PSI from Ring-OLE

Wutichai Chongchitmate
Chulalongkorn University, Thailand
wutichai.ch@chula.ac.th

Yuval Ishai*
Technion, Israel
yuvali@cs.technion.ac.il

Steve Lu
Stealth Software Technologies, Inc.
steve@stealthsoftwareinc.com

Rafail Ostrovsky†
UCLA, USA
rafail@cs.ucla.edu

Abstract

Private set intersection (PSI) is one of the most extensively studied instances of secure computation. PSI allows two parties to compute the intersection of their input sets without revealing anything else. Other useful variants include *PSI-Payload*, where the output includes payloads associated with members of the intersection, and *PSI-Sum*, where the output includes the sum of the payloads instead of individual ones.

In this work, we make two related contributions. First, we construct simple and efficient protocols for PSI and PSI-Payload from a ring version of oblivious linear function evaluation (ring-OLE) that can be efficiently realized using recent ring-LPN based protocols. A standard OLE over a field $\mathbb{F}$ allows a sender with $a, b \in \mathbb{F}$ to deliver $ax + b$ to a receiver who holds $x \in \mathbb{F}$. Ring-OLE generalizes this to a ring $\mathcal{R}$, in particular, a polynomial ring over $\mathbb{F}$. Our second contribution is an efficient *general reduction* of a variant of PSI-Sum to PSI-Payload and secure inner product.

Our protocols have better communication cost than state-of-the-art PSI protocols, especially when requiring security against malicious parties and when allowing input-independent preprocessing. Compared to previous maliciously secure PSI protocols that have a similar computational cost, our online communication is 2x better for small sets ($2^8 - 2^{12}$ elements) and 20% better for large sets ($2^{20} - 2^{24}$). Our protocol is also simpler to describe and implement. We obtain even bigger improvements over the state of the art (4-5x better running time) for our variant of PSI-Sum.

*Research supported in part by ERC Project NTSC (742754), BSF grant 2018393, and ISF grant 2774/20.

†Research supported in part by DARPA under Cooperative Agreement HR0011-20-2-0025, NSF grants CNS-2001096 and CCF-2220450, US-Israel BSF grant 2015782, Cisco Research Award, Google Faculty Award, JP Morgan Faculty Award, IBM Faculty Research Award, Xerox Faculty Research Award, OKAWA Foundation Research Award, B. John Garrick Foundation Award, Teradata Research Award, Lockheed-Martin Research Award and Sunday Group. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of DARPA, the Department of Defense, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes not withstanding any copyright annotation therein.

1 Introduction

Secure computation protocols allow two or more parties to perform a distributed computation while hiding their inputs from each other. A special type of secure computation protocol that has recently attracted a lot of research effort is *private set intersection* (PSI). A PSI protocol allows the parties to compute the intersection of their input sets without revealing anything except the output. PSI has found a wide array of applications, including private contact discovery, matching problems arising in business and data management, contact tracing and much more. See, e.g., [FNP04, IKN+20, PRTY19, DPT20, CM20, MPR+20, RT21, RS21, GPR+21] and references therein.

In this work we make two related contributions: we design new efficient protocols for a basic flavor of PSI, and present a general technique for extending it to a richer flavor that enables aggregating values associated with members of the intersection.

Asymmetric 2-party PSI. We consider *asymmetric* 2-party PSI protocols in which only one party, called the *receiver*, learns the intersection. The other party, called the *sender*, does not learn anything. In the following, the term PSI refers to asymmetric 2-party PSI by default.

1.1 Prior Work

To put our contributions in the proper context, we start with an overview of previous approaches to concretely efficient PSI. The first is based on the hardness of Decisional Diffie–Hellman (DDH) problem [Sha80, Mea86]. PSI protocols based on this approach are less efficient than newer approaches, especially when considering their computational cost. These protocols are difficult to apply to other variants of the PSI problem or strengthened for malicious security [DCKT10], and are inherently not post-quantum secure. Recent protocols that follow this approach but improve and extend it in several ways were given in [IKN+17, IKN+20, MPR+20, RT21].

Another approach, known as *circuit-based* PSI, reduces PSI to efficient secure computation for general circuits [HEK12, PSTY19, RS21]. This approach has an advantage of flexibility, as protocols that rely on this approach can be easily modified to securely compute any function on the intersection. This includes, in particular, the cases of "PSI with payloads" and "PSI-Sum" that we will discuss later. On the down side, achieving security against a malicious adversary is expensive. Even for the easier case of semi-honest security, this approach has relatively large communication and computation costs compared to more specialized approaches.

A more recent approach to practical PSI is via oblivious transfer (OT) [Rab05, EGL85]. The PSI protocols from [KKRT16, PRTY19, RR17a, RR17b, PRTY20, MRR20, CM20, DPT20, RS21, GPR+21] combine random instances of OT and relaxed forms of oblivious pseudorandom functions (OPRF) [FIPR05]. This approach allows efficient communication and running time by exploiting efficient OT extension techniques [IKNP03]. However, these techniques are less modular and thus more difficult to modify to accommodate other PSI variants or to benefit from input-independent preprocessing. Moreover, improving the security from semi-honest to malicious requires more work [RR17a, RR17b, PRTY20, GPR+21] and typically incurs a significant communication overhead.

The final approach, which is the most relevant to the present work, is an algebraic one [FNP04, FIPR05, KS05, GN19]. The high-level idea is to encode the elements in each set as roots of a polynomial, and securely compute a polynomial whose roots are the members of the intersection. The main advantages of this approach are simplicity and modularity, which make it easier to achieve malicious security. Our protocols will be based on this approach, and specifically the recent instance of this approach proposed by Ghosh and Nilges [GN19]. Whereas much of the effort in [GN19] is spent on extending malicious security to the symmetric and multi-party case, our work will focus

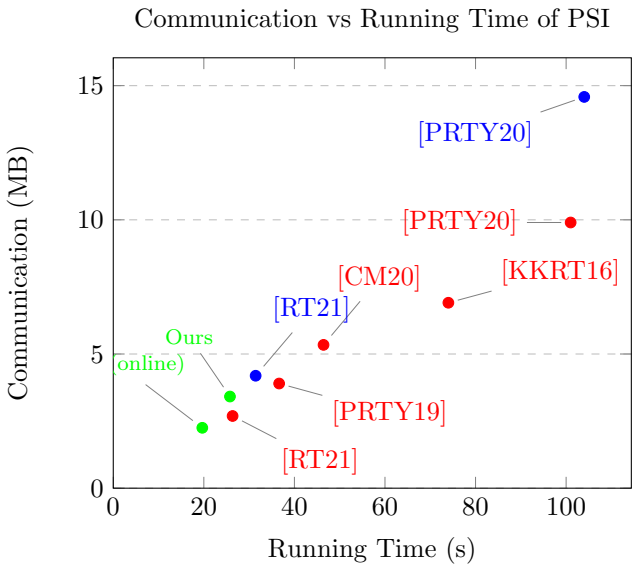

Figure 1: Communication vs running time in low bandwidth setting (1 Mbps) of PSI protocols for $n = 2^{16}$. Red dots represent PSI with semi-honest security. Blue dots represent PSI with malicious security. The green dot represents our PSI protocol (end-to-end and online time, both with malicious security).

on further improving efficiency in the two-party case and extending the functionality to other useful variants of PSI we discuss next.

PSI with payloads. *PSI payload* is a variant of PSI where each member a of the sender's input set has an associated value $t_a \in \{0,1\}^\ell$, referred to as a payload. The receiver obtains the associated payload values of members of the intersection along with the intersection. While some PSI constructions, such as ones that follow the circuit-based approach, can be easily modified to PSI payload, for others this incurs a substantial overhead. The algebraic approach that we follow in this work can be modified to securely compute a polynomial that evaluates to $(0^\lambda \| t_a)$ on member a of the intersection. Thus, our PSI protocol can be extended to PSI payload with only a small communication overhead.

PSI with sum. PSI-Sum (also called Sum-PSI or Private Intersection-Sum) is a variant of PSI where each member of the sender's input set has an associated integer value, similar to PSI payload. The receiver, however, only obtains the sum of the payloads of the elements in the intersection without learning individual payloads. Our implementation of PSI-Sum is different in that it also reveals the intersection to the receiver. That is, our version can be viewed as a standard PSI protocol with an additional output that includes the sum of the payloads. To distinguish this version from the standard one [IKN$^+$17, IKN$^+$20, MPR$^+$20], which only reveals the cardinality of the intersection, we call it *PSI+Sum*. While the extra information revealed to the receiver in PSI+Sum is problematic for some applications, it can be harmless or even useful for others. For instance, the receiver may be an analyst who wishes to learn aggregate statistics of a secret set of individuals (e.g., customers or patients) which she already knows to be part of the sender's data set.

Existing constructions of PSI-Sum are either OT-based, using an OPRF to hide the intersection while allowing computation of the payload sum [IKN$^+$17, IKN$^+$20, MPR$^+$20], or simply use the circuit-based approach [PSTY19, RS21, HMS21]. The general approach for PSI-Sum

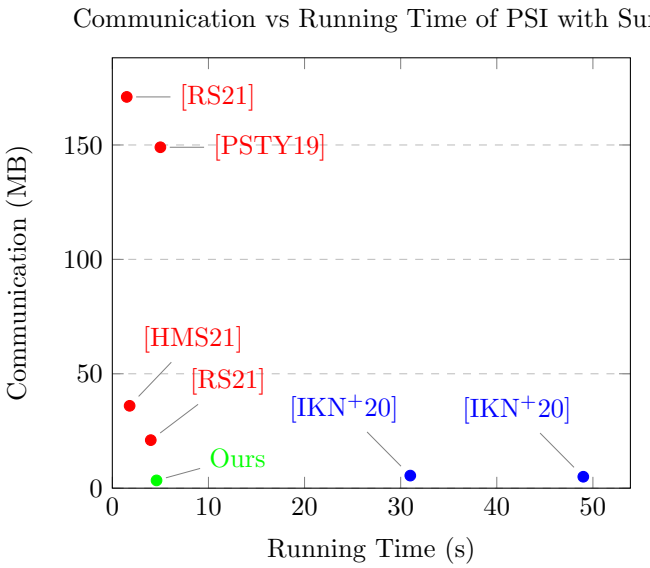

Figure 2: Communication vs running time of "PSI with sum" protocols for $n = 2^{16}$ with 32-bit payloads. Red dots represent circuit-based PSI-Sum protocols. Blue dots represent PSI-Sum protocols using other approaches. The green dot represents our (intersection-revealing) PSI+Sum protocol. The numbers are from the respective papers or estimated.

in previous works is to start with a protocol for PSI payload, and use ad-hoc techniques to mask the payloads in a way that only their sum can be recovered by the receiver. For example, [IKN$^+$17, IKN$^+$20, MPR$^+$20] use different kinds of homomorphic encryption to hide individual payloads while allowing their aggregation. (See [CO18] for discussion of the underlying challenges.) These previous approaches result in either high communication or slow running time, as shown in Figure 2. By revealing the intersection to the receiver, we significantly improve the communication and running time. Jumping ahead, we will present an efficient *general* transformation from PSI with payload to PSI+Sum.

The preprocessing model. While our PSI protocols are competitive also when considering end-to-end costs, they are particularly attractive when allowing input-independent preprocessing as in [RR17a, RR17b, RS21]. A protocol in the preprocessing model consists of two phases. In the *offline* phase, which can be executed before the inputs are known (say, during an idle time of the system), the parties can interact in order to securely generate correlated randomness that is stored for future use. In the *online* phase, which is executed once the inputs are known, the parties use the correlated randomness generated in the offline phase to perform the PSI computation more efficiently. Most of PSI protocols from the literature can benefit from the preprocessing model by moving steps that can be computed without the inputs, such as setting up OTs or Bloom filters [KLS$^+$17], to the offine phase. However, the extent to which this helps varies greatly between different PSI techniques. In particular, PSI protocols based on DDH or OPRF require the inputs to be known before most of the work can be performed, and thus only modestly benefit from preprocessing. In contrast, the algebraic approach we pursue here can be used to shift the vast majority of the work to the offline phase, where the cost of the latter can be minimized using recent techniques of Boyle et al. [BCG$^+$19, BCG$^+$20]. Moreover, the offloading other protocols can perform in the offline phase are mainly computation, not communication. As shown in Figure 1, in a low bandwidth setting, our protocol is the only one benefit from preprocessing. It spends 25-30%

of both communication and running time in the offline phase, and thus represented by two dots for total and online. Other PSI protocols run mostly in the online phase and are represented by one dot.

(Ring) OLE. The oblivious linear-function evaluation (OLE) functionality [NP99, IPS09] is an algebraic generalization of OT. Recall that the OT functionality allows a sender with two inputs $b_0, b_1 \in \{0,1\}$ to send b_i to a receiver whose input is $i \in \{0,1\}$. The receiver only learns b_i but learns nothing about b_{1-i}, while the sender also learns nothing about i. In the OLE functionality over a finite field $\mathbb{F}$, the sender instead holds $a, b \in \mathbb{F}$, the receiver holds an input $x \in \mathbb{F}$, and the receiver obtains the output $ax + b$. An OT functionality is equivalent to a special case of an OLE functionality with $\mathbb{F} = \mathbb{F}_2$. Efficient OLE protocols can be constructed either from OT alone [Gil99, IPS09, KOS16, HMRT22] or, more efficiently, from OT and noisy Reed-Solomon codes [NP99, IPS09, GNN17] or lattice-based additively homomorphic encryption [JVC18, BEP+20]. An efficient "black-box" technique for protecting OLE protocols against malicious parties has been proposed in [HIMV19].

Our main PSI protocol is based on a generalized version of OLE we refer to as *ring-OLE*. In a ring-OLE protocol, the inputs a, b, x are in a (finite) ring $\mathcal{R}$. In this work we consider a quotient ring $\mathcal{R} = \mathbb{F}[X]/\langle p(X)\rangle$ where each element can be represented by a polynomial of bounded degree. Standard OLE over $\mathbb{F}$ can be viewed as a special case in which the degree-bound is 0. Ring-OLE for $\mathcal{R} = \mathbb{F}[X]/\langle p(X)\rangle$ where $p(X)$ has n distinct roots in $\mathbb{F}$ can be constructed from OLE as follows. For each root c of $p(X)$, the parties engage in an instance of OLE to send $a(c)x(c) + b(c)$ to the receiver. Then, the receiver can use polynomial interpolation to reconstruct $ax + b$. In order to ensure that the degree of ax does not exceed that of p, one can use 2 additional instances of OLE to check the result, analogously to the technique used in [GN19]. However, better efficiency can be obtained by using a ring-LPN based construction from [BCG+20], which generates a random instance of degree-n ring-OLE with sublinear communication cost in n while achieving security against malicious parties.

1.2 Our Results

In this paper, we obtain the following results.

Maliciously secure PSI from ring-OLE. We present a 2-party PSI protocol with security against malicious parties from ring-OLE. Our protocol is particularly attractive in the preprocessing model, but also has competitive end-to-end costs. Since ring-OLE can be easily reduced to OLE, our protocol can also be cast in the OLE-hybrid model.

