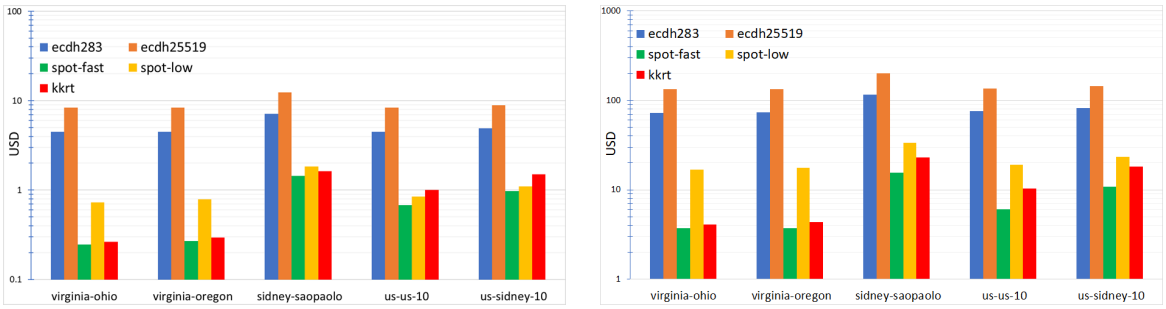


Figure 10: Monetary cost (in USD) per 1000 runs of PSI on 2^{16} (left) and 2^{20} (right) items, in the B2B network scenario.

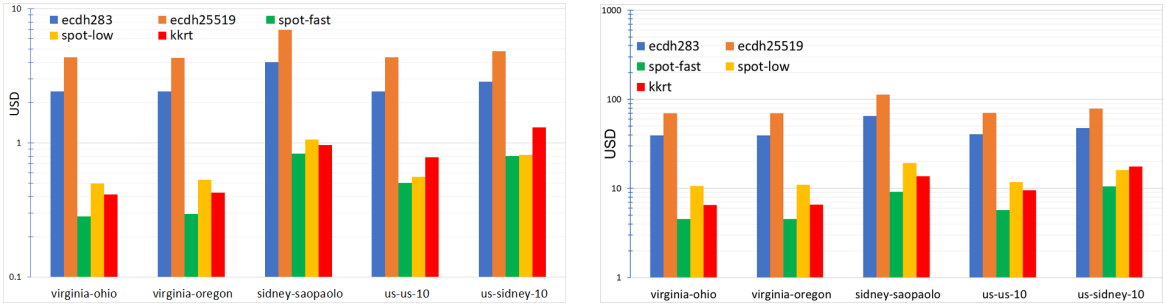


Figure 11: Monetary cost (in USD) per 1000 runs of PSI on 2^{16} (left) and 2^{20} (right) items, in the 'Internet' network scenario.

The data transfer cost differ depending on whether both endpoints are within AWS, and the data-center of the endpoints. We consider two network settings:

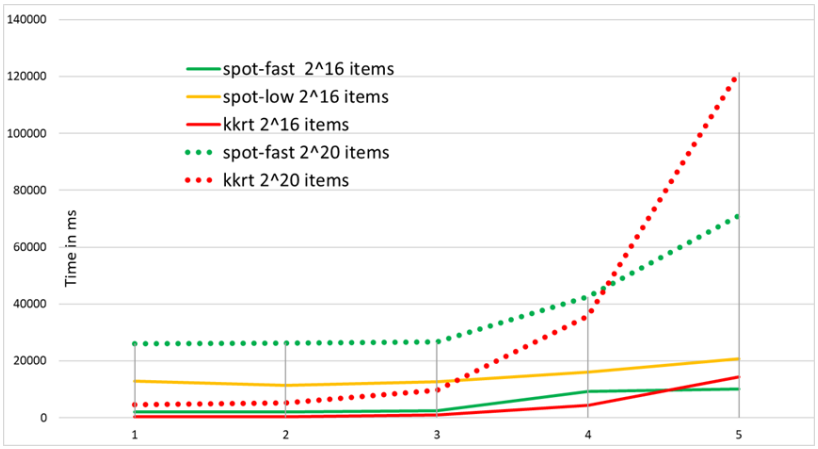
- In a **business-to-business (B2B)** setting between two fixed organizations that want to regularly perform PSI on their dynamic data, both endpoints may be within the AWS network.
- In an **internet** setting where one organization wishes to regularly perform PSI with a dynamically changing partner, only one party may be within the AWS network.

These considerations have the following effect on the cost of data transfer on AWS:

- Inbound data transfer from the Internet to EC2 is free.
- Outbound data transfer from EC2 to the Internet incurs the highest cost. Rates in USD per 1GB are 0.09 (USA), 0.1093 (Mumbai), 0.114 (Sydney), 0.25 (Sao Paolo).
- Outbound data transfer between two instance at the same site cost 0.01 USD/GB per direction.
- Outbound data transfer to another AWS site costs (in USD/GB): 0.02 (USA), 0.086 (Mumbai), 0.14 (Sydney) and 0.16 (Sao Paolo)
- Additional cost is for using a public IP address, which is indeed required for the scenarios we consider; this costs 0.01 USD/GB for all sites.

We compute the total monetary cost of a protocol execution as follows. Let T be the runtime in hours of the protocol; let X_1 and X_2 be the outbound communication of the first and second parties, resp.; let $\mathsf{C}_{\mathsf{T}1}, \mathsf{C}_{\mathsf{T}2}$ be the uptime rate of the machines run by the parties; and let $\mathsf{C}_{\mathsf{X}1}, \mathsf{C}_{\mathsf{X}2}$ be the outbound data transfer rates for the machines/regions of the parties. The cost in USD is then:

$$\mathsf{TotalCost} = \mathsf{T} \cdot (\mathsf{C}_{\mathsf{T}1} + \mathsf{C}_{\mathsf{T}2}) + \mathsf{X}_1 \cdot \mathsf{C}_{\mathsf{X}1} + \mathsf{X}_2 \cdot \mathsf{C}_{\mathsf{X}2} + 0.01 \cdot (\mathsf{X}_1 + \mathsf{X}_2)$$



1	9.6 Gb/s	Virginia-Virginia
2	1.08 Gb/s	Virginia-Ohio
3	0.17 Gb/s	Virginia-Oregon
4	0.031 Gb/s	Sidney-Sao Paolo
5	0.01 Gb/s	Virginia-Virginia (controlled b/w)

Figure 12: Evaluated run times over AWS EC2 with descending bandwidth. Solid and dotted lines are for PSI over 2^{16} and 2^{20} items respectively. The 1-5 numbers at the x-axis of the figure represent the configurations 1-5 described in the table to the right.

6.3 Experimental Results

6.3.1 AWS monetary cost.

To limit the number of protocol executions performed on AWS, we focus on set sizes of 2^{16} and 2^{20} as they are representative of realistic set sizes for aformentioned applications of PSI.

The monetary cost of PSI protocols is presented in Figures 10 and 11. We see that our spot-fast protocol variant is the cheapest protocol in all of the settings we consider. In the B2B scenarios it is 4%-35% for PSI of 2^{16} items and 10%-40% cheaper for PSI of 2^{20} items, compared to the second cheapest protocol (KKRT). In the 'Internet' scenarios it is 13%-38% cheaper for PSI of 2^{16} items and 30%-40% cheaper for 2^{20} items. The numerical costs can be found in Tables 3-6 in Appendix E.

6.3.2 Break-even point with KKRT.

Our protocol has less communication than the faster KKRT protocol. As the network becomes slower, the protocol becomes more network-bound and our advantage in communication eventually leads to faster performance than KKRT. We compared the running time of the PSI protocols on networks of different speeds, in order to identify the "break-even point" where our protocol (spot-fast) becomes faster than KKRT.

