Efficient Linear Multiparty PSI and Extensions to Circuit/Quorum PSI

Nishanth Chandran[1], Nishka Dasgupta[1], Divya Gupta[1], Sai Lakshmi Bhavana Obbattu[1], Sruthi Sekar[2], and Akash Shah[1]

[1] Microsoft Research, Bangalore
{nichandr, t-nidasg, divya.gupta, t-saobb, t-akshah}@microsoft.com
[2] Indian Institute of Science, Bangalore
{sruthi.sekar1@gmail.com}

Abstract. Multiparty Private Set Intersection (mPSI), enables n parties, each holding private sets (each of size m) to compute the intersection of these private sets, without revealing any other information to each other. While several protocols for this task are known, the only concretely efficient protocol is due to the work of Kolesnikov *et al.* (KMPRT, CCS 2017), who gave a semi-honest secure protocol with communication complexity $\mathcal{O}(nmt\lambda)$, where $t < n$ is the number of corrupt parties and λ is the security parameter. In this work, we make the following contributions:
− First, for the natural adversarial setting of semi-honest honest majority (i.e. $t < n/2$), we asymptotically improve upon the above result and provide a concretely efficient protocol with total communication of $\mathcal{O}(nm\lambda)$.
− Second, concretely, our protocol has $6(t + 2)/5$ times lesser communication than KMPRT and is upto 5× and 6.2× faster than KMPRT in the LAN and WAN setting even for 15 parties.
− Finally, we introduce and consider two important variants of mPSI - circuit PSI (that allows the parties to compute a function over the intersection set without revealing the intersection itself) and quorum PSI (that allows P_1 to learn all the elements in his/her set that are present in at least k other sets) and provide concretely efficient protocols for these variants.

1 Introduction

Multiparty PSI. Private set intersection (PSI) [64, 45] enables two parties P_1 and P_2, with respective input sets X and Y, to learn the intersection $X \cap Y$, without revealing any other information to any of the parties. General secure multiparty computation protocols [66, 33, 5, 4] have proven to be inefficient to solve this problem and hence several works have focused on obtaining concretely efficient specialized protocols [39, 36, 17, 18, 22, 54, 49, 42, 59, 60, 57, 52, 55, 13, 53, 43]. The problem of *Multiparty PSI* (mPSI) was first introduced in [28] and it generalizes PSI − i.e., n parties compute the intersection of their n private data sets, without revealing any additional information. While the protocol with best asymptotic communication complexity for mPSI was given in [37], the first and only known practical realization was provided in [43]. This protocol is secure in the semi-honest dishonest majority setting[3] (i.e., the adversary can corrupt up to $n - 1$ parties and follows the protocol specification faithfully) and its total communication complexity is $\mathcal{O}(nmt\lambda)$, where n is the number of parties, $t < n$ is the corruption threshold, m is the set size of each party and λ is the computational security parameter. While such a high communication overhead might be unavoidable for concretely efficient dishonest majority protocols (i.e., not based on homomorphic encryption), in many scenarios, security against honest majority (i.e., $t < n/2$) is acceptable and widely studied in several practical contexts [20, 44, 47, 2, 68, 7, 16]. Hence, it is important to explore the concrete efficiency of mPSI protocols in this setting. Unfortunately, even under this relaxation (also considered in [14]), the best known protocol [43] is no better and has complexity $\mathcal{O}(nmt\lambda)$.

[3] The works of [43, 40] also build concretely efficient mPSI in a weaker augmented semi-honest model; in this work, we focus on standard semi-honest security.

