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

DOI 10.56553/popets-2022-0073
Received 2021-11-30; revised 2022-03-15; accepted 2022-03-16.

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 input 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: $f_1 = \bigcap_i X_i$, 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-

S. Dov Gordon: George Mason University, E-mail: gordon@gmu.edu
Carmit Hazay: Bar-Ilan University, E-mail: carmit.hazay@biu.ac.il
***Corresponding Author: Phi Hung Le:** George Mason University, E-mail: ple13@gmu.edu

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 [Gol09], 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 [AMZ21] 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. [BNOP21] 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 [BNOP21], 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 $\mathsf{Enc}(r \cdot P(y) + y)$. 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 [AMZ21]. 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 pseudorandom 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 κ 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 times[3] [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 $O(mw\kappa^2)$ where m is the number of parties and κ is the security parameter. Their construction provides an output to one party.

3 When considering communication cost rather than end-to-end running time, it is still possible to outperform OT-based solutions, including those based on garbled circuits [CT10, CLR17, MPR+20], by using a linear number of public key operations.

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

Table 1. A comparison of communication complexity. w is the input size, m is the number of parties, λ, κ are the security parameters, and $t = O(\lambda)$. In all protocols, multiples of mw are the dominant terms. All protocols are secure against malicious adversaries.

Protocol	Communication complexity		Fully secure
	Total	P_0	
[HV17]	$O(mw\kappa \log w + m^2\kappa)$	$O(mw\kappa \log w)$	NO
[GPR+21]	$O(mw(\kappa + \lambda + \log w) + m^2\kappa)$	$O(mw(\kappa + \lambda + \log w))$	NO
[BNOP21]	$O(mw\kappa^2 + mw\kappa \log(\kappa w)$	$O(mw\kappa^2 + mw\kappa \log(\kappa w)$	NO
Ours (P_0 receives output)	$O(mw\kappa + m^2\kappa + mt\kappa \log w)$	$O(mw\kappa + mt\kappa \log w)$	NO
Ours (All receive output)	$O(mw\kappa + m^2 t\kappa \log w)$	$O(mw\kappa + mt \log w)$	YES

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: $R(\cdot) = R_1(\cdot) + R_2(\cdot)$, and $S(\cdot) = S_1(\cdot) + S_2(\cdot)$. 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: $T = Q \cdot R + P \cdot S$.

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 benefit 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 κ 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 $O((m^3 + m^2 w)\kappa$. That is just the cost to set up the watchlists [LOP11]. In order to achieve better performance, we use a *customized* version 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 $O(mw\kappa^2 + mw\kappa \log(\kappa w))$ bits. Because of the κ^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 parallelization. Our implementation currently instantiates the OLE instances with either OT [Gil99] 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 $t_j = q_j(r_{1,j} + r_{2,j}) + p_j(s_{1,j} + s_{2,j})$ where w is the input size, $j \in \{1, \ldots, 2w+1\}$, and $t_j, q_j, p_j, r_{1,j}, r_{2,j}, s_{1,j}, s_{2,j}$ are the evaluations of the above polynomials on public points η_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 [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 $R_1(x) = \frac{\sum_{i=0}^{w} a_i x^i}{\sum_{i=0}^{k} b_i x^i}$. These attacks have shown to violate both privacy and correctness. As opposed to [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. [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 κ. 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 $\mathcal{C}$ (with addition and multiplication gates of fan-in 2), and denote the size of $\mathcal{C}$ by $|\mathcal{C}|$. 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 κ, and a statistical security parameter λ that captures a statistical error of up to $2^{-\lambda}$. We assume that $\lambda \leq \kappa$. We let $\mathcal{F}$ be a multi-party functionality that maps a set of m inputs to an output over some field $\mathbb{F}$ (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^λ and 1^κ. 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 $\mathsf{poly}(\kappa)$ (this effectively means that we only require security when the input length is bounded by *some* polynomial in κ). We denote by $\textsc{real}_{\Pi,\mathcal{A}(z)}(x_1, \ldots, x_m, \kappa, \lambda)$ the output of the honest parties and the adversary $\mathcal{A}$ controlling a subset $I \subset [m]$ of parties in the real execution of Π, where z is the auxiliary input, x_i is P_i's initial input, κ is the computational security parameter, and λ is the statistical security parameter. We denote by $\textsc{ideal}_{\mathcal{F},\mathcal{S}(z)}(x_1, \ldots, x_m, \kappa, \lambda)$ the output of the honest parties and the simulator $\mathcal{S}$ in the ideal model where $\mathcal{F}$ is computed by a trusted party. We stress that in this ideal model, the adversary is given the output first, and then instructs $\mathcal{F}$ whether to send output to the honest parties, or to abort. We refer the reader to Goldreich's textbook for more detail [Gol09]. In some of our protocols the parties have access to ideal model implementations of certain cryptographic primitives such as ideal coin tossing ($\mathcal{F}_{\text{COIN}}$). We denote such executions by $\text{REAL}^{\mathcal{F}_{\text{COIN}}}_{\Pi,\mathcal{A}(z)}(x_1,\ldots,x_m,\kappa,\lambda)$. 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* $\Pi = \langle P_1,\ldots,P_m\rangle$ *is said to* securely compute a functionality $\mathcal{F}$ in the presence of active adversaries *if the parties always have the correct output* $\mathcal{F}(x_1,\ldots,x_m)$ *when neither party is corrupted, and moreover the following security requirement holds. For any probabilistic* $\mathsf{poly}(\kappa)$*-time adversary* $\mathcal{A}$ *controlling a subset* $I \subset [m]$ *of parties in the real world, there exists a probabilistic* $\mathsf{poly}(\kappa)$*-time adversary (simulator)* $\mathcal{S}$ *controlling* I *in the ideal model, such that for every non-uniform* $\mathsf{poly}(\kappa)$*-time distinguisher* $\mathcal{D}$ *there exists a negligible function* $\nu(\cdot)$ *such that the following ensembles are distinguished by* $\mathcal{D}$ *with at most* $\nu(\kappa)+2^{-\lambda}$ *advantage:*
$\{\text{REAL}_{\Pi,\mathcal{A}(z)}(x_1,\ldots,x_m,\kappa,\lambda)\}_{\kappa\in\mathbb{N},\lambda\in\mathbb{N},x_1,\ldots,x_m,z\in\{0,1\}^*}$
and
$\{\text{IDEAL}_{\mathcal{F},\mathcal{S}(z)}(x_1,\ldots,x_m,\kappa,\lambda)\}_{\kappa\in\mathbb{N},\lambda\in\mathbb{N},x_1,\ldots,x_m,z\in\{0,1\}^*}$

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 $\perp$. In this work, we use polynomial encodings to share a *set* of secrets in $\mathbb{F}=\mathbb{GF}(q)$. 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 $\mathsf{Reconst}$**:** 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

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

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 $s_{i_1},\ldots,s_{i_{t+w+1}} \subseteq \mathsf{Share}(\{x_1,\ldots,x_w\})$,

$$\Pr\left[\{x_1,\ldots,x_w\}\subseteq \mathsf{Reconst}\left(s_{i_1},\ldots,s_{i_{t+w+1}}\right)\right]=1$$

Secrecy: For any pair of secret sets x,x', and every two sets of shares $s_{i_1},\ldots,s_{i_{t+w}} \subseteq \mathsf{Share}(x)$ and $s'_{i_1},\ldots,s'_{i_{t+w}} \subseteq \mathsf{Share}(x')$, $\{s_{i_1},\ldots,s_{i_t}\}$ and $\{s'_{i_1},\ldots,s'_{i_t}\}$ 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* $\eta=\{\eta_1,\cdots,\eta_n\}\in\mathbb{F}^n$ *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 x 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 *Com***:** On input $x \in \mathbb{F}$, and security parameter κ, the committing algorithm outputs a pair of values (c, d).

Decommitting Algorithm *Decom***:** Given a commitment c and a decomittment value d, the decommitment scheme 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, $\mathsf{poly}(\kappa)$ time distinguisher $\mathcal{D}$, there exists a negligible function $\nu(\cdot)$ such that $|\Pr[\mathcal{D}(Com(x_1) = 1] - \Pr[\mathcal{D}(Com(x_2) = 1]| \leq \nu(\kappa)$.