Our protocol builds on the OLE-based PSI protocol of Ghosh and Nilges [GN19] but improves it in several ways.[1] First, our protocol significantly increases efficiency in the online phase by moving all calls to the (ring-)OLE functionality to the offline phase. Our online phase is "non-cryptographic" and typically has higher efficiency in both communication and running time compared to other PSI protocols, even those with only semi-honest security. Second, we combine $\mathcal{O}(n)$ calls to OLE functionality, where n is the size of the input set, into a single call to ring-OLE functionality for a quotient ring of polynomials of bounded degree. This makes our construction modular and easier to analyze, and allows us to take advantage of recent efficient random (ring-)OLE setup [BCG+20], to minimize communication costs with reasonable amortized overhead. Third, we simplify the approach taken in [GN19] to achieve malicious security. Instead of using two OLE calls to prevent a malicious receiver from sending $x = 0$ and learning $a(0) + b = b$, we

[1] Abadi et al. [AMZ21] recently presented several attacks on PSI protocols from [GN19]. These attacks mainly target the symmetric variant, where both parties receive the output, and do not apply to our version of this protocol.

use the special-purpose ring-OLE to push this check to the online phase where the sender aborts unless $x \in \mathbb{F}[X]$ is a monic polynomial of degree n. Even when instantiating this method with OLE (instead of ring-OLE), it almost halves the number of OLE calls in [GN19]. Combining both the offline and online phase, our protocol decreases the communication complexity of [GN19] by 50%-75% and its running time by 35%-45% even when OLE in [GN19] is generated by the PCG. In particular, the online phase of our PSI protocol communicates 80%-90% less bits than that of [GN19] and is almost twice as fast when compared to the original.

Compared to the state-of-the-art OT-based PSI [PRTY19, PRTY20, CM20, GPR$^+$21] with semi-honest security, the communication complexity of our PSI protocol in the online phase is up to 50% smaller while remaining competitive in the running time for small ($2^8, 2^{12}$ elements) sets. The vector-OLE-based PSI [RS21] and DH-based PSI [RT21] only outperform ours in term of communication for large ($2^{20}, 2^{24}$ elements) sets. In the malicious setting, our advantages are even bigger. The communication complexity of our PSI protocol in the online phase is up to 50% less than any other known PSI for small sets and 20% less for large sets. In the low bandwidth setting, our protocol outperforms other PSI protocols in the running time as well, as shown in Figure 1. See Table 2 for a detailed comparison of communication costs, Section 4.2 for analysis of computation costs, and Section 8 for comparison to other approaches.

A major additional advantage of our protocol is that it can be easily extended to PSI with payload and PSI with sum, which we discuss next.

PSI payload. We modify our PSI construction to support payload. In particular, we construct a protocol for PSI payload with security against malicious parties from ring-OLE. As in the basic PSI case, we may instantiate the ring-OLE with OLE and obtain a PSI payload protocol in the OLE-hybrid model. The online phase of the protocol is very communication efficient and does not involve any use of cryptography. The payloads in our protocol can be encoded using the same polynomials that are used to compute the intersection. The technique can be adapted to oblivious *programmable* PRF (OPPRF)-based PSI [PRTY20, RS21] and oblivious key-value store (OKVS)-based PSI [GPR$^+$21], but not (non-programmable) OPRF [CM20] or DH-based [RT21] PSI. In some cases, masked values of the payload need to be sent separately, usually encrypted [IKN$^+$20]. They cannot be sent as random (hashed) values together with the set elements as only one party knows them. Efficient decryption and exchange of these values are done with only semi-honest security. Our approach avoids these limitations with little additional communication overhead.

From PSI payload to PSI+Sum. Our second main contribution is a general construction of a PSI+Sum protocol from any PSI payload and secure inner product. Informally, the protocol works as follows: The sender masks the payload for each element of her set with an evaluation of a random polynomial at this element. The receiver can only get rid of the masked value of the sum by performing a secure inner product between a sum of each power of the intersection members and the coefficients of the masked polynomial.

Secure inner product can be efficiently reduced to OLE, which in turn has a low cost in the preprocessing model. Instantiating this reduction with our PSI payload protocol, we obtain an efficient PSI+Sum protocol with security against semi-honest parties from OLE. Similar to the previous construction, calls to OLE are made only in the offline phase, and thus the online phase is entirely non-cryptographic. When the (offline) random OLE functionality is instantiated with the recent ring-LPN based protocol from [BCG$^+$20] our protocol is competitive in both communication complexity and running time even when combining the offline and online phases into a single end-to-end protocol. See Table 3 for a comparison with prior works.

1.3 Applications

PSI is motivated by a growing number of real-world applications that are thoroughly discussed in prior works (see, e.g., [FNP04, IKN+20, PRTY19] and references therein). We would like to motivate our two main points of departure from most of these prior works.

Offline-online setting. Many practical MPC protocols, including the popular SPDZ line of protocols, heavily rely on the premise that offline work (including communication and computation) is a much cheaper resource than online work. It is therefore very natural to take advantage of the same premise in the context of PSI. Moreover, given the new techniques for generating (ring-)OLE correlations from ring-LPN [BCG+20], we get significant efficiency gains even with respect to the total cost (counting both offline and online). In particular, we can delegate about 40% of the communication and 25%-50% of the running time to the offline phase in our PSI protocol.

Relaxed security for PSI+Sum. Unlike other PSI-Sum protocols, our general transformation from PSI payload to PSI+Sum reveals not only the sum of the payloads in the intersection but also reveals (to the receiver) the intersection itself. (Note that the set of payloads in the intersection still remains secret, except for revealing the sum.) This can be a significant limitation for some applications. However, in other applications this may not be a concern. For instance, in classical use-cases of PSI (e.g., ones related to matching problems) the ideal functionality reveals the actual set intersection either to both parties or to one of the parties. In the same use-cases, there may be additional payloads containing sensitive information (e.g., salaries) that can only be revealed in aggregate form. Finally, as discussed above, there are many use-cases in which the intersection itself reveals no information to the receiver because its data set is always a subset of the sender's. This is the case, for instance, when the sender is a company or a hospital and the receiver wants to obtain some aggregate statistics of a secret subset of customers or patients that are known to be associated with the sender.

1.4 Organization

The rest of the paper is organized as follows. We present the PSI functionality and fast algorithms for polynomial evaluation, polynomial interpolation and power sum in Section 2. In Section 3, we describe OLE-related functionalities and reductions between them. Section 4 describes our main ring-OLE based PSI protocol in the preprocessing model, along with a proof of malicious security and performance analysis. In Section 5, we extend the protocol to include payloads. In Section 6, we describe and analyze our general construction of PSI+Sum from PSI Payload and secure inner product. In Section 7, we briefly discuss how to extend the ring-OLE based construction to other PSI varaints. Finally, in Section 8, we analyze the concrete performance of our PSI protocols and compare it to other PSI protocols.

2 Preliminaries

We use λ and κ to denote the statistical and computational security parameters, respectively. We use $[n]$ to denote $\{1, 2, \ldots, n\}$. We use the standard definition of negligible functions and computational indistinguishability [GM84]. We will denote by $\Pr_r[X]$ the probability of an event X over coins r, and $\Pr[X]$ when r is not specified. The abbreviation "PPT" stands for probabilistic polynomial time. For a finite set S, we denote $a \leftarrow S$ a uniformly random choice of a from S. For a randomized algorithm A, let $A(x; r)$ denote running A on an input x with random coins r. If r is chosen uniformly at random with an output y, we denote $y \leftarrow A(x)$. For a vector $\vec{v} \in \mathbb{F}^m$, we use $\vec{v}[i]$ to denote the i-th element. We denote the inner product of $\vec{u}, \vec{v} \in \mathbb{F}^m$ by $\langle u, v \rangle$.

2.1 Polynomial Operations

Our algorithms for PSI and its variants use an algebraic approach and thus need to perform several polynomial operations including (multi-point) polynomial evaluation, polynomial interpolation and power sum, that is computing $\sum_{j=1}^{m} c_j^i$ for $i = 0, \ldots, n$.

2.1.1 Polynomial Evaluation

Our PSI protocols and its variants need to evaluate polynomials at a large number of points. We use an efficient algorithm in [BM74, BLS03] to evaluate a given polynomial at all points at once. The idea is that evaluating a polynomial $f(x) \in \mathbb{F}[x]$ at $a \in \mathbb{F}$ is $f(a) = f(x) \bmod (x - a)$. Instead of computing $f(x) \bmod (x - a)$ for each evaluation point, the algorithm first constructs a *subproduct tree*, which is a binary tree where each node contains the product of polynomials in its children. The leaves of the subproduct tree contain $x - a_i$ where $a_1, \ldots, a_n$ are evaluation points. The subproduct tree can be constructed from leaves where polynomial multiplications are computed using FFT. To evaluate a polynomial f, we compute modulo of f down along the subproduct tree from its root. The result at each leaf will be the evaluation. This fast multi-point polynomial evaluation algorithm, called *FastEval*, allows our algorithms to evaluate a polynomial at n distinct points using $17n \log^2 n$ field multiplications [BLS03] as opposed to point-wise evaluation using $\mathcal{O}(n^2)$ field multiplications.

2.1.2 Polynomial Interpolation

The subproduct tree can also be used to compute polynomial interpolation [BM74, BLS03]. We may think of the polynomial evaluation at $a_1, \ldots, a_n \in \mathbb{F}$ as a linear function. Then polynomial interpolation at the same points is simply its inverse. For a polynomial $f(x) = \sum_{i=0}^{n} a_i x^i$, we let $f'(x) = \sum_{i=1}^{n} i a_i x^{i-1}$ denote its formal derivative. From Lagrange interpolation formula, the unique polynomial f of degree $n - 1$ that satisfies $f(a_i) = b_i$ for $i = 1, \ldots, n$ is

$$f(x) = \sum_{i=1}^{n} b_i \left(\frac{\prod_{j \neq i} (x - a_j)}{\prod_{j \neq i} (a_i - a_j)} \right) = \sum_{i=1}^{n} \frac{b_i}{m'(a_i)} \cdot \frac{m(x)}{x - a_i}$$

where $m(x) = \prod_{i=1}^{n} (x - a_i)$ and $m'(x) = \sum_{i=1}^{n} \prod_{j \neq i} (x - a_j)$ is its formal derivative. We can compute $m'(a_i)$ using FastEval and $\frac{m(x)}{x - a_i}$ using the subproduct tree. This algorithm uses $20n \log^2 n$ field multiplications to interpolate n points [BLS03].

2.1.3 Power Sum

Our algorithm for PSI+Sum also requires the receiver to compute $\sum_{c \in I} c^i$ for $i = 0, \ldots, d$ where both $|I|$ and d are linear in the input size n. A direct computation would require $\mathcal{O}(n^2)$ multiplications and additions. We consider this computation as a multiplication between the transpose of the Vandermonde matrix of elements of I and a vector of all 1's. Thus, this computation can

be done using the algorithm in Figure 3. For a polynomial $f(x) = \sum_{i=0}^{n} a_i x^i$ with $a_n \neq 0$, we let $rf(x) = \sum_{i=0}^{n} a_{n-i} x^i = x^n f(x^{-1})$ denote its reverse-coefficient polynomial. This algorithm is a simplified version of an algorithm in [BLS03] using the subproduct tree.

PowerSum

1. Compute $f(x) = \prod_{c \in I}(x - c)$
2. Compute the reverse-coefficient polynomial $rf(x)$
3. Compute the inverse of $rf(x)$ modulo x^{d+1}, denoted $g(x)$
4. Compute the reverse-coefficient polynomial of the derivative $f'(x)$, denoted $t(x)$
5. Output the coefficient vector $\vec{v}$ of $t(x)g(x)$ modulo x^{d+1}

Figure 3: Power Sum

Theorem 2.1. *$v_i = \sum_{c \in I} c^i$ for $i = 0, \ldots, d$.*

Proof. From $f(x) = \prod_{c \in I}(x - c)$, we have $rf(x) = \prod_{c \in I}(1 - cx)$. Since we can write the derivative $f'(x) = \sum_{c \in I} \prod_{c' \neq c}(x - c')$, we get $t(x) = \sum_{c \in I} \prod_{c' \neq c}(1 - c'x)$. Thus,

$$\begin{aligned} t(x)g(x) &= \sum_{c \in I} \prod_{c' \neq c} (1 - c'x) g(x) \pmod{x^{d+1}} \\ &= \sum_{c \in I} (1 - cx)^{-1} \pmod{x^{d+1}} \\ &= \sum_{c \in I} \left(1 + cx + (cx)^2 + \ldots + (cx)^d\right) \pmod{x^{d+1}} \\ &= \sum_{i=0}^{d} \left(\sum_{c \in I} c^i\right) x^i \pmod{x^{d+1}} \end{aligned}$$

□

2.2 Private Set Intersection

We define the security of the two-party private set intersection (PSI) as a special case of a secure two-party computation (2PC). We follow the standard security notions for semi-honest and malicious securities. The ideal functionality of PSI is defined in Figure 4.

2.3 Secure Inner Product

A secure inner product functionality allows two parties to compute an inner product of each party's input vector and output as an additive share to each party. We define an ideal functionality for secure inner product in Figure 5.

$\mathcal{F}_{\mathsf{PSI}}$

Parameters. n is the upper bound of the size of parties' input sets; $n' \geq n$ is the upper bound of the corrupted sender's input set.

Functionality.

1. Upon receiving a message (inputS, A) from the sender with $A \subseteq \{0,1\}^*$, if $|A| > n$ and the sender is honest, or $|A| > n'$ and the sender is corrupted, ignore that message. Otherwise, store A and send (inputS) to $\mathcal{A}$.
2. Upon receiving a message (inputR, B) from the receiver with $B \subseteq \{0,1\}^*$, if $|B| > n$, ignore that message. Otherwise, store B and send (inputR) to $\mathcal{A}$.
3. Upon receiving a message $(\mathsf{deliver})$ from $\mathcal{A}$, check if both A and B are stored, else ignore that message. Otherwise, set $I = A \cap B$ and send (output, I) to the receiver. Ignore all further messages.

Figure 4: Ideal functionality for private set intersection (PSI)

We show how to construct a protocol realizing $\mathcal{F}^n_{\mathsf{InnerProduct}}$ in the $\mathcal{F}_{\mathsf{OLE}}$-hybrid model in Figure 6. The idea behind this construction is similar to the secure inner product protocols in OT-hybrid model in [KOS16, GSB+17]. We note that the degree of the polynomial p'_B is set to $d-1$ instead of d to allow Sender to check that p_B has degree exactly d. This prevents a malicious Receiver from learning some parts of p_A from p_Q. While $p^*_B = p_B - p'_B$ exposes the coefficient of x^d from p_B, this coefficient can be chosen uniformly at random from $\mathbb{F} \setminus \{0\}$ independent of B.

