S. Dov Gordon, Carmit Hazay, and Phi Hung Le*

Fully Secure PSI via MPC-in-the-Head

Abstract: We design several new protocols for private set intersection (PSI) with active security: one for the two party setting, and two protocols for the multi-party setting. In recent years, the state-of-the-art protocols for two party PSI have all been built from OT-extension. This has led to extremely efficient protocols that provide correct output to one party; seemingly inherent to the approach, however, is that there is no efficient way to relay the result to the other party with a provable correctness guarantee. Furthermore, there is no natural way to extend this line of works to more parties. We consider a new instantiation of an older approach. Using the MPC-in-the-head paradigm of Ishai et al. [IPS08], we construct a polynomial with roots that encode the intersection, without revealing the inputs. Our reliance on this paradigm allows us to base our protocol on passively secure Oblivious Linear Evaluation (OLE) (requiring 4 such amortized calls per input element). Unlike state-of-the-art prior work, our protocols provide correct output to all parties. We have implemented our protocols, providing the first benchmarks for PSI that provides correct output to all parties. Additionally, we present a variant of our multi-party protocol that provides output only to a central server.

Keywords: Private Set Intersection; Secure Computation; MPC-in-the-Head

1 Introduction

Secure multi-party computation (MPC) allows two or more parties to perform some agreed upon computation on their private input, while revealing nothing beyond the value of the output. General solutions to the problem were first developed in the 1980s [Yao86, Gol09], and allow for the computation of arbitrary functions over the input: $m$ participants agree on $m$ functions, $f_1, \ldots, f_m$, and each provides an input to the computation. At the end of their interaction, party $i$ learns $f_i(X_1, \ldots, X_m)$, where $X_j$ is the input provided by party $j$. In the last fifteen years, the research has shifted towards the study of concrete efficiency [WRK17, EKR18, BCS19]. While the general solutions, which support arbitrary computations, have become quite efficient, for certain particular computations, tailored protocols can greatly outperform the generic approach, both asymptotically and concretely e.g., [HT10, CHI+21]. Private set intersection (PSI) is an example of such a concrete and well-studied function.

There are many variants of the PSI problem, but, broadly, two or more parties each hold a private set of values, and they all learn the intersection of those sets: $\forall j \in \{1, \ldots, m\}: f_j(X_1, \ldots, X_m) = \bigcap_i X_i$. In reality, it is surprisingly challenging to provide a correct output to all parties. In fact, all prior works on PSI, in both the two-party and multi-party settings, provide an output to only one party: $f_i = \bigcap_i X_i$, and, for $j > i$, $f_j = \perp$. While there are generic ways of “compiling” such protocols to provide output to all parties, e.g., using zero-knowledge,\(^1\) this ruins the efficiency of known constructions. In this work, we consider both variants of PSI, referring to protocols that realize the former as “fully secure”, and those realizing the latter as having “one-sided output”. Ours is the first construction to offer full security in this sense,\(^2\) providing features that are important in many PSI use cases. (We provide some examples in Section 1.2.)

As with general-purpose secure computation, the solution space depends greatly on whether the adversary is assumed to be semi-honest (aka passive), in which case corrupted parties always follow the protocol, or ma-

---

\(^1\) In a semi-honest setting, the party receiving output can forward it to the other participants, and, by assumption, it will do so correctly. In the malicious setting, those parties need a method of verifying the correctness of this output. This issue is independent of the issue of fairness which is discussed below.

\(^2\) We note that the PSI computation, like any other, can be compiled into a Boolean circuit, and evaluated using any appropriate generic protocol for secure computation. This would also yield full security, but the asymptotic complexity, and concrete cost of this approach is worse than custom PSI protocols.
licious (aka active), in which case the corrupted parties might deviate arbitrarily from the protocol specification. In this work, we only consider active adversaries. Finally, another important feature in MPC regards fairness: when multiple parties are meant to receive output, a protocol is considered fair if no party can learn their own output while preventing others from doing the same. (The question of fairness is irrelevant when the computation is meant to be one-sided.) When half or more parties might be malicious, fairness is impossible to achieve [Cle86], and the standard security definition [Go19], called security with abort, ensures only that if a party receives output, it is guaranteed to be correct. Throughout our paper we consider the setting where $m - 1$ parties might be malicious, including the two-party setting; we therefore only consider security with abort.

We construct three new PSI protocols for the malicious model, one in the two-party setting, and two in the multi-party setting. Specifically, we construct the first actively secure two-party protocol that provides output to both parties, or “fully secure”. To our knowledge, the only other PSI protocol that claims to be fully secure is the two-party protocol by Ghosh and Nilges [GN19]. However, a recent analysis [AM21] has shown that [GN19] is insecure and is susceptible to various attacks. Mitigating these attacks seems to imply a much higher complexity than reported.

We then extend our result to the multi-party setting, providing the first fully secure multi-party PSI construction (MPSI). Finally, we show how to relax security of this protocol, providing an output to a single designated party, and improving efficiency. We provide a comparison with several recent MPSI protocols that provide this weaker guarantee in Sections 5-6.

We have implemented our protocols, providing the first benchmarks for fully secure PSI in both the two-party and multi-party settings. Our constructions rely heavily on a cryptographic primitive called Oblivious Linear Evaluation (OLE). In our protocols, the PSI problem is reduced to oblivious polynomial multiplication via OLE where correctness is ensured via the use of a watchlist mechanism that prevents the adversary from using any malformed input or deviating from the protocol. Another important feature of our protocols is in relying on passively secure OLE (while achieving active security), requiring only 4 passive OLE executions per input. When compared with using generic, malicious secure, 2PC (e.g. Overdrive [KPR18]) to compute a PSI circuit (e.g. [HEK12]), our protocol is nearly 1000× faster (assuming trusted key setup for Overdrive).

Since our PSI protocols are the first fully secure constructions, there is no readily available prior work to compare with. When providing an output to only one party, concurrent with our work, Ben-Efraim et al. [BNO21] have provided the only other experimental evaluation of MPSI with active security. We compare our experimental results with theirs as best as we can in Section 6. Beside [BNO21], and also concurrent with our work, Garimella et al. [GPR+21] provide another construction with this weaker output guarantee (but without any experimental evaluation). As demonstrated, our protocols are highly competitive despite the stronger level of security.

### 1.1 A Brief History of PSI

Before describing our precise contributions, we provide a brief, and non-exhaustive history of this highly studied problem. We categorize prior work according to the technical approach.

**PSI From Polynomial Evaluation.** One of the earliest PSI protocols, by Freedman et al. [FNP04], provided an elegant semi-honest solution using additively homomorphic encryption. Party $P_1$ encodes its input as the roots of a polynomial, $P$. It then encrypts the coefficients of this polynomial and sends the ciphertexts to party $P_2$, who evaluates the same polynomial, homomorphically, on each of its own inputs. $P_2$ then randomizes the result of each evaluation as follows, and sends the randomized encodings to party $P_1$ to determine the output: for input $y$, party $P_2$ computes $\text{Enc}(r \cdot P(y) + y)$. If $y$ is a root of $P$, this encodes $y$, while in all other cases, it encodes a random value. Preventing malicious behaviour requires using cut-and-choose and the random oracle, where only the first party learns the PSI result. Specifically, Freedman et al. in [FNP04] uses the random oracle to derandomize the computations of $P_2$, which can be recomputed by $P_1$ for the elements that intersect. Over the next years, several results strengthened the security guarantees and the performance of this approach [KS05, DMRY09, HN10, Haz15, HV17].

Recently, Ghosh and Nilges [GN19] provided a malicious fully secure construction. They also extended their construction to the one-sided multi-party setting. Unfortunately, their constructions are flawed [AM21]. To the best of our knowledge, that was the only attempt to design a PSI construction with this property (excluding generic protocols for secure computation).
PSI From Oblivious PRFs. A separate line of works explores a different approach, using oblivious pseudo-random functions (OPRFs) [FIPR05, HL08, JL09]. In this approach, party $P_1$ samples a random PRF key $k$ for PRF $F$ and computes $F_k(x)$ for every input $x$ in its set, and finally sends the encoded values to the second party. Party $P_2$ then obliviously evaluates the same PRF $F$, without knowing $k$, on each of its own inputs and computes the intersection on the encoded values. In the construction from [HL08], the OPRF is constructed from the number theoretic PRF of Naor and Reingold [NR97], though later variations would improve upon this approach (e.g., [JL09]). None of these constructions is fully secure in the malicious setting.

Generic Solutions. In 2012, Huang et al. [HEK12] demonstrated that generic protocols for secure computation, based on Yao’s garbled circuits, had improved to the point that they were now faster than custom PSI protocols. The computational complexity of garbled circuits is dominated by oblivious transfer (OT). OT extension, introduced by Ishai et al. [IKNP03], allows us to reduce $O(w)$ OTs to $O(\kappa)$ public key operations and $O(w)$ symmetric key operations, where $w$ is the input size and $\kappa$ the security parameter. Huang et al. presented a circuit of size $O(w \log w)$ for performing PSI on two sets of size $w$, which determines the communication complexity. While several existing custom protocols already offered linear communication complexity, they required $O(w)$ public key operations which overcome the cost of sending $O(w \log w)$ data in “reasonable” networks. More recently, Pinkas et al. [PSTY19] showed how to reduce the circuit size to $O(w)$ using cuckoo hashing, but only in the semi-honest setting.

OT-Based PSI. Since 2013, a long line of works that is based on OT extension, have outperformed the generic solutions, providing the best running times3 [DCW13, PSZ14, PSSZ15, KKRT16, RR17, PRTY20]. The most recent constructions of this type are similar to the earlier protocols that embed oblivious PRFs. These works rely on OT extension to construct randomized, correlated encodings of the input values [KKRT16], similarly to oblivious PRFs. The earlier results in this line of works only offered semi-honest security, but with $O(w)$ communication complexity and very few public key operations. Rindal and Rosulek provided the first malicious secure construction from OT extension, requiring $O(w \log w)$ communication [RR17]. Pinkas et al. [PRTY20] introduced the first maliciously secure PSI protocol from OT extension with linear communication complexity. Rindal and Schoppmann [RS21] improved on Pinkas et al. [PRTY20] in concrete terms, though they again required $O(w \log w)$ communication. Building on [RR17], a recent work by Ben-Efraim et al. [BNOP21] designed and implemented a multi-party PSI protocol with malicious security and communication complexity dominated by $O(m \kappa n^2)$ where $m$ is the number of parties and $\kappa$ is the security parameter. Their construction provides an output to one party.

1.2 Our Contributions

Applying MPC-in-the-Head to PSI. We depart from this successful line of works building PSI from OT extension, and return instead to methods based on oblivious polynomial evaluation. We present three new, maliciously secure PSI protocols, one for the two-party case, and two different extensions to the multi-party setting. In a very broad sense, our approach is similar to the old result by Kissner and Song [KS05], in that we arrive at the output by computing a polynomial $T(x) = Q(x) \cdot R(x) + P(x) \cdot S(x)$, where the roots of $Q(x)$ encode the inputs of one party, the roots of $P(x)$ encode the inputs of the other, and the polynomials $S(x)$ and $R(x)$, which are not known to either party, serve to hide the elements that are not in the intersection.

However, while Kissner and Song homomorphically encrypt the coefficients of these polynomials, our approach for computing this polynomial is more similar to the recent protocols proposed by Ghosh and Nilges [GN19]. Like them, we reduce the problem of computing this polynomial $T$ to the problem of OLE. However, we manage to do this while guaranteeing output to all parties. Additionally, we rely on the MPC-in-the-head paradigm of Ishai et al. (IPS) [IPS08] to ensure security, which provides several benefits: (1) This allows us to rely only on semi-honest OLE and, (2) it can support an arbitrary number of parties.

Ishai et al. presented a general compiler for constructing maliciously secure MPC in the dishonest setting out of simpler primitives. At a high level, this is done by combining two protocols: an “outer protocol” for computing the desired function – in our case, the polynomial $T$, and an “inner protocol” for securely simulating the roles of the participants in the outer protocol. The outer protocol is unconditionally secure against an
adversary corrupting a minority of parties, while the inner protocol relies on some cryptographic primitive, and must be secure against a semi-honest adversary corrupting m − 1 parties. (Often, m = 2.) The parties in the inner protocol secret share among themselves the state of the parties in the outer protocol, and securely simulate each of their executions. Using oblivious transfer, the members of the inner protocol obliviously establish “watch channels” through which they can monitor the behavior of a minority of simulated parties. This allows them to catch any cheating with very high probability, without violating privacy.

We describe how our protocol is derived from the IPS paradigm. For simplicity, we stick to the two party setting. As in IPS, we view the outer protocol as involving multiple servers, and two clients: the two parties with input play the role of the clients, and begin by secret sharing their input sets with the servers. This is done by sampling a random polynomial with roots at the points corresponding to the clients’ inputs, and sending a single evaluation of this polynomial to each server. We denote the polynomials encoding the input sets as P and Q. The two clients then separately sample random polynomials to serve as additive shares of the masking polynomials: R(·) = R1(·) + R2(·), and S(·) = S1(·) + S2(·). These polynomials are secret shared with the servers as well. The servers add the shares of R and S, and perform two polynomial multiplications by locally multiplying their threshold shares, doubling the degree of the polynomial. They add the results, each arriving at a secret share of: T = Q · R + P · S.