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

Additional preliminaries are found in Appendix A.

Functionality $\mathcal{F}_{2\text{PSI}}$

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

Functionality. $\mathcal{F}_{2\text{PSI}}$ communicates with parties P_1, P_2, and adversary $\mathcal{A}$.

- Wait for the input $X = (x_1, ..., x_w)$ and $Y = (y_1, ..., y_w)$ from P_1 and P_2 respectively.
- Wait for the adversary $\mathcal{A}$ to add up to $(t+e)$ more items to the input set of the corrupted party. Let $\widetilde{X}$ and $\widetilde{Y}$ be the input sets after this step (only the corrupt party's input is modified).
- Send the output $\widetilde{X} \cap \widetilde{Y}$ to P_2.
- Wait for abort/continue from P_2. Upon receiving abort from P_2, the functionality sends $\perp$ to P_1. Else, the functionality sends $\widetilde{X} \cap \widetilde{Y}$ to P_1.

Fig. 1. Fully Secure Two-Party PSI Ideal Functionality.

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 ω 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_i 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_1'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 $p_i s_{2,i} + u_{1,i}$.

Π_{2PSI} **MPC-in-the-Head Based Two-Party PSI**

Setup: P_1 and P_2 agree on a common finite field $\mathbb{F}_q$ and ω is an n^{th} root of unity of the field (namely, $n|(q-1)$). Let $\eta = \{1, \omega, ..., \omega^{n-1}\}$, 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 $P(x) = [\Pi_{j=1}^{w}(x-x_j)] \cdot T_1(x)$. 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.**[a]
   – P_1 samples random polynomials $Z_1(\cdot)$, $R_1(\cdot)$, $S_1(\cdot)$ and computes the $RS_{\mathbb{F}_q,n,k,\eta}$ encodings: $\mathbf{z}_1 = Z_1(\eta)$, $\mathbf{r}_1 = R_1(\eta)$, $\mathbf{s}_1 = S_1(\eta)$. P_2 samples $Z_2(\cdot)$, $R_2(\cdot)$, $S_2(\cdot)$ and computes $\mathbf{z}_2 = Z_2(\eta)$, $\mathbf{r}_2 = R_2(\eta)$, $\mathbf{s}_2 = S_2(\eta)$. All polynomials have degree at most $(w+t+e)$ and are chosen over the finite field $\mathbb{F}_q$.
   – P_i samples a random polynomial $U_i(\cdot)$ of degree $2k$ and computes $\mathbf{u_i} = U_i(\eta)$.
3. **Coin Tossing.** The parties call $\mathcal{F}_{\text{ComCoin}}$ twice (Functionality 9), each receiving n random strings and decommitments 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: $(\mathsf{com}_{i,1}, \ldots, \mathsf{com}_{i,n})$. P_i will use $\sigma_{i,j}$ as its randomness for the jth OLE execution.
4. **Watchlist Channels Setup via t-out-of-n OT.** The parties call $\mathcal{F}_{\text{OT}}^{t:n}$ (Functionality 5). P_1 receives t tuples $(q_j, u_{2,j}, r_{2,j}, s_{2,j}, z_{2,j}, \tau_{2,j})$ from P_2. They repeat the process with reversed roles, where P_2 receives t tuples $(p_j, u_{1,j}, r_{1,j}, s_{1,j}, z_{1,j}, \tau_{1,j})$ 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 $w+t+e$.
   – The parties call $\mathcal{F}_{\text{COIN}}$ (Functionality 8) to sample random public values $\{\alpha_1, ..., \alpha_8\}$.
   – P_1 sends $\mathbf{a}$ where $a_j \leftarrow \alpha_1 \cdot z_{1,j} + \alpha_2 \cdot r_{1,j} + \alpha_3 \cdot s_{1,j} + \alpha_4 \cdot p_j$ to P_2, $j \in [1, n]$.
   – P_2 sends $\mathbf{b}$ where $b_j \leftarrow \alpha_5 \cdot z_{2,j} + \alpha_6 \cdot r_{2,j} + \alpha_7 \cdot s_{2,j} + \alpha_8 \cdot q_j$ to P_1, $j \in [1, n]$.
   – The parties verify that $\mathbf{a}$ and $\mathbf{b}$ are valid $RS_{\mathbb{F}_q,n,k,\eta}$ codewords. They also check the correctness of these shares against their watched channels.
      – P_1: $\forall j \in I_1 : b_j = \alpha_5 \cdot z_{2,j} + \alpha_6 \cdot r_{2,j} + \alpha_7 \cdot s_{2,j} + \alpha_8 \cdot q_j$.
      – P_2: $\forall j \in I_2 : a_j = \alpha_1 \cdot z_{1,j} + \alpha_2 \cdot r_{1,j} + \alpha_3 \cdot s_{1,j} + \alpha_4 \cdot p_j$
6. **OLE.** The parties make a sequence of calls to $\mathcal{F}_{\text{OLE}}$ (Functionality 7):
   – P_1 provides $\mathbf{r}_1$ whereas P_2 provides $(\mathbf{q}, \mathbf{u_2})$ to $\mathcal{F}_{\text{OLE}}$. P_1 obtains $\mathbf{c_1} = (c_{1,1}, \cdots, c_{1,n})$ where $c_{1,j} = q_j \cdot r_{1,j} + u_{2,j}$.
   – P_1 provides $(\mathbf{p}, \mathbf{u_1})$ whereas P_2 provides $\mathbf{s_2}$ to $\mathcal{F}_{\text{OLE}}$. P_2 obtains $\mathbf{c_2} = (c_{2,1}, \cdots, c_{2,n})$ where $c_{2,j} = p_j \cdot s_{2,j} + u_{1,j}$.
   – P_i verifies that $\mathbf{c_i}$ is a valid $RS_{\mathbb{F}_q,n,2k,\eta}$ codeword.
   – P_1 verifies against the watchlist that for $j \in I_1 : c_{1,j} = q_j \cdot r_{1,j} + u_{1,j}$, and that $\tau_{2,j}$ is consistent with the OLE execution for those inputs of P_2.
   – P_2 verifies against the watchlist that for $j \in I_2 : c_{2,j} = p_j \cdot s_{2,j} + u_{2,j}$, and that $\tau_{1,j}$ is consistent with the OLE execution for those inputs of P_1.
7. **Output Reconstruction.**
   (a) P_1 computes $\mathbf{d_1}$ where $d_{1,j} = c_{1,j} + p_j \cdot s_{1,j} - u_{1,j}$ and sends $\mathbf{d_1}$ to P_2.
   (b) P_2 computes $\mathbf{d_2}$ where $d_{2,j} = c_{2,j} + q_j \cdot r_{2,j} - u_{2,j}$ and sends $\mathbf{d_2}$ to P_1.
   (c) The parties verify that $\mathbf{d_i}$ is a valid $RS_{\mathbb{F}_q,n,2k,\eta}$ codeword. They also verify against the watchlist that
      – For $j \in I_1 : d_{2,j} = c_{2,j} + q_j \cdot r_{2,j} - u_{2,j}$.
      – For $j \in I_2 : d_{1,j} = c_{1,j} + p_j \cdot s_{1,j} - u_{1,j}$.
   (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 (ω^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 $(p_i s_{2,i} + u_{1,i}) + (q_i r_{2,i} - u_{2,i})$. P_1 computes and sends $(p_i s_{1,i} - u_{1,i}) + (q_i r_{1,i} + u_{2,i})$. Summing these together, they each learn $p_i s_i + q_i r_i$, where $s_i = s_{1,i} + s_{2,i}$ is the evaluation of random, private polynomial S at the ith public index (and r_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 $\mathcal{F}$ with active security, while making black-box use of the following two ingredients: (1) an active MPC protocol which realizes $\mathcal{F}$ 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 coin-flipping protocol that provides randomness for the OLE execution to one party, and a commitment to that randomness to the other party. The decommitment 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 $2n/w = 2(2(w+t+e)+1)/w = 4+(4t+4e+2)/w$ where w is the input size and t, e are parameters associated with w such that $(1-e/n)^t < 2^{-\lambda}$. As w increases, $(4t+4e+2)/w \to 0$.