Theorem 2.2. $\Pi^n_{\mathsf{innerProduct}}$ *unconditionally realizes* $\mathcal{F}^n_{\mathsf{InnerProduct}}$ *in the* $\mathcal{F}_{\mathsf{OLE}}$*-hybrid model.*

Proof. By the correctness of $\mathcal{F}_{\mathsf{OLE}}$, we have $z_i = \vec{u}[i]\vec{v}[i] - \vec{r}[i]$. Thus, $z = \langle \vec{u}, \vec{v} \rangle - r$. Hence,

$$\begin{aligned} o_A + o_B &= \langle \vec{a}, \vec{w}_2 \rangle - r + \langle \vec{b}, \vec{w}_1 \rangle - z \\ &= \langle \vec{a}, \vec{b} - \vec{v} \rangle + \langle \vec{a} + \vec{u}, \vec{v} \rangle - \langle \vec{u}, \vec{v} \rangle \\ &= \langle \vec{a}, \vec{b} \rangle - \langle \vec{a}, \vec{v} \rangle + \langle \vec{a}, \vec{v} \rangle + \langle \vec{u}, \vec{v} \rangle - \langle \vec{u}, \vec{v} \rangle \\ &= \langle \vec{a}, \vec{b} \rangle \end{aligned}$$

When the adversary $\mathcal{A}$ corrupts Alice, we construct a simulator $\mathcal{S}_A$ interacting with $\mathcal{A}$ as follows. It first simulates $\mathcal{F}_{\mathsf{OLE}}$ in the offline phase to extract $\vec{u}$ and $\vec{r}$ Upon receiving $\vec{w}_1$ in the online phase, $\mathcal{S}_A$ computes $\vec{a} = \vec{w}_1 - \vec{u}$ and sends it to $\mathcal{F}^n_{\mathsf{InnerProduct}}$ and receives o_A. Finally, it sends a uniformly random $\vec{w}_2$ with $\langle \vec{a}, \vec{w}_2 \rangle = o_A + \sum_i \vec{r}[i]$ to $\mathcal{A}$. Since Bob is honest, $\vec{v}$ is uniformly random, and so is $\vec{w}_2$ in the real world. Thus, the ideal world interaction and the real world interaction are indistinguishable.

When the adversary $\mathcal{A}$ corrupts Bob, we construct a simulator $\mathcal{S}_B$ interacting with $\mathcal{A}$ as follows. It first simulates $\mathcal{F}_{\mathsf{OLE}}$ in the offline phase to extract $\vec{v}$, and send uniformly random z_i to $\mathcal{A}$. It sends a uniformly random $\vec{w}_1$ to $\mathcal{A}$ in the online phase. Upon receiving $\vec{w}_2$, $\mathcal{S}_B$ computes $\vec{b} = \vec{w}_2 + \vec{v}$. and sends it to $\mathcal{F}^n_{\mathsf{InnerProduct}}$ and receives o_B. Since Alice is honest, $\vec{r}$ and $\vec{u}$ are uniformly random, so is $\vec{w}_1$ and $z_i = \vec{u}[i]\vec{v}[i] - \vec{r}[i]$ in the real world. Thus, the ideal world interaction and the real world interaction are indistinguishable. □

$\mathcal{F}_{\mathsf{InnerProduct}}$

Parameters. $\mathbb{F}$ is a finite field; n is the length of input vectors.

Functionality.

1. For $i \in \{1, 2\}$, upon receiving a message $(\mathsf{input}_i, \vec{v}_i)$ from P_i with $\vec{v}_i \in \mathbb{F}^n$, verify that there is no stored vector $\vec{v}_i$, else ignoring that message. Otherwise, store $\vec{v}_i$ and send (input_i) to $\mathcal{A}$.

2. Upon receiving a message $(\mathsf{deliver})$ from $\mathcal{A}$, check if both $\vec{v}_1$ and $\vec{v}_2$ are stored else ignoring that message. Otherwise, set $c = \langle \vec{v}_1, \vec{v}_2 \rangle \in \mathbb{F}$, sample $o_1 \leftarrow \mathbb{F}$, set $o_2 = c - o_1$, and send (output_i, o_i) to P_i for $i \in \{1, 2\}$. Ignore all further messages.

Figure 5: Ideal functionality for secure inner product

The secure inner product functionality can also be instantiated from a random inner product in [BCG+20]. The random inner product instance can replace $\vec{u}$ and r for Alice, and $\vec{v}$ and z for Bob in Figure 6 where $\langle \vec{u}, \vec{v} \rangle = r + z$. This method is more efficient than using OLE or random OLE in both computation and communication in the offline phase. The online phase and the security remain the same.

2.4 Polynomial Ring modulo $p(X)$

We view the ring $\mathcal{R} = \mathbb{F}[X]/\langle p(X) \rangle$ as a collection of equivalence classes of polynomials modulo $p(X)$ where

$$[f] = \{g \in \mathbb{F}[X] : f \equiv g \pmod{p(X)}\}$$

denote the equivalence class of f in $\mathcal{R}$ This means that if two polynomials have the same remainder when divided by $p(X)$, they are in the same class, i.e. the same element in $\mathcal{R}$. This way, any polynomial can be viewed as an element of $\mathcal{R}$ by considering its equivalence class. Moreover, each polynomial f (of any degree) in $\mathbb{F}[X]$ is uniquely identified with the representative $f \pmod{p(X)}$ of degree less than that of p.

3 Oblivious Linear Evaluation

Let $\mathbb{F}$ be a finite field. An oblivious linear function evaluation (OLE) allows two parties, a sender with input $a, b \in \mathbb{F}$ and a receiver with input $x \in \mathbb{F}$, to securely compute $ax + b$ and output to the receiver. The OLE can be seen as a generalization of an oblivious transfer (OT) where $\mathbb{F} = \mathbb{F}_2$. The OLE protocols are usually constructed in batches from various techniques and primitives such as noisy Reed Solomon codes in the OT-hybrid model in [NP99, IPS09, GNN17], an additive homomorphic encryption [JVC18], pseudorandom correlation generator (PCG) based on ring-Learning Parity with Noise (ring-LPN) [BCG+20].

3.1 Ring-OLE

In this work, we consider a generalization of OLE, called *ring-OLE (rOLE)*, where the inputs a, b, x are from a ring $\mathcal{R}$ instead of a field. The rOLE functionality is defined in Figure 7.

$\Pi^n_{\mathsf{innerProduct}}$

Offline Phase.

1. Alice samples random vectors $\vec{u}, \vec{r} \in \mathbb{F}^n$.
2. Bob samples a random vector $\vec{v} \in \mathbb{F}^n$.
3. For $i = 1, \ldots, n$,
   - Alice sends $(\mathsf{inputS}, (\vec{u}[i], -\vec{r}[i]))$ to $\mathcal{F}_{\mathsf{OLE}}$.
   - Bob sends $(\mathsf{inputR}, \vec{v}[i])$ to $\mathcal{F}_{\mathsf{OLE}}$ and receives (output, z_i).
4. Alice computes $r = \sum_i \vec{r}[i] \in \mathbb{F}$.
5. Bob computes $z = \sum_i z_i \in \mathbb{F}$.

Online Phase. Alice on input $\vec{a} \in \mathbb{F}^n$, Bob on input $\vec{b} \in \mathbb{F}^n$

1. Alice sends $\vec{w}_1 = \vec{a} + \vec{u} \in \mathbb{F}^n$ to Bob.
2. Bob sends $\vec{w}_2 = \vec{b} - \vec{v} \in \mathbb{F}^n$ to Alice.
3. Alice outputs $o_A = \langle \vec{a}, \vec{w}_2 \rangle - r$.
4. Bob outputs $o_B = \langle \vec{w}_1, \vec{v} \rangle - z$.

Figure 6: Protocol realizing secure inner product in the $\mathcal{F}_{\mathsf{OLE}}$-hybrid model

When $\mathcal{R} = \mathbb{F}[X]/\langle p(X) \rangle$, a quotient ring of a polynomial ring $\mathbb{F}[X]$, where $p(X)$ is a product of d distinct linear terms, a ring-OLE for $\mathcal{R}$ can be constructed from d instances of (standard) OLE over $\mathbb{F}$. When $\mathcal{R}$ is finite, a variant of rOLE where both parties' inputs are replaced by uniformly sampled elements of $\mathcal{R}$ is called a *random rOLE (rrOLE)*. A malicious variant where malicious party may choose their inputs is defined in [BCG+20] as shown in Figure 8.

In [BCG+20], rrOLE protocol for $\mathcal{R} = \mathbb{F}[X]/\langle p(X) \rangle$ can be constructed from pseudorandom correlation generator (PCG) based on ring-Learning Parity with Noise (ring-LPN). The resulting rrOLE correlation can be split into d instances of random OLE correlation by evaluating at each root of $p(X)$ (or by dividing by each linear factor).

3.2 Ring-OLE from Random Ring-OLE in the Offline Phase

In this section, we construct a rOLE protocol from rrOLE functionality in the offline phase. This protocol will later be used in our PSI protocol. By moving the rrOLE functionality to the offline phase, the rOLE protocol and the upcoming PSI protocol will have efficient online phase in both communication and computation. Let $\mathcal{F}_{\mathsf{rrOLE}}$ be the rrOLE functionality. We describe the rOLE protocol $\Pi^{\mathsf{offline}}_{\mathsf{rOLE}}$ in Figure 9.

Informally, in the offline phase, the sender and the receiver execute rrOLE for randomly chosen ring elements (a', b') and (x', c') where $c' = a'x' + b'$. In the online phase, they use a', b' and x', c' to additively mask their inputs, and send the masked values in clear. We can show that the joint

$\mathcal{F}_{\mathsf{rOLE}}$

Parameters. a ring $\mathcal{R}$

Functionality.

1. Upon receiving a message $(\mathsf{inputS}, (a, b))$ from the sender with $a, b \in \mathcal{R}$, verify that there is no stored tuple else ignoring that message. Otherwise, store (a, b) and send (inputS) to $\mathcal{A}$.
2. Upon receiving a message (inputR, x) from the receiver with $x \in \mathcal{R}$, verify that there is no stored value else ignoring that message. Otherwise, store x and send (inputR) to $\mathcal{A}$.
3. Upon receiving a message $(\mathsf{deliver})$ from $\mathcal{A}$, check if both (a, b) and x are stored else ignoring that message. Otherwise, set $c = ax + b \in \mathcal{R}$ and send (output, c) to the receiver. Ignore all further messages.

Figure 7: Ideal functionality for ring oblivious linear function evaluation (rOLE)

distribution of masked messages completely hide each party's inputs while allowing the receiver to recover $ax + b$. We prove the security of $\Pi^{\mathsf{offline}}_{\mathsf{rOLE}}$ in Figure 9 in the following theorem.

Theorem 3.1. *Assuming $\mathcal{F}_{\mathsf{rrOLE}}$ functionality, $\Pi^{\mathsf{offline}}_{\mathsf{rOLE}}$ securely realizes $\mathcal{F}_{\mathsf{rOLE}}$ using $\mathcal{F}_{\mathsf{rrOLE}}$ only in the offline phase with perfect security.*

Proof. (Sketch) The correctness of $\mathcal{F}_{\mathsf{rOLE}}$ implies that $c' = a'x' + b'$. Thus,

$$\begin{aligned} c = b^* + a^*x - c' &= (b + b' - a'(x - x')) + (a + a')x - (a'x' + b') \\ &= ax + b. \end{aligned}$$

So, $\Pi^{\mathsf{offline}}_{\mathsf{rOLE}}$ is correct. Suppose the sender is corrupted. We construct a simulator $\mathcal{S}_{\mathsf{Sender}}$ that follows the protocol except that it simulates $\mathcal{F}_{\mathsf{rrOLE}}$ in the offline phase to extract a', b'. In the online phase, it sends x^* chosen uniformly at random from $\mathcal{R}$ to the sender. Upon receiving a^*, b^*, it computes $a = a^* - a'$ and $b = b^* - b' + a'x^*$ for the functionality. Since x' is also chosen uniformly at random from $\mathcal{R}$, x^* from $\mathcal{S}_{\mathsf{Sender}}$ and $x^* = x - x'$ in the real protocol as they have the same distribution.

Now suppose the receiver is corrupted. We construct a simulator $\mathcal{S}_{\mathsf{Receiver}}$ that simulates $\mathcal{F}_{\mathsf{rrOLE}}$ in the offline phase to extract x', c'. In the online phase, upon receiving x^* from $\mathcal{A}$, the simulator sends $\tilde{x} = x^* + x'$ to the functionality to get $\tilde{c} = a\tilde{x} + b$. It chooses $\tilde{a}^*$ uniformly at random, computes $\tilde{b}^* = \tilde{c} - \tilde{a}^*(x^* + x') + c'$, and sends $\tilde{a}^*, \tilde{b}^*$ to $\mathcal{A}$.

Lemma 3.2. *The distribution of (a^*, b^*) given (x^*, x', c, c') in $\Pi^{\mathsf{offline}}_{\mathsf{rOLE}}$ and the distribution $(\tilde{a}^*, \tilde{b}^*)$ given $(x^*, x', \tilde{c}, c')$ are identical.*

Proof. Since for any $a', x' \in \mathcal{R}$ there exists $b' = c' - a'x'$ satisfying $c' = a'x' + b'$. Thus, a' is uniform given c', so is a^*. Hence, a^* and $\tilde{a}^*$ has the same distribution. Given x^*, x', c, c', a^*, we can write

$$\begin{aligned} b^* &= b + b' - a'x^* \\ &= (c - ax) + (c' - a'x') - a'x^* \\ &= c - ax - a'(x' + x^*) + c' \\ &= c - a^*(x' + x^*) + c' \end{aligned}$$

$\mathcal{F}_{\mathsf{rrOLE}}$

Parameters. a finite ring $\mathcal{R}$

Functionality.

1. If both parties are honest or $\mathcal{A}$ is semi-honest, uniformly sample $a, b, x \leftarrow \mathcal{R}$ and compute $c = ax + b$.
2. If $\mathcal{A}$ is malicious and the sender is corrupted, upon receiving a message (a, b) from $\mathcal{A}$, if $a, b \in \mathcal{R}$, uniformly sample $x \leftarrow \mathcal{R}$ and compute $c = ax + b$, else ignore the message.
3. If $\mathcal{A}$ is malicious and the receiver is corrupted, upon receiving a message (x, c) from $\mathcal{A}$, if $x, c \in \mathcal{R}$, uniformly sample $a \leftarrow \mathcal{R}$ and compute $b = ax - c$, else ignore the message.
4. Upon receiving a message (deliver) from $\mathcal{A}$, check if a, b, c, x are stored, else ignoring that message. Send (a, b) to the sender and (x, c) to the receiver, respectively. Ignore all further messages.

Figure 8: Ideal functionality for random ring oblivious linear function evaluation (rrOLE) [BCG+20]

Both b^* and $\tilde{b}^*$ can be determined by the same expression given x^*, x', c, c', a^* or $x^*, x', \tilde{c}, c', \tilde{a}^*$. Therefore, the distribution of (a^*, b^*) and the distribution $(\tilde{a}^*, \tilde{b}^*)$ are identical. □

Thus, the real protocol and the simulated one are indistinguishable. □

This protocol is highly efficient. The online phase consists of sending 3 ring elements, and the computation is dominated by 2 ring multiplications. Combining the above theorem with any rrOLE protocol with semi-honest or malicious security gives rOLE protocol with the same level of security. Since OLE is a special case of rOLE with $\mathcal{R} = \mathbb{F}$, the above construction gives an efficient OLE protocol using random OLE in the offline phase, and sending 3 fields elements in the online phase.

However, rOLE for a ring $\mathcal{R} = \mathbb{F}[X]/\langle p(X)\rangle$ can not be used to compute PSI using the algebraic approach in [FNP04, FIPR05, KS05, GN19] as operations modulo $p(X)$ do not preserve roots. We thus need to consider a polynomial ring $\mathcal{R} = \mathbb{F}[X]$. Since the polynomial ring is not finite, the previous construction which relies on rrOLE does not directly apply. We then consider a special-purpose rOLE for $\mathcal{R} = \mathbb{F}[X]$ with some additional restriction on the degree and leading coefficient of each polynomial. The functionality for this special-purpose rOLE is defined in Figure 10.

When using the functionality to compute PSI, this restriction prevents high-degree terms of polynomial b to be known by a malicious receiver choosing polynomial x of low degree. Since the sender only receives a single message, we have lower restriction on the sender's input. Note that we also restrict the leading coefficient of x. This condition allows the protocol to efficiently check for malicious receivers. When we use this protocol has a part of a PSI protocol, x is chosen by its roots. Thus, the receiver may choose the polynomial to be monic. We show that doing so does not reveal the rest of the polynomial, and thus its roots.