Our main observation is that this particular outer protocol lends itself very nicely to the IPS approach. In the IPS compiler, the state of each server is additively secret shared by the clients, and the outer protocol is emulated on these additive shares. This emulation can be quite expensive, depending on the particular instantiations of these protocols. However, for this particular inner protocol, arriving at the polynomial encoding of the input requires only two parallel polynomial multiplications, and a few additions. After providing the additive secret sharing (of the polynomial shares), the clients use the GMW protocol in the OLE hybrid model to emulate the product of the additively shared points on the polynomial. This requires just two OLE calls for each server.

This captures the high-level idea of our construction, but omits several important details. The two clients perform a degree check of all polynomials (simultaneously) in order to defend against any cheating in the server emulation. Furthermore, we have neglected to discuss the implementation of the watch channels, which is a crucial component of the IPS paradigm, and allows us to benefit from the efficiency of semi-honest OLE constructions. All of these details can be found in the formal protocol description.

Although our protocol relies heavily on the ideas behind the IPS compiler, we do not in fact rely on their theorem [IPS08], and instead provide a direct proof of security for our protocol. The IPS protocol is highly general, while we are focusing on a very specific problem. Once the general abstraction has been removed, the resulting PSI protocol is in fact easier to understand without the added complexity of separating an outer and inner protocol. We presented the IPS framework in this introduction only to explain how we arrived at our result, and to provide intuition for why our use of semi-honest secure OLE suffices for our claim of malicious secure PSI.

**Fully Secure PSI.** In many applications, it is highly important that all parties learn the output. Consider, for example, two competing companies that would both benefit from identifying their overlapping customers. They intend to perform this computation on a monthly basis. If one company aborts the computation unfairly, the collaboration can be terminated, and little harm has been done. However, if one party consistently learns the correct intersection and reports only 25% of the resulting set to its competitor, it then receives an unfair advantage, indefinitely! Other PSI applications would ben-

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Communication complexity</th>
<th>Fully secure</th>
</tr>
</thead>
<tbody>
<tr>
<td>HV17</td>
<td>O(mwκ log w + m²κ)</td>
<td>NO</td>
</tr>
<tr>
<td>GPR + 21</td>
<td>O(mw(κ + λ + log w) + m²κ)</td>
<td>NO</td>
</tr>
<tr>
<td>Ours (P₂ receives output)</td>
<td>O(mwκ + m²κ + mλκ log w)</td>
<td>NO</td>
</tr>
<tr>
<td>Ours (All receive output)</td>
<td>O(mwκ + m²κ + mλκ log w)</td>
<td>YES</td>
</tr>
</tbody>
</table>
eit from fully secure protocols for similar reasons, such as distrustful governments comparing satellite positions [HLOI16], or searching for software vulnerabilities.

Constructions from OT extension proceed by providing a list of random encodings to one party, who then computes the intersection on these encodings, locally. There is no simple way to certify this list, so it is trivial for the adversary to lie about what was recovered. A naive approach to certifying the output would be to employ a zero-knowledge proof, where the witness is the input to a random oracle. Instantiating the random oracle with an MPC friendly hash function, e.g., [AAB+19], implies a circuit for this proof statement that contains around $2^9$ AND gates. Estimating the parameters according to Limbo [dSGOT21], a recently designed proof system that also relies on the MPC-in-the-head paradigm, implies a circuit with 20KB proof size and around 0.01s running time for the prover. Other proof systems will achieve different tradeoffs between the proof size and the prover’s running time. This approach is not scalable as the input size grows: on input sets containing 1 million items, this would require close to 3 hours of computing time. For this reason, all custom constructions in the literature only provide output to one party.

Multi-Party PSI (MPSI). Another drawback of the recent constructions based on OT-extension is that they do not readily extend to the multi-party setting. In contrast, we extend our result to the multi-party setting and provide two protocol variants. The main protocol that we present in detail ensures that all parties receive correct output. As we mentioned previously, ours is the first construction of this kind. It has communication complexity of $O((m^2 \log w + mw)\kappa)$, where $m$ denotes the number of parties, $w$ denotes the input set size, and $\kappa$ is a security parameter, and it is based on the MPC-in-the-head paradigm. Applying the IPS compiler directly will result in a very inefficient protocol, which has the communication complexity of at least $O((m^3 + m^2w)\kappa)$. That is just the cost to set up the watchlists [LOP11].

In order to achieve better performance, we use a customised version of the IPS compiler for our multi-party PSI protocol. We redesign the watchlist mechanism for the MPSI protocol, basing it on the commit-and-reveal paradigm. In the context of our MPSI protocol, the new watchlist mechanism reduces the number of watch channels from $O(mt)$ to $O(t)$ where $t$ is the number of channels that each party watches. In Section 4 we discuss in detail how we implement our new watchlist mechanism.

Our MPSI protocol requires $O((m^2 \log w + mw)\kappa)$ bits of communication for setting up the watchlists channels, and then only $O(mw\kappa)$ bits to compute the polynomial encoding the intersection. We also present a variant that provides output to a single designated party. In this construction the communication reduces to $O((m^2 + mw)\kappa)$. Our MPSI constructions need just $4m$ passive OLE calls per input item. Recently, Ben-Efraim et al. [BNOP21] gave a new construction for (one side) multi-party PSI and provided experimental results (their code is not currently available). Their construction is based on Garbled Bloom Filters and requires communicating $O(mw^2 + mw\log(kw))$ bits. Because of the $\kappa^2$ overhead, they quickly run into memory constraints and report only on input sizes up to $2^{18}$, which is relatively small in this line of work.

Performance. Our two-party protocol only requires 4 passively secure OLE instances for each element in the set (amortized). For our multi-party PSI, the bottleneck of the protocol is with respect to the central party that is required to perform $4m$ passive OLE per input item. To test the performance of our protocols, we implemented a prototype that performs an end-to-end PSI functionality (see Section 6).

Black-Box Use of OLE. Since our reliance on OLE is black-box, we can instantiate this functionality with any OLE construction, and can benefit from future improvements, such as new developments in OLE extension and parallelisation. Our implementation currently instantiates the OLE instances with either OT [Gill99] or with the packed, additively homomorphic encryption scheme [BGV12], based on Ring LWE. The latter allows us to pack $2^{12}$ values into a single instance, which greatly contributes to the concrete efficiency.

1.3 Related Work

Ghosh and Nilges. These authors made the observation that the computation of $T = Q(R_1 + R_2) + P(S_1 + S_2)$ can be reduced to computing $t_j = \eta_j(r_{1,j} + r_{2,j}) + p_j(s_{1,j} + s_{2,j})$ where $w$ is the input size, $j \in \{1, \ldots, 2w + 1\}$, and $t_j, \eta_j, p_j, r_{1,j}, r_{2,j}, s_{1,j}, s_{2,j}$ are the evaluations of the above polynomials on public points $\eta_j$ [GN19]. Ghosh and Nilges attempted to make their protocol secure against malicious adversaries by using actively secure OLE to compute $t_j$, where the output polynomial $T(\cdot)$ is verified by checking if $T(x) \equiv Q(x)(R_1(x) + R_2(x)) + P(x)(S_1(x) + S_2(x))$ over two random points $x_1$ and $x_2$, each chosen by a party (in
the two-party setting). Unfortunately, these checks do not catch all the malicious attacks \[\text{[AMZ21]}\]. Specifically, their verification step does not ensure that the evaluations above are well-formed or used consistently throughout the protocol. An adversary can modify these shares during the computation, or use the shares, say \(r_{1,j}\), that are not consistent with a polynomial \(R_1(x) = \sum_{i=0}^{w} a_i x^i\) but some \(R_1(x) = \sum_{i=0}^{w} b_i x^i\). These attacks have shown to violate both privacy and correctness. As opposed to \[\text{[GN19]}\], our protocol guarantees that the shares are consistent and well-formed throughout the computation via the use of MPC-in-the-head.

**Leviosa.** Our construction leverages a lot of the techniques of Hazay et al. \[\text{[HIMV19]}\]. They provide a concrete instantiation of the IPS framework, resulting in a generic two-party secure computation protocol with security against a malicious adversary.

We note that Leviosa makes no claims about the multi-party setting, though, IPS does, and one can consider how Leviosa would generalize: it would not scale well. As we mentioned earlier, the cost of setting up the watchlist channels alone would be \(O((m^3 + m^2 w)\kappa)\).

When focusing on PSI, we are able to extend this approach much more efficiently, primarily because the protocol is amenable to a star topology in the communication network. Specifically, because we can arrive at the output encoding, \(T\), through a series of pairwise computations with an (arbitrarily) designated central party, the “inner protocol” in the IPS framework still only requires pairwise additive secret-sharings of the state of the outer protocol. This allows us to perform pairwise OLEs on additively shared values (as in our two-party protocol), rather than some multi-party execution of the OLE. Furthermore, when verifying correctness of the OLE executions, each party only needs to verify the correctness of the central party; beyond that, they can rely on the central party to perform the verification of the other peers. For this reason, only the central party must send \(m\) decommitments, while the other parties each sends only one. We note that without replacing the OT-based watchlist channels with commit-and-reveal, we could not have benefited from this latter advantage implied by a star topology.

In the two-party setting, Leviosa could be used “off-the-shelf” in order compile the semi-honest polynomial multiplication protocol into a malicious-secure protocol. In our two-party protocol, this would result in roughly twice the number of OLE calls, as their generic input encoding does not leverage the fact that PSI input is already naturally, correctly encoded. Our main contribution in the two-party setting is to recognize the relevance of Leviosa for two-party PSI.

### 2 Preliminaries

**Basic Notations.** We denote a security parameter by \(\kappa\). We denote by \([n]\) the set of elements \(\{1, \ldots, n\}\) for some \(n \in \mathbb{N}\). Throughout the paper, we denote by \(m\) the number of parties, \(w\) the input size. We assume functions to be represented by an arithmetic circuit \(C\) (with addition and multiplication gates of fan-in 2), and denote the size of \(C\) by \(|C|\). By default we define the size of the circuit to include the total number of gates including input gates.

#### 2.1 Secure Multi-Party Computation

We use a standard stand-alone definition of secure multi-party computation protocols. In this work, we only consider static corruptions, i.e. the adversary decides which parties it corrupts before the execution begins. We also only consider security with abort, in which the one party receives their output first, and, if malicious, may choose to abort before others recover the output. Note that in the variant of our multi-party protocol in which only one designated party receives an output, this ability to abort is irrelevant. Nevertheless, for simplicity, we use the same security definition. We use two security parameters in our definition: a computational security parameter \(\kappa\), and a statistical security parameter \(\lambda\) that captures a statistical error of up to \(2^{-\lambda}\). We assume that \(\lambda \leq \kappa\). We let \(F\) be a multi-party functionality that maps a set of \(m\) inputs to an output over some field \(F\) (w.l.o.g.).

Let \(\Pi = \langle P_1, \ldots, P_m \rangle\) denote a multi-party protocol, where each party is given an input \(x_i\) and security parameters \(1^\lambda\) and \(1^\kappa\). We allow honest parties to be PPT in the entire input length (this is needed to ensure correctness when no party is corrupted), but bound adversaries to time \(\text{poly}(\kappa)\) (this effectively means that we only require security when the input length is bounded by some polynomial in \(\kappa\)). We denote by \(\text{REAL}_{\Pi, A}(z)\) the output of the honest parties and the adversary \(A\) controlling a subset \(I \subseteq [m]\) of parties in the real execution of \(\Pi\), where \(z\) is the auxiliary input, \(x_i\) is \(P_i\)’s initial input, \(\kappa\) is the computational security parameter, and \(\lambda\) is the statistical security parameter. We denote by \(\text{IDEAL}_{F, S(z)}(x_1, \ldots, x_m, \kappa, \lambda)\) the
output of the honest parties and the simulator $S$ in the ideal model where $F$ is computed by a trusted party. We stress that in this ideal model, the adversary is given the output first, and then instructs $F$ whether to send output to the honest parties, or to abort. We refer the reader to Goldreich’s textbook for more detail [Go09]. In some of our protocols the parties have access to ideal model implementations of certain cryptographic primitives such as ideal coin tossing ($F_{\text{COIN}}$). We denote such executions by $\text{REAL}_{I,A}(z)(x_1, \ldots, x_m, \kappa, \lambda)$. Due to Canetti’s stand-alone composition theorem [Can00], it suffices to prove that this hybrid execution is indistinguishable from the ideal execution.