1.1 Our Contributions

In this work, we build the first concretely efficient mPSI protocol in the semi-honest honest majority setting, with total communication of $\mathcal{O}(nm\lambda)$, thus obtaining an $\mathcal{O}(t)$-factor improvement over [43]. While theoretically, this matches the complexity of the protocol from [37] based on homomorphic encryption[4], concretely, our protocol is approximately $6(t+2)/5$ times more communication frugal than [43]. This amounts to more than an order of magnitude lesser communication than [43] when the number of parties > 15 and $t \approx n/2$; even when $t = 1$, our protocol has nearly $4\times$ lesser communication than [43]. We also implement our protocol and show it to be up to $5\times$ and $6.2\times$ faster than [43] in the LAN and WAN settings, respectively in the honest majority setting considered in their experiments (as an example for $n = 15, t = 7$ and set size $m = 2^{20}$, our protocol executes under 40s and 245s in LAN and WAN settings).

Next, we consider 2 important variants of the mPSI problem - circuit PSI and quorum PSI providing semi-honest security in honest majority setting.

Circuit PSI. The problem of circuit PSI was introduced in the 2 party setting [38] and enables parties P_1 and P_2, with their private input sets X and Y, respectively, to compute $f(X \cap Y)$, where f is any *symmetric* function (i.e., f operates on $X \cap Y$ and is oblivious to the order of elements in it). Circuit PSI allows to keep the intersection $X \cap Y$ itself secret from the parties while allowing to securely compute $f(X \cap Y)$ and has found many interesting applications. Some examples of symmetric functions include cardinality, set intersection sum [67], and threshold intersection [35]. Circuit PSI has received a lot of attention and has also shown to be practically feasible in the 2-party context [54, 56, 55, 24, 15, 12]. The problem of circuit PSI is equally well-motivated in the multiparty setting. However, to the best of our knowledge, it has remained unexplored.

In our work, we provide the first multiparty ciruit PSI protocol achieving a communication of approximately $\mathcal{O}(mn(\lambda\kappa + \log^2 n))$. Concretely, its communication is only $\approx 4\times$ the cost of mPSI. We also give a circuit PSI protocol achieving a communication complexity of $\mathcal{O}(mn(\lambda + (\log m + \kappa)^2))$, which is asymptotically linear in n. However, this protocol is concretely less efficient than the first one.

Quorum PSI. We consider another variant of mPSI, called *quorum PSI* (qPSI), where a leader P_1 wishes to obtain the elements of his/her set that are also present in at least k of the other $n-1$ parties' sets. Such a variant lends itself to natural applications - e.g. in the context of anti-money laundering [26, 25] and checking if a list of entities are present in multiple blacklists. We provide an efficient qPSI protocol in the semi-honest honest majority setting achieving a communication cost of $\mathcal{O}(nm\kappa(\lambda + \kappa \log n))$.

We implement both circuit PSI and qPSI protocols showing that these protocols are concretely efficient as well. These are the first implementations of multiparty circuit PSI and quorum PSI.

Protocol blueprint. Our protocols for all three problem settings, namely, mPSI, circuit PSI and qPSI, broadly have two phases. At a high level, in the first phase, a fixed designated party, say P_1, interacts with all other parties $P_2, \ldots, P_n$ using 2-party protocols. In the second phase, all parties engage in n-party protocols to compute a circuit to get the requisite output. We describe these phases in the context of mPSI and then discuss the changes for the other variants.

For mPSI, in the first phase, we invoke a two-party functionality, which we call *weak private set membership* (wPSM) functionality, between a leader, P_1 and each P_i (for $i \in \{2, \cdots, n\}$). Informally, the wPSM functionality, when invoked on inputs of P_1 and P_i (their individual private sets[5]) does the following: for each element in P_1's set, it outputs the same random value to both P_1 and P_i, if that

[4] We remark here that [37], through the use of homomorphic encryption, provide an mPSI protocol in the semi-honest (as well as malicious) dishonest majority setting, achieving a communication of $\mathcal{O}(nm\lambda)$; however, they do not show it to be concretely efficient. As mentioned in [43] the protocol of [37] is expected to be much slower than [43] due to its use of homomorphic encryption.