We construct a protocol realizing $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],d}$ using $\mathcal{F}_{\mathsf{rrOLE}}$ only in the offline phase in Figure 11. The main distinction between this protocol and the previous one is that a corrupted receiver of this protocol may try to choose x of degree less than d to expose some part of b. In order to do so while passing the check on the degree of x^*, he must choose x' of degree d instead. However, we

$\Pi^{\mathsf{offline}}_{\mathsf{rOLE}}$

Offline Phase.

1. Sender receives (a', b') from $\mathcal{F}_{\mathsf{rrOLE}}$.
2. Receiver receives (x', c') from $\mathcal{F}_{\mathsf{rrOLE}}$ where $c' = a'x' + b'$.

Online Phase. Sender on input $a, b \in \mathcal{R}$, Receiver on input $x \in \mathcal{R}$

1. Receiver sends $x^* = x - x'$ to Sender.
2. Sender computes $b^* = b + b' - a'x^*$ and $a^* = a + a'$, and sends a^*, b^* to Receiver.
3. Receiver outputs $c = b^* + a^*x - c'$.

Figure 9: Protocol for $\mathcal{F}_{\mathsf{rOLE}}$ using $\mathcal{F}_{\mathsf{rrOLE}}$ in the offline phase

can show that any malicious choice of x_i's in the offline phase yields x' of degree at most $d-1$. We refer to Subsection 2.4 for the ring structure and notations.

The following theorem shows security of $\Pi^{\mathbb{F}[X],d}_{\mathsf{rOLE}}$ in the $\mathcal{F}_{\mathsf{rrOLE}}$-hybrid model.

Theorem 3.3. *$\Pi^{\mathbb{F}[X],d}_{\mathsf{rOLE}}$ realizes $\mathcal{F}^{\mathbb{F}[X],d}_{\mathsf{rOLE}}$ in the $\mathcal{F}_{\mathsf{rrOLE}}$-hybrid model.*

Proof. By the correctness of $\mathcal{F}_{\mathsf{rrOLE}}$, we have $c_0 = a_0x_0 + b_0 \pmod p$. Then

$$\begin{aligned} c' &= (a_0x_0 + b_0) - a''x'p_1 \pmod p \\ &= (a' + a''p_1)(x' + x''p_2) + (b' - a'x''p_2) - a''x'p_1 \pmod p \\ &= a'x' + b' + a''x''p_1p_2 = a'x' + b' \pmod p \end{aligned}$$

Since $\deg c' \le 2d-1$, $\deg a' = d$, $\deg x' \le d-1$, and $\deg b' \le 2d-1$, $c' = a'x' + b'$. The same proof correctness as $\Pi^{\mathsf{offline}}_{\mathsf{rOLE}}$ gives the correctness of the protocol.

Security against corrupted Sender: We construct a simulator $\mathcal{S}_{\mathsf{Sender}}$ that interacts with $\mathcal{A}$ as follows. It simulates $\mathcal{F}_{\mathsf{rrOLE}}$ honestly to extract $a_0, b_0, c_0, x_0, x', x''$. Upon receiving a'' from $\mathcal{A}$, it computes $\tilde{a}' = a_0 - a''p_1$. If $\tilde{a}'x'$ has degree at most $2d-1$, let $\tilde{b}' = b_0 + \tilde{a}'x''p_2 \pmod p$. Otherwise, it sets $\tilde{a}' = 0$ and $\tilde{b}' = c' = c_0 - a''x'p_1 \pmod p$. In the online phase, it sends x^* chosen uniformly at random from monic polynomials of degree d to the sender. Upon receiving a^*, b^*, if $\deg a^* \le d$ and $\deg b^* \le 2d$, it computes $\tilde{a} = a^* - \tilde{a}'$ and $\tilde{b} = b^* - \tilde{b}' + \tilde{a}'x^*$, and sends $(\mathsf{inputS}, (\tilde{a}, \tilde{b}))$ to $\mathcal{F}^{\mathbb{F}[X],d}_{\mathsf{rOLE}}$. Otherwise, it aborts.

We prove the indistinguishability through the following hybrids:

- H_0: This is the real world interaction of the protocol $\Pi^{\mathbb{F}[X],d}_{\mathsf{rOLE}}$.
- H_1: The same as H_0 except that $\mathcal{S}_{\mathsf{Sender}}$ simulates $\mathcal{F}_{\mathsf{rrOLE}}$ honestly to extract a_0, b_0, c_0, x_0 (which gives x', x''). Upon receiving a'' from $\mathcal{A}$, $\mathcal{S}_{\mathsf{Sender}}$ computes $\tilde{a}' = a_0 - a''p_1$. If $\tilde{a}'x'$ has degree at most $2d-1$, let $\tilde{b}' = b_0 + \tilde{a}'x''p_2 \pmod p$. Otherwise, it sets $\tilde{a}' = 0$ and $\tilde{b}' = c' = c_0 - a''x'p_1 \pmod p$. Note that $c' = \tilde{a}'x' + \tilde{b}'$ in both cases, and $\tilde{b}' = c'$ is uniformly distributed among polynomials of degree at most $2d-1$. This hybrid is identical to H_0 as $\tilde{a}'$ and $\tilde{b}'$ are not used.

$\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],d}$

Parameters. a finite field $\mathbb{F}$, an upper bound parameter d **Functionality.**

1. Upon receiving a message $(\mathsf{inputS}, (a, b))$ from the sender with $a, b \in \mathbb{F}[X]$, verify that there is no stored tuple, $\deg a \leq d$ and $\deg b \leq 2d$, else ignore the message. Otherwise, store (a, b) and send (inputS) to $\mathcal{A}$.
2. Upon receiving a message (inputR, x) from the receiver with $x \in \mathbb{F}[X]$, verify that there is no stored value, x is monic and $\deg x = d$, else ignore the message. Otherwise, store x and send (inputR) to $\mathcal{A}$.
3. Upon receiving a message $(\mathsf{deliver})$ from $\mathcal{A}$, check if both (a, b) and x are stored else ignore the message. Otherwise, set $c = ax + b$, send (output, c) to the receiver. Ignore all further messages.

Figure 10: Ideal functionality for the special-purpose rOLE for $\mathcal{R} = \mathbb{F}[X]$ where $\deg a \leq d$, $\deg b \leq 2d$ and $\deg x = d$

- H_2: The same as H_1 except that $\mathcal{S}_{\mathsf{Sender}}$, upon receiving a^*, b^*, if $\deg a^* \leq d$ and $\deg b^* \leq 2d$, it $\tilde{a} = a^* - \tilde{a}'$ and $\tilde{b} = b^* - \tilde{b}' + \tilde{a}'x^*$. Otherwise, it aborts. The receiver outputs $\tilde{a}x + \tilde{b}$ instead of $b^* + a^*x - c'$. From

$$(a^* - \tilde{a}')x + (b^* - \tilde{b}' + \tilde{a}'x^*) = a^*x + b^* - (\tilde{a}'(x + x^*) + \tilde{b}') = a^*x + b^* - c'$$

  This hybrid is identical to H_1.

- H_3: The same as H_2 except that $\mathcal{S}_{\mathsf{Sender}}$ uniformly samples a monic polynomial $\tilde{x}^*$ of degree d and sends to $\mathcal{A}$ instead of $x^* = x - x'$ and $\tilde{b} = b^* - \tilde{b}' + \tilde{a}'\tilde{x}^*$. Since the receiver is honest, x is a monic polynomial of degree d and x' the remainder of dividing a uniformly chosen x_0 of degree at most $2d - 1$ by p_2. Thus, it is uniform among polynomials of degree $d - 1$. Thus, x^* and $\tilde{x}^*$ has the same distribution. Hence, H_2 and H_3 have the same distribution.

- H_4: The same as H_3 except that $\mathcal{S}_{\mathsf{Sender}}$ sends $(\mathsf{inputS}, (\tilde{a}, \tilde{b}))$ to $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],d}$ instead of interacting with the receiver. This is the ideal world interaction. If $\mathcal{S}_{\mathsf{Sender}}$ does not abort, $\deg a^* \leq d$ and $\deg b^* \leq 2d$, so are $\tilde{a}$ and $\tilde{b}$. The receiver in the ideal world will receive $(\mathsf{output}, \tilde{a}x + \tilde{b})$. This hybrid is identical to H_3.

Security against corrupted Receiver: We construct a simulator $\mathcal{S}_{\mathsf{Receiver}}$ that interacts with $\mathcal{A}$ as follows. It simulates $\mathcal{F}_{\mathsf{rrOLE}}$ honestly to extract a_0, b_0, c_0, x_0, samples a'' honestly and computes a' of degree d. Upon receiving x'' from $\mathcal{A}$, it computes $\tilde{x}' = x_0 - x''p_2$. If $\tilde{x}'$ has degree at most $d - 1$, let $\tilde{c}' = c_0 - a''\tilde{x}'p_1 \pmod p$. Otherwise, it sets $\tilde{x}' = 0$ and $\tilde{c}' = b' = b_0 + a''x''p_1 \pmod p$.In the online phase, upon receiving x^* from $\mathcal{A}$, if x^* is not a monic polynomial of degree d, it aborts. Otherwise, it computes $\tilde{x} = x^* + \tilde{x}'$, and sends $(\mathsf{inputR}, \tilde{x})$ to $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],d}$ and receives $(\mathsf{output}, \tilde{c})$. It sends $\tilde{a}^*$ chosen uniformly at random from polynomials of degree at most d, and $\tilde{b}^* = \tilde{c} - \tilde{a}^*\tilde{x} + \tilde{c}'$ to $\mathcal{A}$.

We prove the indistinguishability through the following hybrids:

$\Pi_{\mathsf{rOLE}}^{\mathbb{F}[X],d}$

Parameters. $\mathbb{F}$ is a finite field. $\mathcal{R} = \mathbb{F}[X]/\langle p(X)\rangle$ where $p(X)$ is a product of $2d$ distinct linear terms. Elements in $\mathcal{R}$ can be represented by a polynomial of degree at most $2d-1$. Let $p(X) = p_1(X)p_2(X)$ where p_i has degree d.

Offline Phase.

1. Sender receives polynomials $([a_0], [b_0])$ from $\mathcal{F}_{\mathsf{rrOLE}}$ where $a_0, b_0 \in \mathbb{F}[X]$ are representatives of the equivalence classes with degree at most $2d-1$.
2. Receiver receives polynomial $([x_0], [c_0])$ from $\mathcal{F}_{\mathsf{rrOLE}}$ where $x_0, c_0 \in \mathbb{F}[X]$ are representatives of the equivalence classes with degree at most $2d-1$.
3. Sender computes (random) $a'' \in \mathbb{F}[X]$ of degree at most $d-1$ such that $a' := a_0 - a''p_1$ has degree d. Sender sends a'' to Receiver.
4. Receiver computes (unique) $x'' \in \mathbb{F}[X]$ of degree at most $d-1$ such that $x' := x_0 - x''p_2$ has degree at most $d-1$. Receiver sends x'' to Sender.
5. If $\deg x'' \geq d$, Sender aborts; otherwise, computes $b' = b_0 + a'x''p_2 \pmod p$.
6. If $\deg a'' \geq d$, Receiver aborts; otherwise, computes $c' = c_0 - a''x'p_1 \pmod p$.

Online Phase. Sender on input $a, b \in \mathbb{F}[X]$ with $\deg a \leq d$ and $\deg b \leq 2d$, Receiver on input $x \in \mathbb{F}[X]$ monic with $\deg x = d$

1. Receiver sends $x^* = x - x'$ to Sender.
2. If x^* is not a monic polynomial of degree d, Sender aborts. Otherwise, Sender computes $b^* = b + b' - a'x^*$ and $a^* = a + a'$, sends a^*, b^* to Receiver
3. If $\deg b^* > 2d$ or $\deg a^* > d$, Receiver aborts; otherwise, outputs $c = b^* + a^*x - c'$.

Figure 11: Protocol realizing special-purpose ring-OLE $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],d}$ using $\mathcal{F}_{\mathsf{rrOLE}}$ only in the offline phase

- H_0: This is the real world interaction of the protocol $\Pi_{\mathsf{rOLE}}^{\mathbb{F}[X],d}$.
- H_1: The same as H_0 except that $\mathcal{S}_{\mathsf{Receiver}}$ simulates $\mathcal{F}_{\mathsf{rrOLE}}$ honestly to extract a_0, b_0, c_0, x_0. It receives a'' from the receiver and passes it to $\mathcal{A}$. Upon receiving x'' from $\mathcal{A}$, it computes $\tilde{x}' = x_0 - x''p_2$. If $\tilde{x}'$ has degree at most $d-1$, let $\tilde{c}' = c_0 - a''\tilde{x}'p_1 \pmod p$. Otherwise, it sets $\tilde{x}' = 0$ and $\tilde{c}' = b' = b_0 + a''x''p_1 \pmod p$. Note that $\tilde{c}' = a'\tilde{x}' + b'$ in both cases, and $\tilde{c}' = b'$ is uniformly distributed among polynomials of degree at most $2d-1$. This hybrid is identical to H_0 as $\tilde{x}'$ and $\tilde{c}'$ are not used.
- H_2: The same as H_1 except that $\mathcal{S}_{\mathsf{Receiver}}$, upon receiving x^* from $\mathcal{A}$, if x^* is not a monic polynomial of degree d, aborts. Otherwise, it computes $\tilde{x} = x^* + \tilde{x}'$. Let $\tilde{c} = a\tilde{x} + b$. It uniformly samples a polynomial of degree at most d $\tilde{a}^*$, compute $\tilde{b}^* = \tilde{c} - \tilde{a}^*\tilde{x} + \tilde{c}'$, and send $\tilde{a}^*$ and $\tilde{b}^*$ to $\mathcal{A}$. Due to the bounded degrees, the additions and multiplications of polynomials and the ring are the same. By the same argument as Lemma 3.2, the distribution (a^*, b^*) and $(\tilde{a}^*, \tilde{b}^*)$ are identical.
- H_3: The same as H_2 except that $\mathcal{S}_{\mathsf{Receiver}}$ samples a'' on behalf of the sender, and sends $(\mathsf{inputR}, \tilde{x})$ to $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],d}$ to receive $(\mathsf{output}, \tilde{c})$. This is the ideal world interaction. If $\mathcal{S}_{\mathsf{Receiver}}$ does not abort, x^* is monic of degree d, so is $\tilde{x}$. By the correctness of $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],d}$, $\tilde{c} = a\tilde{x} + b$. This hybrid is identical to H_2.

□

3.3 Ring-OLE from PCG based on Ring-LPN

In this section, we describe an efficient construction of rrOLE protocol for $\mathcal{R} = \mathbb{F}[X]/\langle p(X)\rangle$ that can be instantiated in the previous constructions. The PCG protocol in [BCG+20] allows two parties to efficiently generate n pairs of random OLE correlated randomness (a, b) and $(x, ax + b)$ for $a, b, x \in \mathbb{F}$ by communicating only $\mathcal{O}(\log n)$ bits. Their protocol generates a batch of random OLE instances through rrOLE for the ring $\mathcal{R} = \mathbb{F}[X]/\langle p(X)\rangle$ where $p(X)$ is a product of n distinct linear terms by breaking up the result into n instances of random OLE. Since our goal is to generate rrOLE, we modify this construction by simply not breaking up the rrOLE. The security of the protocol in [BCG+20] is based on the following assumption, which is a variant of the ring-LPN assumption.