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 [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 $\Pi_{2\text{PSI}}$ (cf. Figure 2) securely computes functionality $\mathcal{F}_{2\text{PSI}}$ (cf. Figure 1) with two parties in the $\{F_{\text{ComCoin}}, \mathcal{F}_{\text{OT}}^{t:n}, \mathcal{F}_{\text{COIN}}, \mathcal{F}_{\text{OLE}}\}$-hybrid, tolerating static adversaries with a statistical error of $d/|\mathbb{F}| + (1 - e/n)^t + 2(w+t+e)/|\mathbb{F}|$ where $d = n-k+1$ is the distance of the underlying code.*

The proof can be found in Appendix C.

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

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

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

- 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 $\mathcal{A}$ to add as many as $((m-1)t + e)$ additional items to the input set of the corrupted parties. Let $\widetilde{X}^{(i)}$ be the modified input set of party P_i.
- Send output $S \leftarrow \bigcap_{i=0}^{m-1} \widetilde{X}^{(i)}$ 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.

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 [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_i. For each pair (P_0, P_i), the central party P_0 learns an $RS_{\mathbb{F}_q,n,2k,\eta}$ encoding of the polynomial $T_i = Q_0 \cdot R_i + Q_i(S_0^i + S_i) + V_i$ 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_i, respectively, and V_i is the masked polynomial used to hide the intersection between P_0 and P_i. 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 $RS_{\mathbb{F}_q,n,2k,\eta}$ 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 [LOP11] requires $O(m^3 + m^2 n)$ bits where $n = O(w+mt+e)$. To set up the watch channels, each party P_i executes the multi-sender t-out-of-n OT protocol, called $\mathcal{F}_{\mathbf{mOT}}^{\mathbf{t:n}}$ (see Figure 6), which allows P_i to watch all other parties at the same t channels of its choice. The authors proposed an instantiation for $\mathcal{F}_{\mathbf{mOT}}^{\mathbf{t:n}}$ based on DDH assumption, where each channel requires $O(n) = O(w + mt + e)$ exponentiations 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.

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 using a Merkle tree [Mer87]. They broadcast only the root of each tree, followed by a hash of the received roots to verify consistency.

The cost for all parties to commit to their shares is $O(m^2\kappa)$, and the cost to perform one check is $O(m^2 t\kappa \log n)$. Note that we need to pick t and e such that $(1 - e/n)^t < 2^{-\lambda}$ where $n > 2(w + t + e)$. When the input size w is large, we can choose $t = 5\lambda$ then $e = n(1 - \sqrt[5]{1/2}) = 2(1 - \sqrt[5]{1/2})(w + t + e) \Rightarrow e = 0.35w + 1.76\lambda$. Now $t = O(\lambda)$ (typically $\lambda = 40$) and can be dropped from our asymptotic cost. The total communication for our watchlist set up and watchlist verification is $O(m^2\kappa \log n)$ if one check is performed. The computational cost is the cost to generate the Merkle tree, and to reveal and verify t servers. For our multi-party PSI protocol, we need just three checks (Figure 4). It is extremely cheap compared to the cost of the multi-party t-out-of-n OT used in the IPS compiler.

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.

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 $O((m^2+nm)\kappa)$. 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 $O((m^2 + nm)\kappa)$. 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 $O(t\kappa \log n)$ 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.

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 Π_{MPSI} (cf. Figure 4) securely computes functionality $\mathcal{F}_{\text{MPSI}}$ (cf. Figure 3) with m parties in the $\{\mathcal{F}_{\text{COIN}}, \mathcal{F}_{\text{OLE}}\}$-hybrid, tolerating static adversaries with a statistical error of $d/|\mathbb{F}| + (1-e/n)^t + m(w+3t+e)/|\mathbb{F}|$ 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 $\mathbb{F}$ be such that our statistical error $(1 - e/n)^t + d/|\mathbb{F}| + (w + t + e)/|\mathbb{F}|$ is bound by $2^{-\lambda}$ (where $n = 2(w + t + e) + 1$) and λ 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 $t + e \leq t_0 + e_0 < 3 \cdot \sqrt{\lambda \ln 2 \cdot (2w)}$.

Π_{MPSI} **MPC-in-the-Head Based MPSI**

1. **Setup.** Parties $P_0, \ldots, P_{m-1}$ agree on a common finite field $\mathbb{F}_q$ and ω is an n^{th} root of unity of the field (namely, $n|(q-1)$). Let $\boldsymbol{\eta} = \{1, \omega, \ldots, \omega^{n-1}\}$, 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 $(1 - e/n)^t \le 2^{-\lambda}$.
2. **Input Sharing Phase.** Each party P_i has an input $X_i = \{x_1^i, ..., x_w^i\}$. P_i samples a random polynomial $T_i(x)$ of degree $t + e$, computes $Q_i(x) = T_i(x)\Pi_{j=1}^{w}(x - x_j^i)$ and the $RS_{\mathbb{F}_q,n,k,\boldsymbol{\eta}}$ encoding $\mathbf{q_i} = Q_i(\boldsymbol{\eta})$.
3. **Sample Random Masked Polynomials.** For each pair (i, j) where $1 \le i < j \le (m-1)$, P_i and P_j call $\mathcal{F}_{\text{COIN}}$ (Functionality 8) to sample a common seed $seed_{ij}$. Let $V_{ij} \leftarrow PRG(seed_{ij})$ 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 $V_i = \sum_{1 \le j \ne i \le m-1} V_{ij}$ and computes the $RS_{\mathbb{F}_q,n,2k,\boldsymbol{\eta}}$ encoding $\mathbf{v}_i = V_i(\boldsymbol{\eta})$.
4. **Random Polynomials Sampling.**
   - P_0 samples random polynomials $R_0(\cdot)$, $Z_0(\cdot)$, and $S_0^i(\cdot)$ 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 finite field $\mathbb{F}_q$. P_0 computes the $RS_{\mathbb{F}_q,n,k,\boldsymbol{\eta}}$ encodings: $\mathbf{z_0^i} = Z_0^i(\boldsymbol{\eta})$, $\mathbf{r_0^i} = R_0^i(\boldsymbol{\eta})$, and $\mathbf{s_0^i} = S_0^i(\boldsymbol{\eta})$. P_i $\mathbf{z_i} = Z_i(\boldsymbol{\eta})$, $\mathbf{r_i} = R_i(\boldsymbol{\eta})$, and $\mathbf{s_i} = S_i(\boldsymbol{\eta})$.
   - For $i \in [1, m-1]$, P_i samples random $seed_i$ and polynomial $U_i(\cdot) \leftarrow PRG(seed_i)$ of degree $2k$. They compute the encoding $\mathbf{u_i} = U_i(\boldsymbol{\eta})$. P_i broadcasts the $Com(seed_i)$ to all other parties.
5. **Coin Tossing.** Each pair (P_0, P_i) call $\mathcal{F}_{\text{ComCoin}}$ twice (Functionality 9), each receiving n random strings and decommitments for those strings. The random values are used for the OLE invocations.
6. **Watchlist Channels Commitment.** P_0 commits its shares $(q_{0,j}, r_{0,j}, z_{0,j}, s_{0,j}^1, \ldots, s_{0,j}^{m-1})$ to all parties using Merkle tree. P_i commits its shares $(q_{i,j}, r_{i,j}, s_{i,j}, z_{i,j}, v_{i,j}, u_{i,j})$ 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 $\mathcal{F}_{\text{COIN}}$ to sample random public values $\{\alpha_1, \alpha_2, \alpha_3, \alpha_4^i, ..., \alpha_8^i\}$ together $(1 \le i \le m-1)$.
   - P_i, $1 \le i \le m-1$, computes $\mathbf{b_i}$ where $b_{i,j} \leftarrow \alpha_5^i \cdot z_{i,j} + \alpha_6^i \cdot r_{i,j} + \alpha_7^i \cdot s_{i,j} + \alpha_8^i \cdot q_{i,j}$, then sends it to P_0.
   - P_0 computes $\mathbf{b_0}$ where $b_{0,j} \leftarrow \alpha_1 \cdot z_{0,j} + \alpha_2 \cdot r_{0,j} + \alpha_3 \cdot q_{0,j} + \sum_{i=1}^{m-1} \alpha_4^i \cdot s_{0,j}^i$, $\mathbf{b} = \sum_{k=0}^{m-1} \mathbf{b}_i$, then broadcasts it to all other parties.
   - **Select watchlist channels for first check.** All parties make a call to $\mathcal{F}_{\text{COIN}}$ 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** $Q_i(\cdot) \ne 0$. All parties verify that $q_{i,j} \ne 0$ for $j \in I$.
   - **Degree test.** All parties verify that $\mathbf{b}$ is a valid $RS_{\mathbb{F}_q,n,k,\boldsymbol{\eta}}$ code word and for every $j \in I_1$: $b_j = \sum_{i=1}^{m} [\alpha_1^i \cdot z_{0,j} + \alpha_2^i \cdot r_{0,j} + \alpha_3^i \cdot s_{0,j}^i + \alpha_4^i \cdot q_{0,j} + \alpha_5^i \cdot z_{i,j} + \alpha_6^i \cdot r_{i,j} + \alpha_7^i \cdot s_{i,j} + \alpha_8^i \cdot q_{i,j}]$
8. **OLE.** Each pair (P_0, P_i) makes a sequence of calls to $\mathcal{F}_{\text{OLE}}$ on inputs. P_0 provides $\mathbf{s}_0^i$ whereas P_i provides $(\mathbf{q_i}, \mathbf{u_i})$ to $\mathcal{F}_{\text{OLE}}$. P_0 obtains $\mathbf{c_0^i} = (c_{0,1}^i, \ldots, c_{0,n}^i)$ where $c_{0,j}^i = q_{i,j} \cdot s_{0,j}^i + u_{i,j}$. P_0 provides $(\mathbf{q_0}, \mathbf{u_0^i})$ whereas P_i provides $\mathbf{r_i}$ to $\mathcal{F}_{\text{OLE}}$. P_i obtains $\mathbf{c_i} = (c_{i,1}, \ldots, c_{i,n})$ where $c_{i,j} = q_{0,j} \cdot r_{i,j} + u_{0,j}^i$. For one-sided output: (P_0, P_i) executes only one OLE, P_0 provides $\mathbf{q_0}$, P_i provides $(\mathbf{r_i}, \mathbf{q}_i\mathbf{s}_i + \mathbf{v_i})$, P_0 learns $\mathbf{f_0^i} = \mathbf{q_0 r_i} + \mathbf{q_i s}_i + \mathbf{v_i}$.
9. **Verify OLE (Second Check).**
   - **Select watchlist channels for OLE verification.** All parties make a call to $\mathcal{F}_{\text{COIN}}$ to sample a common random coin, and use that coin to sample t indices $I_2 = \{i_{2,1}, \ldots, i_{2,t}\}$. The parties reveal all the shares and randomnesses used at these indices to all other parties.
   - **Verify** $\mathcal{F}_{\text{OLE}}$. For each pair (P_0, P_i)
     - P_0 verifies that $\mathbf{c_0^i}$ is a valid $RS_{\mathbb{F}_q,n,2k,\boldsymbol{\eta}}$ code word and for $j \in I_2$: $c_{0,j}^i = q_{i,j} \cdot s_{0,j}^i + u_{i,j}$.
     - P_i verifies that $\mathbf{c_i}$ is a valid $RS_{\mathbb{F}_q,n,2k,\boldsymbol{\eta}}$ code word and for $j \in I_2$: $c_{i,j} = q_{0,j} \cdot r_{i,j} + u_{0,j}^i$.