[5] Strictly speaking, as is common in PSI protocols, a phase of local hashing is done before invoking this functionality.

element is in P_i's set, and outputs independent random values, otherwise[6]. By invoking only n instances of the wPSM functionality overall, we ensure that the total communication complexity of this phase is linear in n. In the second phase, all the parties together run a secure multiparty computation to obtain shares of 0 for each element in P_1's set that is in the intersection and shares of a random element for other elements. Having invoked wPSM between P_1 and every other party, this can be computed using a *single multiplication* protocol. We evaluate this multiplication using the MPC protocol from [20, 44] in the second phase, resulting in the total communication complexity being *linear in n*.

In our circuit and quorum PSI protocols, the first phase additionally includes conversion of the outputs from the wPSM functionality to arithmetic shares of 1 if P_1 and P_i received the same random value, and shares of 0, otherwise (this is similar to how 2-party circuit-PSI protocols work). In the second phase, in circuit-PSI, for every element of P_1, all parties must get shares of 1 if that element belongs to the intersection, and shares of 0, otherwise. To do this, we use the following trick: for every element x in P_1's set, count the number of other sets q_x in which element x is present (the first phase of our protocol does indeed give us such a count). Now, if we compute $w_x = (q_x - (n-1))^{p-1}$ over $\mathbb{F}_p$, where $p > n$ is prime, then $w_x = 0$ if $q_x = n-1$ (and 1 otherwise), which precisely gives us whether or not x is in the intersection. Hence, one can compute shares of whether x is in the intersection or not by simply computing this polynomial (which can be securely done using $2 \log p$ multiplications). In the case of qPSI, we appropriately choose another polynomial such that for each element in P_1's set, the polynomial evaluates to 0 if and only if that element belongs to the quorum intersection, and random otherwise.

Next, we make a few observations on our protocol blueprint. As already mentioned, this blueprint allows us to get sub-quadratic complexity in n for all our protocols. Moreover, in the first phase, P_i for $i \neq 1$ interacts with P_1 alone. As an example, in mPSI, P_i only engages in one instance of weak-PSM, whereas P_1 engages in $n-1$ instances of the same. Also, we show in technical sections, the complexity of phase-one significantly dominates the overall complexity. With these observations, our protocols give a desirable property of all-but-1 parties being light-weight, making them suitable to be used in client-server setting, where only one party needs to be computationally heavy and is played by the server. Note that unlike prior works in client-server setting [46, 1], we allow collusion between the server, P_1, and any subset of the clients, $P_2, \ldots, P_n$ as long as $t < n/2$ parties are corrupt. Finally, protocol in [43] also had an asymmetry between load on different parties, and our clients require $7(2t+3)/10$ times less communication than clients in [43].

To summarize our contributions:

- We give the first concretely efficient protocol for mPSI, with communication complexity of $\mathcal{O}(nm\lambda)$ and constant rounds.
- We construct the first multiparty circuit-PSI and qPSI protocols and show them to be concretely efficient.
- Finally, we implement our protocols and show that our mPSI protocol is up to $5\times$ and $6.2\times$ faster than prior state of the art [43] in LAN and WAN settings, respectively even for 15 parties.

Our protocols are semi-honest secure in the honest majority setting.

1.2 Other Related Works

The works of [28, 14, 61, 62, 41, 37] build theoretical multiparty PSI protocols in the malicious setting, relying on homomorphic encryption (or bilinear groups in [62]); however, none of these works are concretely efficient. The work of [40] build an mPSI protocol in the semi-honest dishonest majority setting, using garbled bloom filters, but provide no implementation. The works of [30, 3] and [9] consider variants of mPSI - *multiparty threshold PSI* (where the parties learn the intersection only if its size is greater than a threshold) and *multiparty cardinality testing for threshold PSI* (where the parties learn

[6] This resembles the two-party *oblivious programmable pseudorandom function (OPPRF)* functionality [43], and we indeed show that it can be instantiated using an OPPRF.

the cardinality of the intersection under the same condition). Setting the threshold to 0 in multiparty threshold PSI gives a protocol for mPSI with complexity matching our protocol [3]; however, these protocols are based on homomorphic encryption and are not concretely efficient. The works of [46, 1] build mPSI protocols in the server-aided model (which assumes the existence of a server that does not collude with the clients).