Definition 3.4 (Module-LPN [BCG+20])**.** *Let $\mathcal{R} = \mathbb{F}[X]/\langle F(X)\rangle$ for some prime-order field $\mathbb{F} = \mathbb{F}_p$ and a degree-N polynomial $F(X) \in \mathbb{F}[X]$, and let $c, m, t \in \mathbb{N}$. Let $\mathcal{HW}_t$ be the distribution of uniformly random polynomials in $\mathcal{R}$ with exactly t nonzero coefficients. The $\mathcal{R}^c-\mathsf{LPN}_{p,m,t}$ problem is hard if for any PPT adversary $\mathcal{A}$, it holds that $\Pr[\mathcal{A}((a_i, \langle a_i, e\rangle + f_i)_{i=1}^m) = 1] - \Pr[\mathcal{A}((a_i, u_i)_{i=1}^m) = 1] \le \mathsf{negl}(\kappa)$ where the probabilities are taken over $a_1, \ldots, a_m \leftarrow \mathcal{R}^{c-1}$, $u_1, \ldots, u_m \leftarrow \mathcal{R}$, $e \leftarrow \mathcal{HW}_t^{c-1}$, $f_1, \ldots, f_m \leftarrow \mathcal{HW}_t$ and the randomness of $\mathcal{A}$*

We denote $w = c \cdot t = \mathcal{O}(\kappa)$ the total number of noise positions.

Theorem 3.5 (PCG for OLEs [BCG+20])**.** *Assuming $\mathcal{R}^c - \mathsf{LPN}_{p,1,t}$, there exists a protocol that securely generates correlated randomness (a, b) and $(x, ax + b)$ where $a, b, x \leftarrow \mathcal{R} = \mathbb{F}[X]/\langle F(X)\rangle$ where $F(X)$ is a product of n distinct linear terms. The communication complexity of the protocol is $\mathcal{O}(\log n)$ and the computation complexity is $\mathcal{O}(n \log n)$.*

More specifically, the communication for setting up the seed for rrOLE is $w^2 \cdot (2\log n + 3\log|\mathbb{F}| + (2\kappa+3)\log(2n$ for semi-honest security, and $w^2 \cdot (34\log n + 10\log|\mathbb{F}| + (2\kappa+3)\log(2n) + 4\kappa) + w \cdot (\log n + \log|\mathbb{F}|)$

for malicious security. The computation consists of $4w^2n$ PRG and $\mathcal{O}(c^2 n \log n)$ field operations for semi-honest security and additional w^2 random oracle evaluations for malicious security.

Combining this theorem with our rOLE construction gives a protocol computing rOLE in the preprocessing model communicating $\mathcal{O}(n \log n)$ bits and the computation complexity is $\mathcal{O}(n \log^2 n)$ based on $\mathcal{R}^c - \mathsf{LPN}_{p,1,t}$.

Amortized setup for rrOLE. Since the communication is $\mathcal{O}(\log n)$ with relatively large constant, the setup communication is small fraction of the total communication when n is large ($n \geq 2^{20}$). For smaller n, it is more efficient to run a single setup for a large number of rOLE (and PSI) instances. As shown in Table 1, the amortized communication cost for $n = 2^8, 2^{12}, 2^{16}$ can be reduced significantly by setup for large number of instances. The same techniques may not apply to other setup such as VOLE [RS21].

4 PSI from Ring-OLE

In this section we construct a maliciously secure two-party PSI with output for one party in Figure 12 realizing the PSI functionality in Figure 4. We first assume that the input sets A, B are subsets of some finite field $\mathbb{F}$. This assumption can be dropped by assuming a random oracle $H : \{0,1\}^* \to \mathbb{F}$ when $|\mathbb{F}|$ is sufficiently large as shown in [RT21]. The idea of the construction is similar to the protocol in [KS05, GN19]. The receiver obtains a polynomial p_Q that is a linear combination of polynomials p_A and p_B whose roots include elements in the input sets of the sender and the receiver, respectively. Thus, all elements in the intersection will be roots of p_Q. Note that p_A has degree $2n$ in order to match the degree of the product $p_B p_R$. The sender chooses p_A uniformly random subjected to at most n elements in his input set. Our protocol assumes the special-purpose rOLE $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],n}$ described in Figure 10.

Π_{PSI}

Parameters. Sender on input $A \subseteq \mathbb{F}$ of size at most n for honest sender or $2n$ for corrupt. Receiver on input $B \subseteq \mathbb{F}$ of size at most n.

1. Receiver samples a monic polynomial p_B of degree n such that $p_B(b) = 0$ for all $b \in B$, and sends (inputR, p_B) to $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],n}$.
2. Sender samples a polynomial p_A of degree at most $2n$ such that $p_A(a) = 0$ for all $a \in A$ and a random polynomial p_R of degree at most n, and sends $(\mathsf{inputS}, (p_R, p_A))$ to $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],n}$.
3. Receiver receives (output, p_Q) and outputs $I = \{b \in B : p_Q(b) = 0\}$.

Figure 12: Protocol realizing PSI in the $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],n}$-hybrid model

The following theorem shows the security of Π_{PSI} in the $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],n}$-hybrid model.

Theorem 4.1. *Suppose* $\log |\mathbb{F}| \geq 2 \log n + \lambda$. *Then the protocol* Π_{PSI}^d *realizes* $\mathcal{F}_{\mathsf{PSI}}$ *in the* $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],n}$*-hybrid model with* λ *bits of statistical security.*

Proof. We first show the correctness of Π_{PSI}. By the correctness of $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],n}$, $p_Q = p_A + p_B p_R$. Thus,

for b such that $p_B(b) = 0$, $p_Q(b) = 0$ iff $p_A(b) = 0$. Since except with probability

$$1 - \left(1 - \frac{2n}{|\mathbb{F} \setminus A|}\right)^{|B \setminus A|} \leq \frac{2n|B \setminus A|}{|\mathbb{F}| - n} \leq \frac{2n^2}{|\mathbb{F}| - n}$$

all $b \in B \setminus A$ are not roots of p_A. Therefore, $I = A \cap B$ except with probability at most $2^{-\lambda}$ when $\log |\mathbb{F}| \geq 2 \log n + \lambda$. Now we consider the security of Π_{PSI}.

Security against corrupted Sender: We construct a simulator $\mathcal{S}_{\mathsf{Sender}}$ that interacts with $\mathcal{A}$ as follows. It simulates $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],n}$ honestly to receive p_A and p_R from $\mathcal{A}$. It computes $\tilde{A} = \{a \in \mathbb{F} : p_A(a) = 0\}$. Note that $|\tilde{A}| \leq \deg p_A = 2n$. It sends $(\mathsf{inputS}, \tilde{A})$ and $\mathsf{deliver}$ to $\mathcal{F}_{\mathsf{PSI}}$. We prove the indistinguishability through the following hybrids:

- H_0: This is the real world interaction of the protocol Π_{PSI}.
- H_1: The same as H_0 except that $\mathcal{S}_{\mathsf{Sender}}$ simulates $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],n}$ honestly. This hybrid is identical to H_0.
- H_2: The same as H_1 except that $\mathcal{S}_{\mathsf{Sender}}$ sends $(\mathsf{inputS}, \tilde{A})$ and $\mathsf{deliver}$ to $\mathcal{F}_{\mathsf{PSI}}$ instead of sending $p_Q = p_A + p_B p_R$ to the receiver. This is the ideal world interaction. Since the receiver is honest, $p_B(b) = 0$ for all $b \in B$. For each $b \in B$, $p_Q(b) = 0$ if and only if $p_A(b) = 0$. Thus, $\tilde{A} \cap B = \{b \in B : p_Q(b) = 0\}$. This hybrid is identical to H_1.

Security against corrupted Receiver: We construct a simulator $\mathcal{S}_{\mathsf{Receiver}}$ that interacts with $\mathcal{A}$ as follows. It simulates $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],n}$ honestly to receive p_B from $\mathcal{A}$. It computes $\tilde{B} = \{b \in \mathbb{F} : p_B(b) = 0\}$. Note that $|\tilde{B}| \leq \deg p_B = n$. It sends $(\mathsf{inputR}, \tilde{B})$ and $\mathsf{deliver}$ to $\mathcal{F}_{\mathsf{PSI}}$ and receives $(\mathsf{output}, \tilde{I})$. It samples uniformly $\tilde{p}_Q$ of degree at most $2n$ such that $\tilde{p}_Q(c) = 0$ for all $c \in \tilde{I}$ and $\tilde{p}_Q(c) \neq 0$ for all $c \in \tilde{B} \setminus \tilde{I}$. Finally, it sends $(\mathsf{output}, \tilde{p}_Q)$ to $\mathcal{A}$. We prove the indistinguishability through the following hybrids:

- H_0: This is the real world interaction of the protocol Π_{PSI}.
- H_1: The same as H_0 except that $\mathcal{S}_{\mathsf{Receiver}}$ simulates $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],n}$ honestly. This hybrid is identical to H_0.
- H_2: The same as H_1 except that $\mathcal{S}_{\mathsf{Receiver}}$ computes $\tilde{I} = \{c \in \mathbb{F} : p_B(c) = 0 \text{ and } p_A(c) = 0\}$, then uses it to samples uniformly $\tilde{p}_Q$ of degree at most $2n$ such that $\tilde{p}_Q(c) = 0$ for all $c \in \tilde{I}$ and $\tilde{p}_Q(c) \neq 0$ for all $c \in \tilde{B} \setminus \tilde{I}$. We consider the following lemma, which is similar to Lemma 1 of [SCK12].

  Lemma 4.2. *Let $f, g \in \mathbb{F}[X]$ with $\max(\deg f, \deg g) = c \leq d$ and $\gcd(f, g) = 1$ and $r, s \in \mathbb{F}[X]$ are chosen uniformly and independently with degree at most d. Then $fr+gs$ is uniformly distributed among polynomials degree at most $c + d$.*

  Proof. Define $M(r, s) = fr+gs$ for polynomials $r, s \in \mathbb{F}[X]$ of degree at most d. By identifying polynomials with the vectors of its coefficients, we may consider $M : \mathbb{F}^{d+1} \times \mathbb{F}^{d+1} \to \mathbb{F}^{d+c+1}$. Suppose r, r', s, s' be polynomials of degree at most d with $M(r, s) = M(r', s')$. Then $(r - r')f = (s' - s)g$. Since $\gcd(f, g) = 1$, there exists a polynomial h of degree at most $d - c$ such that $r - r' = gh$ and $s' - s = fh$. For any r, s of degree at most d, we can find r', s' with the same image for each h of degree at most $d - c$. Thus, the number of pre-images is $|\mathbb{F}|^{d-c+1}$ for polynomial in the range of M. Since $|\mathbb{F}^{d+1} \times \mathbb{F}^{d+1}|/|\mathbb{F}|^{d-c+1} = |\mathbb{F}|^{d+c+1}$, M is surjective. Therefore, choosing r, s uniformly and independently gives uniform $M(r, s) = fr + gs$. □

Since the sender is honest, we can write $p_A = p_a \cdot p_i \cdot r$ where $p_i(X) = \prod_{c \in \tilde{I}}(x - c)$ and $p_a(X) = \prod_{a \in A \setminus \tilde{I}}(x - a)$ and $r = p_A/(p_a \cdot p_i)$. Then $p_B = p_b \cdot p_i$ for some polynomial p_b. Applying the above lemma to p_a, p_b gives $p_a r + p_b p_R$ being uniform if r, p_R are chosen uniformly with degree at most n. This is the case as the sender is honest. (Assuming $|A| = n$ or add dummy elements.) We can write $p_Q = p_A + p_B p_R = p_i(p_a r + p_b p_R)$ and $\tilde{p}_Q = p_i \tilde{p}$ for a uniformly chosen $\tilde{p}$ of degree at most $2n - \deg p_i = n + \deg p_b$. Thus, $p_Q = p_A + p_B p_R$ has the same distribution as $\tilde{p}_Q$. Thus, H_1 and H_2 have identical distribution.

- H_3: The same as H_2 except that $\mathcal{S}_{\mathsf{Receiver}}$ sends $(\mathsf{inputR}, \tilde{B})$ and $\mathsf{deliver}$ to $\mathcal{F}_{\mathsf{PSI}}$ and receives $(\mathsf{output}, \tilde{I})$. It samples uniformly $\tilde{p}_Q$ of degree at most $2n$ such that $\tilde{p}_Q(c) = 0$ for all $c \in \tilde{I}$ and $\tilde{p}_Q(c) \neq 0$ for all $c \in \tilde{B} \setminus \tilde{I}$. Finally, it sends $(\mathsf{output}, \tilde{p}_Q)$ to $\mathcal{A}$. This is the ideal world interaction. As discussed above, except with probability at most $2^{-\lambda}$, all $b \in B \setminus A$ are not roots of p_A. If this is the case, $\tilde{I}$ output from $\mathcal{F}_{\mathsf{PSI}}$ and $\tilde{I}$ computed in H_2 are identical. Thus, H_2 and H_3 are statistically close.

□

We note that by assuming $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],n}$, the protocol is statistically secure with statistical security parameter λ. By instantiating $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}[X],n}$ with $\Pi_{\mathsf{rOLE}}^{\mathbb{F}[X],n}$, we obtain the following corollary.

Corollary 4.3. *There exists a protocol realizing $\mathcal{F}_{\mathsf{PSI}}$ in the $\mathcal{F}_{\mathsf{rrOLE}}$-hybrid model with $n' = 2n$ and statistical security parameter λ having communication complexity $\mathcal{O}(n(\log n + \lambda))$ and computation complexity $\mathcal{O}(n \log^2 n)$.*

Next, we will analyze communication and computation complexity of this protocol.

4.1 Communication

The communication complexity of Π_{PSI} consists of bits communicated during the rrOLE for $\mathcal{R} = \mathbb{F}[X]/\langle p(X) \rangle$, two polynomials of degree at most $n - 1$ in the offline phase, two polynomials of degree at most n and a polynomial of degree at most $2n$ in the online phase. Let $c_{\mathsf{rrOLE}}(n)$ be the number of bits exchanged for rrOLE for $\mathcal{R} = \mathbb{F}[X]/\langle p(X) \rangle$ with $\deg p = n$. Then the total number of bits communicated is

$$(c_{\mathsf{rrOLE}}(2n) + 2n \log |\mathbb{F}|) + (4n + 3) \log |\mathbb{F}|$$

where $c_{\mathsf{rrOLE}}(2n) + 2n \log |\mathbb{F}|$ bits are communicated in the offline phase and $(4n + 3) \log |\mathbb{F}|$ bits in the online phase. Since $c_{\mathsf{rrOLE}}(2n) = \mathcal{O}(\log n)$, the total communication is approximately $6n \log |\mathbb{F}|$.

The previous OLE-based construction uses $4n + \mathcal{O}(1)$ instances of OLE in the online phase [GN19]. Each instance requires at least $9 \log |\mathbb{F}|$ bits using homomorphic encryption (GAZELLE) for semi-honest security. Upgrading to malicious security doubles the communication [GN19, HIMV19]. Using the random OLE in the offline phase using PCG setup [BCG+20] can reduce the communication to $3 \log |\mathbb{F}|$ bits in the online phase. Even in that case, the online communication is at least $12 \log |\mathbb{F}|$ which is 3x our construction while the total communication is 2x our construction.