From the running times in Figure 12, we find that the spot-fast variant overtakes KKRT as the fastest PSI protocol when network bandwidth drops below the 10–30 Mbps range. The concrete times are detailed in Tables 7-8 in Appendix E.

6.3.3 Detailed, controlled local benchmarks.

A more detailed benchmark for set sizes $2^{12} - 2^{24}$ and controlled network configurations is given in Table 2. We also considered the effect of multi-threading on protocol performance, with $T \in \{1, 4\}$ threads. The implementation of KKRT does not support multi-threading.

The communication of our protocol is approximately $2\times$ smaller than that of [KKRT16]. For example, computing the intersection of sets of size $n = 2^{20}$, spot-fast and spot-low variants require 76.43 MB and 63.18 MB respectively, whereas [KKRT16] requires 127 MB of communication, (a $1.7 - 2.0\times$ improvement).

In a single-threaded LAN setting, spot-fast variant is several times slower than KKRT, requiring 25.62 seconds with $n = 2^{20}$. Applying the same parameters to [KKRT16] results in a running time of 4.1 seconds. The running time of spot-fast variant is improved significantly by multi-threading, improving to 7.61 seconds when utilizing 4 threads.

In the WAN setting, spot-fast becomes the fastest protocol on slow (10Mbps and 1Mbps) network, due to its lower communication cost. For example, in the 10Mpbs network, for sets of size $n = 2^{20}$, spot-fast takes 66.2 seconds, while [KKRT16] requires 120.13 seconds, a $1.8\times$ improvement.

Params.		Protocol	Comm.	Total time (seconds)							
				10 Gbps		100 Mbps		10 Mbps		1 Mbps	
n_1	n_2		(MB)	$T=1$	4	1	4	1	4	1	4
2^{24}	2^{24}	DH-PSI (K-283)	—	—	—	—	—	—	—	—	—
		DH-PSI (25519)	—	—	—	—	—	—	—	—	—
		KKRT	1955.2	**63.3**	—	**261.9**	—	1852.1	—	—	—
		spot-low	—	—	—	—	—	—	—	—	—
		spot-fast	1254.5	440.1	146.1	474.6	173.3	**1071.8**	**1062.8**	—	—
2^{20}	2^{20}	DH-PSI (K-283)	84.0	1141.8	338.5	1152.5	336.9	1158.2	334.2	1472.4	854.3
		DH-PSI (25519)	76.1	2110.6	632.8	2290.5	634.5	2325.7	673.0	2497.8	1014.0
		KKRT	127	**4.61**	—	**17.47**	—	120.1	—	1154.5	—
		spot-low	**63.1**	270.3	179.2	273.4	185.3	299.6	206.67	687.2	**311.16**
		spot-fast	76.4	25.6	**7.6**	27.8	**10.53**	**66.2**	**66.0**	**646.3**	645.3
2^{16}	2^{16}	DH-PSI (K-283)	5.2	69.8	20.20	70.77	21.93	71.10	22.8	80.1	44.4
		DH-PSI (25519)	4.7	136.9	39.4	140.4	40.1	142.8	40.8	151.3	48.2
		KKRT	8.06	**0.43**	—	**1.99**	—	8.4	—	74.5	—
		spot-low	**3.9**	12.8	8.8	13.7	9.8	15.1	10.9	41.1	**39.1**
		spot-fast	4.71	1.90	**0.77**	2.91	**2.02**	**5.46**	**5.36**	**40.19**	40.08
2^{12}	2^{12}	DH-PSI (K-283)	0.32	4.59	1.87	4.65	1.67	4.82	1.56	5.18	**2.75**
		DH-PSI (25519)	0.29	8.72	2.58	8.90	27.5	9.10	2.80	9.59	2.98
		KKRT	0.53	**0.22**	—	**0.87**	—	1.24	—	5.7	—
		spot-low	**0.25**	0.87	0.61	1.4	1.2	1.4	13.23	3.17	3.0
		spot-fast	0.3	0.4	**0.21**	1.14	**0.99**	**1.16**	**1.01**	**3.58**	3.51

Table 2: Total communication cost in MB and running time in seconds comparing our protocol to [KKRT16] and HD-PSI, with $T \in \{1, 4\}$ threads; each item has 128-bit length. 10Gbps network assumes 0.2ms RTT, and others use 80ms RTT. Cells with "—" denote setting not supported or program out of memory.

Both of our protocols outperformed DH-PSI. For example, spot-low requires 63 MB while DH-PSI (Curve25519) requires 76 MB, a $\sim 12\%$ improvement.

In terms of computation, even our slower spot-low variant is based on symmetric-key operations, and is significantly faster than DH-PSI. We also examined the effect of multi-threading. Similar to DH-PSI, spot-fast variant is extremely amenable to parallelization. Concretely, we parallelize our algorithm at the level of bins. Both DH-PSI and spot-fast yield a similar speedup of about 3.5× by using 4 threads.

References

[ACT11] Giuseppe Ateniese, Emiliano De Cristofaro, and Gene Tsudik. (if) size matters: Size-hiding private set intersection. In *PKC*, pages 156–173, 2011.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM CCS*, pages 535–548, 2013.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *STOC*, pages 479–488, 1996.

[Ber06] Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In *PKC*, pages 207–228, 2006.

[BPSW07] Justin Brickell, Donald E. Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In *ACM CCS*, pages 498–507, 2007.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CCS18] Andrea Cerulli, Emiliano De Cristofaro, and Claudio Soriente. Nothing refreshes like a repsi: Reactive private set intersection. In *Applied Cryptography and Network Security (ACNS)*. Springer Berlin Heidelberg, 2018.

[CDJ16] Chongwon Cho, Dana Dachman-Soled, and Stanislaw Jarecki. Efficient concurrent covert computation of string equality and set intersection. In *CT-RSA*, pages 164–179, 2016.

[CGT12] Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Fast and private computation of cardinality of set intersection and union. In *CANS 2012*, pages 218–231, 2012.

[CKT10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *ASIACRYPT*, pages 213–231, 2010.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *ACM CCS, 2017*, pages 1243–1255, 2017.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, 3rd Edition*. MIT Press, 2009.

[CRS03] Artur Czumaj, Chris Riley, and Christian Scheideler. Perfectly balanced allocation. In *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*, 2003.

[CT12] Emiliano De Cristofaro and Gene Tsudik. Experimenting with fast private set intersection. In *TRUST 2012*, pages 55–73, 2012.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *ACM CCS 2013*, pages 789–800, 2013.

[DRRT18] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. PIR-PSI: scaling private contact discovery. *Proceedings on Privacy Enhancing Technologies*, 2018(4):159–178, 2018.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28:637–647, 1985.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, 2005.

[FNO18] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Private set intersection with linear communication from general assumptions. ePrint Archive, Report 2018/238, 2018.

[GN19] Satrajit Ghosh and Tobias Nilges. An algebraic approach to maliciously secure private set intersection. EUROCRYPT, 2019.

[GNN17a] Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. Maliciously secure oblivious linear function evaluation with constant overhead. Cryptology ePrint Archive, Report 2017/409, 2017. http://eprint.iacr.org/2017/409.

[GNN17b] Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. Maliciously secure oblivious linear function evaluation with constant overhead. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 629–659. Springer, Heidelberg, December 2017.

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

[HFH99] Bernardo A. Huberman, Matthew K. Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *EC*, pages 78–86, 1999.

[HL08] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC*, pages 155–175, 2008.