1.3 Organization

We begin with the details of the security model and cryptographic primitives in Section 2 on preliminaries. Then, we describe our multiparty PSI protocol in Section 3, our circuit PSI protocol in Section 4, and our quorum PSI protocol in Section 5. Finally, we present our experimental results in Section 6 on implementation and performance.

2 Preliminaries

Notations. Let κ and λ denote statistical and computational security parameters respectively. For a positive integer k, $[k]$ denotes the set $\{1, 2, \cdots, k\}$. For a set S, $|S|$ denotes the cardinality of S. For two sets S and S', $S \setminus S'$ denotes the set of elements that are present in S but not in S'. For $x \in \{0,1\}^*$, $|x|$ denotes the bit-length of x. For integers a and b such that $(a < b)$, $[a, b]$ denotes the closed interval of integers between a and b. We use log to denote logarithms with base 2. For any $x \in \{0,1\}^\ell$, $\ell > 1$, we also use its natural interpretation as an integer in the range $\{-2^{\ell-1}, 2^{\ell-1} - 1\}$ using 2's complement representation. $\mathbb{F}_p$ denotes a finite field with prime order p.

Secret Sharing. An $(n, t)-$ secret sharing scheme [63, 6] for $t < n$ allows to distribute a secret s amongst n parties as *shares* $s_1, \cdots, s_n$, such that any $t + 1$ parties can collectively reconstruct the secret s from their shares and no collusion of t parties learn any information about s. We instantiate $(n, t)-$ secret sharing for a secret $s \in \mathbb{F}$ with the Shamir secret sharing scheme [63]. Additionally, we make use of the additive secret sharing scheme, which is an $(n, n-1)$- secret sharing scheme. Here, to share $s \in \mathbb{F}$, shares of n parties $\langle s \rangle_1, \cdots, \langle s \rangle_n$ are chosen uniformly from the field $\mathbb{F}$ subject to the constraint that $\langle s \rangle_1 + \cdots + \langle s \rangle_n = s$, where $+$ is the addition operation in $\mathbb{F}$. We use the additive secret sharing both in the general n-party setting and also more specifically in the 2-party setting. To secret share a boolean value $b \in \{0, 1\}$ we use additive secret sharing scheme over the field $\mathbb{F}_2$. We use boolean sharing only in the two party setting. If a bit b is shared amongst two parties P_1 and P_2, the shares are denoted by $\langle b \rangle_1^B$ and $\langle b \rangle_2^B$ respectively.

2.1 Security Model

We consider the multiparty setting with n parties: $P_1, \cdots, P_n$. We consider a semi-honest adversary $\mathcal{A}$ that corrupts $t < n/2$ parties and tries to learn as much information as possible from the protocol execution but faithfully follows the protocol specification. This is called the semi-honest honest majority setting. To capture semi-honest security of a protocol in the simulation based model [34, 31, 10], we show that for any semi-honest adversary, there exists a simulator such that the view of a distinguisher in the following two executions are indistinguishable: one is the view of the real execution of the protocol in the presence of a semi-honest adversary and the second is the view of an ideal execution of the protocol where a simulator interacts with the ideal functionality (which, given the inputs of all parties, computes the function being evaluated and returns the outputs). We further also consider semi-honest security in a hybrid model [10], where, in addition to communicating as usual in the standard execution of the protocol, the parties have access to an ideal functionality. Specifically, in an $\mathcal{F}$-hybrid protocol, the parties may give inputs to and receive outputs from this functionality $\mathcal{F}$. By the universal composition theorem [10], if we have any semi-honest secure protocol π realizing the functionality $\mathcal{F}$, then any $\mathcal{F}$-hybrid protocol can be realized in the standard model, by replacing $\mathcal{F}$ with the protocol π.