4.2 Computation

We consider only rrOLE, polynomial evaluation and interpolation as they are much slower than other operations. Note that polynomial interpolation where the given points are roots is much faster than the general case. In the offline phase, the sender's and the receiver's computation are dominated by one instance of rrOLE($2n$). In the online phase, the sender's computation is dominated by degree $2n$ polynomial interpolation on n roots. The receiver's computation is dominated by degree n polynomial interpolation on n roots and degree $2n$ polynomial evaluation on n values.

More Detailed Analysis of Computation Complexity Using FFT-based polynomial evaluation and interpolation algorithms, we can approximate the number of field operations for our PSI algorithm. One FFT on a polynomial of degree $n-1$ uses $\frac{1}{2}n\log n$ field operations. Polynomial multiplication using FFT thus uses $\frac{3}{2}n\log n$ field operations. The FastEval algorithm consists of building a subproduct tree using $\log n$ multiplications and using the tree to compute the evaluation. The total field operations used in FastEval to evaluate n values is $17n\log^2 n$. Note that the degree of the polynomial has little influence on this number when n is large. Using FastEval to interpolate a polynomial on n points requires additional $2\log n$ multiplications. Thus, interpolation uses $20n\log^2 n$ field operations. Its special case of interpolation on roots can be done similar to computing subproduct tree. Thus, it requires $\frac{3}{2}n\log^2 n$ field operations.

5 PSI payload from Ring-OLE

In this section, we modify our PSI protocol to support payload. Payload is a value associated with each element in the sender's set. PSI payload protocol allows the receiver to learn not only the intersection but also the payload associated with elements in the intersection. On the other hand, the receiver learns nothing about the payload of elements not in the intersection. PSI payload functionality is described in Figure 13.

$\mathcal{F}_{\mathsf{PSIPayload}}$

Parameters. n is the upper bound of the size of parties' input sets; $n' \geq n$ is the upper bound of the corrupted sender's input set; ℓ is the bit length of payloads.

Functionality.

1. Upon receiving a message $(\mathsf{inputS}, A, (t_a)_{a\in A})$ from the sender with $A \subseteq \{0,1\}^*$ and $t_a \in \{0,1\}^\ell$ for each $a \in A$, if $|A| > n$ and the sender is honest, or $|A| > n'$ and the sender us corrupted, ignoring that message. Otherwise, store $A, (t_a)_{a\in A}$ and send (inputS) to $\mathcal{A}$.
2. Upon receiving a message (inputR, B) from the receiver with $B \subseteq \{0,1\}^*$, if $|B| > d$, ignoring that message. Otherwise, store B and send (inputR) to $\mathcal{A}$.
3. Upon receiving a message $(\mathsf{deliver})$ from $\mathcal{A}$, check if both A and B are stored, else ignoring that message. Otherwise, set $I = A \cap B$ and send $(\mathsf{output}, I, (t_c)_{c\in I})$ to the receiver. Ignore all further messages.

Figure 13: Ideal functionality for PSI payload

As in the PSI case, we may assume $A, B \subseteq \mathbb{F}$ and the payloads are in a different field $\mathbb{F}'$ with $\ell = \lfloor \log |\mathbb{F}'| \rfloor$. For $\lambda_0 \in \mathbb{N}$ (to be determined later), let $\mathbb{F}''$ be a field of size at least $\mathsf{max}(2^{\lambda_0}|\mathbb{F}'|, |\mathbb{F}|)$. Throughout the protocol, we may consider an element $x \in \mathbb{F}$ as its embedded element in $\mathbb{F}''$ as well. Our PSI payload protocol is described in Figure 14.

$\Pi_{\mathsf{PSIPayload}}$

Parameters. Sender on input A of size at most n for honest sender or $2n$ for corrupt, and associated payload $T = (t_a)_{a \in A}$. Receiver on input B of size at most n.

1. Receiver samples a polynomial p_B of degree n with coefficients in $\mathbb{F}''$ such that $p_B(b) = 0$ for all $b \in B$, and sends (inputR, p_B) to $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}''[X],n}$.
2. Sender samples a polynomial p_A of degree $2n$ with coefficients in $\mathbb{F}''$ such that $p_A(a) = (0^{\lambda_0} \| t_a)$ for all $a \in A$ and a random polynomial p_R of degree n, and sends $(\mathsf{inputS}, (p_R, p_A))$ to $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}''[X],n}$.
3. Receiver receives (output, p_Q) and outputs $I = \{b \in B : \exists t_b \in \mathbb{F}', p_Q(b) = (0^{\lambda_0} \| t_b)\}$ and $(t_b)_{b \in I}$.

Figure 14: Protocol realizing PSI payload in the $\mathcal{F}_{\mathsf{rOLE}}^{\mathbb{F}''[X],n}$-hybrid model

5.1 Correctness and Security

Similar to the non-payload version, we have $p_Q = p_A + p_B p_R$. Thus, for b such that $p_B(b) = 0$, $p_Q(b) = p_A(b)$. By the union bound, the probability that some $b \in B \setminus A$ will have $p_A(b) = (0^{\lambda_0} \| t_b)$ for some t_b, that is $p_A(b) \leq |\mathbb{F}'|$, is $|B \setminus A| \cdot \frac{|\mathbb{F}'|}{|\mathbb{F}''|} \leq \frac{n}{2^{\lambda_0}}$ Therefore, $I = A \cap B$ except with probability $2^{-\lambda}$ when $\lambda_0 = \log n + \lambda$.

The proof of security is similar to the non-payload version.

5.2 Communication

To achieve the correctness, we have $|\mathbb{F}''| \geq 2^{\lambda_0} |\mathbb{F}'|$. Let ℓ be the bit length of payload. We have $\log |\mathbb{F}'| \geq \ell$ and thus $\log |\mathbb{F}''| \geq \ell + \log n + \lambda$. The communication complexity of $\Pi_{\mathsf{PSIPayload}}$ then is the same as Π_{PSI} except with $\log |\mathbb{F}|$ is replaced by $\log |\mathbb{F}''| = \ell + \lambda + \log n$. When λ is the security parameter, the communication is $c_{\mathsf{rrOLE}}(2n) + 2n(\ell + \lambda + \log n)$ in the offline phase and $(4n + 3)(\ell + \lambda + \log n)$ in the online phase. Note that this communication is exactly the same as the non-payload version if the size of payload is the same as the set size.

5.3 Computation

Unlike the communication, the computation of our PSI payload for the sender increases from the non-payload version quite significantly even when the field size is the same. This increase comes from the interpolation in the Online phase. In the non-payload version, the sender computes p_A by interpolating at roots, which is easy to compute. However, in PSI payload, the sender has to compute interpolation at $p_A(a) = (0^{\lambda_0} \| t_a)$. This almost doubles the computation of the Online phase. Also, the increase in the field size discussed above also causes the computation of every evaluation and interpolation to increase as well.

$\mathcal{F}_{\mathsf{PSI+Sum}}$

Parameters. n is the upper bound of the size of parties' input sets; $n' \geq n$ is the upper bound of the corrupted sender's input set; payloads are in a finite field $\mathbb{F}'$.

Functionality.

1. Upon receiving a message $(\mathsf{inputS}, A, (t_a)_{a\in A})$ from the sender with $A \subseteq \{0,1\}^*$ and $t_a \in \mathbb{F}'$, if $|A| > n$ and the sender is honest, or $|A| > n'$ and the sender is corrupted, ignoring that message. Otherwise, store $A, (t_a)_{a\in A}$ and send (inputS) to $\mathcal{A}$.
2. Upon receiving a message (inputR, B) from the receiver with $B \subseteq \{0,1\}^*$, if $|B| > n$, ignoring that message. Otherwise, store B and send (inputR) to $\mathcal{A}$.
3. Upon receiving a message $(\mathsf{deliver})$ from $\mathcal{A}$, check if both A and B are stored, else ignoring that message. Otherwise, set $I = A \cap B$, $s = \sum_{c\in I} t_c$, sample $o_A \leftarrow \mathbb{F}'$, send $(\mathsf{outputS}, o_A)$ to the sender, and send $(\mathsf{outputR}, S_\cap, o_B = s - o_A)$ to the receiver. Ignore all further messages.

Figure 15: Ideal functionality for PSI+Sum

6 PSI+Sum

Private set intersection sum (PSI-Sum) is a natural extension of PSI payload. Instead of learning the intersection and the associated payloads, a receiver only learns the sum of the associated payloads. In this setting, similar to PSI payload, there are two parties, a sender with a set A and associated payload set T, and a receiver with set B. The goal of PSI-Sum is to compute the sum s of $\{t_c\}_{c\in A\cap B}$. In most cases, the receiver learns s while one of the party or both learn the size of the intersection $|A \cap B|$ [IKN$^+$17, IKN$^+$20, MPR$^+$20].

In this work, we consider a variant of PSI-Sum where the receiver also learn the intersection as well, but not individual associated payload. Moreover, the sum value is secret-shared between the sender and the receiver. To distinguish from the above variant, we call this variant *PSI+Sum*, as it is the basic PSI functionality (computing the intersection) plus computing the sum. The ideal functionality is defined in Figure 15. The secret share allows any party to learn s by obtaining other party's share and thus more flexible compared to [IKN$^+$17, IKN$^+$20] where different protocols are defined to give each party the sum, or to both parties in [MPR$^+$20]. We refer to Section 7 for more details.

6.1 PSI+Sum from *Any* PSI Payload and Secure Inner Product

We construct a PSI+Sum from PSI payload functionality and a secure inner product functionality in Figure 16. Informally, the protocol runs PSI payload on masked payload by a random polynomial evaluated at each element of A (embedded in the same field as the payload). The mask can be computed via the inner product. The secure inner product outputs secret shares of the mask, which then combine with the sum of the output of the PSI payload. The computation of $\vec{v}_B$ in Step 5 can be done efficiently in $\mathcal{O}(n \log^2 n)$ using $\mathsf{PowerSum}$ described in Section 2.1.3.

We note that the intersection-revealing property follows from the intersection revealed in $\mathcal{F}_{\mathsf{PSIPayload}}$.

$\Pi_{\mathsf{PSI+Sum}}$

Parameters. Sender on input A of size at most n for honest sender or n' for corrupt, and associated payload $T = (t_a)_{a \in A}$. Receiver on input B of size at most n.

1. Sender samples a random polynomial p of degree n, and compute $T' = (t_a - p(a))_{a \in A}$. Let $\vec{v}_A \in \mathbb{F}'^{n+1}$ be the coefficient vector of p.
2. Sender sends (inputS, A, T') to $\mathcal{F}_{\mathsf{PSIPayload}}$.
3. Receiver sends (inputR, B) to $\mathcal{F}_{\mathsf{PSIPayload}}$ and receives $(\mathsf{output}, I, T'_I)$ where $T'_I = (t'_c)_{c \in I}$.
4. Sender sends $(\mathsf{input}_1, \vec{v}_A)$ to $\mathcal{F}^{n+1}_{\mathsf{InnerProduct}}$.
5. Receiver computes $\vec{v}_B \in \mathbb{F}'^{n+1}$ where $\vec{v}_B[i] = \sum_{c \in I} c^i$ and sends $(\mathsf{input}_2, \vec{v}_B)$ to $\mathcal{F}_{\mathsf{InnerProduct}}$.
6. Sender and Receiver receive (output_1, o_A) and $(\mathsf{output}_2, o'_B)$ from $\mathcal{F}_{\mathsf{InnerProduct}}$, and output o_A and $o_B = \sum_{c \in I} t'_c + o'_B$, respectively.

Figure 16: Protocol realizing PSI+Sum in the $\mathcal{F}_{\mathsf{PSIPayload}}$ and $\mathcal{F}_{\mathsf{InnerProduct}}$-hybrid model

Replacing the functionality with intersection-hiding PSI payload gives standard PSI-Sum in [IKN+20]. Similar to the protocols in [IKN+20], our protocol is only secure against semi-honest receiver. A malicious adversary corrupting the receiver can choose to proceed with any subset of the intersection after learning it. On the other hand, as we can see in the following security analysis, a malicious sender for $\Pi_{\mathsf{PSI+Sum}}$ learns nothing from the protocol. Thus, by instantiating $\mathcal{F}_{\mathsf{PSIPayload}}$ and $\mathcal{F}_{\mathsf{InnerProduct}}$ with maliciously secure construction, we obtain a more secure sum PSI protocol.

6.2 Correctness and Security

Here we provide a security proof for $\Pi_{\mathsf{PSI+Sum}}$.

Theorem 6.1. $\Pi_{\mathsf{PSI+Sum}}$ *realizes* $\mathcal{F}_{\mathsf{PSI+Sum}}$ *in the* $\mathcal{F}_{\mathsf{PSIPayload}}$ *and* $\mathcal{F}_{\mathsf{InnerProduct}}$*-hybrid model against malicious sender and semi-honest receiver.*

Proof. (Sketch) By the correctness of $\mathcal{F}_{\mathsf{PSIPayload}}$, $I = A \cap B$ and $t'_c = t_c - p(c)$. By the correctness of $\mathcal{F}_{\mathsf{InnerProduct}}$, $o_A + o'_B = \langle \vec{v}_A, \vec{v}_B \rangle = \sum_{c \in I} p(c)$. Thus, $o_A + o_B = \left(\sum_{c \in I} t'_c\right) + \sum_{c \in I} p(c) = \sum_{c \in A \cap B} t_c$.

Now suppose the sender is corrupted. We construct the simulator by simulating $\mathcal{F}_{\mathsf{PSIPayload}}$ and $\mathcal{F}_{\mathsf{InnerProduct}}$ to extract A, T' and $\vec{v}_A$. It then reconstructs the polynomial p of degree n and uses it to compute T. It sends (inputS, A, T) to $\mathcal{F}_{\mathsf{PSI+Sum}}$ and receive $(\mathsf{outputS}, o_A)$ and sends o_A to the adversary. o_A from $\mathcal{F}_{\mathsf{PSI+Sum}}$ and from $\mathcal{F}_{\mathsf{InnerProduct}}$ have the same distribution.

Now suppose the receiver is corrupted. We construct the simulator by simulating $\mathcal{F}_{\mathsf{PSIPayload}}$ to extract B It sends (inputR, B) to $\mathcal{F}_{\mathsf{PSI+Sum}}$ and receives $(\mathsf{outputR}, I, o_A)$. It generates T'_I by sampling each element from $\mathbb{F}'$ and sends I, T'_I to the adversary. The simulator then simulates $\mathcal{F}_{\mathsf{InnerProduct}}$ to extract $\vec{v}_B$, and sends $o'_B = o_B - \sum_{c \in I} t'_c$. Since $|I| < d = \deg p$, then T'_I is also uniformly random in $\Pi_{\mathsf{PSI+Sum}}$. Since the receiver is semi-honest, $\vec{v}_B$ is computed from T'_I and thus o'_B has the same distribution. □

We can instantiate $\mathcal{F}_{\mathsf{PSIPayload}}$ and $\mathcal{F}_{\mathsf{InnerProduct}}$ by $\Pi_{\mathsf{PSIPayload}}$ in Figure 14 and $\Pi_{\mathsf{innerProduct}}$ in Figure 6. We obtain the following corollary.

Corollary 6.2. *There exists a protocol realizing $\mathcal{F}_{\mathsf{PSI+Sum}}$ from rrOLE against malicious sender and semi-honest receiver.*

We will then analyze the performance of $\Pi_{\mathsf{PSI+Sum}}$ with the above instantiation.