   For one-sided output: (P_0, P_i) verify the execution of only one OLE that computes $\mathbf{f_0^i}$.
10. **Output Aggregation and Verification.**
    - P_0 computes $\mathbf{d_0} = \mathbf{q_0} \cdot \mathbf{r_0} + \sum_{i=1}^{m-1}(\mathbf{c_0^i} - \mathbf{u_0^i}) = \mathbf{q_0} \cdot \mathbf{r_0} + \sum_{i=1}^{m-1} \mathbf{q_i} \cdot \mathbf{s_0^i} + \sum_{i=1}^{m-1}(\mathbf{u_i} - \mathbf{u_0^i})$.
    - P_i computes $\mathbf{d_i} = \mathbf{c_i} + \mathbf{q_i} \cdot \mathbf{s_i} - \mathbf{u_i} + \mathbf{v_i} = \mathbf{q_0} \cdot \mathbf{r_i} + \mathbf{q_i} \cdot \mathbf{s_i} + \mathbf{v_i} + (\mathbf{u_0^i} - \mathbf{u_i})$.
    - P_i sends $\mathbf{d_i}$ to P_0. P_0 computes $\mathbf{t} = \sum_{i=0}^{m-1} \mathbf{d_i}$ and broadcasts it to all P_i.
    - **Select watchlist channels for third check.** All parties make a call to $\mathcal{F}_{\text{COIN}}$ to sample a common random coin, and use that coin to sample t indices $I_3 = \{i_{3,1}, \ldots, i_{3,t}\}$. The parties reveal all the shares used at these indices to all other parties.
    - P_0 verifies that $d_{i,j} = q_{0,j} \cdot r_{i,j} + q_{i,j} \cdot s_{i,j} + v_{i,j} + (u_{0,j}^i - u_{i,j})$ for $i \in [1, m-1]$ and $j \in I_3$. P_i verifies that $t_j = q_{0,j} \cdot \sum_{i=0}^{m-1} r_i + \sum_{i=1}^{m-1} q_{i,j}(s_{0,j}^i + s_{i,j})$ for $j \in I_3$.
    - Each P_i reconstructs the polynomial $T(\cdot)$ from the points (ω^j, t_j) and outputs the intersection $S = \{x \in X_i | T(x) = 0\}$.

    For one-sided output: P_0 computes $\mathbf{t} = \mathbf{q}_0 \cdot \mathbf{r}_0 + \sum_{i=1}^{m-1} \mathbf{f_0^i}$, reconstructs $T(\cdot)$, and computes S.

Fig. 4. Fully Secure Active Multi-Party PSI Protocol.

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 setting: the Reed-Solomon shares are partitioned into groups of N each, each consumes one randomness. The parameters for MPC-in-the-head are set based on the number of groups, $n' = n/N$. The probability that an adversary cheats without getting caught when executing $\mathcal{F}_{\text{OLE}}$ becomes $(1 - e/n')^t$. Let $N = 2^{12}$ and $w = 2^{20}$, then $n' = 256$ and $t + e \approx 360$. In order to verify one batched OLE, all N input shares in that batch will be revealed. As an adversary could learn or corrupt up to $(t+e)$ OLE executions, we need to multiply a random polynomial of degree $N \cdot (t+e)$ with the input polynomials. Our protocol slackness is therefore $N(t+e)/w \times 100\% = 140\%$. Our slackness will be smaller when the input size increases.

Communication Complexity. The overall communication cost of our two-party PSI protocol is linear in the inputs sizes:

$$\underbrace{2 \cdot \mathrm{CC}_{\text{t-out-of-n OT}}}_{\text{watchlists setup}} + \underbrace{2 \cdot n \cdot \mathrm{CC}_{\mathrm{OLE}}}_{\text{passive OLE}} + \underbrace{6n \cdot \kappa}_{\text{coin toss}} + \underbrace{10 \cdot n \cdot \log|\mathbb{F}|}_{\text{watchlist comm.}} + \underbrace{n \cdot \log|\mathbb{F}|}_{\text{degree test}}$$

where $\mathrm{CC}_{\text{t-out-of-n OT}}$ and $\mathrm{CC}_{\mathrm{OLE}}$ are the communication complexities of the underlying OT and OLE protocols, respectively. As $n = O(w)$, the overall communication complexity is $O(w(\kappa + \log|\mathbb{F}|))$ 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 $q > N \cdot |\mathbb{F}|^2$ 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

$$\underbrace{O(n/N \cdot (N \cdot \log N))}_{\text{input encodings}} + \underbrace{O(n/N \cdot (N \cdot \log N))}_{\text{passive OLE}} + \underbrace{O(n)}_{\text{degree test}} + \underbrace{O(n \cdot \log n))}_{\text{input sharing}} + \underbrace{O(n)}_{\text{compute t}} + \underbrace{O(w \cdot \log^2 w))}_{\text{output reconstruction}}$$

Overall, our asymptotic computational complexity is $O(w \log^2 w)$ 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.