**Definition 1.** A protocol $I = (P_1, \ldots, P_m)$ is said to securely compute a functionality $F$ in the presence of active adversaries if the parties always have the correct output $F(x_1, \ldots, x_m)$ when neither party is corrupted, and moreover the following security requirement holds. For any probabilistic poly($t$)-time adversary $A$ controlling a subset $I \subset [m]$ of parties in the real world, there exists a probabilistic poly($\kappa$)-time adversary (simulator) $S$ controlling $I$ in the ideal model, such that for every non-uniform poly($\kappa$)-time distinguisher $D$ there exists a negligible function $\nu(\cdot)$ such that the following ensembles are distinguished by $D$ with at most $\nu(\kappa) + 2^{-\lambda}$ advantage:

\[
\text{REAL}_{I,A}(z)(x_1, \ldots, x_m, \kappa, \lambda)_{z \in \{0,1\}^*} \quad \text{and} \quad \text{IDEAL}_{I,S}(z)(x_1, \ldots, x_m, \kappa, \lambda)_{z \in \{0,1\}^*}
\]

### 2.2 Secret-Sharing

A secret-sharing scheme allows distribution of a secret among a group of $n$ players, each of whom in a sharing phase receives a share of the secret. In its simplest form, the goal of secret-sharing is to allow only subsets of players of size at least $t + 1$ to reconstruct the secret. More formally a $(t + 1)$-out-of-$n$ secret sharing scheme comes with a sharing algorithm that on input a secret $s$ outputs $n$ shares $s_1, \ldots, s_n$ and a reconstruction algorithm that takes as input $\{(i, s_i)\}_{i \in S}$ where $|S| > t$ and outputs either a secret $s'$ or $\bot$. In this work, we use polynomial encodings to share a set of secrets in $\mathbb{F} = GF(q)$. We only require that the output of the reconstruction algorithm includes every secret, and it may output a superset of the secret set. We present the sharing and reconstruction algorithms below:

**Sharing Algorithm Share:** For any input set $\{x_1, \ldots, x_w\} : x_i \in \mathbb{F} \setminus \{1, \ldots, n\}$, pick a random polynomial $p(\cdot)$ of degree $t + w$ in the polynomial ring $\mathbb{F}[x]$ with the condition that $p(x_i) = 0$. Output $p(1), \ldots, p(n)$.

**Reconstruction Algorithm Reconst:** For any input $\{(i, s'_i)\}_{i \in S}$, compute a polynomial $g(x)$ such that $g(i) = s'_i$ for every $i \in S$. This is possible using Lagrange interpolation where $g$ is given by

\[
g(x) = \sum_{i \in S} s'_i \prod_{j \in S \setminus \{i\}} \frac{x - j}{i - j}.
\]

Finally the reconstruction algorithm outputs the roots of $g$.

A secure secret sharing scheme is required to satisfy the following properties:

**Correctness:** For every secret set $\{x_1, \ldots, x_w\}$ and every set of $t + w + 1$ shares $s_{t_1}, \ldots, s_{t_{t+w+1}} \subseteq \text{Share}(\{x_1, \ldots, x_w\})$, we have

\[
\Pr[\{x_1, \ldots, x_w\} \subseteq \text{Reconst}(s_{t_1}, \ldots, s_{t_{t+w+1}})] = 1
\]

**Secrecy:** For any pair of secret sets $x, x'$, and every two sets of shares $s_{t_1}, \ldots, s_{t_{t+w}} \subseteq \text{Share}(x)$ and $s'_{t_1}, \ldots, s'_{t_{t+w}} \subseteq \text{Share}(x')$, the sets $\{s_{t_1}, \ldots, s_t\}$ and $\{s'_{t_1}, \ldots, s'_{t_t}\}$ are identically distributed.

### 2.3 Coding Notation

For a code $C \subseteq \mathbb{F}^n$ and a vector $v \in \mathbb{F}^n$, denote by $d(v, C)$ the minimal distance of $v$ from $C$, namely the number of positions in which $v$ differs from the closest codeword in $C$, and by $\Delta(v, C)$ the set of positions in which $v$ differs from such a closest codeword (in case of ties, take the lexicographically first closest codeword). We further denote by $d(V, C)$ the minimal distance between a vector set $V$ and a code $C$, namely $d(V, C) = \min_{v \in V} d(v, C)$.

**Definition 2 (Reed-Solomon code.).** For positive integers $n, k$, finite field $\mathbb{F}$, and a vector $\eta = \{\eta_1, \ldots, \eta_n\} \in \mathbb{F}^n$ of distinct field elements, the code $RS_{\mathbb{F}, n, k, \eta}$ is the $[n, k, n - k + 1]$ linear code over $\mathbb{F}$ that consists of all $n$-tuples $(p(\eta_1), \ldots, p(\eta_n))$ where $p$ is a polynomial of degree $< k$ over $\mathbb{F}$ and $d = n - k + 1$ is the minimum distance.
2.4 Commitment Schemes

We use cryptographic commitments in our coin-flipping functionality. In a commitment scheme, a sender holds some secret value \( x \in \mathbb{F} \). It sends a commitment to \( z \) to a receiver, which reveals nothing about \( x \). At a later time, the sender can send a decommitment, which proves that \( x \) was the value used at the time of commitment.

**Committing Algorithm** \( \text{Com} \): On input \( x \in \mathbb{F} \), and security parameter \( \kappa \), the committing algorithm outputs a pair of values \( (c,d) \).

**Decommitting Algorithm** \( \text{Decom} \): Given a commitment \( c \) and a decommitment value \( d \), the decommitment algorithm outputs a value \( x \). We note that the decommitment algorithm might take \( x \) as part of \( d \), and return \( \perp \) in case \( x \) and \( d \) are inconsistent.

A secure commitment scheme is required to satisfy the following properties.

**Hiding:** For every pair of inputs, \( x_1, x_2 \in \mathbb{F} \), and for every non-uniform, \( \text{poly}(\kappa) \) time distinguisher \( D \), there exists a negligible function \( \nu(\cdot) \) such that 
\[
| \Pr[D(\text{Com}(x_1)) = 1] - \Pr[D(\text{Com}(x_2)) = 1] | \leq \nu(\kappa).
\]

**Binding:** For every non-uniform, \( \text{poly}(\kappa) \) time adversary \( A \) outputting \( (c,d_1,d_2) \), there exists a negligible function \( \nu(\cdot) \) such that 
\[
\Pr[\text{Decom}(c,d_1) = \text{Decom}(c,d_2)] \leq \nu(\kappa).
\]

Additional preliminaries are found in Appendix A.

3 Fully Secure Active PSI

In this section we present our two-party actively secure protocol for computing the PSI functionality (cf. Figure 1) where both parties learn the output.

Our protocol follows the basic design of Kissner and Song [KS05], where the parties generate two polynomials \( P(\cdot) \) and \( Q(\cdot) \) that correspond to their inputs (namely, the roots of these polynomials are the input sets). Next, the parties jointly compute \( T(\cdot) = P(\cdot)S(\cdot) + Q(\cdot)R(\cdot) \), where \( S \) and \( R \) are random polynomials, and all polynomials have the same degree. They can then extract the intersection from the roots of \( T \). Specifically, Kissner and Song proved that if \( S(\cdot) \) and \( R(\cdot) \) are chosen uniformly at random, and privately, then \( T(\cdot) \) can be represented as \( T(\cdot) = I(\cdot)W(\cdot) \) where \( I(\cdot) \) is the polynomial with the roots at the intersection items, and \( W(\cdot) \) is a random polynomial. Intuitively, note that if \( P(\omega) = 0 \), then \( P(\omega)S(\omega) = 0 \). If \( Q(\omega)R(\omega) = 0 \), then \( P(\omega)S(\omega) + Q(\omega)R(\omega) = 0 \), and \( \omega \) is a root of \( T(\cdot) \). On the other hand, if \( Q(\omega) \neq 0 \), then because \( R(\omega) \) is uniform, \( T(\omega) \) is uniform, and unlikely to be 0. Because both \( S(\cdot) \) and \( R(\cdot) \) are uniform and unknown, it follows that \( T(\cdot) \) is a uniform polynomial, subject to have the intersecting roots. Furthermore, note that if \( P(\cdot) \neq 0 \) and \( Q(\cdot) \neq 0 \), revealing \( T(\cdot) \) to \( P_1 \) does not leak any information about the other party's input other than the intersection. In order to guarantee that \( R(\cdot) \) and \( S(\cdot) \) are sampled uniformly at random, each party \( P_i \) independently samples \( R_i(\cdot) \) and \( S_i(\cdot) \) uniformly at random. Following that, the parties compute 
\[
T(\cdot) = P(\cdot)(S_1(\cdot) + S_2(\cdot)) + Q(\cdot)(R_1(\cdot) + R_2(\cdot)).
\]
Then as long as one party honestly samples its polynomials shares, \( R(\cdot) \) and \( S(\cdot) \) will be uniformly random polynomials and \( T(\cdot) \) will be distributed as explained above.

We use OLE (see Figure 7 for the OLE functionality) to perform the polynomial multiplications, as follows. All polynomials have degree \( w \), and we fix a set of \( n > 2w \) public indices. Let \( p_i = P(i) \) denote the evaluation of \( P_i \)'s input polynomial at public index \( i \), and define \( q_i \) similarly. \( P_1 \) samples random polynomials \( R_1(\cdot), U_1(\cdot) \) and \( S_1(\cdot) \) and evaluates them at all \( n \) public indices: let \( r_{1,i} = R_1(i) \), and we use a similar notation for the remaining random polynomials. \( P_2 \) does the same with random polynomial \( R_2(\cdot), U_2(\cdot) \) and \( S_2(\cdot) \). \( P_1 \) submits \( r_{1,i} \) to the ith OLE instance, and \( P_2 \) submits \( (q_i, u_{2,i}) \); \( P_1 \) receives \( q_i r_{1,i} + u_{2,i} \). Symmetrically, \( P_2 \) receives from a parallel OLE instance \( p_i s_{2,i} \).
Setup: $P_1$ and $P_2$ agree on a common finite field $\mathbb{F}_p$ and $\omega$ is an $n^{th}$ root of unity of the field (namely, $n=(q-1)$). Let $\eta = \{1, \omega, \ldots, \omega^{n-1}\}$, $w$ be the input size and $t, e, n$ be positive integers where $k = (w + t + e)$, $2k < n$ and $(1 - e/n)^{t} \leq 2^{-\lambda}$.

Inputs: $P_1$ and $P_2$ have inputs $X = \{x_1, ..., x_w\}$ and $Y = \{y_1, ..., y_w\}$ respectively. (Assume $\eta \cap X = \emptyset, \eta \cap Y = \emptyset$.)

The Protocol:

1. **Input Sharing Phase.** $P_1$ samples $T_1(x)$ and $P_2$ samples $T_2(x)$, each a random polynomial of degree $t + e$. $P_1$ computes $P(x) = [\Pi_{j=1}^{w} (x - y_j)] \cdot T_1(x)$. $P_2$ computes $Q(x) = [\Pi_{j=1}^{w} (x - y_j)] \cdot T_2(x)$. $P_1$ computes $p_j = P(\omega^j)$, $P_2$ computes $q_j = Q(\omega^j)$ for all $j \in \{1, n\}$.

2. **Random Polynomials Sampling.**
   - $P_1$ samples random polynomials $Z_1(\cdot)$, $R_1(\cdot)$, $S_1(\cdot)$ and computes the $RS\mathcal{F}_{q,n,k,\eta}$ encodings: $z_1 = Z_1(\eta)$, $r_1 = R_1(\eta)$, $s_1 = S_1(\eta)$. $P_2$ samples $Z_2(\cdot)$, $R_2(\cdot)$, $S_2(\cdot)$ and computes $z_2 = Z_2(\eta), r_2 = R_2(\eta), s_2 = S_2(\eta)$.
   - All polynomials have degree at most $(w + t + e)$ and are chosen over the finite field $\mathbb{F}_q$.
   - $P_1$ samples a random polynomial $U_1(\cdot)$ of degree $2k$ and computes $u_1 = U_1(\eta)$.

3. **Coin Tossing.** The parties call $\mathcal{F}_{\text{Coin}}$ twice (Functionality 9), each receiving $n$ random strings and de-commitments for those strings: $((\sigma_{i,1}, \tau_{i,1}), \ldots, (\sigma_{i,n}, \tau_{i,n}))$, as well as $n$ commitments to the other party’s randomness: $(\text{com}_1, \ldots, \text{com}_n)$. $P_1$ will use $\sigma_{i,j}$ as its randomness for the $j$th OLE execution.

4. **Watchlist Channels Setup via t-out-of-n OT.** The parties call $\mathcal{F}_{\text{OT}}$ (Functionality 5). $P_1$ receives $t$ tuples $(q_1, u_{2,j}, r_{2,j}, s_{2,j}, \tau_{2,j})$ from $P_2$. They repeat the process with reversed roles, where $P_2$ receives $t$ tuples $(p_1, u_{1,j}, r_{1,j}, s_{1,j}, \tau_{1,j})$ from $P_1$. Let $I_1$ and $I_2$ be the sets of indices chosen by $P_1$ and $P_2$, respectively, defined by the receiver’s input to each OT instance.

5. **Degree Test.** The parties perform degree test on $Z_1, Z_2, R_1, R_2, S_1, S_2, P, Q$ to verify that they have a degree of at most $w + t + e$.