[HMFS17] Xi He, Ashwin Machanavajjhala, Cheryl J. Flynn, and Divesh Srivastava. Composing differential privacy and secure computation: A case study on scaling private record linkage. In *ACM CCS*, pages 1389–1406, 2017.

[HS13] Wilko Henecka and Thomas Schneider. Faster secure two-party computation with less memory. In *ASIA CCS*, pages 437–446, 2013.

[HV17] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Scalable multi-party private set-intersection. In Serge Fehr, editor, *PKC 2017, Part I*, volume 10174 of *LNCS*, pages 175–203. Springer, Heidelberg, March 2017.

[IKN+17] Mihaela Ion, Ben Kreuter, Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. Private intersection-sum protocol with applications to attributing aggregate ad conversions. *ePrint Archive 2017/738*, 2017.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, pages 145–161, 2003.

[IPS09] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 294–314. Springer, Heidelberg, March 2009.

[IR88] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *CRYPTO*, pages 8–26, 1988.

[KK13] Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In *CRYPTO*, pages 54–70, 2013.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched OPRF with applications to PSI. In *ACM CCS*, 2016.

[KLS+17] Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. Private set intersection for unequal set sizes with mobile applications. *PoPETs*, 2017(4):177–197, 2017.

[KMP+17] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multiparty private set intersection from symmetric-key techniques. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 1257–1272. ACM Press, October / November 2017.

[Knu05] Donald E Knuth. The art of computer programming, volume 4: Generating all combinations and partitions, fascicle 3, 2005.

[Lam16] Mikkel Lambæk. Breaking and fixing private set intersection protocols. Master's thesis, Aarhus University, 2016.

[LPSY15] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round mpc combining BMR and SPDZ. In *CRYPTO*, 2015.

[MB72] R. Moenck and Allan Borodin. Fast modular transforms via division. In *Switching and Automata Theory*, pages 90–96, 1972.

[Mea86] C. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *IEEE S&P*, 1986.

[MPP10] Mark Manulis, Benny Pinkas, and Bertram Poettering. Privacy-preserving group discovery with linear complexity. In *ACNS 2010*, pages 420–437, 2010.

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *31st ACM STOC*, pages 245–254. ACM Press, May 1999.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *12th SODA*, pages 448–457. ACM-SIAM, January 2001.

[PSS17] Arpita Patra, Pratik Sarkar, and Ajith Suresh. Fast actively secure OT extension for short secrets. In *NDSS*, 2017.

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX 2015*, 2015.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *USENIX 2014*, pages 797–812, 2014.

[PSZ18] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. *ACM Trans. Priv. Secur.*, 21, 2018.

[RA17] Amanda C. Davi Resende and Diego F. Aranha. Unbalanced approximate private set intersection. *ePrint Archive 2017/677*, 2017.

[Rab05] Michael O. Rabin. How to exchange secrets with oblivious transfer. *ePrint Archive 2005/187*, 2005, 2005.

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

[SEK03] Peter Sanders, Sebastian Egner, and Jan Korst. Fast concurrent access to parallel disks. *Algorithmica*, 35(1):21–55, 2003.

[Sha80] Adi Shamir. On the power of commutativity in cryptography. In *Automata, Languages and Programming*, 1980.

[TKC07] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Utku Celik. Privacy preserving error resilient dna searching through oblivious automata. In *ACM CCS*, pages 519–528, 2007.

[TLP+17] Sandeep Tamrakar, Jian Liu, Andrew Paverd, Jan-Erik Ekberg, Benny Pinkas, and N. Asokan. The circle game: Scalable private membership test using trusted hardware. In *ASIA CCS*, pages 31–44, 2017.

A Security Proofs

A.1 Semi-Honest Security

Below we give the details of the proof of Theorem 3.4.

Corrupt Alice. The simulator observes Alice's inputs to $\mathcal{F}_{\mathsf{ROT}}$ primitive and gives random q_i as OT outputs in Step 2. The only other message Alice receives is the polynomial P in step 4. Instead of $P := \mathsf{Interp}_{\mathbb{F}}(\{y, R(y)\}_{y \in Y})$, the simulator sends uniformly random polynomial to Alice.

We formally show the simulation by proceeding the following hybrids:

Hybrid 1.. The first hybrid is the real interaction described in Figure 5. Here, a honest Bob uses his input Y, honestly interacts with the corrupt Alice via $\mathcal{F}_{\mathsf{ROT}}$, and builds the polynomial P. We observe Alice's choice bits s used in step 2.

Hybrid 2.. In this hybrid, we consider each value of bit s_i. We replace $\{F(t_i, y) \mid y \in Y\}$ with uniformly random bits if $s_i = 0$, and $\{F(u_i, y) \mid y \in Y\}$ with uniformly random bits if $s_i = 1$. Alice knows one of $\{t_i, u_i\}$ from $\mathcal{F}_{\mathsf{ROT}}$, and the output of pseudo-random function F is uniformly distributed. Hence, the modification is indistinguishable. Note that in this hybrid, all $R(y)$ values are random from Alice's view.

Hybrid 3.. This hybrid chooses the polynomial P uniformly at random, as in the final simulation. Note that in the previous hybrid P is chosen as $P := \mathsf{Interp}_{\mathbb{F}}(\{y, R(y)\}_{y \in Y})$. From Proposition 3.1, this distribution on P is uniform; hence, the hybrids are identically distributed.

Corrupt Bob. The simulator for a corrupt Bob first obtains $X \cap Y$ from the ideal PSI functionality. It simulates random outputs t_i, q_i from $\mathcal{F}_{\mathsf{ROT}}$. The only other message received by Bob is the set $\mathcal{O}$ in step 5. To simulate this message, the simulator computes $n' = n_1 - |X \cap Y|$ and uniformly samples values $z_1, \ldots, z_{n'}$. It then simulates Alice's message as (a random permutation of) $\mathcal{O} = \{H(T(x)) \mid x \in X \cap Y\} \cup \{z_1, \ldots, z_{n'}\}$.

To formally prove the security of this simulation, we consider a sequence of hybrids:

Hybrid 1.. The first hybrid is the real interaction described in Figure 5. Here, a honest Alice uses her input X, honestly interacts with the corrupt Bob via $\mathcal{F}_{\mathsf{ROT}}$, computes $\mathcal{O}$ and sends it to Bob.

Hybrid 2.. In this hybrid, before doing anything else the simulation predicts what the function R will be (recall that the function R is defined in terms of the $\mathcal{F}_{\mathsf{ROT}}$ outputs), and uses Bob's input Y to predict what the polynomial P will be. The simulation immediately aborts in the event that $P(x) \oplus R(x)$ has hamming weight less than κ for any value $x \in X \setminus Y$. Otherwise, the simulation continuse as in Hybrid 1.

Note that the simulator aborts exactly when $\mathsf{BadPoly}^R_{\mathbb{F}}(X, Y) = 1$ (defined in Definition 3.2), where R is the function defined in the protocol. To show that the hybrids are indistinguishable, it suffices to show that $\Pr[\mathsf{BadPoly}^R_{\mathbb{F}}(X, Y) = 1]$ is negligible.

Following Proposition 3.3, we have chosen parameters in Table 6 so that the desired probability is at most 2^{λ}. However, Proposition 3.3 deals with the case that R is a random function, whereas in this hybrid R is derived from a PRF. What's more, corrupt Bob sees the keys to those PRFs (the $\boldsymbol{t_i}$'s and $\boldsymbol{u_i}$'s)!

Fortunately it does not matter that Bob's view includes the key material for the R function. The algorithm $\mathsf{BadPoly}_{\mathbb{F}}$ (*i.e.*, the logic that determines whether this simulation aborts) needs only black-box access to R. Hence the output probability of $\mathsf{BadPoly}_{\mathbb{F}}$ is changed only by a negligible amount when using a random R or R defined in terms of a PRF as in the protocol.