6.3 Communication

The communication complexity of $\Pi_{\mathsf{PSI+Sum}}$ consists entirely of the communication of $\Pi_{\mathsf{PSIPayload}}$ and $\Pi_{\mathsf{innerProduct}}$ in both Offline and Online phases. From Section 5, the communication complexity of $\Pi_{\mathsf{PSIPayload}}$ consists of the communication of rrOLE and $2n(\ell + \lambda + \log n)$ in the Offline phase, and $4n(\ell + \lambda + \log n)$ bits in the Online phase, for ℓ-bit payload and statistical security parameter λ. From Section 2.3, the communication complexity of $\Pi_{\mathsf{InnerProduct}}$ consists of the communication of n instances of random OLE over $\mathbb{F}'$ in the Offline phase and $2n\ell$ bits in the Online phase. Using PCG setup [BCG+20] reduces the communication to $\mathcal{O}(\log n)$ bits in the offline phase. Thus, the communication in the offline phase is $\mathcal{O}(\log n) + 2n(\ell + \lambda + \log n)$, and the communication in the online phase is $4n(\ell + \lambda + \log n) + 2n\ell$.

Table 1: Communication of PSI protocols when instantiated with computational security parameter $\kappa = 128$ and statistical security parameter $\lambda = 40$ and $n = 2^8, 2^{12}, 2^{16}, 2^{20}, 2^{24}$. In Tiny-PSI, $\phi = 256$ is the element size for an elliptic curve. In PaXoS and OKVS-PSI, ℓ, ℓ_1 for $n = 2^{12}, 2^{16}, 2^{20}, 2^{24}$ are provided in [PRTY20] based on the security parameters. We amortize the rrOLE setup over 2^{20} and split it into $2^{20}/n$ smaller instances. See Section 3.3 for more information. The lowest online communication is denoted in bold font. The lowest total communication is underlined.

Protocol	Communication	Set size					Assumption
		2^8	2^{12}	2^{16}	2^{20}	2^{24}	
semi-honest							
OLE [GN19]+PCG [BCG+20] (online)	$12(\lambda + 2\log n)n$	$672n$	$768n$	$864n$	$960n$	$1056n$	rLPN
SpOT-low [PRTY19]	$3.5\kappa n + 1.02(2 + \lambda + \log n)n$	$501n$	$505n$	$509n$	$513n$	$517n$	CDH
SpOT-fast [PRTY19]	$3.5(1 + 1/\lambda)\kappa n + 2(\lambda + 2\log n)n$	$571n$	$587n$	$603n$	$619n$	$635n$	CDH
PaXoS [PRTY20]	$2.4n\ell + (\lambda + 2\log n)n$	-	$1139n$	$1207n$	$1268n$	$1302n$	CDH
CM [CM20]	$4.8\kappa n + (\lambda + 2\log n)n$	$670n$	$678n$	$686n$	$694n$	$702n$	CDH
VOLE+PaXoS (online) [RS21]	$2.4\kappa n + (\lambda + 2\log n)n + 2^{13}\kappa n^{1/8}$	$8547n$	$1083n$	$502n$	$398n$	$396n$	LPN+CDH
VOLE+PaXoS (total) [RS21]	$2.4\kappa n + (\lambda + 2\log n)n + 2^{17}\kappa n^{1/20}$	$86838n$	$6580n$	$914n$	$426n$	$398n$	
MiniPSI [RT21]	$\phi n + (\lambda + 2\log n)n$	$\underline{312n}$	$\underline{320n}$	$\underline{328n}$	$\underline{336n}$	$\underline{\mathbf{344n}}$	CDH
OKVS-PSI [GPR+21]	$1.3n\ell + (\lambda + 2\log n)n$	-	$646n$	$687n$	$724n$	$746n$	CDH
Ours (online)	$4(\lambda + 2\log n)n$	$\mathbf{224n}$	$\mathbf{256n}$	$\mathbf{288n}$	$\mathbf{320n}$	$352n$	rLPN
Ours (total)	$6(\lambda + 2\log n)n + (\kappa^3/8 + \kappa^2)\log n$	$9360n$	$1215n$	$500n$	$485n$	$528n$	
Ours (total, amortized)		$341n$	$389n$	$437n$	$485n$	$528n$	
malicious							
OLE [GN19]+PCG [BCG+20] (online)	$24(\lambda + 2\log n)n$	$1344n$	$1536n$	$1728n$	$1920n$	$2112n$	rLPN
PaXoS [PRTY20]	$2.4n\ell + \ell_1 n + 2.4\lambda n$	-	$2191n$	$2164n$	$2098n$	$2061n$	CDH
VOLE+PaXoS (online) [RS21]	$3.4\kappa n + 2^{13}\kappa n^{1/8}$	$8627n$	$1159n$	$558n$	$446n$	$436n$	LPN+CDH
VOLE+PaXoS (total) [RS21]	$3.4\kappa n + 2^{17}\kappa n^{1/20}$	$87038n$	$6772n$	$960n$	$\underline{474n}$	$\underline{438n}$	
MiniPSI [RT21]	$2\kappa n + \phi n$	$512n$	$512n$	$512n$	$512n$	$512n$	CDH
OKVS-PSI [GPR+21]	$1.3n\ell + \ell_1 n + 1.3\lambda n$	-	1294	$1275n$	$1236n$	$1213n$	CDH
Ours (online)	$4(\lambda + 2\log n)n$	$\mathbf{224n}$	$\mathbf{256n}$	$\mathbf{288n}$	$\mathbf{320n}$	$\mathbf{352n}$	rLPN
Ours (total)	$6(\lambda + 2\log n)n + (\kappa^3/8 + 3\kappa^2)\log n$	$11960n$	$1423n$	$516n$	$486n$	$528n$	
Ours (total, amortized)		$\underline{342n}$	$\underline{390n}$	$\underline{438n}$	$486n$	$528n$	

6.4 Computation

The computation complexity of $\Pi_{\mathsf{PSI+Sum}}$ consists of the computation of $\Pi_{\mathsf{PSIPayload}}$ and $\Pi_{\mathsf{innerProduct}}$ in both Offline and Online phases, n-point polynomial evaluation by the Sender in the Offline phase and PowerSum by the Receiver in the Online phase. The computation of $\Pi_{\mathsf{PSIPayload}}$ consists of the computation of rOLE in the Offline phase while the Online phase is dominated by $2n$-point

polynomial interpolation and n point polynomial evaluation. The computation of $\Pi_{\mathsf{innerProduct}}$ consists of the computation of n OLE in the Offline phase, and $4n$ field additions and $2n$ field multiplications, which are dominated by other operations. The computation complexity of ring-OLE and OLE depend on how each OLE implementation handles fields of different size. In any case, the increase is at most 50%, from PSI payload, which is the same as standard PSI when the field size is the same. Since PowerSum is as difficult as multi-point evaluation, the computation in the Online phase thus increases by about 50%.

7 Other Variations

In this section, we briefly discuss the possibility of extending our algebraic construction to other PSI variants, in particular, threshold PSI and PSI cardinality.

7.1 Threshold PSI

We consider a variant of PSI, called threshold PSI where the receiver only learns the intersection if the intersection size satisfies a certain condition. Concretely, we consider the version of *threshold PSI* defined in [GS19, BMRR21], where the receiver only learns the intersection when the size of the *difference* between the two input sets is smaller than a given (publicly known) threshold. Since the protocol in [GS19] is also using algebraic technique, we can modify it to our ring-OLE and preprocessing model. The protocol in [GS19] consists of two steps. First, the protocol checks if the difference satisfies the condition using additively homomorphic encryption. Second, the protocol computes the intersection using the rational function $\frac{V(x)}{p_B(x)}$ where p_A, p_B are polynomials whose roots are sender's and receiver's input set, respectively, and $V(x)$ is a linear combination of p_A, p_B. The linear factors corresponding to the intersection cancel out, and the rational function can be written as $\frac{U(x)}{p_{B\setminus A}(x)}$ which can be reconstructed to find $B\setminus A$ and the intersection. $V(x)$ are computed point-wise via OLE.

To increase the efficiency, especially in the ring-OLE and preprocessing model, we consider the Reversed Laurent Series (RLS) introduced for algebraic construction of private set union (PSU) protocol in [SCK12]. Any rational functions can be represented by the RLS, which in turn can be represented by a polynomial up to a certain number of terms. With sufficient number of terms, the rational function can be recovered. Similar to their reconstruction of rational functions, this number of terms comes from the threshold. Since the RLS can be represented by a polynomial, we can use ring-OLE in the preprocessing model to compute the linear combination.

7.2 PSI Cardinality with Payload

We consider a variation of PSI payload where the receiver only learns the payload of the elements of the intersection, but not the intersection itself. Since the number of payloads the receiver learns is the same as the size of the intersection, the receiver also learns the size of the intersection, hence, PSI cardinality.

Before discussing this PSI payload variant, we consider how the algebraic technique can be used to compute PSI cardinality. Our algebraic technique computes a polynomial whose roots include the members of the intersection. Unfortunately, this polynomial is a random linear combination of polynomials representing input sets. It also has other roots that, with overwhelming probability, do not correspond to members of either set. Thus, we cannot simply compute the cardinality from its degree. In our PSI protocol, the receiver computes the intersection from this polynomial by

evaluating it at each member of his input set. To hide the intersection, we need to replace each member of the receiver's input set by a random number (possibly in a different field).

We can use an oblivious pseudorandom function (OPRF), as described in [KKRT16, PSTY19, PRTY19, IKN$^+$20]. An OPRF allows the receiver to learn $F_k(b)$ for each b in his input set without learning the pseudorandom function F_k. The sender knows F_k but not $F_k(b)$, and he can compute $F_k(a)$ for each a in his input set. Finally, the sender simply sends his $F_k(a)$'s to the receiver. The OPRF by itself already allows one to compute PSI as in [KKRT16].

We can apply batch OPRFs to each element in the set and permute the resulting random numbers for the receiver's input set. We call this variation of batch OPRFs a batch shuffled OPRFs, where the receiver evaluates F_k on every member of his input set at once without learning their order. The receiver will construct his polynomial from this numbers while the sender can compute the same PRF for elements in his input set. This way, the receiver will know only the payload associated with each element in the intersection but not the element itself. Thus, we get a PSI cardinality. Finally, combining the OPRF with our PSI payload gives PSI cardinality with payload.

We can construct a batch shuffled OPRF using polynomials and homomorphic encryption as follows. The receiver computes a polynomial whose roots are members of his input set. He then encrypts the coefficients with homomorphic encryption and sends them to the sender. The sender picks a random polynomial f of degree n and homomorphically computes a polynomial whose roots are $f(b_i)$'s where b_i's are the roots. This transformation only consists of fixed number of additions and multiplications of the coefficients. Finally, the sender sends the transformed polynomial to the receiver, who can decrypt and factor to find $f(b_i)$'s in a random order. The problem of computing the transformed polynomial using only ring-OLE in the preprocessing model is still open.

Finally, we consider the Sum PSI variant in [IKN$^+$20] where the receiver learns the size of the intersection and the sum of the payload associated with the members of the intersection, but not the intersection itself. As we mentioned above, we can achieve this result by applying Π^d_{SumPSI} to the PSI cardinality with payload.

7.3 PSI Payload from Other PSI Protocols

We can modify the existing PSI protocols based the OPRF approach from [PRTY19, PRTY20, RS21, GPR$^+$21], and other techniques from [CM20, RT21] to allow payload as follows. In these protocols, each element in the sender's and the receiver's input sets are mapped in some ways using pseudorandom functions or encodings. This mapping is defined in a way that the receiver can interactively computed the mapping on only their input set. The sender then sends hashed values of the output of this mapping on his input set. This allows the receiver to compare by hashing his own, but not other elements that are not in his input set. The sender can use the output of this mapping to derive keys to encrypt the payload associating to each element of the sender's set, and send the encrypted payloads to the receiver. Since the receiver can only compute these keys associating to elements in his set, he can only decrypt the payload associating to the intersection.

Comparing to our PSI payload, this modification is not as efficient. As we mentioned in Section 5, our PSI payload communicates the same number of bits as our PSI protocol when the payload length is smaller than $\log n$ where n is the set size. While both this modification and our PSI payload require more computation from the sender, the receiver in our protocol does about the same work as without payload.

Among these protocols, the protocols in [RS21, GPR$^+$21] use a programmable PRF (PPRF) instead, which allows the sender to choose the outputs of the function for his inputs. Thus, a similar technique to our PSI payload can be used by attaching the payload to the programmed outputs.

Thus, they can be modified into PSI payload quite efficiently as well.

8 Performance Evaluation

In this section, we compare the communication and computation complexity of our PSI protocol and its variants against other recent results both theoretically and experimentally.

8.1 Theoretical Comparison

PSI. We instantiate our PSI protocol in Figure 12 with rOLE in Figure 11 and rrOLE from [BCG$^+$20]. We compare the theoretical communication complexity of the resulting PSI protocol against several PSI protocols in literature including OLE-based PSI [GN19], OPRF-based PSI [PRTY19, PRTY20, CM20, RS21] and its variant [GPR$^+$21], and Diffie-Hellman (DH)-based PSI [RT21]. We instantiate the OLE PSI from [GN19] using random OLE from [BCG$^+$20]. We exclude VOLE+Interpolation variant of [RS21] as it has impractical computation complexity of $\mathcal{O}(n^3)$ despite it has the lowest communication for $n = 2^{20}, 2^{24}$. We only consider the case where both parties' input sets have size n, computational security parameter κ and statistical security parameter λ. Since some protocols have additional parameters, we also consider the case where all parameters are fixed for $n = 2^8, 2^{12}, 2^{16}, 2^{20}, 2^{24}$, $\kappa = 128$ and $\lambda = 40$. The result is shown in Table 1. While our protocol originally supports elements of size $\lambda + 2\log n$ which would be $72 - 88$ bits for the parameters used in the table, while other OPRF-based PSI protocols allow elements of size κ which would be 128 bits, we may use a random oracle mapping larger elements to the smaller set as in [RT21].

Our protocol has better online communication than other PSI protocols when the set size of both parties are the same $n = 2^8, 2^{12}, 2^{16}, 2^{20}$ in both semi-honest and malicious settings. In semi-honest setting, the communication of our protocol is up to 30% lower than the most communication efficient PSI protocol. When $n = 2^{24}$, the online communication of Tiny-PSI [RT21] is slightly smaller. Even when the total communication is concerned, our protocol still remains competitive, especially for smaller set size.

In malicious setting, our protocol has the best online communication among the PSI protocols considered in the table, between 20% to 60%. In fact, it should remain competitive up to very large set of size $n = 2^{30}$. In this setting, the total communication is also better for smaller set of size $n = 2^8, 2^{12}, 2^{16}$. For larger set sizes, VOLE+PaXoS [RS21] takes over.

Our protocol is statistically secure given the rrOLE functionality in the offline phase. Thus, the communication is exactly the same as in the semi-honest case. The VOLE PSI also requires minimal change in the field size of OPRF output from $\lambda + 2\log n$ to κ. Thus, the communication increases by the size difference per element. When compared to the other OLE-based PSI [GN19], our protocol reduces the total communication by half in semi-honest setting and 75% in malicious setting. When only the online communication is considered, the reduction doubles in both cases.

The theoretical computation complexity is more difficult to compare as each protocol relies on different kinds of primitive computation steps. Our PSI protocol and OLE-based PSI [GN19] mainly uses field operations. Thus, the main computation measurement is the number of field multiplications. On the other hand, the OPRF-based PSI protocol such as solving linear systems via garbled Cuckoo hash graph [PRTY20, RS21, GPR$^+$21] and Elliptic curve group operations [RT21].