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

$$\underbrace{m \cdot \kappa}_{\text{watchlists commit}} + \underbrace{mn \cdot \mathrm{CC}_{\mathrm{OLE}}}_{\text{passive OLE}} + \underbrace{6mn \cdot \kappa}_{\text{coin toss}} + \underbrace{mt \log n \cdot \log|\mathbb{F}|}_{\text{watchlist comm.}} + \underbrace{\cdot mn \cdot \log|\mathbb{F}|}_{\text{degree test}}$$

$\mathrm{CC}_{\mathrm{OLE}}$ is the communication complexities of the underlying OLE protocols. Next, the communication cost for each non-central party is

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 $\mathbb{F}_p$. Our OLE is instantiated with OT for $w \in \{2^8, 2^{12}, 2^{16}\}$ and with Ring-LWE for $w \in \{2^{18}, 2^{20}\}$.

	$w = 2^8$		$w = 2^{12}$		$w = 2^{16}$		$w = 2^{18}$		$w = 2^{20}$	
Parties	**Runtime**	**Comm**	**Runtime**	**Comm**	**Runtime**	**Comm**	**Runtime**	**Comm**	**Runtime**	**Comm**
2	0.16	4.8	0.83	40.2	8.10	230	12.28	323	26.23	654
4	0.27	14.5	1.55	120	12.56	689	18.29	970	35.42	1963
8	0.19	33.8	3.00	282	20.50	1606	31.43	2261	63.00	4582
16	1.05	72.5	6.40	602	40.53	3442	58.33	4845	117.4	9818
32	2.26	150	12.7	1245	77.16	7114	116.8	10013	-	-

$$\underbrace{m \cdot \kappa}_{\text{watchlists commit}} + \underbrace{2n \cdot \mathrm{CC}_{\mathrm{OLE}}}_{\text{passive OLE}} + \underbrace{6n \cdot \kappa}_{\text{coin toss}} + \underbrace{mt \log n \cdot \log|\mathbb{F}|}_{\text{watchlist comm.}} + \underbrace{n \cdot \log|\mathbb{F}|}_{\text{degree test}}$$

Computational Complexity. In terms of computational cost, the central party has to make $O(mw)$ 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

$$\underbrace{O(n/N \cdot (N \cdot \log N))}_{\text{input encodings}} + \underbrace{O(m \cdot n/N \cdot (N \cdot \log N))}_{\text{passive OLE}} + \underbrace{O(mn)}_{\text{degree test}} + \underbrace{O(n \cdot \log n))}_{\text{input sharing}} + \underbrace{O(n)}_{\text{compute t}} + \underbrace{O(w \cdot \log^2 w))}_{\text{output reconstruction}}$$

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

$$\underbrace{O(n/N \cdot (N \cdot \log N))}_{\text{input encodings}} + \underbrace{O(n/N \cdot (N \cdot \log N))}_{\text{passive OLE}} + \underbrace{O(mn)}_{\text{degree test}} + \underbrace{O(n \cdot \log n))}_{\text{input sharing}} + \underbrace{O(n)}_{\text{compute t}} + \underbrace{O(w \cdot \log^2 w))}_{\text{output reconstruction}}$$

We can see that the heaviest work is done by the central party. The actual runtime of the central party is dominated by the cost to compute the OLE, which is $O(mn \log N)$ field multiplications. Compared with the two-party PSI, the main extra cost that the central party has to handle is the cost to perform the OLE with all the other parties.

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.

Table 3. One-sided output MPSI: Runtime (in seconds) and communication cost (in MB). Input items are represented by elements of a 64-bit prime field $\mathbb{F}_p$. Ours uses single thread, while [KMP+17] and [BNOP21] both use multi-threading. [BNOP21] uses 32 threads on a 32-core machine for their central party, [KMP+17] users $(m-1)$ threads for all parties where m is the number of parties. In these experiment, we consider an adversary corrupting at most $(m-1)$ parties. The numbers for [KMP+17] and [BNOP21] are taken directly from their paper. Our OLE is instantiated with OT for $w \in \{2^8, 2^{12}, 2^{16}\}$ and with Ring-LWE for $w \in \{2^{18}, 2^{20}\}$. (SH: semi-honest, M: malicious secure)

		$w=2^8$		$w=2^{12}$		$w=2^{16}$		$w=2^{18}$		$w=2^{20}$	
Protocols	**Parties**	**Runtime**	**Comm**	**Runtime**	**Comm**	**Runtime**	**Comm**	**Runtime**	**Comm**	**Runtime**	**Comm**
[KMP+17] (SH)	4	-	-	0.34	4.9	3.16	78	-	-	52.25	1402
	15	-	-	1.85	23	20.61	363	-	-	304.36	6547
[BNOP21] (M)	4	0.20	-	0.55	-	6.62	-	27.09	-	128.25	-
	8	0.25	-	0.66	-	7.62	-	30.82	-	143.20	-
	16	0.37	-	0.91	-	13.18	-	57.33	-	-	- -
	32	0.80	-	1.60	-	21.54	-	85.37	-	-	-
Ours (M)	4	0.15	3.67	0.64	30.5	8.52	416	16.34	684	33.26	1386
	8	0.22	8.55	1.10	71.1	14.04	972	24.77	1596	51.59	3234
	16	0.37	18.3	2.05	152	24.36	2430	45.41	3420	96.72	6931
	32	0.70	37.9	4.24	315	53.33	4305	86.10	7068	-	-

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× faster. (See Table 3). Asymptotically, their protocol also requires much higher communication complexity than ours, namely, $O(mw\kappa^2 + mw\kappa \log(\kappa w))$ 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 × 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× 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.

Acknowledgments

This first author was supported by DARPA and SPAWAR under contract N66001-15-C-4070, and by NSF Grants CNS-1942575 and CNS-1955264. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. The second author was supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office, and by ISF grant No. 1316/18.

References

[AAB+19] Abdelrahaman Aly, Tomer Ashur, Eli Ben-Sasson, Siemen Dhooghe, and Alan Szepieniec. Efficient symmetric primitives for advanced cryptographic protocols (A marvellous contribution). *IACR Cryptol. ePrint Arch.*, page 426, 2019.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *CCS*, pages 2087–2104, 2017.

[AMZ21] Aydin Abadi, Steven J. Murdoch, and Thomas Zacharias. Polynomial representation is tricky: Maliciously secure private set intersection revisited. Cryptology ePrint Archive, Report 2021/1009, 2021. https://ia.cr/2021/1009.

[BCS19] Carsten Baum, Daniele Cozzo, and Nigel P. Smart. Using topgear in overdrive: A more efficient zkpok for SPDZ. In *SAC*, pages 274–302, 2019.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

[BNOP21] Aner Ben-Efraim, Olga Nissenbaum, Eran Omri, and Anat Paskin-Cherniavsky. Psimple: Practical multiparty maliciously-secure private set intersection. *IACR Cryptol. ePrint Arch.*, 2021:122, 2021.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[CHI+21] Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, Abhi Shelat, Muthu Venkitasubramaniam, and Ruihan Wang. Diogenes: Lightweight scalable RSA modulus generation with a dishonest majority. In *IEEE SP*, 2021.

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

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

[CT10] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In Radu Sion, editor, *Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, Spain, January 25-28, 2010, Revised Selected Papers*, volume 6052 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 2010.

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

[DMRY09] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In *ACNS*, pages 125–142, 2009.

[dSGOT21] Cyprien Delpech de Saint Guilhem, Emmanuela Orsini, and Titouan Tanguy. Limbo: Efficient zero-knowledge mpcith-based arguments. *IACR Cryptol. ePrint Arch.*, 2021:215, 2021.

[EKR18] David Evans, Vladimir Kolesnikov, and Mike Rosulek. A pragmatic introduction to secure multi-party computation. *Foundations and TrendsÂ® in Privacy and Security*, 2(2-3):70–246, 2018.

[FIPR05] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, pages 303–324, 2005.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, pages 1–19, 2004.

[Gil99] Niv Gilboa. Two party RSA key generation. In *CRYPTO*, pages 116–129, 1999.

[GN19] Satrajit Ghosh and Tobias Nilges. An algebraic approach to maliciously secure private set intersection. In *EUROCRYPT*, pages 154–185, 2019.