In other words, the probability that this simulation aborts is at most $1/2^{\lambda}$ plus a negligible function, so the hybrids are indistinguishable.

Hybrid 3.. In the previous hybrid, $\mathcal{O}$ is being computed as

$$\begin{aligned}&\{H(Q(x) \oplus s \cdot P(x)) \mid x \in X\}\\ &\quad = \{H(T(x)) \mid x \in X \cap Y\} \cup \{H(T(x) \oplus s \cdot \underbrace{(P(x) \oplus R(x))}_{\text{hamming weight} \geq \kappa}) \mid x \in X \setminus Y\}\end{aligned}$$

Conditioned on not aborting, $P(x) \oplus R(x)$ has hamming weight at least κ for $x \in X \setminus Y$. Hence, the corresponding outputs of H are pseudorandom because of the κ-Hamming correlation robust property of H (Definition 2.1). In this hybrid, we modify $\mathcal{O}$ to be computed as in the final simulation. That is, the simulator chooses $z_1, \ldots, z_{n'}$ where $n' = |X \setminus Y|$ and computes $\mathcal{O}$ as:

$$\{H(T(x)) \mid x \in X \cap Y\} \cup \{z_1, \ldots, z_{n'}\}$$

Hybrid 4.. This hybrid no longer artificially aborts when $P(x) \oplus R(x)$ has low hamming weight for $x \in X \setminus Y$. The change is indistinguishable for the same reason as before. Now the hybrid no longer uses any information about the items in $X \setminus Y$, and it corresponds to our final simulation.

A.2 Security Against Malicious Sender

Hybrid 1.. This is the real interaction in which the simulator honestly plays the role of $\mathcal{F}_{\mathsf{ROT}}$ and Bob, with input Y.

Hybrid 2.. In the semi-honest proof, we first replace all $F(t_i, y)$ with uniformly random bits if $s_i = 0$, and all $F(u_i, y)$ with uniformly random bits if $s_i = 1$. In this case, F is a random oracle rather than a PRF, so these values are already uniform. Instead, we abort if Alice makes an oracle query of the form $F(t_i, \cdot)$ for $s_i = 0$ or $F(u_i, \cdot)$ for $s_i = 1$. The probability of such an event is negligible and, conditioned on not aborting, all $R(y)$ values are random from Alice's point of view.

Hybrid 3.. Instead of computing $P := \mathsf{Interp}_{\mathbb{F}}(\{y, R(y)\}_{y \in Y})$, the simulator sends a random polynomial. When computing honest Bob's output, the simulator checks for $H(Q(y) \oplus s \cdot P(y)) \in \mathcal{O}$ instead of checking for $H(T(y)) \in \mathcal{O}$. These changes have no effect on the adversary's view.

Hybrid 4.. In this hybrid, the simulator computes $\widetilde{X}$ as described above, and it artificially aborts if honest Bob outputs an item $y \notin \widetilde{X}$. There are two cases for such a y:

- If $y \notin C$ then the adversary has no information about $Q(y)$ — this value is distributed independently of the adversary's view. Then the target value $H(Q(y) \oplus s \cdot P(y))$ is random only fixed in step 6 when Bob computes his output — i.e., after Alice has already sent $\mathcal{O}$ in step 5. If the output of H is σ bits, then it is only with probability $|\mathcal{O}|/2^\sigma = n_1/2^\sigma$ that the target value is in $\mathcal{O}$, causing Bob to include y in the output.
- If $y \in C$ but $y \notin \widetilde{X}$, it means that the value $H(Q(y) \oplus s \cdot P(y))$ has been fixed but is not in $\mathcal{O}$. In this case, Bob will never include y in the output.

Taking a union bound over the values $y \in Y \setminus \widetilde{X}$, the total probability of abort is $n_1 n_2 / 2^\sigma$, which we assume to be negligible by the choice of σ.

Hybrid 5.. In the previous hybrid the simulation computes $\widetilde{X}$ in step 5 by just observing Alice's behavior (including random oracle queries), then Bob outputs $Y \cap \widetilde{X}$. Hence, we can modify the interaction so that the simulator sends $\widetilde{X}$ to the ideal PSI functionality, Bob sends Y, and Bob receives $\widetilde{X} \cap Y$ as output instead. The adversary's view is identical to the previous hybrid, and this hybrid correponds to our final simulation.

A.3 Insecurity against Malicious Receiver

We now demonstrate a concrete attack against our protocol, which can be carried out by a malicious receiver. Let $X = \{x_1, \ldots, x_n\}$ be the sender's set. Our attack will allow the receiver to perform a dictionary attack to determine X, after the protocol has finished. That is, given any $\tilde{x}$, the receiver can easily test whether $\tilde{x} \in X$ by merely inspecting the protocol transcript. Such an ability is inconsistent with security.

The attack requires that the receiver knows ℓ items, denoted $x_1, \ldots, x_\ell$, in X, where ℓ is the number of base OTs and the number of bits in the sender's private randomness $\boldsymbol{s}$. Generally speaking, $\ell \ll n$.

The receiver is supposed to generate a polynomial P such that $P(y) = R(y)$ for every $y \in Y$, where R is a target value determined by the OT extension process. Instead, the malicious receiver interpolates a polynomial $\tilde{P}$ so that $\tilde{P}(x_i) = R(x_i) \oplus e_i$ for $i \in [\ell]$, where e_i is a string with a 1 in position i and 0s everywhere else. The polynomial $\tilde{P}$ can be interpolated to go through any other desired points — such points don't matter.

Now the sender will send $H(Q(x) \oplus s \cdot \tilde{P}(x))$ for all $x \in X$. From Equation 1, this is equal to:

$$H(T(x) \oplus s[\tilde{P}(x) \oplus R(x)])$$

In particular, for the special items $x_1, \ldots, x_\ell$, the corresponding value is:

$$H(T(x_i) \oplus s \cdot e_i) = \begin{cases} H(T(x_i)) & \text{if } s_i = 0 \\ H(T(x_i) \oplus e_i) & \text{if } s_i = 1 \end{cases}$$

In particular, there are only two choices for this value, depending on a *single bit* of s. For each i, the receiver can compute both of the above possibilities and check which one was included by the sender. In this way the receiver can learn all the bits of s. Note that s is the only randomness of Alice in the protocol.

Later on, to test whether the sender held an item $\tilde{x}$, the receiver simply checks whether $H(T(x) \oplus s[\tilde{P}(x) \oplus R(x)])$ was sent by the sender. All of the values in this expression are now known to the receiver.

A.4 Insecurity of the Fast Variant Against Malicious Receiver

The speed-optimized protocol is **no longer** secure against a malicious sender! The problem is that the sender is supposed to send a pair of masks for each item $x \in X$, but nothing stops a malicious sender from sending the OT-extension output for $x\|1$ but omitting the value for $x\|2$. If the sender does this, then Bob will include x in his output iff he assigned x to location $h_1(x)$. But the final bin assignment of x depends (indirectly) on *all* of Bob's input items. Hence, this behavior of a malicious sender (apparently) cannot be simulated in the ideal world.

B Reducing Alice's Communication

Let us abstract the last 2 steps of the basic PSI protocol. Alice computes a value $A(x) = H(Q(x) \oplus \boldsymbol{s} \cdot P(x))$ for each $x \in X$ and sends all $A(x)$ values to Bob as an unstructured set $\mathcal{O}$. Bob computes $B(y) = H(T(y))$ for each $y \in Y$ and outputs $\{y \mid B(y) \in \mathcal{O}\}$.