6. **OLE.** The parties make a sequence of calls to $\mathcal{F}_{\text{OLE}}$ (Functionality 7):
   - $P_1$ provides $r_1$ whereas $P_2$ provides $(q, u_2)$ to $\mathcal{F}_{\text{OLE}}$. $P_1$ obtains $c_1 = (c_{1,1}, \ldots, c_{1,n})$ where $c_{1,j} = q_1 \cdot r_{1,j} + u_{2,j}$.
   - $P_1$ provides $(p, u_1)$ whereas $P_2$ provides $s_2$ to $\mathcal{F}_{\text{OLE}}$. $P_2$ obtains $c_2 = (c_{2,1}, \ldots, c_{2,n})$ where $c_{2,j} = p_2 \cdot s_{2,j} + u_{1,j}$.
   - $P_1$ verifies that $c_1$ is a valid $RS\mathcal{F}_{q,n,2k,\eta}$ codeword.
   - $P_2$ verifies against the OLE execution for those inputs of $P_1$.
   - $P_2$ verifies against the OLE execution for those inputs of $P_1$.

7. **Output Reconstruction.**
   (a) $P_1$ computes $d_1$ where $d_{1,j} = c_{1,j} + p_j \cdot s_{1,j} - u_{1,j}$ and sends $d_1$ to $P_2$.
   (b) $P_2$ computes $d_2$ where $d_{2,j} = c_{2,j} + q_j \cdot r_{2,j} - u_{2,j}$ and sends $d_2$ to $P_1$.
   (c) The parties verify that $d_1$ is a valid $RS\mathcal{F}_{q,n,2k,\eta}$ codeword. They also verify against the OLE.

   - For $j \in I_1$: $d_{1,j} = c_{1,j} + p_j \cdot r_{1,j} - u_{1,j}$.
   - For $j \in I_2$: $d_{2,j} = c_{2,j} + q_j \cdot s_{2,j} - u_{2,j}$.
   (d) Both parties compute $t_j = d_{1,j} + d_{2,j} = p_j s_{1,j} + s_{2,j} + q_j r_{1,j} + r_{2,j}$.
   (e) $P_1$ and $P_2$ obtain $T(\cdot) = P(\cdot)S(\cdot) + Q(\cdot)R(\cdot)$ by interpolating the points $(\omega^j, t_j)$ and evaluate $T(\cdot)$ on their input.
   (f) $P_1$ outputs $X \cap Y = \{x_j \mid T(x_j) = 0\}$ and $P_2$ outputs $X \cap Y = \{y_j \mid T(y_j) = 0\}$.

(a) The random polynomials $Z_i(\cdot)$ are used in the degree test to verify that all shares are valid Reed-Solomon codes. The random polynomials $U_i(\cdot)$ are used to randomize the output of the OLE.

---

**Fig. 2.** Fully Secure Active Two-Party PSI Protocol.
$P_2$ computes and sends $(p_1 s_{2,i} + u_{1,i}) + (q r_{2,i} - u_{2,i})$. $P_2$ computes and sends $(p_1 s_{1,i} - u_{1,i}) + (q r_{1,i} + u_{2,i})$. Summing these together, they each learn $p_1 s_{1,i} + q r_{1,i}$, where $s_{1,i} = s_{1,i} + s_{2,i}$ is the evaluation of random, private polynomial $S$ at the $i$th public index (and $r_{1,i}$ is similar).

To ensure security, our protocol follows the blueprint of the two-party protocol designed by Hazay et al. [HIMV19], which is based on the IPS compiler [IPS08], and achieves malicious security using the “MPC-in-the-head” paradigm. This powerful paradigm securely realizes an arbitrary functionality $F$ with active security, while making black-box use of the following two ingredients: (1) an active MPC protocol which realizes $F$ in the honest majority setting, and (2) a passive MPC protocol in the dishonest majority setting (e.g. a two-party protocol) that realizes the next-message functions\(^4\) for each party in protocol (1). When applied to our setting, protocol (1) is the point-wise multiplication and addition of the polynomials previously described, and protocol (2) is the OLE execution above.

To enforce correct behaviour, Ishai et al. introduced a novel concept called *watchlists*: the parties run an emulation of protocol (1) by securely executing protocol (2) for each message. Each party obliviously checks the other party's behavior in (2) through OT channels, and because (1) is secure against a minority of corruptions, privacy is still guaranteed. Namely, the parties commit to the input and the randomness used in each OLE execution. Then, each party is allowed to obliviously open $t$ committed values to be checked against the messages received in the OLE execution. This oblivious choice is made via OT instances. With an appropriate choice of parameters (see Section 5), any attack will be caught with high probability. In this work, we build on the concrete analysis of [HIMV19] the honest majority building block of IPS, for concrete PSI protocols.

We use $w$ denote the size of each user's input set. We describe here how we set the degree of the polynomials. Namely, since every party will open and verify $t$ OLE instances, they will immediately learn $t$ shares of every polynomial. Furthermore, a malicious party might cheat in $e$ OLE instances, breaking the privacy of that execution, and learning an additional $e$ shares of each polynomial. We therefore use polynomials with degree greater than $w + t + e$, ensuring that $t + e$ shares do not leak anything about the roots of the polynomials. (Note that $T$ has double this degree, due to the polynomials multiplication.)

Our protocol is black-box in the implementation details of the underlying OLE and can be instantiated with any OLE protocol. To verify correctness of the OLE executions, the users begin by executing a secure co-flipping protocol that provides randomness for the OLE execution to one party, and a commitment to that randomness to the other party. The decomposition to the randomness is sent over the watchlist channels, together with the OLE inputs. This allows the receiving party to verify the correctness of all OLE messages in that execution. To maintain the reliance on black-box OLE, we treat the OLE executions as ideal function calls; however, the verification procedure just described will require knowledge of the particular OLE instantiation.

Overall, it costs only two passive OLE to compute each $T(\eta_j)$. Based on the analysis from [HIMV19], the amortized number of passive OLE needed for each item is $2n/w = 2[(w + t + e + 1)/w]$, or even if it sets more than $e$ evaluations $P(\eta_j)$ or $Q(\eta_j)$ to 0, this will be caught immediately by the honest party since the error probability will be $(1 - e/n)^t < 2^{-\lambda}$. As $w$ increases, $(4t + 4e + 2)/w \rightarrow 0$.

The watchlist mechanism used in our protocol also allows us to prevent the adversary from setting $P(\cdot)$ or $Q(\cdot)$ to 0 for free. In particular, via the use of a watchlist, each party can verify the computation of $t$ evaluations $T(\eta_j)$. Therefore, if the adversary deliberately sets $P(\cdot)$ or $Q(\cdot)$ to 0, or even if it sets more than $e$ evaluations $P(\eta_j)$ or $Q(\eta_j)$ to 0, this will be caught immediately by the honest party since the error probability will be $(1 - e/n)^t$ (the concrete parameters are fixed in Section 5). See Figure 2 for our two-party PSI protocol.

**Slackness.** We note that our approach introduces some slackness in terms of the input size: while honest parties will provide an input set of size $w$, a malicious party might include an additional $t + e$ inputs of its choosing without detection, by embedding the chosen values in an additional $t + e$ roots.\(^5\) This leaks some additional information about the honest input set.

Rather than attempting to prevent this, we weaken the functionality to reflect this attack. This weakening admits a more efficient protocol. In our protocols, the

---

\(^4\) The function computing the next outgoing message, given the current state of the participating party.

\(^5\) Technically, even in an honest execution, a random polynomial consistent with the honest input may contain some additional roots. However, if these are random points, the probability that they end up in the intersection is negligible. In an adversarial setting, the adversary could embed values of interest, based on some auxiliary information about the honest input.
slackness is defined as $\epsilon = (t + e)/w$. The slackness also has a negative impact on the efficiency, thus it is desirable to keep the slackness as low as possible. Concretely, for an input size $2^{24}$, our slackness is 24% of the input size (See Section 5.1). We stress that input size slackness exists in many efficient PSI constructions. For example, the fastest semi-honest protocol \cite{PSTY19} is based on cuckoo hashing, and has slackness at least 100%.

**Theorem 1.** Let $k, t, e, w, n$ be positive integers such that $k \geq t + e + w$, $e < d/3$, and $2k < n$, then protocol $\Pi_{\text{2PSI}}$ (cf. Figure 2) securely computes functionality $\mathcal{F}_{\text{2PSI}}$ (cf. Figure 1) with two parties in the $\{\text{ComdCoin}, \mathcal{F}_{\text{OTT}}, \mathcal{F}_{\text{COIN}}, \mathcal{F}_{\text{OLE}}\}$-hybrid, tolerating static adversaries with a statistical error of $d/|\mathbb{F}| + (1 - e/n)^3 + 2(w + t + e)/|\mathbb{F}|$ where $d = n - k + 1$ is the distance of the underlying code.

The proof can be found in Appendix C.

4 Fully Secure Active PSI: The Multi-Party Extension

Another important benefit of our paradigm is that, in contrast to most prior two-party approaches, it can be extended to any number of parties. At the heart of our multi-party protocol is an extension of the protocol shown in Section 3. The $m$ parties compute the polynomial $T$ of the form $T = Q_0 \sum_{i=0}^{m-1} R_i + \sum_{i=1}^{m-1} Q_i (S_0^i + S_i)$. Namely, all parties contribute their input polynomials as well as the masking polynomials. Our two-party PSI functionality is adapted to the multi-party setting in Figure 3. Throughout this section we highlight in blue any text related to our modified protocol that provides output only to $P_0$.

Our protocol uses a hybrid approach between a fully connected network and a star topology network, where the parties communicate with a single central party. That is, our protocol considers both types of networks, which is similar to the approach proposed by \cite{GPR+21}. Nevertheless, our MPSI protocol is fully secure while theirs only provides output the central party. We use watchlists to enforce honest behavior of all the parties. A fully connected network is needed to set up the watchlist channels among the parties, then a star topology is used to compute a masked intersecting polynomial between $P_0$ and $P_1$. For each pair ($P_0, P_1$), the central party $P_1$ learns an $RS\hat{g}_{\hat{w}, n, 2k, \eta}$ encoding of the polynomial $T_i = Q_0 \cdot R_i + Q_i (S_0^i + S_1^i) + V_i$ where $Q_i$ is the polynomial that encodes $P_i$’s input, $S_0^i$ and $(R_i, S_i)$ are random polynomials sampled by $P_0$ and $P_1$, respectively, and $V_i$ is the masked polynomial used to hide the intersection between $P_0$ and $P_1$. Specifically, the $V_i$’s are random polynomials that are sampled such that $\sum_{i=1}^{m-1} V_i = 0$, allowing $P_0$ to add all $T_i$ together to compute the intersection.

After $P_0$ obtains the $RS\hat{g}_{\hat{w}, n, 2k, \eta}$ encoding of the polynomial $T = Q_0 \cdot R_0 + \sum_{i=1}^{m-1} T_i$, it broadcasts the encoding to all other parties who can verify it against their watchlists. As all parties must commit their inputs during the watchlist channel setup steps, $P_0$ cannot drop or add anything to the intersection without being caught by the watchlists.

**Naive IPS Watchlist Setup.** The watchlist channels setup step proposed in \cite{LOP11} requires $O(m^3 + m^2 n)$ bits where $n = O(w + mt + e)$. To set up the watch channels, each party $P_1$ executes the multi-sender $t$-out-of-$n$ OT protocol, called $\mathcal{F}_{\text{mOT}}^{t,n}$ (see Figure 6), which allows $P_1$ to watch all other parties at the same $t$ channels of its choice. The authors proposed an instantiation for $\mathcal{F}_{\text{mOT}}^{t,n}$ based on DDH assumption, where each channel requires $O(n) = O(w + mt + e)$ expositions for input length $w$. It is clear that if the watchlist channels for our multi-party PSI protocol are set up using their instantiation, the protocol will be very inefficient.

**Functionality $\mathcal{F}_{\text{MPSI}}$**

**Setup.** Let $t, e, w, n$ be positive integers where $w$ is the parties’ input size, $k = w + t + e$, $n > 2k$, $d = n - k + 1$, $e < d/3$, $(1 - e/n)^3 < 2^{-\lambda}$.

**Functionality.** $\mathcal{F}_{\text{MPSI}}$ communicates with parties $P_0, \cdots, P_{m-1}$, and adversary $A$.

- Wait for the input $X^{(i)} = (x_1^{(i)}, ..., x_w^{(i)})$ from $P_i$.
- If $P_0$ is one of the corrupted parties, replace all other corrupted parties’ input with $P_0$’s input.
- Wait for the adversary $A$ to add as many as $(m-t+e)$ additional items to the input set of the corrupted parties. Let $X^{(i)}$ be the modified input set of party $P_i$.
- Send output $S \leftarrow \bigcap_{i=0}^{m-1} X^{(i)}$ to $P_0$.
- If all parties are supposed to receive output:
  - If $P_0$ is corrupted, wait for abort/continue from $P_0$. Upon receiving abort, send $\perp$ to all honest parties. Else, send $S$ to all honest parties.
  - If $P_0$ is honest, send $S$ to all parties.