[Gol09] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*, volume 2. Cambridge University Press, 2009.

[GPR+21] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In *CRYPTO*, pages 395–425, 2021.

[Haz15] Carmit Hazay. Oblivious polynomial evaluation and secure set-intersection from algebraic prfs. In *TCC*, pages 90–120, 2015.

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

[HIMV19] Carmit Hazay, Yuval Ishai, Antonio Marcedone, and Muthuramakrishnan Venkitasubramaniam. Leviosa: Lightweight secure arithmetic computation. In *CCS*, pages 327–344, 2019.

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

[HLOI16] Brett Hemenway, Steve Lu, Rafail Ostrovsky, and William Welser IV. High-precision secure computation of satellite collision probabilities. In *SCN*, pages 169–187, 2016.

[HN10] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. In *PKC*, pages 312–331, 2010.
[HT10] Carmit Hazay and Tomas Toft. Computationally secure pattern matching in the presence of malicious adversaries. In *ASIACRYPT*, pages 195–212, 2010.
[HV17] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Scalable multi-party private set-intersection. In *PKC*, pages 175–203, 2017.
[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, pages 145–161, 2003.
[IPS08] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.
[JL09] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *TCC*, pages 577–594, 2009.
[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *CCS*, pages 818–829, 2016.
[KMP+17] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *CCS*, pages 1257–1272, 2017.
[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In *EUROCRYPT*, pages 158–189, 2018.
[KS05] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *CRYPTO*, pages 241–257, 2005.
[LOP11] Yehuda Lindell, Eli Oxman, and Benny Pinkas. The IPS compiler: Optimizations, variants and concrete efficiency. In *CRYPTO*, pages 259–276, 2011.
[Mer87] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, pages 369–378, 1987.
[MPR+20] Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. Two-sided malicious security for private intersection-sum with cardinality. In *CRYPTO*, pages 3–33, 2020.
[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *FOCS*, pages 458–467, 1997.
[PRTY20] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXoS: Fast, malicious private set intersection. In *EUROCRYPT*, pages 739–767, 2020.
[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX*, pages 515–530, 2015.
[PSTY19] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In *EUROCRYPT*, pages 122–153, 2019.
[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *USENIX*, pages 797–812, 2014.
[RR17] Peter Rindal and Mike Rosulek. Malicious-secure private set intersection via dual execution. In *CCS*, pages 1229–1242, 2017.
[RS21] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: fast OPRF and circuit-psi from vector-ole. *IACR Cryptol. ePrint Arch.*, 2021:266, 2021.
[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *CCS*, pages 21–37, 2017.
[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 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.

Functionality $\mathcal{F}_{\mathrm{OT}}^{t:n}$

Functionality $\mathcal{F}_{\mathrm{OT}}^{t:n}$ communicates with sender S and receiver R, and adversary $\mathcal{A}$.

1. Upon receiving input $(sid, v_1, \ldots, v_n)$ from S where $v_i \in \{0,1\}^\kappa$ for all $i \in [n]$, record $(sid, v_1, \ldots, v_n)$.
2. Upon receiving $(sid, u_1, \ldots, u_t)$ from R where $u_i \in \{0,1\}^{\log n}$ for all $i \in [t]$, send $(v_{u_1}, \ldots v_{u_t})$ to R. Otherwise, abort.

Fig. 5. The oblivious transfer functionality.

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 $\mathcal{F}^{\mathbf{t:n}}_{\mathbf{mOT}}$

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

1. Upon receiving input $(sid, v^i_1, \cdots, v^i_n)$ from S_i where $i \in [m]$ and $v_j \in \{0,1\}^\kappa$ for all $j \in [n]$, record $(sid, v^i_1, \cdots, v^i_n)$.
2. Upon receiving $(sid, u_1, \cdots, u_t)$ from R where $u_i \in \{0,1\}^{\log n}$ for all $i \in [t]$, send $(v^i_{u_1}, \cdots, v^i_{u_t})$ 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 $\mathcal{F}_{\mathrm{OLE}}$

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

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 use a standard coin tossing functionality F_{COIN} for generating the randomness for the degree test. This functionality can be implemented using commitments. We further use functionality F_{ComCoin} for generating the randomness used in the OLE instances.

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

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

Fig. 8. Public coin tossing functionality.

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

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

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.

B Optimizations for Two-Party PSI

In our two-party PSI protocol, we compute each share $t_j = p_j(s_{1,j} + s_{2,j}) + q_j(r_{1,j} + r_{2,j})$ via blackbox access to $\mathcal{F}_{\mathrm{OLE}}$. Our $\mathcal{F}_{\mathrm{OLE}}$ (the first one) is instantiated with Ring-LWE as below:

- P_1 partitions shares p_j into blocks of N, and encrypts each block using the public key and the randomness obtained from the coin tossing (Step 3).
- P_2 also partitions shares $s_{2,j}, u_{2,j}$ into blocks of N, computes $Enc(PK, p_j \cdot s_{2,j} + u_{2,j})$ for the whole block using the randomness from the coin tossing step.
- The process is reversed with P_2 provides q_j and P_1 provides $r_{1,j}, u_{1,j}$ and P_1 is the receiver.