B.1 Difference Encoding of Tamrakar *et al.*

In our basic protocol, the $A(x)$ values have length $\sigma = \lambda + \log(n_1 n_2)$ bits. With overwhelming probability, each of these items is distinct. Interpret these values as numbers in the range $\{0, \ldots, 2^\sigma - 1\}$ and suppose they are sorted. Since there are n_1 values, the average difference between consecutive values is $2^\sigma / n_1 = 2^{\lambda + \log n_2}$. This suggests that by sending only the *differences* between items, Alice could send expect to send only $\lambda + \log n_2$ bits per item.

Tamrakar *et al.* [TLP+17] use this idea in their PSI protocol, and suggest the following difference encoding. Let $A_1 < \ldots < A_{n_1}$ denote the values that Alice wishes to send. If $A_i - A_{i-1}$ can be written in $\lambda + \log n_2 + 2$ bits, then it is sent as an integer with that length. Otherwise, an all-zeroes string of $\lambda + \log n_2 + 2$ bits is sent, and A_i is sent explicitly. The two cases are easily distinguished, since $A_i - A_{i-1}$ is never zero (except with negligible probability).

Tamrakar *et al.* use a simulation to argue that only roughly 2% of the differences $A_i - A_{i-1}$ exceed $\lambda + \log n_2 + 2$ bits (*i.e.*, are 4 times larger than the expectation). Hence the total communication of this approach is roughly $1.02(\lambda + \log_2 n_2 + 2)n_1$, and therefore Bob can easily run a decoding of this message in linear time. Note that we are simply encoding/compressing the set of values sent by Alice, and not introducing any additional failure probability. The change also has no effect on security. Suppose that in the unaltered protocol, Alice sends the $A(x)$ values in a random order. Then this message distribution is computable from her "compressed" message, and vice-versa. Hence, both the original and modified protocol leak the same information.

Note that other methods of encoding Alice's message are possible: For example, Alice can send a 0 followed by $A_i - A_{i-1}$, if the difference fits in $\lambda + \log n_2 + 2$ bits, or send a 1 followed by A_i otherwise. One can even use an optimal encoding (ranking/unranking) scheme for k-combinations of $[N]$ — *e.g.*, using a combinatorial number system [Knu05]. Either of these would save a marginal amount of communication (for example, when $n = 2^{20}$ it saves 16 bits per item, hence only 2.1 MB in total). The optimization would be much more expensive or cumbersome to implement since it requires all OPRF outputs to be computed and sorted, but without this optimization they can be sent as they are computed. The difference encoding optimization can indeed be applied to other PSI protocols as well (DH-PSI, KKRT, and spot-fast), but we only implement this optimization in our spot-low.

B.2 New Polynomial Encoding

Let us review why the $A(x)$ values in our protocol have length $\lambda + \log(n_1 n_2)$. If for any $x \in X$ and $y \in Y$, we have $x \neq y$ and $A(x) = B(y)$, then Bob's output will be incorrect. To bound the correctness error by $2^{-\lambda}$ we consider a union bound over all $|X| \times |Y|$, and set the length of these strings to be $\lambda + \log_2(n_1 n_2)$ bits.

We suggest a different method based on polynomial interpolation: Instead of sending the $A(x)$ hash values back to Bob as a set, Alice interpolates and sends a polynomial π such that $\pi(x) = A(x)$ for all $x \in X$. Then Bob can output $\{y \mid B(y) = \pi(y)\}$. Now correctness is only violated when $y \notin X$ and yet $\pi(y) = B(y)$. The analysis of correctness error now involves a union bound over only $|Y|$ events. As such, the length of these values now can be $\lambda + \log_2(n_2)$ bits, a savings of $\log_2(n_1)$ bits per item. Note that since $A(x)$ values for $x \in X \setminus Y$ look random to Bob, so the polynomial π hides these x-values.

This approach reduces communication to exactly $\lambda + \log_2(n_2)$ bits per item of Alice. Note however that we can only consider a polynomial mapping items x to $A(x)$ if the length of items is less than the length of $A(x)$. For example, with 2^{20} items and statistical security 2^{-40}, the length of $A(x)$ is 60 bits, so this optimization works whenever the PSI items are 60 bits or less.

This optimization involves an expensive interpolation of a high-degree polynomial by Alice. In this case we *cannot* use the 2-choice hashing technique of Section 4 to reduce the cost of this interpolation. To see why, suppose Alice chooses $h'_1, h'_2 : \{0,1\}^* \to [m]$ (independent of the hash functions Bob would use in Section 4), assigns her items to bins, and in each bin interpolates a polynomial of the relevant $A(x)$ values. Then Bob would be able to detect whether Alice has placed item x in location $h'_1(x)$ vs $h'_2(x)$ (this is possible since Bob identifies in which bin he sees a match). Unfortunately, this leaks some minor information about Alice's set, and cannot be simulated. Specifically, the choice of placing x at $h'_1(x)$ vs $h'_2(x)$ depends indirectly on Alice's *entire* set X.

C Fast Interpolation and Multi-point Evaluation

Two problems that the parties in our protocol have to solve are:

- **Interpolation.** Given a set of n points $\{(x_i, y_i)\}_{i \in [n]}$ where $x_i, y_i \in \mathbb{F}$, output the (unique) polynomial $P(x)$ over $\mathbb{F}$ that satisfies $P(x_i) = y_i$ for all $i \in [n]$.
- **Multi-point Evaluation.** Given a polynomial $P(x)$ and a set of n elements $x_1, \ldots, x_n \in \mathbb{F}$, output $P(x_1), \ldots, P(x_n)$.

It is known that these problems could be solved in $O(n^2)$ arithmetic operations, however, when n is very large (e.g. 2^{20}) this becomes impractical. In the following we describe the algorithms of Moenck and Borodin from 1972 [MB72] to solve these problems in $O(n \log^2 n)$ arithmetic operations, which make them a better fit for our needs.

Binary tree representation.. Denote a binary tree with n leafs by $T(n)$. In the following we may refer to $T(n)$ (with n a power of 2) as a simple array of size $2 \cdot n - 1$, denoted $A(2 \cdot n - 1)$ (and has entries $A[0], \ldots, A[2 \cdot n - 2]$. This is a common representation of a binary tree, such that the i-th leafs resides at $A[n - 2 + i]$. For an internal node $A[i]$, its left and right childs reside at $A[2 \cdot i]$ and $A[2 \cdot i + 1]$ respectively and if $i \neq 0$ its parent resides at $A[(i-1)/2]$.

C.1 Detailed Description of the Algorithms

In a high level the algorithms can be separated into sub-algorithms as follows:

C.1.1 Interpolation.

The input to this algorithm is the set $\{(x_i, y_i)\}_{i\in[n]}$ where $x_i, y_i \in \mathbb{F}$. The output is a polynomial $P(x)$ over $\mathbb{F}$ s.t. $P(x_i) = y_i$ for all $i \in [n]$.

1. *Construct a sub-product tree.* A sub-product tree is a binary tree whose nodes and leafs contains polynomials. Its i-th leaf contain the degree-1 polynomial $(x - x_i)$ and each node contains the polynomial that is the multiplication of the polynomials of its childs. For example, let $M(2\cdot n-1)$ be the array representation of this tree, then $M[0] = \prod_{i\in[n]}(x - x_n)$, $M[1] = \prod_{i\in[n/2]}(x - x_n)$, $M[2] = \prod_{i\in[n/2+1,n]}(x - x_n)$ and so on.