2.2 Cuckoo Hashing

Cuckoo hashing [51] uses K random hash functions $h_1, \cdots, h_K : \{0,1\}^\sigma \to [\beta]$ to map m elements into β bins. The mapping procedure is as follows. An element x is inserted into the bin $h_i(x)$, if this bin is empty for some $i \in [K]$ (if there are multiple empty bins, then we pick the first one in the lexicographic ordering of the bins). Otherwise, pick a random $i \in [K]$, insert x in bin $h_i(x)$, evict the item currently in $h_i(x)$ and recursively insert the evicted item. The recursion proceeds until no more evictions are necessary or until a threshold number of re-allocations are done. If the recursion stops because of the latter reason, it is considered as a failure event. This failure signifies existence of an element that didn't map to any of the bins. Some variants of Cuckoo hashing maintain a set called the *stash*, to store such elements. Stash-less cuckoo hashing is where no special stash is maintained.

In stash-less Cuckoo hashing, Pinkas *et al.* [57] showed that for $K = 3, 4$ and 5 and $\beta = 1.27m, 1.09m$ and $1.05m$ respectively, the failure probability is atmost 2^{-40}, by extrapolating their experimental analysis for the failure probability 2^{-30}. All protocols in this work are in this stash-less setting. To bound the overall failure probability of our proposed protocols to 2^{-40}, we require an analysis of the parameters of Cuckoo hashing such that the failure probability in stash-less Cuckoo hashing scheme is atmost $2^{-41}/2^{-42}/2^{-46}$. Extrapolating, similar to [57], we get $\beta = 1.28m/1.28m/1.31m$ to ensure that the failure probability in stash-less Cuckoo hashing is atmost $2^{-41}/2^{-42}/2^{-46}$ respectively for $K = 3$.

2.3 Two-party Functionalities

Equality Test We use a two-party equality test functionality $\mathcal{F}_{\mathsf{EQ}}^{\ell}$. In this functionality, parties P_1 and P_2 have $a \in \{0,1\}^\ell$ and $b \in \{0,1\}^\ell$ respectively as private inputs and receive boolean shares of the bit 1 if $a = b$ and 0 otherwise, as the output. We make use of the protocol given in [12] that builds on the ideas of [29, 21, 58] to realize this functionality. The simplified expression of the concrete communication complexity of this protocol is $3\ell\lambda/4 + 8\ell$ and round complexity is $\log \ell$.

Boolean to Arithmetic Share Conversion We also use a two-party functionality $\mathcal{F}_{\mathsf{B2A}}^{\mathbb{F}}$, which converts boolean shares of a bit to its additive shares (in a field $\mathbb{F}$). More specifically, the functionality requires parties P_1 and P_2 to input their boolean shares b_1 and b_2 (of a bit b) respectively and outputs the additive shares x_1 and x_2 of b over $\mathbb{F}$ to P_1 and P_2 respectively. We instantiate this functionality with the share conversion protocol given in [58] that uses one corelated OT and has total communication of $\lambda + \lceil \log |\mathbb{F}| \rceil$ bits and round complexity 2.
We remark here that OT extension using the recent line of work on SilentOT [8, 65] can be used to improve the communication cost of both the equality test and boolean to arithmetic share conversion functionalities. Our implementations do not incorporate these recent optimizations, which would only improve their performance.

2.4 Weak Private Set Membership Test Functionality

We define a 2-party functionality, $\mathcal{F}_{\mathsf{w-PSM}}^{\beta,\sigma,N}$, called *weak private set membership (weak-*PSM*) test* that allows a clean exposition of our protocols. We note that this functionality is similar in spirit to the batch oblivious programmable PRF considered in [55] and as we discuss later, that is indeed one way to realize this functionality efficiently. In a single instance of the weak PSM test, one party holds an element q and another party holds a set X. Parties learn the same random element w if $q \in X$, else one party learns y and other party learns w, where y and w are independent random values. A weak PSM test is where the two parties do multiple instances of membership tests together as a batch. We define the functionality $\mathcal{F}_{\mathsf{w-PSM}}^{\beta,\sigma,N}$ formally in Figure 1, where β is the batch size, σ is length of input and output elements, and N is the total size of all sets input by the second party.