Instead of making two calls to $\mathcal{F}_{\mathrm{OLE}}$ 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, $\widetilde{m}$ 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 $\mathcal{F}_{\text{ComCoin}}$ honestly, generating random coins and commitments.
2. **Watchlists.**
$\widetilde{q}_I, \widetilde{u}_{2,I}, \widetilde{z}_{2,I}, \widetilde{r}_{2,I}, \widetilde{s}_{2,I}$: The simulator samples random polynomials $\widetilde{Q}(\cdot)$, $\widetilde{Z}_2(\cdot)$, $\widetilde{R}_2(\cdot)$, $\widetilde{S}_2(\cdot)$ of degree $w+t+e$, and $U_2(\cdot)$ of degree $2(w+t+e)$. It evaluates the polynomials on the roots of unity $\eta = (1, \omega, \ldots, \omega^{n-1})$ and obtains $RS_{\mathbb{F}_q,n,k,\eta}$ encodings $\widetilde{\mathbf{q}}$, $\widetilde{\mathbf{r}}_2$, $\widetilde{\mathbf{s}}_2$, $\widetilde{\mathbf{z}}_2$, and $RS_{\mathbb{F}_q,n,2k,\eta}$ encoding $\mathbf{u_2}$. The simulator sees P_1's choice bits when it submits them to the ideal functionality $\mathcal{F}_{\text{OT}}^{t:n}$ in Step 4. If more than t bits are set to 1, the simulator aborts and outputs whatever P_1 outputs. Else, the simulator obtains the indices $I = \{i_1, ..., i_t\}$ where the choice bits are 1. It hands P_1 the values $\widetilde{q}_I$, $\widetilde{u}_{2,I}$, $\widetilde{z}_{2,I}$, $\widetilde{r}_{2,I}$, $\widetilde{s}_{2,I}$ to simulate the messages P_1 receives. (Note that the simulation follows similarly also in the case where P_1 set fewer than t selection bits.)
The simulator extracts $\{p_j,\ u_{1,j},\ r_{1,j},\ s_{1,j},\ z_{1,j}\}_j$ when P_1 sends its input to $\mathcal{F}_{\text{OT}}^{t:n}$ in Step 4. The simulator reconstructs the polynomials $p(\cdot)$, $R_1(\cdot)$, $S_1(\cdot)$, $Z_1(\cdot)$ from these values. If any of the polynomials has a degree greater than $w+t+e$, the simulator sets $\mathsf{abort}_0 = 1$. Otherwise, $\mathsf{abort}_0 = 0$.
3. **Degree Test.**
$\widetilde{\alpha}_1, \widetilde{\alpha}_2, \widetilde{\alpha}_3, \widetilde{\alpha}_4, \widetilde{\alpha}_5, \widetilde{\alpha}_6, \widetilde{\alpha}_7, \widetilde{\alpha}_8, \widetilde{b}_j$: The simulator samples $\widetilde{\alpha}_1, \ldots, \widetilde{\alpha}_8$ uniformly at random and hands them to P_1 to simulate the output of $\mathcal{F}_{\text{COIN}}$. It computes $\mathbf{b} = (b_1, \ldots, b_n)$ where $\widetilde{b}_j = \alpha_5 \cdot \widetilde{z}_{2,j} + \alpha_6 \cdot \widetilde{r}_{2_j} + \alpha_7 \cdot \widetilde{s}_{2_j} + \alpha_8 \cdot \widetilde{q}_j$ and hands $\mathbf{b}$ to P_1 to simulate the messages P_1 receives in Step 5. If $\mathsf{abort}_0 == 1$ the simulator aborts and outputs whatever P_1 outputs.
Input Extraction. If $\mathsf{abort}_0 == 0$, the simulator interpolates the polynomial $\widetilde{P}(\cdot)$ from the points (ω^j, p_j). If $\widetilde{P}(\cdot) \equiv 0$, the simulator aborts and outputs whatever P_2 outputs. Else, the simulator extracts P_1's input $\widetilde{X}$ defined by $\widetilde{X} = \{x \mid \widetilde{P}(x) = 0\}$. The extracted input must be embedded with the following slackness: $\widetilde{X} = X \cup X'$ where X' are the roots of $T_1(\cdot)$ chosen by P_1 in Step 1.
Synthesize P_2's Input. The simulator submits $\widetilde{X}$ to the ideal functionality and obtains $\widetilde{X} \cap Y$. It recomputes $\widetilde{Q}(\cdot)$ such that $\widetilde{Q}(\cdot) = W(\cdot) \prod_{z \in \widetilde{X} \cap Y} (X - z)$ such that $deg(\widetilde{Q}) = w+t+e$, $\widetilde{Q}(\eta^j) = \widetilde{q}_j$ for $j \in I_1$, and $W(z) \neq 0$ for $z \in \widetilde{X}/(\widetilde{X} \cap Y)$.
4. **OLE.** $\widetilde{\mathbf{c}_1} = (\widetilde{c}_{1,1}, \ldots, \widetilde{c}_{1,n})$: The simulator verifies the messages sent in all n executions of the passive OLE in Step 6, verifying correctness against the decommitted randomness.[6] If more than $d/3$ executions are inconsistent with the watchlists yet the adversary is not caught, the simulator sets $\mathsf{abort}_1 = 1$. The simulator extracts $r_{1,j}$ when P_1 sends its input to the first $\mathcal{F}_{\text{OLE}}$ in Step 6. The simulator hands $\widetilde{\mathbf{c}}_\mathbf{1}$ where $\widetilde{c}_{1,j} = q_j \cdot r_{1,j} + u_{2,j}$ to P_1 to simulate this step. The simulator extracts $(p_j, u_{1,j})$ when P_1 sends its input to the second $\mathcal{F}_{\text{OLE}}$ in Step 6.
5. **Output Reconstruction.**
   - $\widetilde{\mathbf{d}}_\mathbf{2}$: The simulator computes $\widetilde{\mathbf{d}}_\mathbf{2}$ from $\mathbf{c_2}$ and its shares, and hands it to P_1.
   - The simulator receives $\mathbf{d_1} = (d_{1,1}, \ldots, d_{1,n})$ from P_1. It verifies that $d_{1,j} = c_{1,j} + p_j \cdot s_{1,j} - u_{1,j} = q_j \cdot r_{1,j} + u_{2,j} + p_j \cdot s_{1,j} - u_{1,j}$ for all $j \in [1, n]$. If $\mathsf{abort}_1 = 1$ 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 $RS_{\mathbb{F}_q,n,k,\eta}$ codeword to $\mathcal{F}_{\text{OT}}^{t:n}$ 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.

6 As explained in the second footnote in Figure 2, for simplicity, in the remainder of the simulation we will treat these OLE executions as ideal. However, to verify correctness of the executions, we have to examine the messages of the particular instantiation of the ideal primitive. Since this can be done generically, for any instantiation, we allow ourselves this simplification.

$$\left\{\mathrm{REAL}^{F_{\mathrm{ComCoin}},\mathcal{F}_{\mathrm{OT}}^{t:n},\mathcal{F}_{\mathrm{COIN}},\mathcal{F}_{\mathrm{OLE}}}_{\Pi_{2\mathrm{PSI}},\mathcal{A}(z)}(X,Y,\kappa,s))\right\}_{\kappa,s,X,Y,z} \stackrel{c}{\equiv} \left\{\mathrm{IDEAL}_{\mathcal{F}_{2\mathrm{PSI}},\mathcal{S}(z)}(X,Y,\kappa,s))\right\}_{\kappa,s,X,Y,z}$$

- H_0 : This game is a hybrid execution of the protocol.
- H_1 : Similar to H_0, except that when E_1 happens, the simulator aborts.
- H_2 : Similar to H_1, except that when E_2 happens, the simulator aborts.
- H_3 : Ideal execution of the protocol.

$\mathbf{H_0}$ and $\mathbf{H_1}$: We prove that H_0 and H_1 are statistically close. Note that H_0 and H_1 can be distinguished if and only if one of the random polynomials sampled in Step 2 in H_0 has a degree greater than $w+t+e$, yet the degree test passes. Whereas in H_1, the simulator always knows if the adversary cheats and aborts when the degree test is executed, even if it passes.

Recall first that a random polynomial p that is sampled honestly in Step 2, $p(\eta) \equiv (p(1), ..., (\omega^{n-1}))$ is a $[n, k, n-k+1]$ Reed-Solomon codeword. Moreover, the degree test checks whether a random linear combination of the codewords generated from those random polynomials belongs to $RS_{\mathbb{F}_q,n,k,\eta}$. When one of these random polynomials has a degree higher than $(w+t+e)$, in order to pass the degree test, P_1 must come up with a tuple $(a_1, ..., a_n)$ that is a valid $RS_{\mathbb{F}_q,n,k,\eta}$ codeword.

To bound this error, we rely the following Lemma from Ames et al. [AHIV17] to bound the error probability that the degree test passes by $d/\mathbb{F}$. We denote by the matrix U_i the list of codewords to be proven by party P_i.

Lemma 3. *[AHIV17] Let $L = RS_{\mathbb{F}_q,n,k,\eta}$ and e a positive integer such that $e < d/3$, where d is the minimum distance of L. Suppose $d(U_i, L^m) > e$ where U_i is as defined as above. Then, for a random l^* in the row-span of U_i, it holds that*

$$\Pr[d(l^*, L) \leq e] \leq d/|\mathbb{F}|.$$

$\mathbf{H_1}$ and $\mathbf{H_2}$: We prove that H_1 and H_2 are statistically close. Note that H_1 and H_2 can be distinguished if and only if in H_1 the event E_2 happens but it is not caught by the watchlists. This error probability is bounded by $(1-e/n)^t$. Meaning, if the adversary is deviating in at least e positions overall, it will be caught except with this probability.

$\mathbf{H_2}$ and $\mathbf{H_3}$: We prove that the views in H_2 and H_3 are statistically close to each other.

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 $\widetilde{Q}(\cdot)$ and generates the messages $\widetilde{q}_{I_1}$. Only after the OT, the simulator extracts P_1's input $\widetilde{X}$. The output is determined based on the extracted input of the adversary. Once the simulator obtains the output $\widetilde{X} \cap Y$, the simulator needs to recalculate the polynomial $\widetilde{Q}(\cdot)$ used in the following steps of the simulation. Now $\widetilde{Q}(\cdot) = \Pi_{i=1}^{w+t+e-|\widetilde{X}\cap Y|}(X-z_i)\Pi_{y_i \in \widetilde{X}\cap Y}(X-y_i) \cdot \widetilde{T}_2'(\cdot)$ where $z_i \notin \widetilde{X} \cap Y$ and $\widetilde{T}_2'(\cdot)$ are random polynomials of degree $(t+e)$, and chosen such that the new $\widetilde{Q}(\cdot)$ is consistent with the shares $\widetilde{q}_{I_1}$ sent through the watchlist channels. Denote $\widetilde{Y} = \{z_i\} \cup (\widetilde{X} \cap Y)$. 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 $\widetilde{Q}(\cdot)$. However, as $\Pi_{i=1}^{w}(X-y_i)$ and $\Pi_{i=1}^{w}(X-\widetilde{y}_i)$ are both masked with random polynomials of degree $t+e$, nothing is leaked about y_i or $\widetilde{y}_i$ from observing t evaluations.