2. *Calculate derivative.* In this step we calculate the derivative of $M[0]$ that was produced before. Let $M(x)$ be the polynomial at the root of M (i.e. $M[0]$), its derivative is:

$$\begin{aligned} M'(x) &= \left(\prod_{i\in[n]}(x - x_n)\right)' \\ &= \left(\prod_{i\in[n]}(x - x_i)\right) \cdot \left(\sum_{i\in[n]} \frac{1}{(x - x_i)}\right) \\ &= \sum_{j\in[n]} \prod_{\substack{i\in[n] \\ i\neq j}} (x - x_i) \end{aligned}$$

3. *Evaluate derivative.* When evaluating $M'(x)$ on x_k for $k \in [n]$ we get:

$$\begin{aligned} M'(x_k) &= \sum_{j\in[n]} \prod_{\substack{i\in[n] \\ i\neq j}} (x_k - x_i) \\ &= \sum_{\substack{j\in[n] \\ j\neq k}} \prod_{\substack{i\in[n] \\ i\neq j}} (x_k - x_i) + \sum_{j=k} \prod_{\substack{i\in[n] \\ i\neq k}} (x_k - x_i) \\ &= \prod_{\substack{i\in[n] \\ i\neq k}} (x_k - x_i) \end{aligned}$$

   since $\sum_{\substack{j\in[n] \\ j\neq k}} \prod_{\substack{i\in[n] \\ i\neq j}} (x_k - x_i) = 0$. We denote $a'_i = M'(x_i)$. In this step we evaluate M' over $x_1 \ldots, x_n$ and obtain $a'_1, \ldots, a'_n$.

4. *Actual interpolation.* Using the Lagrange interpolation formula the desired polynomial P can be written as:

$$P(x) = \sum_{i\in[n]} y_i \cdot L_i(x)$$

   where $L_i(x) = \frac{\prod_{\substack{j\in[n]^{(x-x_j)} \\ j\neq i}}}{\prod_{\substack{j\in[n]^{(x_i-x_j)} \\ j\neq i}}} = \begin{cases} 1 & x = x_i \\ 0 & x = x_k \neq x_i \end{cases}$.
   Now, setting $a_i = \frac{1}{\prod_{\substack{j\in[n]^{(x_i-x_j)} \\ j\neq i}}}$ we get

$$P(x) = \sum_{i\in[n]} y_i \cdot a_i \prod_{\substack{j\in[n] \\ j\neq i}} (x - x_j)$$

To enable the "divide and conquer" method to the above, we write:

$$\begin{aligned} P(x) &= \left(\prod_{i=\frac{n}{2}+1}^{n}(x-x_i)\right)\cdot\left(\sum_{k=1}^{\frac{n}{2}} y_k\cdot a_k\left(\prod_{\substack{i=1\\ i\neq k}}^{\frac{n}{2}}(x-x_i)\right)\right) \\ &+ \left(\prod_{i=1}^{\frac{n}{2}}(x-x_i)\right)\cdot\left(\sum_{k=\frac{n}{2}+1}^{n} y_k\cdot a_k\left(\prod_{\substack{i=\frac{n}{2}+1\\ i\neq k}}^{n}(x-x_i)\right)\right) \end{aligned}$$

Note that $\prod_{i=\frac{n}{2}+1}^{n}(x-x_i)$ and $\prod_{i=1}^{\frac{n}{2}}(x-x_i)$ were already computed in sub-algorithm (1) and are equal to $M[1]$ and $M[2]$ respectively. Moreover note that $a_i = 1/a'_i$ where all a_i's were computed in sub-algorithms (2) and (3).

Putting it all together, let $Y = y_1,\ldots,y_n$, $A = a_1,\ldots,a_n$ and the sub-product tree of polynomials M be global objects precomputed by sub-algorithms (1-3) above, interpolation is done by running INTERPOLATE(0) where INTERPOLATE is defined in Algorithm 2. The procedure ISLEAF(i) simply returns whether the index i is a leaf in an array that represent a binary tree with n leafs (recall that such an array has $2\cdot n-1$ entries indexed $0,\ldots,2\cdot n-2$. More formally ISLEAF(i) $= \left(i \stackrel{?}{\in} [n-2, 2\cdot n-2]\right)$ where $x \stackrel{?}{\in} X = 1$ if and only if $x \in X$. In addition, the procedure LEAFNUM(ℓ) is given a leaf index $\ell \in [0, 2\cdot n-2]$ in an array representing a binary tree and returns the index of the leaf *among the leafs*, i.e. a number from $[n]$. Formally, LEAFNUM(ℓ) $= \ell - n + 2$. Finally, we define the procedures LEFT(i)$= 2\cdot i$ and RIGHT(i)$= 2\cdot i+1$ to obtain the indexes of the childs for a node indexed i.

Algorithm 2 The algorithm for interpolation.

```
1: procedure INTERPOLATE(i)
2:     if ISLEAF(i) then
3:         j ← LEAFNUM(i).
4:         return y_j · a_k
5:     ℓ ← LEFT(i); r ← RIGHT(i).
6:     P_ℓ ← INTERPOLATE(ℓ); P_r ← INTERPOLATE(r).
7:     return P_ℓ · M[r] + P_r · M[ℓ].
```

C.1.2 Multi-point evaluation.

The input to this algorithm is a degree-(n-1) polynomial P and the set $X = \{x_1,\ldots,x_n\}$. The output is the set $Y = \{P(x_1),\ldots,P(x_n)\}$.

1. *Construct a sub-product tree.* This is the same sub-algorithm as in the interpolation, which returns the sub-product tree of polynomials M.
2. *Actual evaluation.* This part exploits the following:
   - Let $A(x), B(x)$ be two polynomials, then there exist two polynomials $Q(x)$ and $R(x)$ such that
     $$A(x) = Q(x)\cdot B(x) + R(x)$$
     and $\deg R < \deg B$. We write $R = A\%B$.
   - Let $A(x), B(x)$ be two polynomials, and let $x_1,\ldots,x_m$ be roots of $B(x)$, then for every $i \in [m]$ it holds that
     $$A(x_i) = (A\%B)(x_i)$$
     This is correct since $A(x) = Q(x)\cdot B(x) + R(x)$ and $B(x_i) = 0$ as x_i is B's root, therefore $A(x_i) = R(x_i)$. Finally, observe that if $B(x)$ is of degree 1, as happens in the leafs of M, then R has degree 0 (i.e. R is a constant).

Using the above, the multi-point evaluation algorithm is presented in Algorithm 3. We assume that the sub-product tree of polynomials M (constructed using elements $x_1, \ldots, x_n$) is a global object. We assume a pre-allocated, initially empty, global array $Y(n)$ with indices $[n]$ that is being filled by the algorithm, that is, $Y[i]$ for $i \in [n]$ will be assigned with $P(x_{i+1})$. The algorithm is given i as the root of the current sub-tree to process and P as the current polynomial to evaluate. The algorithm is executed with MultipointEvaluate$(0, P)$.

Algorithm 3 The algorithm multi-point evaluation.

```
1:  procedure MultipointEvaluate(i, A)
2:      B = M[i].
3:      R = A%B.
4:      if IsLeaf(i) then
5:          j ← LeafNum(i).
6:          Y[j − 1] = R.                                  ▷ R is a constant.
7:          return .
8:      ℓ ← Left(i); r ← Right(i).
9:      MultipointEvaluate(ℓ, R).
10:     MultipointEvaluate(r, R).
```

C.2 Concrete Times

Times for each of the above sub-algorithms were measured separately and presented in Figure 13.