![Fig. 3. Multi-Party PSI Ideal Functionality.](image-url)
Watchlist Channels via the Commit-and-Reveal Paradigm. We propose a new way to set up our watchlist channels. We send only $O(m^2 \log n)$ bits (where $n = O(w + t + e)$), and the construction has very low computational cost. The number of watched channels in [LOP11] is $O(mt)$, as each party independently chooses $t$ servers to watch. If $(m-1)$ parties are corrupt, they can collectively learn $(m-1)t$ shares from the honest party. This would force us to pad the input polynomials with a random polynomial of degree of $(m-1)t + e$, resulting in $n > 2(w + (m-1)t + e)$. If we could instead arrange for all parties to watch the same $t$ channels, we could set $n > 2(w + t + e)$ instead of $n > 2(w + (m-1)t + e)$. The challenge is that a colluding party will tell $P_0$ which channels are being watched; the adversary can avoid being caught when it cheats.

To solve this problem, we replace the OT watchlist with a commit-and-reveal protocol. Instead of “quietly” watching a live channel, the parties are asked to commit to their shares before the computation, and only when they perform a check, after the messages are sent, do they agree on a random set of $t$ channels. They decommit those shares, together with any randomness used in these channels, to all other parties. To reduce the cost of broadcasting the commitments, the parties commit to their shares before the computation, and only when required to pad the input polynomials with a random polynomial of degree of $(m-1)t + e$, resulting in $n > 2(w + (m-1)t + e)$. If we could instead arrange for all parties to watch the same $t$ channels, we could set $n > 2(w + t + e)$ instead of $n > 2(w + (m-1)t + e)$. The challenge is that a colluding party will tell $P_0$ which channels are being watched; the adversary can avoid being caught when it cheats.

Theorem 2. Let $k, t, e, w, n$ be positive integers such that $k \geq w + 3t + e$, $e < d/3$, and $2k < n$, then protocol $\Pi_{\text{MPSI}}$ (cf. Figure 4) securely computes functionality $F_{\text{MPSI}}$ (cf. Figure 3) with $m$ parties in the $\{F_{\text{Coin}}, F_{\text{OLE}}\}$-hybrid, tolerating static adversaries with a statistical error of $d/|F| + (1-e/n)^t + m(w+3t+e)/|F|$ where $d = n-k+1$ is the distance of the underlying code.

The proof can be found in Appendix D.

5. The Efficiency of Our Protocols

5.1 The Two-Party Setting

In this section we will explore the concrete parameters of our two-party protocol. Let $w$ be the input length, and let $t$, $e$, and $F$ be such that our statistical error $(1-e/n)^t + d/|F| + (w+t+e)/|F|$ is bound by $2^{-\lambda}$ (where $n = 2(w+t+e)+1$) and $\lambda$ is the security parameter. Note that for the case of two-party PSI, it is desirable that $t+e$ is as small as possible, because the slackness of our protocol is defined by $(t+e)/w$. We show that the optimal solution for $t+e$ is bound by $O(\sqrt{\lambda \cdot w})$. Fixing $e_0 = \sqrt{\lambda \ln 2 \cdot (2w)}$, we now look for $t_0$ such that $(1-e_0/n)^{t_0} \approx \exp(-\lambda \ln 2) = 2^{-\lambda}$, or $t_0 \cdot \log(1-e_0/n) \approx -\lambda \ln 2$. As $e_0 \ll w < n$, $t_0 \cdot \log(1-e_0/n)$ can be approximated with $t_0 \cdot (-e_0/n)$ using Taylor’s approximation. Thus $t_0 \approx \lambda \ln 2 \cdot (2(w + t_0 + e_0) + 1)/e_0 \approx e_0 + 2\lambda \ln 2 \cdot t_0/e_0 + 2 \cdot \lambda \ln 2$. It is clear that $t_0 < 2 \cdot e_0$. If $t$ and $e$ are optimized, then $t+e \leq t_0 + e_0 < 3 \cdot \sqrt{\lambda \ln 2 \cdot (2w)}$. 

MPSI With One Party Output. While we have mainly focused on achieving full security, where every party receives correct output, it is worth noting that if we relax security as in [GPR+21], then with a few modifications our communication cost is only $O((m^2 + mn)\kappa)$. First, during the watchlist channels setup step, instead of requiring $P_i \neq P_0$ to watch all other parties, $P_i$ only needs to watch $P_0$. As a result, the communication cost of the watchlist channels setup is now reduced to $O((m^2 + mn)\kappa)$. Specifically, $P_0$ commit to its shares via a Merkle tree and broadcasts the tree’s root to everyone. This costs $O(m^2\kappa)$ bits. The OLE verification cost is $O(\kappa \log n)$ for each pair $(P_0, P_i)$, which is very small compared to the cost to compute the OLE. Concretely, our relaxed multi-party PSI has the amortized cost of $4m$ passive OLE per input item.

Our MPSI Asymptotic Communication Cost. Our new watchlist mechanism has communication cost of $O(m^2 \kappa \log n)$ bits where $n = O(w + t + e)$, and $t + e \ll w$. Additionally, the pairwise OLE executions (Step 8) cost $O(nm\kappa)$ bits. In total, the communication cost of our protocol is $O((m^2 \log n + mn)\kappa)$ bits. Our protocol is presented in Figure 4.
1. **Setup.** Parties $P_0, \ldots, P_{m-1}$ agree on a common finite field $\mathbb{F}_q$ and $\omega$ is an $n^{th}$ root of unity of the field (namely, $\eta(q-1)$). Let $n = \{1, \omega, \ldots, \omega^{n-1}\}$, $w$ the input size and $t, e, n$ be positive integers such that $2k < n$, $k = (w + 3t + e)$, $e < (n - k + 1)/3$, and $(1 - e/n) < 2^{-\lambda}$.

2. **Input Sharing Phase.** Each party $P_i$ has an input $X_i = \{x_i^1, \ldots, x_i^n\}$. $P_i$ samples a random polynomial $T_i(x)$ of degree $t + e$, computes $Q_i(x) = T_i(x)\Pi_{j=1}^{n} (x - x_i^j)$ and the $RS_{\mathbb{F}_q, n, k, q}$ encoding $\mathbf{q}_i = Q_i(\eta)$.

3. **Sample Random Masked Polynomials.** For each pair $(i, j)$ where $1 \leq i < j \leq (m - 1)$, $P_i$ and $P_j$ call $F_{\text{COIN}}$ (Functionality 8) to sample a common seed $\text{seed}_{ij}$. Let $V_{ij} \leftarrow PRG(\text{seed}_{ij})$ be a random polynomial of degree $2k$. $P_i$ stores $V_{ij}$ while $P_j$ stores $V_{ji} = -V_{ij}$. For $i \in \{1, m - 1\}$, $P_i$ sets $V_i = \sum_{1 \leq j < i \leq m-1} V_{ij}$ and computes the $RS_{\mathbb{F}_q, n, 2k, q}$ encoding $\mathbf{v}_i = V_i(\eta)$.

4. **Random Polynomials Sampling.**
   - $P_0$ samples random polynomials $R_0(\cdot), Z_0(\cdot),$ and $S_0(\cdot)$ for $i \in \{1, m - 1\}$. $P_i$ samples $Z_i(\cdot), R_i(\cdot),$ and $S_i(\cdot)$.
   - All polynomials have degree at most $k$ and are chosen over the common finite field $\mathbb{F}_q$. $F_0$ computes the $RS_{\mathbb{F}_q, n, k, q}$ encodings: $\mathbf{z}_0^i = Z_0(\eta), \mathbf{r}_0^i = R_0(\eta),$ and $\mathbf{s}_0^i = S_0(\eta)$. $P_i$ sets $z_0^i = Z_i(\eta), r_0^i = R_i(\eta),$ and $s_0^i = S_i(\eta)$.
   - For $i \in \{1, m - 1\}$, $P_i$ samples random seed and polynomial $U_i(\cdot) \leftarrow PRG(\text{seed}_{ij})$ of degree $2k$. They compute the encoding $\mathbf{u}_i = U_i(\eta)$. $P_i$ broadcasts the $\text{Com}(\text{seed}_{ij})$ to all other parties.

5. **Coin Tossing.** Each pair $(P_0, P_1)$ call $F_{\text{ComCoin}}$ twice (Functionality 9), each receiving $n$ random strings and decommissions for those strings. The random values are used for the OLE invocations.

6. **Watchlist Channels Commitment.** $P_0$ commits its shares $(q_{0,j}, r_{0,j}, \ldots, s_{0,j}, \ldots, s_{0,j-1})$ to all parties using Merkle tree. $P_i$ commits its shares $(q_{i,j}, r_{i,j}, s_{i,j}, v_{i,j}, u_{i,j})$ to all parties using Merkle tree. For one-sided output: $P_i$ only sends the commitment to $P_0$.

7. **Degree Test (First Check).**
   - The parties call $F_{\text{COIN}}$ to sample random public values $\{a_1, a_2, a_3, \alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_k\}$ together ($1 \leq i \leq m - 1$).
   - $P_i$, $1 \leq i \leq m - 1$, computes $b_i$ where $b_i = \alpha_i - \omega^{-j} \cdot z_i + \alpha_i^2 \cdot r_i + \alpha_i^3 \cdot s_i + j - 1 \cdot q_i$. Then, it sends $b_i$ to $P_0$.
   - $P_0$ computes $b_0$ where $b_0 = \alpha_1 - \omega^{-j} \cdot z_0 + \alpha_2 \cdot r_0 + \alpha_3 \cdot q_0 + \sum_{j=1}^{m-1} \alpha_i^j \cdot s_i + j - 1 \cdot q_i$. Then, it broadcasts $b_0$ to all other parties.
   - Select watchlist channels for first check. All parties make a call to $F_{\text{COIN}}$ to sample $I_1 = \{i_{1,1}, \ldots, i_{1,t}\}$ and reveals to all other parties their shares and OLE randomness for these indices.
   - Verify $Q_i(\cdot) \neq 0$. All parties verify that $q_i \neq 0$ for $i \in I_1$.
   - **Degree test.** All parties verify that $b$ is a valid $RS_{\mathbb{F}_q, n, k, q}$ code word and for every $j \in I_1$:
     $$b_j = \sum_{i=1}^{m} [\alpha_i^j \cdot z_0 + \alpha_i^j \cdot r_0 + \alpha_i^j \cdot s_i + j - 1 \cdot q_i]$$

8. **OLE.** Each pair $(P_0, P_i)$ makes a sequence of calls to $F_{\text{OLE}}$ on inputs. $P_0$ provides $s_i^0$ whereas $P_i$ provides $(q_0, u_i^0)$ to $F_{\text{OLE}}$. $P_0$ obtains $c_i^0 = (c_{i,1}^0, \ldots, c_{i,n}^0)$ where $c_{i,j} = q_{i,j} \cdot s_{i,j} + u_{i,j}$. $P_i$ provides $(q_0, u_i^0)$ to $F_{\text{OLE}}$. $P_i$ obtains $c_i = (c_{i,1}, \ldots, c_{i,n})$ where $c_{i,j} = q_{i,j} \cdot r_{i,j} + u_{i,j}$. For one-sided output: $(P_0, P_i)$ executes only one OLE, $P_0$ provides $q_0$. $P_i$ provides $(q_1, q_i s_i + v_i)$. $P_i$ learns $f_i^0 = q_0 r_i + q_i s_i + v_i$.

9. **Verify OLE (Second Check).**
   - Select watchlist channels for OLE verification. All parties make a call to $F_{\text{COIN}}$ to sample a common random coin, and use that coin to sample to sample $I_2 = \{i_{2,1}, \ldots, i_{2,t}\}$. The parties reveal all the shares and randomnesses used at these indices to all other parties.
   - Verify $F_{\text{OLE}}$. For each pair $(P_0, P_i)$:
     - $P_0$ verifies that $c_i^0$ is a valid $RS_{\mathbb{F}_q, n, 2k, q}$ code word and for $j \in I_2$: $c_{i,j} = q_{i,j} \cdot s_{i,j} + u_{i,j}$.
     - $P_i$ verifies that $c_i$ is a valid $RS_{\mathbb{F}_q, n, 2k, q}$ code word and for $j \in I_2$: $c_{i,j} = q_{i,j} \cdot r_{i,j} + u_{i,j}$.
   - For one-sided output: $(P_0, P_i)$ verify the execution of only one OLE that computes $f_i^0$.