PSI payload. We instantiate our PSI payload protocol in Figure 14 with rOLE in Figure 11 and rrOLE from [BCG$^+$20]. While few PSI papers consider this variant, we can turn linear-solver-based PSI into PSI payload using a similar technique as our construction. In particular, a payload can be concatenated to the value part of a key-value pair where the key relates to the element associated

Table 2: Running time of PSI protocols for computational security parameter $\kappa = 128$, statistical security parameter $\lambda = 40$, 128-bit elements, and for set size $d = 2^8, 2^{12}, 2^{16}, 2^{20}$ in LAN and WAN (10 Mbps and 1 Mbps) network. The lowest runtime/communication for each setting is denoted in bold font. Our protocol uses amortized rrOLE setup with degree parameter 2^{20}. See Section 3.3 for more information about amortization. The method of adding payload is not discussed in [PRTY19, PRTY20]. See Section 7.3 for an extension of these protocols to PSI payloads, albeit at a higher cost that has not been benchmarked.

protocols	runtime (s)									communication (MB)		
	LAN			10 Mbps			1 Mbps					
	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
semi-honest												
KKRT [KKRT16]	0.135	**0.31**	**5.22**	2.79	9.07	135.1	5.41	74.44	1155	0.43	6.91	114.3
SpOT-low [PRTY19]	0.463	8.74	296.1	2.37	16.86	374.3	2.94	36.63	713.6	0.25	3.9	63.2
SpOT-fast [PRTY19]	**0.116**	1.51	28.55	4.31	17.64	**42.50**	4.06	38.42	493.4	0.3	4.61	76.5
PaXoS [PRTY20]	0.146	0.345	5.41	1.903	9.97	128	5.57	101.2	1719	0.59	9.9	169.7
CM [CM20]	0.145	0.482	9.28	2.83	7.15	85.45	3.61	46.43	754.8	0.36	5.34	86.9
MiniPSI [RT21]	0.722	11.45	305.9	1.12	13.29	342.8	1.82	26.35	615.1	**0.16**	**2.69**	**44**
ours - online	0.1	1.6	45.05	0.198	3.65	76.11	1.11	19.66	359.2	0.125	2.25	30
ours - total*	0.203	2.93	72.45	**0.332**	**5.45**	110.4	**1.48**	**25.77**	**468.5**	0.191	3.415	50.47
malicious												
RR [RR17b]	0.267	3.045	-	8.93	101.1	-	91.1	1525	-	9.08	154.17	-
PaXoS [PRTY20]	0.209	**0.359**	**6.07**	2.2	10.04	133.3	5.74	104	1882	0.93	14.58	236
MiniPSI [RT21]	0.729	11.24	313.4	1.14	15.68	363	1.87	31.44	659.6	0.26	4.19	67.1
ours - online	0.1	1.6	45.05	0.198	3.65	76.11	1.11	19.66	359.2	0.125	2.25	30
ours - total*	**0.204**	2.95	72.48	**0.333**	**5.45**	**110.5**	**1.49**	**25.84**	**469.3**	**0.192**	**3.421**	**50.57**

to the payload. Thus, the PSI protocols in [PRTY20, RS21, GPR$^+$21] can be transformed into PSI payload. However, some parameters in these constructions are experimental.

PSI+Sum. We instantiate our PSI+Sum protocol in Figure 16 with PSI payload protocol in Figure 14, rOLE in Figure 11 and rrOLE from [BCG$^+$20]. We also instantiate the PSI+Sum protocol with PSI payload protocol transformed from the PSI protocols in [RS21].

8.2 Experimental Evaluation

In this section, we implement our PSI protocols using NTL library to measure concrete communication and computation complexity. We compare them against some previous results. We benchmarked our protocol on Intel Core i5 2.3 GHz, 4GB RAM, 4 physical cores (all implementations are single-threaded).

PSI. We implement our PSI protocol described in Section 4, and compare our communication and computation complexity against other PSI protocols.Table 2 gives a comparison between our protocol against other protocols in the semi-honest setting [KKRT16, PRTY19, PRTY20, CM20, RT21] and in the malicious setting [RR17b, PRTY20, RT21]. We do not include PSI protocols from [RS21, GPR$^+$21] as we do not have codes to run their protocols. We consider computational security parameter $\kappa = 128$, statistical security parameter $\lambda = 40$ and 128-bit elements for set size $d = 2^{12}, 2^{16}, 2^{20}$. Thus, our protocol uses a random oracle to map each element into the field of appropriate size as described in each protocol's description. Our protocol uses amortized correlation setup over $n = 2^{20}$ for $n = 2^{12}, 2^{16}$.

The communication complexity of our protocol is better than all but MiniPSI [RT21] in semi-honest setting. This leads to better running time in low-bandwidth settings. On the other hand, our protocol is much faster than MiniPSI, up to 5x. Thus, our protocol still has better running time even in low-bandwidth settings. In malicious setting, our protocol communicates fewer bits

Table 3: Communication cost and running time of our PSI+Sum protocol and PSI-Sum protocols for 128-bit elements with 32-bit payload and statistical security parameter $\lambda = 40$. Each party's input set has $n = 2^{12}, 2^{16}, 2^{20}$ elements. The lowest runtime/communication for each setting is denoted in bold font. Our protocol uses amortized rrOLE setup over 2^{20}. See Section 3.3 for more information about amortization.

protocols	communication (MB)			running time (s)			malicious
	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	
PSI-Sum							
Bloom Filter+RLWE [IKN+20]	37	629	-	76.13	1318	-	none
Random-OT+RLWE [IKN+20]	5	84	1380	40.05	644.1	10601	none
DDH+RLWE [IKN+20]	0.38	5.5	87	1.95	30.98	499.47	none
DDH+Paillier [IKN+20]	0.33	5.1	84	3.165	48.93	776.46	none
DOPRF [MPR+20]	1.26	17.3	269.21	1150	17865	284075	both
Circuit [PSTY19]	9	149	2540	1.20	8.49	120.7	both
Circuit [HMS21]	-	36.1	585	-	1.85	24.7	both
VOLE+PaXoS+Circuit (IKNP) [RS21]	13.4	171	2830	0.495	**1.52**	**23.3**	both
VOLE+PaXoS+Circuit (SilentOT) [RS21]	4.79	21.1	277	0.737	4.05	103	both
PSI+Sum							
ours - online	0.203	3.25	56.73	0.25	5.01	118.55	sender
ours - total*	**0.291**	**4.664**	**81.2**	**0.339**	6.41	144.72	

than all previous known PSI protocols. While PaXoS [PRTY20] is faster than our protocol in LAN setting, our protocol performs better in low-bandwidth settings. Comparing to the experimental results in [RS21, GPR+21], our protocol has better communication complexity than both, and should perform better in low-bandwidth settings as well.

The PSI protocols in [PRTY19, CM20, PRTY20], the running time are mostly in the online phase as most computation requires inputs. Our protocol, similar to MiniPSI [RT21], can perform some computation in the offline phase. Our protocol, in particular, communicates about 40% in the offline phase. Thus, the performance in the online phase compares even better to other PSI protocols when restrict to the online phase. We note that the experimental results are performed on a single-core machine. We expect a similar comparison on a multiple-core machine as well.

PSI-Sum and PSI+Sum. We implement our PSI+Sum protocol in Section 6 based on our PSI Payload protocol in Section 5. Unlike its standard variant, we do not have codes of other PSI Payload and PSI-Sum protocols. Thus, the comparison here is estimated based on the published results.

Table 3 shows the comparison of the communication and computation complexity between our PSI+Sum protocol and other PSI-Sum protocols, both semi-honest [IKN+20] and malicious [MPR+20] security, some of which are circuit-based [PSTY19, HMS21, RS21]. The two previous approaches give lower communication (DDH-based in [IKN+20]) versus faster running time (circuit-based). Our protocol communicates fewer bits than all PSI-Sum protocols. In particular, the communication of the circuit-based PSI-Sum is 3.5x-35x compared to our PSI+Sum. On the other hand, our protocol is much faster compared to non circuit-based protocols while still quite competitive compared to circuit-based ones. Our protocol would outperform other PSI-Sum in WAN setting. The transformation can be applied to other PSI such as VOLE-PSI [RS21] or OKVS-PSI [GPR+21] to achieve faster running time at the cost of additional communication. See Section 7.3 for more details.

References

[AMZ21] Aydin Abadi, Steven J Murdoch, and Thomas Zacharias. Polynomial representation is tricky: Maliciously secure private set intersection revisited. In *European Symposium on Research in Computer Security*, volume 12973 of *Lecture Notes in Computer Science*, pages 721–742. Springer, 2021.

[BCG+19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *Advances in Cryptology - CRYPTO 2019*, volume 11694 of *Lecture Notes in Computer Science*, pages 489–518. Springer, 2019.

[BCG+20] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators from Ring-LPN. In *Advances in Cryptology – CRYPTO 2020*, volume 12171 of *Lecture Notes in Computer Science*, pages 387–416. Springer, 2020.

[BEP+20] Carsten Baum, Daniel Escudero, Alberto Pedrouzo-Ulloa, Peter Scholl, and Juan Ramón Troncoso-Pastoriza. Efficient protocols for oblivious linear function evaluation from Ring-LWE. In Clemente Galdi and Vladimir Kolesnikov, editors, *International Conference on Security and Cryptography for Networks. SCN 2020*, volume 12238 of *Lecture Notes in Computer Science*, pages 130–149. Springer, 2020.

[BLS03] Alin Bostan, Grégoire Lecerf, and Éric Schost. Tellegen's principle into practice. In *Proceedings of the 2003 international symposium on Symbolic and algebraic computation, ISSAC 2003*, pages 37–44. ACM, 2003.

[BM74] Allan Borodin and Robert Moenck. Fast modular transforms. *Journal of Computer and System Sciences*, 8(3):366–386, 1974.

[BMRR21] Saikrishna Badrinarayanan, Peihan Miao, Srinivasan Raghuraman, and Peter Rindal. Multi-party threshold private set intersection with sublinear communication. In *IACR International Conference on Public-Key Cryptography*, volume 12711 of *Lecture Notes in Computer Science*, pages 349–379. Springer, 2021.

[CM20] Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious prf. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, pages 34–63, Cham, 2020. Springer International Publishing.

[CO18] Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In *International Conference on Security and Cryptography for Networks*, volume 11035 of *Lecture Notes in Computer Science*, pages 464–482. Springer, 2018.

[DCKT10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *International Conference on the Theory and Application of Cryptology and Information Security*, volume 6477 of *Lecture Notes in Computer Science*, pages 213–231. Springer, 2010.

[DPT20] Thai Duong, Duong Hieu Phan, and Ni Trieu. Catalic: Delegated psi cardinality with applications to contact tracing. In *International Conference on the Theory and*

Application of Cryptology and Information Security, volume 12493 of *Lecture Notes in Computer Science*, pages 870–899. Springer, 2020.

[EGL85] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, 1985.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography Conference, TCC 2005*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer, 2005.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 1–19. Springer, 2004.

[Gil99] Niv Gilboa. Two party RSA key generation. In *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 116–129. Springer, 1999.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[GN19] Satrajit Ghosh and Tobias Nilges. An algebraic approach to maliciously secure private set intersection. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 154–185. Springer, 2019.

[GNN17] Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. Maliciously secure oblivious linear function evaluation with constant overhead. In *International Conference on the Theory and Application of Cryptology and Information Security*, volume 10624 of *Lecture Notes in Computer Science*, pages 629–659. Springer, 2017.

[GPR+21] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In *Annual International Cryptology Conference*, volume 12826 of *Lecture Notes in Computer Science*, pages 395–425. Springer, 2021.

[GS19] Satrajit Ghosh and Mark Simkin. The communication complexity of threshold private set intersection. In *Annual International Cryptology Conference*, volume 11693 of *Lecture Notes in Computer Science*, pages 3–29. Springer, 2019.

[GSB+17] Adrià Gascón, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. Privacy-preserving distributed linear regression on high-dimensional data. *Proceedings on Privacy Enhancing Technologies*, 2017(4):345–364, 2017.

[HEK12] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *19th Annual Network and Distributed System*

Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012, 2012.

[HIMV19] Carmit Hazay, Yuval Ishai, Antonio Marcedone, and Muthuramakrishnan Venkitasubramaniam. Leviosa: Lightweight secure arithmetic computation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, United Kingdom, November 11 - 15, 2019*, pages 327–344. ACM, 2019.

[HMRT22] Iftach Haitner, Nikolaos Makriyannis, Samuel Ranellucci, and Eliad Tsfadia. Highly efficient ot-based multiplication protocols. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 13275 of *Lecture Notes in Computer Science*, pages 180–209. Springer, 2022.

[HMS21] Kyoohyung Han, Dukjae Moon, and Yongha Son. Improved circuit-based psi via equality preserving compression. Cryptology ePrint Archive, Paper 2021/1440, 2021. `https://eprint.iacr.org/2021/1440`.

[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. Cryptology ePrint Archive, Paper 2017/738, 2017. `https://eprint.iacr.org/2017/738`.

[IKN+20] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. On deploying secure computing: Private intersection-sum-with-cardinality. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 370–389. IEEE, 2020.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161. Springer, 2003.

[IPS09] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *Theory of Cryptography Conference*, volume 5444 of *Lecture Notes in Computer Science*, pages 294–314. Springer, 2009.

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 1651–1669, Baltimore, MD, August 2018. USENIX Association.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious prf with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 818–829. ACM, 2016.

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

[KOS16] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 830–842. ACM, 2016.

[KS05] Lea Kissner and Dawn Song. Privacy-preserving set operations. In *Annual International Cryptology Conference*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257. Springer, 2005.

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

[MPR+20] Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. Two-sided malicious security for private intersection-sum with cardinality. In *Annual International Cryptology Conference*, volume 12172 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2020.

[MRR20] Payman Mohassel, Peter Rindal, and Mike Rosulek. Fast database joins and psi for secret shared data. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1271–1287, New York, NY, USA, 2020. Association for Computing Machinery.

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 245–254. ACM, 1999.

[PRTY19] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse ot extension. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 401–431, Cham, 2019. Springer International Publishing.

[PRTY20] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Psi from paxos: Fast, malicious private set intersection. In *Advances in Cryptology – EUROCRYPT 2020*, volume 12106 of *Lecture Notes in Computer Science*, pages 739–767. Springer, 2020.

[PSTY19] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 11478 of *Lecture Notes in Computer Science*, pages 122–153. Springer, 2019.

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

[RR17a] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 10210 of *Lecture Notes in Computer Science*, pages 235–259. Springer, 2017.

[RR17b] Peter Rindal and Mike Rosulek. Malicious-secure private set intersection via dual execution. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1229–1242. ACM, 2017.

[RS21] Peter Rindal and Phillipp Schoppmann. Vole-psi: Fast oprf and circuit-psi from vector-ole. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, volume 12697 of *Lecture Notes in Computer Science*, pages 901–930. Springer, 2021.

[RT21] Mike Rosulek and Ni Trieu. Compact and malicious private set intersection for small sets. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1166–1181. ACM, 2021.

[SCK12] Jae Hong Seo, Jung Hee Cheon, and Jonathan Katz. Constant-round multi-party private set union using reversed laurent series. In *International Workshop on Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 2012.

[Sha80] Adi Shamir. On the power of commutativity in cryptography. In *International Colloquium on Automata, Languages, and Programming*, volume 85 of *Lecture Notes in Computer Science*, pages 582–595. Springer, 1980.