C.3 Parallelization

There are two options to use multithreading for the interpolation and evaluation algorithms:

- *All slices at once.* Meaning that if we have $\rho = \beta/\alpha$ slices then we can use up to ρ completely independent threads to process them simultaneously.
- *Boost each slice separately.* Meaning that we use as many threads as possible to speedup the processing of an individual slice. This is possible due to the convenient tree-based structure of the algorithms (more details are given in Appendix C). That is, for each slice, we can assign each thread the task of processing a completely independent sub-tree and then combine all results to obtain the lowest levels of the tree and finally its root.

In our implementation we chose the second options as it is much more flexible and can leverage any number of threads and separate the work almost equally between them. On the contrary, in the first option, if the number of threads does not divides ρ then there is clearly a waste of computational power.

D Choice of Finite Field

It would be natural to implement the polynomial algorithms (interpolation and evaluation) over $GF(2^\ell)$ as the parties use the bit-string output of the OT extension as input to these algorithms. However, our implementation for the low communication variant (Figure 5) works over a prime field since existing libraries[9] for polynomial multiplication and division have optimizations for this field type but not for the extension fields. As such, we leave it to a future work to implement polynomial multiplication and division over fields that allow for fast arithmetics (e.g. extension fields or prime fields with Mersenne or Fermat primes). Therefore, in order to capture the ℓ-bit inputs/outputs of the polynomial we use the smallest prime p such that $p > 2^\ell$. From the density of primes it is guaranteed that the gap $p - 2^\ell$ is negligible (in ℓ) and does not affect the security of the protocol (this is a known method, see [LPSY15, Sec 3.1] for example). For $\ell = 436$ which we use in our implementation we have $p = 2^\ell + 295$, for which $p \approx 2^\ell(1 + 1.66 \cdot 10^{-129})$.

[9] e.g. http://www.shoup.net/ntl

β	Evaluation				Interpolation			
	n	2^{18}	2^{19}	2^{20}	n	2^{18}	2^{19}	2^{20}
64	B.T.	2092	4702	10616	B.T.	2153	4333	9406
	Eval.	5054	11146	24458	Der.	22	43	86
					Eval.	5030	10644	23202
					Int.	3378	6867	14549
	Total	7146	15848	35074	Total	10585	21889	47246
128	B.T.	2820	6044	13959	B.T.	2765	6012	12889
	Eval.	7032	15581	35835	Der.	21	42	86
					Eval.	6934	15337	33144
					Int.	4343	9246	19471
	Total	9852	21625	49794	Total	14065	30638	65592
256	B.T.	4755	10811	24015	B.T.	4721	10551	23513
	Eval.	12685	29010	61905	Der.	33	66	135
					Eval.	12564	27857	62862
					Int.	7054	15174	33338
	Total	17440	39281	85920	Total	24373	53651	119849
512	B.T.	8428	19227	42209	B.T.	8232	18389	43995
	Eval.	24482	56687	125394	Der.	37	74	173
					Eval.	24212	53797	119701
					Int.	12235	26322	69921
	Total	32910	75914	167603	Total	44718	98584	233791

Figure 13: Time in millisecond to execute the multi-point evaluation and interpolation sub-algorithms over fields of sizes $2^{64} - 2^{512}$ and $n \in \{2^{18}, 2^{19}, 2^{20}\}$. Abbreviations: Build-Tree (B.T.), Evaluation (Eval.), Derivation (Der.) and Interpolation (Int.). Runtimes obtained over Intel(R) Core(TM) i7-4800MQ CPU @ 2.70GHz with 1 thread.

We applied our slicing technique described in Section 5.2 and found that the number of slices that performs best is 3. For example, instead of interpolating a single polynomial with points (x_i, y_i) in $\{\{0,1\}^{128}, \{0,1\}^{436}\}$ (see Figure 6) we interpolate three polynomials P_1, P_2, P_3 where $P_{j\in\{1,2\}}$ are interpolated with points (x_i, y_i^j) in $\{\{0,1\}^{128}\}^2$ and the last polynomial P_3 with $(x_i||0\dots0, y_i^3)$ in $\{\{0,1\}^{180}\}^2$. Here $y_i = y_i^1||y_i^2||y_i^3$ which y_i^1, y_i^2 are of length 128, and y_i^3 is of length 180. Note that the x_i's are the items of the interpolating party, and that this means that our protocol works and performs exactly the same for every domain of elements $\{0,1\}^d$ for $d \leq 128$. This is in contrast to other protocols with a different performance for each domain size.

For our fast protocol (Figure 7), Bob interpolates a small polynomial within a bin. Our empirical experiments show that the straightforward Lagrange interpolation and straightforward polynomial evaluation give the best results for this variant. We interpolate and evaluate the polynomial over $GF(2^\ell)$.

E Supplementary Evaluation Results

In Tables 3–8 we report the concrete numerical costs and running times of our various benchmarks on the AWS network.

F PSI with Unequal Set Size

Several protocols are optimized for PSI with sets of significantly different sizes. We compare the performance of these protocols to our own, for set sizes $n_1 = \{2^{16}, 2^{20}, 2^{24}\}$ and $n_2 = 11041$. These parameters were chosen to match the numbers used in by Chen, Laine, and Rindal [CLR17]. $n_2 = 11041$ is one of two input sizes for which their homomorphic encryption parameters are ideal. Their use of fully homomorphic encryption also restricts their implementation to support 32-bit items only. Hence, for these tests we run all protocols with 32-bit items, even though ours and others support longer items. A summary of results is in Table 9.

Comparison with CLR Chen, Laine, Rindal [CLR17] describe a PSI protocol based on homomorphic encryption. The protocol contains a considerable number of parameters that are set based on experimentation

	virginia-ohio	virginia-oregon	sidney-saopaolo	us-us-10	us-sidney-10
ecdh283	4.48 (3.5%)	4.50 (3.5%)	7.12 (11.7%)	4.49 (3.5%)	4.91 (9.5%)
ecdh25519	8.36 (1.7%)	8.33 (1.7%)	12.46 (6.0%)	8.36 (1.7%)	8.85 (4.8%)
spot-fast	0.25 (56.1%)	0.27 (51.3%)	1.44 (51.3%)	0.68 (20.3%)	0.97 (42.7%)
spot-low	0.73 (16.1%)	0.79 (14.8%)	1.84 (33.9%)	1.22 (9.6%)	0.39 (91.1%)
kkrt	0.26 (91.4%)	0.29 (82.3%)	1.63 (79.3%)	1.00 (24.1%)	1.51 (48.1%)

Table 3: Monetary costs (USD) in the B2B scenario per 1000 runs with set size of 2^{16} items (values in parentheses denote the percentage of the cost due to data transfer).

	virginia-ohio	virginia-oregon	sidney-saopaolo	us-us-10	us-sidney-10
ecdh283	72.64 (3.5%)	73.13 (3.4%)	115.17 (11.7%)	75.28 (3.3%)	82.22 (9.2%)
ecdh25519	133.90 (1.7%)	134.39 (1.7%)	200.92 (6.1%)	136.11 (1.7%)	144.16 (4.7%)
spot-fast	3.69 (62.1%)	3.71 (61.8%)	15.46 (79.1%)	6.09 (37.7%)	10.77 (63.9%)
spot-low	16.91 (11.2%)	17.47 (10.9%)	33.71 (30.0%)	19.07 (9.9%)	23.31 (24.4%)
kkrt	4.09 (93.2%)	4.33 (88.0%)	23.04 (88.2%)	10.29 (37.0%)	18.08 (63.2%)

Table 4: Monetary costs (USD) in the B2B scenario per 1000 runs with set size of 2^{20} items (values in parentheses denote the percentage of the cost due to data transfer).