10. **Output Aggregation and Verification.**
    - $P_0$ computes $d_0 = q_0 \cdot r_0 + \sum_{i=1}^{m-1} (c_{i,j}^0 - u_i^0) = q_0 \cdot r_0 + \sum_{i=1}^{m-1} q_i \cdot s_i + \sum_{i=1}^{m-1} (u_i - u_i^0)$.
    - $P_i$ computes $d_i = c_{i,1} + q_{i,1} \cdot s_{i,1} + v_i = q_{i,1} \cdot r_{i,1} + q_{i,1} \cdot s_{i,1} + v_i + (u_i^0 - u_i)$.
    - $P_i$ sends $d_i$ to $P_0$. $P_0$ computes $t = \sum_{i=0}^{m-1} d_i$ and broadcasts it to all $P_i$.
    - Select watchlist channels for third check. All parties make a call to $F_{\text{COIN}}$ to sample a common random coin, and use that coin to sample to sample $I_3 = \{i_{3,1}, \ldots, i_{3,t}\}$. The parties reveal all the shares used at these indices to all other parties.
    - $P_0$ verifies that $d_{i,j} = q_{i,j} \cdot r_{i,j} + q_{i,j} \cdot s_{i,j} + v_{i,j} + (u_{i,j}^0 - u_{i,j})$ for $i \in \{1, m - 1\}$ and $j \in I_3$.
    - $P_i$ verifies that $t_j = q_{i,j} \cdot r_{i,j} + \sum_{i=1}^{m-1} q_{i,j} \cdot s_{i,j} + s_{i,j}$ for $j \in I_3$.
    - Each $P_i$ reconstructs the polynomial $T_i(x)$ from the points $(\omega^j, t_j)$ and outputs the intersection $S = \{x \in X_j \mid T(x) = 0\}$.

For one-sided output: $P_0$ computes $t = q_0 \cdot r_0 + \sum_{i=1}^{m-1} f_i^0$, reconstructs $T_i(x)$, and computes $S$.
Instantiate OLE With OT [Gil99]. To compute the OLE with input $x \in \mathbb{Z}_p$ from the receiver and $a, b \in \mathbb{Z}_p$ from the sender, the receiver first decomposes $x$ into bits $(x_1, \ldots, x_{|x|})$ where $|x|$ is the bit length of $x$. Both parties execute $|x|$ 1-out-of-2 OT where for the $j$th OT the sender provides the messages $(b_j, 2^{j-1}a + b_j)$ such that $b = \sum_j b_j$ and the receiver has the selection bit $x_j$. The receiver obtains $2^{j-1}ax_j + b_j$. Upon concluding the OTs, the receiver sums all the values it receives and gets $ax + b = \sum_j (2^{j-1}ax_j + b_j)$. The advantage of this instantiation is that it is very efficient, computationally, due to the use of OT extension. However, the communication cost is high, at $O(|x|^2)$ bits per OLE.

Slackness Parameters for Ring-LWE Based OLE. When the packed additive homomorphic encryption scheme is instantiated with Ring-LWE, each ciphertext encodes $N$ plaintexts where $N$ is the degree of the polynomial used in Ring-LWE scheme. Each randomness generated for the OLEs will be used in the batch polynomial used in Ring-LWE scheme. Each randomly generated OLE computation incurs at least $N$·$|x|$ bits, the number of AES calls is linear in terms of input size. The number of multiplications in our protocol is $O(n/|x|)$. The computation incurs at least $N$·$|x|$ bits, the number of AES calls is linear in terms of input size.

Computational Complexity. We measure the computational complexity in terms of number of field multiplications and the number of local AES operations, which we use to sample random field elements, primarily when sampling random polynomials. It is clear that our protocol make $O(n)$ calls to local AES, thus the number of AES calls is linear in terms of input size. The number of multiplications in our protocol is

$$O(n/|x|) + O(n/|x|) + O(n/|x|)$$

The overall asymptotic computational complexity is $O(n^2)$.

5.2 The Multi-Party Setting

We implement our fully secure multi-party PSI protocol (see Table 2). Here, we provide an estimation for the theoretical communication and computation cost for the central party and for non-central ones.

Communication Complexity. We distinguish between the central party and the remaining parties. First, the cost for the central party is

$$O(|x|^2) + O(|x|^2) + O(|x|^2)$$

where $O(|x|^2)$ is the communication complexity of the underlying OT and OLE protocols, respectively. As $n = O(|x|)$, the overall communication complexity is $O(|x|^2)$ bits.

The dominant communication cost of our protocol is due to computing the OLEs. Each OLE invocation requires the parties to communicate $4 \cdot \log(q)$ bits, where $q > N \cdot |F|^2$ is the ciphertext modulus and $N$ is the degree of the polynomial ring used in the underlying encryption scheme. With a conservative estimation, the OLE computation incurs at least 50% of the total communication.

Computational Complexity. We measure the computational complexity in terms of number of field multiplications and the number of local AES operations, which we use to sample random field elements, primarily when sampling random polynomials. It is clear that our protocol make $O(n)$ calls to local AES, thus the number of AES calls is linear in terms of input size. The number of multiplications in our protocol is

$$O(n/|x|) + O(n/|x|) + O(n/|x|)$$

Overall, our asymptotic computational complexity is $O(n^2)$ field multiplications (due to the output reconstruction step that requires polynomial evaluation on $w$ points) and $O(n)$ local AES calls to sample the random polynomials. Even though polynomial evaluation has higher asymptotic computational complexity, the actual running time is dominated by the OLE costs. Further optimizations can be found in Appendix B.
Table 2. Fully secure MPSI: Runtime (in seconds) and communication cost (in MB). Input items are represented by elements of a 64-bit prime field $F_p$. Our OLE is instantiated with OT for $w \in \{2^8, 2^{12}, 2^{16}\}$ and with Ring-LWE for $w \in \{2^{18}, 2^{20}\}$.

<table>
<thead>
<tr>
<th>Parties</th>
<th>$w = 2^8$</th>
<th>$w = 2^{12}$</th>
<th>$w = 2^{16}$</th>
<th>$w = 2^{18}$</th>
<th>$w = 2^{20}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Runtime</td>
<td>Comm</td>
<td>Runtime</td>
<td>Comm</td>
<td>Runtime</td>
</tr>
<tr>
<td>2</td>
<td>0.16</td>
<td>4.8</td>
<td>0.83</td>
<td>40.2</td>
<td>8.10</td>
</tr>
<tr>
<td>4</td>
<td>0.27</td>
<td>14.5</td>
<td>1.55</td>
<td>120</td>
<td>12.56</td>
</tr>
<tr>
<td>8</td>
<td>0.19</td>
<td>33.8</td>
<td>3.00</td>
<td>282</td>
<td>20.50</td>
</tr>
<tr>
<td>16</td>
<td>1.05</td>
<td>72.5</td>
<td>6.40</td>
<td>602</td>
<td>40.53</td>
</tr>
<tr>
<td>32</td>
<td>2.26</td>
<td>150</td>
<td>12.7</td>
<td>1245</td>
<td>77.16</td>
</tr>
</tbody>
</table>

Computational Complexity. In terms of computational cost, the central party has to make $O(mn)$ AES calls to sample the random polynomials and the encryption randomness, while each non-central party makes $O(w)$ AES calls. Next, we count the number of field multiplications that are performed by each party. For the central party, the number of field multiplications is

$$O(n/N \cdot (N \log N)) + O(m \cdot n/N \cdot (N \log N)) + \underbrace{2n \cdot \text{CCOLE}}_{\text{watchlists commit}} + \underbrace{6n \cdot \kappa}_{\text{passive OLE}} + \underbrace{m t \log n \cdot \log |F|}_{\text{coin toss}} + \underbrace{n \cdot \log |F|}_{\text{watchlist comm.}} + \underbrace{(w \cdot \log w)}_{\text{degree test}}$$

For each non-central party, the number of field multiplications is

$$O(n/N \cdot (N \log N)) + O(n/N \cdot (N \log N)) + O(mn) + O(n \cdot \log n) + O(n) + O(w \cdot \log w)$$

6 Implementation Details

Experiments Setting. We implemented our protocols using C++ and the NTL library, and deployed it over AWS servers. We demonstrate our protocol performances in the LAN network where the AWS instances are located in the same region (Northern Virginia). We ran our experiments for sets of input sizes $2^8, 2^{12}, 2^{16}, 2^{18},$ and $2^{20}$ with 2, 4, 8, 16, or 32 parties. We report the running times of the average over 5 executions. For all our experiments, the standard deviation is at most 4.2% of the average runtime.

In these experiments, two AWS instances of type c5.24xlarge were used. Each instance has 48 physical cores supporting 96 threads, CPU clock speed of 3.6 GHz, 192 GB RAM, and its LAN network bandwidth is 25 Gbps. We deployed the central party $P_0$ on one instance and parallelize $P_0$’s code with 32 threads. The second AWS instance hosted the remaining $(m - 1)$ parties. When $m = 2$, we used 32 threads for $P_1$. For $m \geq 4$, all parties $P_1, \ldots, P_{m-1}$ share 96 threads (on average each party runs with 96/$(m - 1)$ threads).

Our Protocol Is Fully Parallelizable. Number theoretic transform is used extensively in our protocol: computation of Reed-Solomon encodings for input and random polynomials, Ring-LWE operations, polynomial multiplication, polynomial division, polynomial evaluation over $w$ points, etc. Fortunately, number theoretic transform is fully parallelizable and we utilize it as much as possible in our implementation. The part that is not fully parallelizable in our protocol is the construction of the Merkle trees. However, instead of generating one Merkle tree, we can divide the data into $p$ chunks and generate $p$ trees in parallel (assume $p$ is the number of threads used). The commitment is $p$ hash digests instead of one. This increases the communication cost a bit, but in return our implementation is fully parallelizable.
Results. Our experiment results are reported in Tables 2 and 3. Table 2 shows the running time and communication cost for our fully secure PSI protocols, while Table 3 shows the results for the variant in which only the central party receives output.

Our efficiency depends partly on the slackness, which is disproportional to the input size. In our experiments, we instantiate the OLE instances with OT when the input size is small (i.e., $2^8$, $2^{12}$, $2^{16}$) and with Ring-LWE when $w \geq 2^{18}$. The reason for that is because the slackness of the Ring-LWE based OLE is large for small input sizes, causing the protocol to be less concretely efficient than for the OT-based OLE. For example, when $w = 2^{16}$, the slackness of Ring-LWE based OLE is 1250% whereas that of OT-based OLE is 7%. On the other hand, the Ring-LWE based OLE has the asymptotic communication cost of $O(p)$ bits per OLE while the OT-based has the cost of $O(p^2)$, where $p$ is the bit length of the field.

As there is no other fully secure multi-party PSI to compare with, we only provide a comparison between our relaxed one-sided output MPSI with prior similar protocols. Among them, only PSimple [BNOP21] and [KMP+17] report experimental results. [BNOP21] uses at least 36 threads for the central party which was deployed on a c5.18xlarge machine (36 cores, 3.6 GHz clock speed, and 144 GB RAM) for $P_0$ and one c5.4xlarge machine (8 cores, 3.6 GHz clock speed, and 32 GB RAM) for each other $P_i$. Even though we are somewhat less parallelized than [BNOP21] (in terms of the number of threads and cores used for $P_0$ and $P_i$ when there are more than 6 parties), our protocol is still competitive and outperforms [BNOP21] when the input size is at least $2^{18}$. When $w = 2^{20}$ our protocol is at least $3\times$ faster. (See Table 3). Asymptotically, their protocol also requires much higher communication complexity than ours, namely, $O(mw^2 + mw \log(kw))$ vs. $O((mw + m^2 + mt \log w)\kappa)$ (see Table 1).

For large input sizes, e.g., $n = 2^{20}$, our protocol is also very competitive against [KMP+17] which is only semi-honest secure. [KMP+17] ran their experiments with all parties deployed on the same machine, a $2 \times 36$-core Intel Xeon with 2.30 GHz CPU and 256 GB of RAM. Considering 15 parties and 14 threads per party (in their implementation, each party uses $(m-1)$ threads where $m$ is the number of parties), [KMP+17] is $3\times$ slower than ours. Garimella et al. [GPR+21] modifies the augmented semi-honest version of [KMP+17] and make it malicious secure with one-sided output. As no experiment results are provided for [GPR+21], we used the available results from [KMP+17] for comparison.

### 7 Conclusions

In this paper, we present two new fully secure PSI constructions with active security: a two-party and a multi-party protocols that provide correct output to all parties (if they ever receive it). Unlike existing state-of-the-art prior work that provides output to only one party, ours are the first practical PSI protocols to provide this feature. Our protocols are constructed based on the MPC-in-the-head paradigm and can be instantiated with any passively secure OLE. Beside the fully secure protocols, we also provide a more efficient multi-party PSI protocol when only one party obtains the output.
References


[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In STOC, pages 364–369, 1986.


A Additional Preliminaries

A.1 Oblivious Transfer

1-out-of-2 oblivious transfer (OT) is a fundamental functionality in secure computation that is engaged between a sender $S$ and a receiver $R$ where a receiver learns only one of the sender’s inputs whereas the sender does not learn anything about the receiver’s input. Here we consider a generalized version of $t$-out-of-$n$ OT where the receiver learns $t$ values and which will be useful in establishing the watchlist channels; see Figure 5 for its formal description.