We consider three instantiations of this functionality using primitives considered in the line of oblivious programmable pseudorandom functions (OPPRF) [43]. We provide details on instantiations in Appendix A and summarize their costs below.

P_1 and P_2 are the receiver and the sender respectively.
Receiver P_1**'s Inputs**: The queries $q_1, \cdots, q_\beta \in \{0,1\}^\sigma$.
Sender P_2**'s Inputs**: Sets $\{X_j\}_{j\in[\beta]}$, where $|X_j(i)| = \sigma$ for every $j \in [\beta]$ and $i \in [|X_j|]$ and $\sum_j |X_j| = N$.
Output:

- For each $j \in [\beta]$, sample w_j uniformly from $\{0,1\}^\sigma$.
- For each $j \in [\beta]$, if $q_j \in X_j$, set $y_j = w_j$, else sample y_j uniformly from $\{0,1\}^\sigma$.
- Return $\{y_j\}_{j\in[\beta]}$ to P_1 and $\{w_j\}_{j\in[\beta]}$ to P_2.

Fig. 1: Weak PSM Functionality $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$

- **Polynomial-based batch-OPPRF** [55]: When we instantiate using the polynomial-based OPPRF from [55], the concrete communication cost is $3.5\lambda\beta + N\sigma$ and round complexity is 4.
- **Table-based OPPRF** [43]: The instantiation using table-based OPPRF [43] assumes an upper-bound on the size of the input sets, which is derived specific to its application. Let $d \in \mathbb{N}$ be the minimum value such that the aforementioned upper-bound is bounded by 2^d. When we instantiate using the table-based OPPRF, the concrete communication cost is $(4.5\lambda + 2^d\sigma)\beta$ and round complexity is 4.
- **Relaxed batch OPPRF**: We can instantiate $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality by invoking relaxed batch OPPRF [12] followed by an invocation of table-based OPPRF [43]. The concrete communication of this case is $(8\lambda + 4\sigma)\beta + 1.31N\sigma$ and round complexity is 8.

Execution Cost: Instantiations of the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality using the above 3 approaches provide trade-offs between computation and communication [43, 55, 12]. Due to this, different protocols are more efficient in different experimental settings as is evident from the empirical results given in Section 6.

2.5 Multiparty Functionalities

Our protocols invoke several n-party functionalities in the honest majority setting and we describe them below. The protocols from [20, 44] can be used to realize these functionalities and we summarize their communication complexity in Table 1.

Let $\mathbb{F}(+,\cdot)$ be a finite field. Let n be the number of parties and $t < n/2$ be the corruption threshold. We use $[a]$ to denote an $(n,t)-$ linear secret sharing of element $a \in \mathbb{F}$ such that each party P_i holds $[a]_i$. Further, $\langle a\rangle$ denotes additive sharing of $a \in \mathbb{F}$ where P_i holds the additive share $\langle a\rangle_i$. For any $a,b,c \in \mathbb{F}$, $c\cdot[a]+[b]$ (resp. $c\cdot\langle a\rangle + \langle b\rangle$) represents that, for each $i \in [n]$, P_i computes $c\cdot[a]_i + [b]_i$ (resp. c $\cdot\langle a\rangle_i + \langle b\rangle_i$)). Linearity ensures that for any $a,b,c \in \mathbb{F}$, $c\cdot[a]+[b] = [c\cdot a + b]$. For $a,c \in \mathbb{F}$, $[a]+c$ and $\langle a\rangle + c$ represent the local computation required to get $[a+c]$ and $\langle a+c\rangle$.

- $\mathsf{RandomF}^{n,t}(\ell)$: Generates $[r_1], \cdots, [r_\ell]$ for uniform elements $r_1, \cdots, r_\ell$ in $\mathbb{F}$.
- $\mathsf{MultF}^{n,t}([a],[b])$: Takes $[a],[b]$ for $a,b \in \mathbb{F}$ and outputs $[a\cdot b]$.