	virginia-ohio	virginia-oregon	sidney-saopaolo	us-us-10	us-sidney-10
ecdh283	2.42 (10.7%)	2.43 (10.7%)	4.02 (12.4%)	2.43 (10.7%)	2.86 (20.3%)
ecdh25519	4.35 (5.4%)	4.33 (5.4%)	7.01 (6.4%)	4.34 (5.4%)	4.85 (10.9%)
spot-fast	0.28 (81.0%)	0.30 (77.8%)	0.83 (53.0%)	0.50 (45.9%)	0.80 (64.4%)
spot-low	0.50 (38.9%)	0.53 (36.7%)	1.06 (35.5%)	0.74 (26.2%)	0.45 (96.2%)
kkrt	0.41 (97.2%)	0.43 (93.9%)	0.96 (80.4%)	0.78 (51.4%)	1.30 (69.2%)

Table 5: Monetary costs (USD) in the 'Internet' scenario per 1000 runs with set size of 2^{16} items (values in parentheses denote the percentage of the cost due to data transfer).

	virginia-ohio	virginia-oregon	sidney-saopaolo	us-us-10	us-sidney-10
ecdh283	39.26 (10.7%)	39.51 (10.6%)	65.08 (12.4%)	40.58 (10.4%)	47.69 (19.7%)
ecdh25519	69.61 (5.5%)	69.86 (5.4%)	113.08 (6.5%)	70.72 (5.4%)	78.92 (10.8%)
spot-fast	4.52 (84.5%)	4.53 (84.3%)	9.15 (80.2%)	5.72 (66.8%)	10.56 (81.1%)
spot-low	10.66 (29.6%)	10.94 (28.9%)	19.29 (31.4%)	11.75 (26.9%)	16.11 (43.9%)
kkrt	6.49 (97.9%)	6.61 (96.1%)	13.72 (88.9%)	9.59 (66.2%)	17.63 (80.7%)

Table 6: Monetary costs (USD) in the 'Internet' scenario per 1000 runs with set size of 2^{20} items (values in parentheses denote the percentage of the cost due to data transfer).

	lan9.6	virginia-ohio	virginia-oregon	sidney-saopaolo	us-us-10
ecdh283	81049.80	81074.90	81431.50	82875.80	81254.40
ecdh25519	153667.70	154187.00	153630.20	154327.30	154045.40
spot-fast	1936.60	2028.10	2465.20	9216.60	10189.10
spot-low	12850.00	11469.70	12586.70	16036.60	20623.00
kkrt	361.00	428.00	976.00	4433.00	14301.00

Table 7: Times in miliseconds for a set of 2^{16} items.

	lan9.6	virginia-ohio	virginia-oregon	sidney-saopaolo	us-us-10
ecdh283	1304656.10	1314656.10	1323987.30	1341503.30	1364300.80
ecdh25519	2457893.80	2467833.30	2477097.90	2488985.10	2509353.00
spot-fast	26105.30	26270.80	26618.50	42545.50	71139.50
spot-low	270240.00	281463.00	291945.10	311266.00	322077.30
kkrt	4668.00	5195.00	9721.00	35901.00	121494.00

Table 8: Times in miliseconds for a set of 2^{20} items.

rather than according to some formula. This make it challenging to describe its communication via a concrete formula. Instead, we give only an empirical comparison (in Table 9) based on their implementation.

The CLR protocol has *asymptotically* better communication complexity that is logarithmic in the size of the larger set, and linear in the size of the smaller set. However, our experiments show that spot-low variant still outperforms CLR for somewhat large input sizes. In the single-threaded 1Mbps setting, with $n_1 = 2^{20}$ and $n_2 = 11041$, our protocol obtains a running time of 68 seconds, whereas CLR takes 105.4 seconds. When comparing the two protocols in terms of communication overhead, our protocol achieves $1.5 - 4.4\times$ improvement when the sender's set size is reasonable $n_1 \leq 2^{20}$. For larger set size n_1, the CLR protocol scales much better than our protocol, due to its communication complexity being only $O(n_2 \log(n_1))$. As mentioned above, CLR's implementation is restricted to 32-bit items, whereas our implementation supports 128 bits.

Comparison with KLSAP. Kiss *et al.* [KLS+17] describe a protocol for unbalanced PSI, which defers a significant amount of computation to an offline pre-processing phase. Essentially, Alice encodes her set as a Bloom filter which she sends to Bob. In the online phase, the parties evaluate an oblivious PRF on each

Params.		Protocol	Comm.	Total time (seconds)							
n_1	n_2		Size (MB)	10 Gbps		100 Mbps		10 Mbps		1 Mbps	
				$T=1$	4	1	4	1	4	1	4
2^{24}	11041	[CLR17]	**23.2, 21.1**	115.4	40.3	117.8	42.7	**134.4**	59.3	**290.8**	**215.1**
		[KLS+17]	2049 (43.3)	90.4	—	265.1	—	1640.21	—	—	—
		[PSZ18]	480.9	**40.5**	**23.3**	**88.0**	66.4	449.5	427.5	4084.8	4067.2
		Ours	114.8	89.3	31.1	104.1	**41.45**	189.2	129.7	1054.0	994.4
2^{20}	11041	[CLR17]	11.5	12.8	5.7	14.0	6.9	22.2	15.1	105.4	98.3
		[KLS+17]	1968 (43.3)	82.1	—	259.9	—	1638.43	—	—	—
		[PSZ18]	30.9	**3.3**	**2.1**	**7.0**	5.6	29.8	28.3	263.7	262.1
		Ours	**7.7**	7.1	2.8	8.56	**4.3**	**15.1**	**10.4**	**68.0**	**67.2**
2^{16}	11041	[CLR17]	4.1, 4.4	3.0	1.7	3.4	2.1	6.4	5.3	36.0	35.0
		[KLS+17]	1963 (43.3)	81.8	—	259.6	—	1636.22	—	—	—
		[PSZ18]	2.6	**0.7**	**0.6**	**1.5**	**1.4**	**3.3**	**3.1**	21.6	21.1
		Ours	**1.0**	2.3	1.7	3.1	2.5	3.5	3.2	**10.8**	**10.5**

Table 9: Total communication cost in MB and running time in seconds comparing our protocol to [PSZ18, KLS+17, CLR17], with $T \in \{1, 4\}$ threads. Each item has 32-bit length in [PSZ18, CLR17, KLS+17], and 128-bit length in our protocol. 10Gbps network assumes 0.2ms RTT, and others use 80ms RTT. Cells with "—" denote setting not supported or program out of memory.

item of Bob, which Bob can probe in the Bloom filter. In their most communication-efficient variant, they use a Diffie-Hellman approach to compute $y \mapsto H(y)^{\alpha\beta}$.

In our comparison, we separate the communication into the offline and online phases. The online communication cost is reported in parenthesis in Table 9. Comparing the two protocols, our protocol scales better than KLSAP.

Comparison with PSZ. Pinkas et al. [PSZ18] propose different hashing parameters for the case of unequal set sizes. The PSZ and KKRT both use a similar paradigm, but PSZ uses less communication when the items have length 32, as we consider for this section. The experiments show that PSZ requires more communication than our protocol. Namely, for $n_1 = 2^{20}$ our protocol requires 114.8 MB of communication compared to PSZ requiring 480.9 MB. When comparing the running time of the two protocols, we observe that in the LAN setting, with $n_1 = 2^{20}$, our protocol takes 89.3 seconds, which is about $2\times$ slower than PSZ in this settings. When restricting the parties to a 10Mbps or 1Mbps network, our protocol is $2.37 - 3.87\times$ faster than PSZ.