![Fig. 5. The oblivious transfer functionality.](image-url)

We also define a variance of $t$-out-of-$n$ OT where there are multiple senders. In this setting the receiver learns $t$ values from each sender. The indices of these values are the same across all the senders. See Figure 6 for its formal description.
Functionality $\mathcal{F}_{\text{mOT}}^{1:n}$

Functionality $\mathcal{F}_{\text{mOT}}^{1:n}$ communicates with senders $S_i$ and receiver $R$, and adversary $A$.

1. Upon receiving input $(sid, v_1, \ldots, v_n)$ from $S_1$ where $i \in [m]$ and $v_j \in \{0,1\}^n$ for all $j \in [n]$, record $(sid, v_1, \ldots, v_n)$.
2. Upon receiving $(sid, v_1, \ldots, v_n)$ from $R$ where $v_i \in \{0,1\}^\log n$ for all $i \in [t]$, send $(v_1, \ldots, v_t)$ for all $i \in [m]$ to $R$. Otherwise, abort.

Fig. 6. The multi-sender $t$-out-of-$n$ OT functionality.

A.2 Oblivious Linear Evaluation

An extension of the oblivious transfer functionality for larger fields is the OLE functionality. More concretely, OLE over a field $\mathbb{F}$ takes a field element $x \in \mathbb{F}$ from the receiver and a pair $(a, b) \in \mathbb{F}^2$ from the sender and delivers $ax + b$ to the receiver. Note that in the case of binary fields, OLE can be realized via a single call to standard (bit-) 1-out-of-2 OT functionality; see Figure 7 for its formal description.

Functionality $\mathcal{F}_{\text{OLE}}$

Functionality $\mathcal{F}_{\text{OLE}}$ communicates with sender $S$ and receiver $R$, and adversary $A$.

1. Upon receiving the input $(sid, (a, b))$ from $S$ where $a, b \in \mathbb{F}$, record $(sid, (a, b))$.
2. Upon receiving $(sid, x)$ from $R$ where $x \in \mathbb{F}$, send $a \cdot x + b$ to $R$. Otherwise, abort.

Fig. 7. The oblivious linear evaluation functionality.

A.3 Coin Tossing

We further use functionality $\mathcal{F}_{\text{COIN}}$ for generating the randomness for the degree test. This functionality can be implemented using commitments. We use a standard coin tossing functionality $\mathcal{F}_{\text{COIN}}$ for generating the randomness used in the OLE instances.

Functionality $\mathcal{F}_{\text{COIN}}$

Upon receiving $(\text{rand}, S)$ from all parties, where $S$ is any efficiently sampleable set,
- Sample $r \leftarrow S$, send $r$ to $A$ and wait for its input.
- If $A$ inputs 'continue' then output $r$ to all parties, otherwise output $\bot$.

Fig. 8. Public coin tossing functionality.

Functionality $\mathcal{F}_{\text{ComCoin}}$

Upon receiving $(\text{rand}, S)$ from both parties, where $S$ is any efficiently sampleable set,
- For $i \in [n]$, sample $\sigma_i \leftarrow S$, and compute $(\text{com}_i, \tau_i) \leftarrow \text{Commit}(\sigma_i)$.
- Send $(\text{com}_i)$ to $P_1$ and $\tau_i$ to $P_2$ and wait for its response.
- If $P_1$ inputs 'continue' then output $(\sigma_i, \tau_i)$ to $P_1$, otherwise output $\bot$.

Fig. 9. Committed coin tossing functionality for two parties.

Instead of calling this functionality $n$ times, we describe the functionality as returning $n$ random strings. When realizing this functionality, this allows us to use succinct commitments, e.g. through the use of Merkle trees.

Instead of making two calls to $\mathcal{F}_{\text{OLE}}$ to compute $t_j$, we just use Ring-LWE in a way that allows us to have better MPC-in-the-head parameters.

- $P_1$ encrypts $p_j, r_{1,j}$ and sends to $P_2$.
- $P_2$ computes $Enc(PK, p_j s_{2,j} + q_j r_{1,j} + q_j r_{2,j})$ and sends it back to $P_1$.
- $P_1$ decrypts the ciphertext, adds $p_j s_{1,j}$ itself to the output, and obtains $t_j$. $P_1$ sends $t_j$ to $P_2$.

This new way of computing $t_j$ also needs just semi-honest Ring-LWE operations; honest behavior is enforced with the use of MPC-in-the-head. The value $p_j s_{2,j} + q_j r_{1,j} + q_j r_{2,j}$ does not leak any additional information beyond what was presented in Figure 2, as $P_1$ can learn $p_j s_{2,j} + q_j r_{1,j} + q_j r_{2,j}$ from $t_j = p_j(s_{1,j} + s_{2,j}) + q_j(r_{1,j} + r_{2,j})$ anyway (as it knows $p_j$ and $s_{1,j}$).
C Proof of Theorem 1

We will consider each corruption case separately. In our simulations, $\tilde{m}$ is a message generated by the simulator to simulate the message $m$ in the hybrid protocol.

Simulation for a Corrupted $P_1$.

1. Coin-Tossing. The simulator plays the role of the trusted party in $F_{\text{CoinCoin}}$, honestly, generating random coins and commitments.

2. Watchlists.

   $q_t, \tilde{w}_{1, t}, \tilde{z}_{2, t}, \tilde{r}_{2, t}, \tilde{s}_{2, t}$: The simulator samples random polynomials $\tilde{Q}(\cdot), \tilde{Z}(\cdot), \tilde{R}(\cdot), \tilde{S}(\cdot)$ of degree $w + t + e$, and $\tilde{U}(\cdot)$ of degree $2(w + t + e)$. It evaluates the polynomials on the roots of unity $\eta = (1, \omega, \ldots, \omega^{a-1})$ and obtains $RS_{\tilde{q}, \eta}^{\tilde{t}}$, $RS_{\tilde{z}, \eta}^{\tilde{t}}$, $RS_{\tilde{r}, \eta}^{\tilde{t}}$, and $RS_{\tilde{Z}, \eta}^{\tilde{t}}$ encoding $\tilde{q}, \tilde{z}, \tilde{r}, \tilde{z},$ and $\tilde{Z}$, respectively. The simulator sends its input to $P_1$, simulates the output of the protocol, and proceeds to the next step.

3. Degree Test.

   $\tilde{\alpha}_1, \tilde{\alpha}_2, \tilde{\alpha}_3, \tilde{\alpha}_4, \tilde{\alpha}_5, \tilde{\alpha}_6, \tilde{\alpha}_7, \tilde{\alpha}_8, \tilde{\beta}_j$: The simulator samples $\tilde{\alpha}_1, \ldots, \tilde{\alpha}_8$ uniformly at random and hands them to $P_1$ to simulate the output of $F_{\text{Coin}}$. It computes $b = (b_1, \ldots, b_a)$ where $\tilde{b}_j = \alpha_1 \cdot \tilde{\alpha}_j + \alpha_2 \cdot \tilde{\alpha}_j + \alpha_3 \cdot \tilde{\alpha}_j + \alpha_4 \cdot \tilde{\alpha}_j + \alpha_5 \cdot \tilde{\alpha}_j + \alpha_6 \cdot \tilde{\alpha}_j + \alpha_7 \cdot \tilde{\alpha}_j + \alpha_8 \cdot \tilde{\alpha}_j$ and hands $b$ to $P_1$ to simulate the messages $P_1$ receives in Step 5. If abort$_0 = 1$ the simulator aborts and outputs whatever $P_1$ outputs.

Input Extraction. If abort$_0 = 0$, the simulator interpolates the polynomial $\tilde{P}(\cdot)$ from the points $(\omega_j, \tilde{p}_j)$. If $\tilde{P}(\cdot) \equiv 0$, the simulator aborts and outputs whatever $P_2$ outputs. Else, the simulator extracts $P_1$’s input $\tilde{X}$ defined by $\tilde{X} = \{ x | \tilde{P}(x) = 0 \}$. The extracted input must be embedded with the following slackness: $\tilde{X} = X \cup X'$ where $X'$ are the roots of $T_1(\cdot)$ chosen by $P_1$ in Step 1.

Synthesize $P_3$’s Input. The simulator submits $\tilde{X}$ to the ideal functionality and obtains $\tilde{X} \cap Y$. It recomputes $\tilde{Q}(\cdot)$ such that $\tilde{Q}(\cdot) = \tilde{W}(\cdot) \prod_{z \in X \cap Y} (X - z)$ such that $deg(\tilde{Q}) = w + t + e$, $\tilde{Q}(\eta_j) = \tilde{q}_j$ for $j \in I_1$, and $W(z) \neq 0$ for $z \in \tilde{X} \cap Y$.

4. OLE. $\tilde{c}_1 = (\tilde{c}_{1,1}, \ldots, \tilde{c}_{1,n})$: The simulator verifies the messages sent in all $n$ executions of the passive OLE in Step 6, verifying correctness against the decomposed randomness. If more than $d/3$ executions are inconsistent with the watchlists yet the adversary is not caught, the simulator sets abort$_1 = 1$. The simulator extracts $\tilde{r}_{1,j}$ when $P_1$ sends its input to the first $F_{\text{OLE}}$ in Step 6. The simulator hands $\tilde{c}_1$ where $c_{1,j} = q_j \cdot r_{1,j} + u_{2,j}$ to $P_1$ to simulate this step. The simulator extracts $(p_j, u_{1,j})$ when $P_1$ sends its input to the second $F_{\text{OLE}}$ in Step 6.

5. Output Reconstruction.

   - $d_2$: The simulator computes $\tilde{d}_2$ from $c_2$ and its shares, and hands it to $P_1$.
   - $d_1$: The simulator receives $d_1 = (d_{1,1}, \ldots, d_{1,n})$ from $P_1$. It verifies that $d_{1,j} = c_{1,j} + p_j \cdot s_{1,j} - u_{1,j}$ for all $j \in [1, n]$. If abort$_1 = 1$ or if the check fails for at least $d/3$ positions, the simulator aborts and outputs whatever $P_1$ outputs.

The simulator completes the simulation and outputs whatever $P_1$ outputs.

We define the event where a malicious $P_1$ deviates from the protocol.

1. $E_1$: In Step 2, $P_1$ sends at least one invalid $RS_{\tilde{q}}^{\tilde{t}}$ codeword to $F_{\text{Coin}}$ where the number of errors is bounded by $d/3$.

2. $E_2$: At least $d/3$ of the OLE instances, or the $d_{1,j}$ values sent in Step 7 or the degree test values, are inconsistent with the watchlists.

We prove that the joint distributions in the hybrid and ideal worlds are computationally indistinguishable by a sequence of hybrid games.
Recall that in $H_3$, instead of using the actual $P_2$’s input to simulate the watchlist messages in Step 4, the simulator samples a random polynomial $\tilde{Q}(\cdot)$ and generates the messages $\tilde{q}_1$. Only after the OT, the simulator extracts $P_1$’s input $\tilde{X}$. The output is determined based on the extracted input of the adversary. Once the simulator obtains the output $\tilde{X} \cap Y$, the simulator needs to recalculate the polynomial $\tilde{Q}(\cdot)$ used in the following steps of the simulation. Now $\tilde{Q}(\cdot) = \prod_{i=1}^{w+t+e-|X \cap Y|} (X - z_i)\Pi_{y_i \in \tilde{X} \cap Y} (X - y_i) \cdot \tilde{T}_2(\cdot)$ where $z_i \notin \tilde{X} \cap Y$ and $\tilde{T}_2(\cdot)$ are random polynomials of degree $(t + e)$, and chosen such that the new $\tilde{Q}(\cdot)$ is consistent with the shares $\tilde{q}_1$, sent through the watchlist channels. Denote $\tilde{Y} = \{z_i\} \cup (\tilde{X} \cap Y)$. On the other hand, in $H_2$ we have $Q(\cdot) = \prod_{i=1}^{w} (X - y_i) \cdot T_2(\cdot)$.

Due to the watchlist mechanism, the adversary sees $t$ evaluations of $Q(\cdot)$ and $\tilde{Q}(\cdot)$. However, as $\Pi_{w+t+e-|X \cap Y|} (X - y_i)$ and $\Pi_{w+t+e-|X \cap Y|} (X - \tilde{y}_i)$ are both masked with random polynomials of degree $t + e$, nothing is leaked about $y_i$ or $\tilde{y}_i$ from observing $t$ evaluations.

In $H_2$, $P_1$ receives the $RS_{F_\eta,n,k,\eta}$ encoding $(t_1, \ldots, t_n)$ of a polynomial $T = P(S_1 + S_2) + Q(R_1 + R_2)$. While in $H_3$, upon extracting $P_1$’s input $\tilde{X}$, the simulator submits $\tilde{X}$ to the ideal functionality and obtains the output $\tilde{X} \cap Y$, so $P_1$ receives $(\tilde{t}_1, \ldots, \tilde{t}_n)$ of $\tilde{T} = \tilde{W} \cdot \Pi_{z_i \in \tilde{X} \cap Y} (X - z_i)$ where $\tilde{W}$ is a random polynomial of degree $2(w + t + e) - |\tilde{X} \cap Y|$ that does not contain any roots of $P_1$ or of $P_2$. According to Kissner and Song [KS05] (Lemma 2), whenever $R = R_1 + R_2$ and $S = S_1 + S_2$ are random polynomials, then the polynomial $T = W \cdot \Pi_{z_i \in \tilde{X} \cap Y} (X - z_i)$ where $W$ is distributed as a random polynomial of degree $2(w + t + e) - |X \cap Y|$. In $H_3$, the simulator obtains the exact $\tilde{X} \cap Y$, however, in $H_2$, $W$ may contain extra roots that are in $\tilde{X} \setminus (\tilde{X} \cap Y)$ or $Y \setminus (\tilde{X} \cap Y)$. $H_2$ and $H_3$ will be indistinguishable if $W$ does not have common roots with the input polynomials of both parties. We claim that the probability that this happens is bounded by $2(w + t + e)/|F|$.  