Additionally, we use the following functionalities which can be realized using techniques from [20].

- $\mathsf{Reveal}^{n,t}([a])$: Takes $[a]$ where $a \in \mathbb{F}$ and outputs a to P_1.
  To realize this functionality, P_i, for all $i \in \{2,\ldots,n\}$, sends $[a]_i$ to P_1, who reconstructs and learns a.
- $\mathsf{RevealInReshareF}^{n}(d,\langle a\rangle)$: Takes $\langle a\rangle$ where $a \in \mathbb{F}$ and $d < n$. Outputs an (n,d)- sharing of a and in addition outputs a to P_1.
  To realize this, parties send additive shares of a to P_1, who reconstructs a, and distributes (n,d) shares of a to all parties.
- $\mathsf{DoubleRandomF}^{n,t}(\ell)$: Generates $[r_1], \cdots, [r_\ell]$ and $\langle r_1\rangle, \cdots, \langle r_\ell\rangle$ for uniform elements $r_1, \cdots, r_\ell$ in $\mathbb{F}$.
  We show a realization of this functionality in Appendix E and present its cost in Table 1.

We summarize the communication and round complexity of realizing the above functionalities as per [20] in Table 1. In our results we invoke $\mathsf{RandomF}^{n,t}$ and $\mathsf{DoubleRandomF}^{n,t}$ on $\ell \gg n$ and for simplicity we let $\lceil \ell/(n-t) \rceil$ to be $\ell/(n-t)$. In the complexity analysis of our results, for ease of exposition, we approximate t/n with $1/2$. This approximation only overestimates our costs as $t < n/2$.

Functionality	Communication	Rounds
$\mathsf{RandomF}^{n,t}(\ell)$	$\left\lceil \frac{\ell}{n-t} \right\rceil n(n-1)\lceil \log \lvert\mathbb{F}\rvert \rceil$ $< 2\ell(n-1)\lceil \log \lvert\mathbb{F}\rvert \rceil$	1
$\mathsf{MultF}^{n,t}([a],[b])$ (amortized cost)	$2(\frac{2n}{n-t}+3)(n-1)\lceil \log \lvert\mathbb{F}\rvert \rceil$ $< 14(n-1)\lceil \log \lvert\mathbb{F}\rvert \rceil$	5
$\mathsf{Reveal}^{n,t}([a])$	$(n-1)\lceil \log \lvert\mathbb{F}\rvert \rceil$	1
$\mathsf{RevealInReshareF}^{n}(d, \langle a \rangle)$	$2(n-1)\lceil \log \lvert\mathbb{F}\rvert \rceil$	2
$\mathsf{DoubleRandomF}^{n,t}(\ell)$	$2\left\lceil \frac{\ell}{n-t} \right\rceil n(n-1)\lceil \log \lvert\mathbb{F}\rvert \rceil$ $< 4\ell(n-1)\lceil \log \lvert\mathbb{F}\rvert \rceil$	1

Table 1: Communication costs of n-party functionalities. The upper bounds given are for $t < n/2$.

2.6 Weak Comparison Functionality

We define a weak form of multiparty comparison functiontionality, $\mathcal{F}_{\mathsf{w\text{-}CMP}}^{p,k,n,t}$ (where k is an element in $\mathbb{F}_p$, n, t denotes the number of parties and corruption threshold). Here n parties $P_1, \cdots, P_n$ input their (n,t)-shares of some $0 \leq a < n$ and the functionality outputs the indicator bit comp, which is 1 iff $a \geq k$, to the leader P_1 and the other parties receive no output. We formally describe this functionality in Figure 2. We show two instantiations of this functionality, which offer different trade-offs to our communication costs. The details of these instantiations are in Section 5.2.