In H_2, P_1 receives the $RS_{\mathbb{F}_q,n,k,\eta}$ 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 $\widetilde{X}$, the simulator submits $\widetilde{X}$ to the ideal functionality and obtains the output $\widetilde{X} \cap Y$, so P_1 receives $(\widetilde{t}_1,\ldots,\widetilde{t}_n)$ of $\widetilde{T} = \widetilde{W} \cdot \Pi_{z_i \in \widetilde{X}\cap Y}(X-z_i)$ where $\widetilde{W}$ is a random polynomial of degree $2(w+t+e) - |\widetilde{X} \cap Y|$ 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 $T = W \cdot \Pi_{z_i \in X\cap Y}(X-z_i)$ 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 $\widetilde{X} \cap Y$, however, in H_2, W may contain extra roots that are in $\widetilde{X} \backslash (\widetilde{X} \cap Y)$ or $Y \backslash (\widetilde{X} \cap Y)$. 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 $2(w+t+e)/|\mathbb{F}|$.

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 $\mathbb{F}$ is bounded by $t/|\mathbb{F}|$.*

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 $1/|\mathbb{F}|$. 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 $\widetilde{P}(\cdot)$ is bounded by $(w+t+e)/|\mathbb{F}|$, and that $W(\cdot)$ has a common root with $\Pi_{i=1}^{w}(X-y_i)$, 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_0 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_i, running the protocol honestly until Step 6 (in Step 2 it uses random input $\widetilde{Q}_i$ 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 $(\widetilde{q}_{i,j}, j \in I)$.
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.
   - P_0's input: $(q_{0,j},\ r_{0,j},\ z_{0,j},\ s^1_{0,j}, \cdots, s^{m-1}_{0,j},\ u^1_{0,j}, \cdots, u^{m-1}_{0,j})$ 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^i_0, U^i_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^i_0, or Q_i, R_i, Z_i, S_i has degree higher than $w+3t+e$, the simulator sets $\mathsf{abort}_0 = 1$. Else, it sets $\mathsf{abort}_0 = 0$.
3. **Synthesize Honest Parties' Input.** If $\mathsf{abort}_0 == 0$, the simulator submits the corrupt parties' input to the ideal functionality, receiving $X = \cap_{i=0}^{m-1} X_i$. For each honest party P_i, the simulator uses $\widetilde{X}_i = X \cup Z_i$ where each Z_i consists of $(w-|X|)$ random values such that $X \cap Z_i = \emptyset$. It recomputes $\widetilde{Q}_i(\cdot)$ such that $\widetilde{Q}_i(\cdot) = \Pi_{z\in\widetilde{X}_i}(X-z)\cdot T_i(\cdot)$, where $deg(T_i) = 3t+e$. It sets $\widetilde{q}_{i,j} = \widetilde{Q}_i(\eta^j)$ 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.

   We note that, whenever the parties need to reveal the shares to perform the checks, $\widetilde{q}_{i,j}$ 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 $\mathsf{abort}_0 == 1$, it aborts and outputs whatever P_0 outputs.
5. **OLE.** $\widetilde{\mathbf{c}}_1 = (\widetilde{c}_{1,1}, \cdots, \widetilde{c}_{1,n})$: 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 $\mathsf{abort}_1 = 1$. Specifically, the simulator extracts $\mathbf{s^i_0}$ when P_0 sends its input to the first $\mathcal{F}_{\text{OLE}}$ in Step 8. The simulator hands $\widetilde{\mathbf{c}}^{\mathbf{i}}_{\mathbf{0}}$ where $\widetilde{c}^i_{0,j} = q_{i,j}\cdot s^i_{0,j} + u_{i,j}$ to P_0 to simulate this step.

   The simulator extracts $(\mathbf{q_0}, \mathbf{u^i_0})$ when P_0 sends its input to the second $\mathcal{F}_{\text{OLE}}$ in Step 8. If P_0 uses encodings that are not consistent with the watchlist (that also enforces the use of the same $\mathbf{q_0}$ in all $\mathcal{F}_{\text{OLE}}$ invocations), the simulator aborts. The simulator uses the extracted input to $\mathcal{F}_{\text{OLE}}$ to compute $\mathbf{c_i}$ for the honest party P_i and stores it.
6. **Output Reconstruction.**
   - $\widetilde{\mathbf{d}}_{\mathbf{i}}$: The simulator computes $\widetilde{\mathbf{d}}_{\mathbf{i}}$ from $\mathbf{c_i}$ and its shares, and hands it to P_0 on behalf of honest parties P_i.
   - The simulator receives $\mathbf{t} = (t_{1,1}, \cdots, t_{1,n})$ from P_0. It verifies that $t_j = q_{0,j}\cdot \sum_{i=0}^{m-1} r_i + \sum_{i=1}^{m-1} q_{i,j}(s^i_{0,j} + s_{i,j})$ for $j \in [1,n]$. If $\mathsf{abort}_1 = 1$ 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 $RS_{\mathbb{F}_q,n,k,\eta}$ 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 $\widetilde{Q}_i$ 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 $(\widetilde{q}_{i,j}, j \in I)$.
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 $(i \in A \setminus \{P_0\})$: $(q_{i,j}, r_{i,j}, s_{i,j}, z_{i,j}, v_{i,j}, u_{i,j})$ for $j \in [1, n]$.
The simulator reconstructs the corresponding polynomials Q_i, R_i, Z_i, S_i, U_i. If any of the polynomials Q_i, R_i, Z_i, S_i has degree higher than $w+3t+e$, the simulator sets $\mathsf{abort}_0 = 1$. Else, it sets $\mathsf{abort}_0 = 0$.
3. **Synthesize Honest Parties' Input.** If $\mathsf{abort}_0 == 0$, the simulator submits the corrupt parties' input to the ideal functionality, receiving $X = \cap_{i=0}^{m-1} X_i$. For each honest party P_i, the simulator uses $\widetilde{X}_i = X \cup Z_i$ where each Z_i consists of $(w - |X|)$ random values such that $X \cap Z_i = \emptyset$. It recomputes $\widetilde{Q}_i(\cdot)$ such that $\widetilde{Q}_i(\cdot) = \Pi_{z \in \widetilde{X}_i}(X - z) \cdot T_i(\cdot)$ such that $deg(T_i) = 3t+e$, $\widetilde{Q}_i(\eta^j) = \widetilde{q}_{i,j}$ 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, $\widetilde{q}_{i,j}$ 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 $\mathsf{abort}_0 == 1$, it aborts and outputs whatever the corrupt parties output.
5. **OLE.** $\widetilde{\mathbf{c_1}} = (\widetilde{c}_{1,1}, \cdots, \widetilde{c}_{1,n})$: 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 $\mathsf{abort}_1 = 1$.
Specifically, the simulator extracts $\mathbf{q_i}, \mathbf{u_i}$ when P_i sends its input to the first $\mathcal{F}_{\text{OLE}}$ in Step 8. The simulator extracts $\mathbf{r_i}$ when P_i sends its input to the second $\mathcal{F}_{\text{OLE}}$ in Step 8. The simulator hands $\widetilde{\mathbf{c_i}}$ where $\widetilde{c}_{i,j} = q_{0,j} \cdot r_{i,j} + u^i_{0,j}$ 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 $\mathcal{F}_{\text{OLE}}$ to compute $\mathbf{c^i_0}$ and $\mathbf{c_i}$ for honest party P_i and stores it.
6. **Output Reconstruction $\widetilde{\mathbf{d}}$**: For each honest parties P_i, the simulator computes $\widetilde{\mathbf{d_i}}$ from $\mathbf{c_i}$ and its shares. For each corrupt party P_i, the simulator receives $\widetilde{d_i}$ and verifies that $d_{i,j} = \widetilde{q}_{0,j} r_{i,j} + q_{i,j} s_{i,j} + v_{i,j} + (u^i_{0,j} - u_{i,j})$ for $i \in A$ and $j \in [1, n]$.
If $\mathsf{abort}_1 = 1$ or if the check fails for at least $d/3$ positions, the simulator aborts and outputs whatever the adversary outputs. Otherwise, the simulator computes $\widetilde{\mathbf{d}}$ 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.