Lemma 4. The probability that at least one of the values $\{x_1, \ldots, x_t\}$ is a root of a uniformly random polynomial $P(X) = a_0 + \cdots + a_n X^n$ over the field $F$ is bounded by $t/|F|$.  

Proof. It is clear that for any value $x$ and for any combinations of $(a_1, \ldots, a_n)$, there is only one value $a_0$ that makes $P(x) = 0$. So, the probability that $x$ is a root of the random polynomial $P(X)$ is exactly $1/|F|$. Taking the union bound, the probability that at least one value
of the set \(\{x_1, \ldots, x_t\}\) is the root of \(P(X)\) is bounded by \(t/|\mathbb{F}|\).

From Lemma 4, the probability that \(W(\cdot)\) has a common root with \(\overline{P}(\cdot)\) is bounded by \((w + t + e)/|\mathbb{F}|\), and that \(W(\cdot)\) has a common root with \(\Pi_{i=1}^n(X - y_i)\), where \(y_i\) values are \(P_2\)'s input, is bounded by \(w/|\mathbb{F}|\). In overall, the chance that \(H_2 \) and \(H_3\) are different is bounded by \(2(w + t + e)/|\mathbb{F}|\).

Furthermore, we claim that the adversary’s input is well defined. This is because the simulator did not abort either in the degree test and either due to the watchlists checks. This implies that the adversary followed the degree test correctly and provided polynomials that are consistent with the values committed via the watchlists.

We conclude that \(H_2\) and \(H_3\) are statistically close with an error bounded by \(2(w + t + e)/|\mathbb{F}|\). Note that \(H_3\) is identically distributed to the simulation.

This concludes the proof for the first case.

**Simulation for a Corrupted** \(P_2\). The roles of \(P_1\) and \(P_2\) in our two-party PSI protocol are symmetric and thus the simulation and proof are identical.

## D Proof of Theorem 2

We consider two cases. In the first case \(P_0\) is corrupt. In the second case \(P_3\) is not corrupt. Let \(A\) be the set of indices of corrupt parties.

**Simulation For a Corrupted** \(P_0\).

1. **Merkle Tree Commitment.** The simulator acts on behalf of honest parties \(P_1\), running the protocol honestly until Step 6 (in Step 2 it uses random input \(\tilde{Q}_i\) for the honest party \(P_1\)). The simulator samples three random coins, each is used to generate the set of indices \(I_1, I_2, I_3\) of the watch channels in each check. Let \(I = I_1 \cup I_2 \cup I_3\). It then uses the random input and random polynomials to generate the Merkle tree’s root. It stores all honest \(P_i\)'s shares at these indices \((\tilde{q}_{i,j}, j \in I)\).

2. **Input Extraction.** In Step 7, the simulator hands corrupt parties a random coin, which correspond to a list of \(t\) indices used for the degree test, and learns \(t\) shares. The simulator rewind the process until it extracts all the corrupt parties’ input and randomness used in the protocol.

   - \(P_i\)’s input: \((q_{0,i}, r_{0,i}, z_{0,i}, s_{0,j}^1, \ldots, s_{0,j}^{m-1}, u_{0,j}, \ldots, u_{0,j}^{m-1})\) for \(j \in [1, n]\).
   - \(P_i\)’s input \((i \in A \setminus \{P_0\})\): \((q_{i,j}, r_{i,j}, s_{i,j}, z_{i,j}, v_{i,j}, u_{i,j})\) for \(j \in [1, n]\).

   The simulator reconstructs the corresponding polynomials \(Q_0, R_0, Z_0, S_0^0, U_0^0\) and \(Q_i, R_i, Z_i, S_i, U_i\) for \(i \in A \setminus \{P_0\}\). If any of the polynomials \(Q_0, R_0, Z_0, S_0^0, U_0^0\), or \(Q_i, R_i, Z_i, S_i\) has degree higher than \(w + 3t + e\), the simulator sets \(\text{abort}_0 = 1\). Else, it sets \(\text{abort}_0 = 0\).

3. **Synthesize Honest Parties’ Input.** If \(\text{abort}_0 = 0\), the simulator submits the corrupt parties’ input to the ideal functionality, receiving \(X = \bigcap_{i=0}^n X_i\). For each honest party \(P_i\), the simulator uses \(\tilde{X}_i = X \cup Z_i\) where each \(Z_i\) consists of \((w - |X|)\) random values such that \(X \cap Z_i = \emptyset\). It recomputes \(\tilde{Q}_i(\cdot)\) such that \(\tilde{Q}_i(\cdot) = \prod_{z \in \tilde{X}_i} (X - z) \cdot T_i(\cdot)\), where \(\text{deg}(T_i) = 3t + e\). It sets \(\tilde{q}_{i,j} = \tilde{Q}_i(u_j^i)\) for \(j \in I\). This is always possible as \(T_i(\cdot)\) is defined as a random polynomial of degree \(3t + e\). There is always a \(T_i\) that satisfies the above conditions.

   We note that, whenever the parties need to reveal the shares to perform the checks, \(\tilde{q}_{i,j}\) will be opened, and they are always consistent with the committed Merkle tree’s root.

4. **Degree Test.** The simulator runs the degree test on behalf of honest parties. If \(\text{abort}_0 = 1\), it aborts and outputs whatever \(P_0\) outputs.

5. **OLE.** \(\tilde{c}_i = (\tilde{c}_{i,1}, \ldots, \tilde{c}_{i,n})\): The simulator monitors the randomness used in all \(n\) executions of the passive OLE in Step 8, verifying correctness. If more than \(d/3\) executions are inconsistent with the watchlists yet the adversary is not caught, the simulator sets \(\text{abort}_1 = 1\). Specifically, the simulator extracts \(s_{0,j}^i\) when \(P_0\) sends its input to the first \(F_{\text{OLE}}\) in Step 8. The simulator hands \(\tilde{c}_0^i\) where \(\tilde{c}_{0,j}^i = q_{i,j} + s_{0,j} + u_{i,j}\) to \(P_0\) to simulate this step.

   The simulator extracts \((q_0, u_0^i)\) when \(P_0\) sends its input to the second \(F_{\text{OLE}}\) in Step 8. If \(P_0\) uses encodings that are not consistent with the watchlist (that also enforces the use of the same \(q_0\) in all \(F_{\text{OLE}}\) in-vocations), the simulator aborts. The simulator uses the extracted input to \(F_{\text{OLE}}\) to compute \(c_i\) for the honest party \(P_i\) and stores it.

6. **Output Reconstruction.**
   - \(\tilde{d}_i\): The simulator computes \(\tilde{d}_i\) from \(c_i\) and its shares, and hands it to \(P_0\) on behalf of honest parties \(P_i\).
   - The simulator receives \(t = (t_{1,1}, \ldots, t_{1,n})\) from \(P_0\). It verifies that \(t_j = q_{0,j} + \sum_{i=0}^{m-1} r_i + \sum_{i=1}^{m-1} q_{i,j}(s_{0,j} + s_{i,j})\) for \(j \in [1, n]\). If \(\text{abort}_1 = 1\) or if the check fails for at least \(d/3\) positions,
the simulator aborts and outputs whatever the adversary outputs.

The simulator completes the simulation and outputs whatever the adversary outputs.

We define the event where a malicious adversary (including \( P_0 \)) deviates from the protocol.

1. \( E_1 \): In Step 4, at least one of the corrupted parties commits to an invalid RS\( \bar{g}_{S_{n,k,n}} \) codeword where the number of errors is bounded by \( d/3 \).

2. \( E_2 \): At least \( d/3 \) of the OLE instances or the \( d_{1,j} \) values sent in Step 10, or the degree test values are inconsistent with the commitments.

In the full version we prove that the joint distributions in the hybrid and ideal worlds are computationally indistinguishable via a sequence of hybrid games.

**Simulation for Honest \( P_0 \).**

1. **Merkle Tree Commitment.** The simulator acts on behalf of honest parties \( P_i \), running the protocol honestly until Step 6 (in Step 2 it uses random input \( \bar{Q}_i \) for the honest party \( P_i \)). The simulator samples three random coins, each is used to generate the set of indices \( I_1, I_2, I_3 \) of the watch channels in each check. Let \( I = I_1 \cup I_2 \cup I_3 \). It then uses the random input and random polynomials to generate the Merkle tree’s root. It stores all honest \( P_i \)’s shares at these indices \( \bar{Q}_{i,j}, j \in I \).

2. **Input Extraction.** In Step 7, the simulator hands corrupt parties a random coin, which correspond to a list of \( t \) indices used for the degree test, and learns \( t \) shares. The simulator rewinds the process until it extracts all the corrupt parties’ input and randomness used in the protocol. For \( i \in A \), the simulator obtains \( P_i \)’s input \( \bar{P}_i \) of indices \( I_1, I_2, I_3 \) of the watch channels in each check.

The simulator reconstructs the corresponding polynomials \( Q_i, R_i, Z_i, S_i, U_i \). If any of the polynomials \( Q_i, R_i, Z_i, S_i, U_i \) has degree higher than \( w + 3t + e \), the simulator sets \( \text{abort} = 1 \). Else, it sets \( \text{abort} = 0 \).

3. **Synthesize Honest Parties’ Input.** If \( \text{abort} \) is 0, the simulator submits the corrupt parties’ input to the ideal functionality, receiving \( X = \cap_{j=0}^{m-1} X_i \). For each honest party \( P_i \), the simulator uses \( \bar{X}_i = X \cup Z_i \) where each \( Z_i \) consists of \( (w - |X|) \) random values such that \( X \cap Z_i \). It recomputes \( \bar{Q}_i(\cdot) \) such that \( \bar{Q}_i(\cdot) = \Pi_{z \in \bar{X}_i} (X - z) T_i(\cdot) \) such that \( \deg(T_i) = 3t + e \). \( \bar{Q}_i(q^j) = \bar{q}_{i,j} \) for \( j \in I \). This is always possible as \( T_i(\cdot) \) is defined as a random polynomial of degree \( 3t + e \). There is always a \( T_i \) that satisfies the above conditions. Whenever the parties need to reveal the shares to perform the checks, \( \bar{q}_{i,j} \) will be opened, and they are always consistent with the committed Merkle tree’s root.

4. **Degree Test.** The simulator runs the degree test on behalf of honest parties. If \( \text{abort} = 1 \), it aborts and outputs whatever the corrupt parties output.

5. **OLE.** \( \bar{c}_1 = (\bar{c}_{1,1}, \cdots, \bar{c}_{1,n}) \): The simulator monitors the randomness used in all \( n \) executions of the passive OLE in Step 8, verifying correctness. If more than \( d/3 \) executions are inconsistent with the watchlists yet the adversary is not caught, the simulator sets \( \text{abort} = 1 \).

Specifically, the simulator extracts \( q_{1, i}, u_{1, i} \) when \( P_i \) sends its input to the first \( F_{\text{OLE}} \) in Step 8. The simulator extracts \( r_1 \) when \( P_i \) sends its input to the second \( F_{\text{OLE}} \) in Step 8. The simulator hands \( \bar{c}_1 \) where \( \bar{c}_{i,j} = q_{0, j} r_{i,j} + u_{0, j} \) to \( P_i \) to simulate this step.

If \( P_i \) uses encodings that are not consistent with the watchlist, the simulator aborts. The simulator uses the extracted input to the \( F_{\text{OLE}} \) to compute \( c_0^1 \) and \( c_1 \) for honest party \( P_i \) and stores it.

6. **Output Reconstruction \( d \):** For each honest party \( P_i \), the simulator computes \( \bar{d}_i \) from \( c_1 \) and its shares. For each corrupt party \( P_i \), the simulator receives \( \bar{d}_i \) and verifies that \( d_{i,j} = q_{0, j} r_{i,j} + q_{i,j} s_{i,j} + v_{i,j} + (u_{0, j} - u_{i, j}) \) for \( i \in A \) and \( j \in [1, n] \).

If \( \text{abort} = 1 \) or if the check fails for at least \( d/3 \) positions, the simulator aborts and outputs whatever the adversary outputs. Otherwise, the simulator computes \( d \) and sends it to corrupt parties.

The simulator completes the simulation and outputs whatever the adversary outputs.

The arguments to prove the the joint distributions in the hybrid and the ideal world is statistically close is similar to the case of corrupt \( P_0 \). We omit the proof.