> There are n parties $P_1, \cdots, P_n$. All elements are considered over $\mathbb{F}_p$ such that $k < n < p$.
> **Inputs**: For each $i \in [n]$, P_i inputs its $(n,t)-$ share $[a]_i$ corresponding to some $0 \leq a < n$ and $[a]_i \in \mathbb{F}_p$.
> **Output**: Reconstruct the shares to get a, and if $a \geq k$, set $\mathsf{comp} = 1$, else set $\mathsf{comp} = 0$. Send comp to P_1. Other parties receive no output.

Fig. 2: Weak Comparison Functionality $\mathcal{F}_{\mathsf{w\text{-}CMP}}^{p,k,n,t}$

3 Multiparty PSI

We begin by formally defining the multiparty private set intersection functionality, $\mathcal{F}_{\mathsf{PSI}}^{n,m}$ in Figure 3 that computes the intersection of private sets of all the parties.

> There are n parties $P_1, \cdots, P_n$.
> **Inputs**: For each $i \in [n]$, P_i has a set X_i of size m.
> **Output**: Return $\cap_{i=1}^{n} X_i$ to each P_i.

Fig. 3: Private Set Intersection Functionality $\mathcal{F}_{\mathsf{PSI}}^{n,m}$

Parameters: There are n parties $P_1, \ldots, P_n$ with private sets of size m. Let $\beta = 1.28m, \sigma = \kappa + \lceil \log m \rceil + 3$ and $p > 2^\sigma$ is a prime. Additions and multiplications in the protocol are over $\mathbb{F}_p$.
Input: Each party P_i has input set $X_i = \{x_{i1}, \cdots, x_{im}\}$, where $x_{ij} \in \{0,1\}^\sigma$. Note that element size can always be made σ bits by first hashing the elements using an appropriate universal hash function.
Protocol:

1. **Pre-processing** (*Randomness generation (required for Step (4))*): $P_1, \cdots, P_n$ invoke the following multiparty functionalities.
   - $([r_1], \cdots, [r_\beta], \langle r_1 \rangle, \cdots, \langle r_\beta \rangle) \leftarrow \mathsf{DoubleRandomF}^{n,t}(\beta)$
   - $([s_1], \cdots, [s_\beta]) \leftarrow \mathsf{RandomF}^{n,t}(\beta)$
2. **Hashing**: Parties agree on hash functions $h_1, h_2, h_3 : \{0,1\}^\sigma \rightarrow [\beta]$.
   P_1 does stashless cuckoo hashing on X_1 using h_1, h_2, h_3 to generate Table_1 and inserts dummy elements into empty bins.
   For $i \in \{2, \cdots, n\}$, P_i does simple hashing of X_i using h_1, h_2, h_3 into Table_i, i.e., stores each $x \in X_i$ at locations $h_1(x), h_2(x)$ and $h_3(x)$. If the three locations are not distinct, dummy elements are inserted in bin with collision.
3. **Invoking the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality**: For each $i \in \{2, \cdots, n\}$, P_1 and P_i invoke the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality for $N = 3m$ as follows:
   - P_i is the sender with input $\{\mathsf{Table}_i[j]\}_{j\in[\beta]}$.
   - P_1 is the receiver with input $\{\mathsf{Table}_1[j]\}_{j\in[\beta]}$.
   - P_1 receives the outputs $\{y_{ij}\}_{j\in[\beta]}$ and P_i receives $\{w_{ij}\}_{j\in[\beta]}$.
4. **Evaluation**: For $j \in [\beta]$,
   - P_1 computes $\langle a_j \rangle_1 = \sum_{i=2}^{n} (-y_{ij} \mod p)$ and for $i \in \{2, \cdots, n\}$, P_i sets $\langle a_j \rangle_i = (w_{ij} \mod p)$.
   - For $i \in [n]$, P_i computes $\langle z_j \rangle_i = \langle a_j \rangle_i - \langle r_j \rangle_i$.
   - $P_1, \cdots, P_n$ compute $[z_j] \leftarrow \mathsf{RevealInReshareF}^n(t, \langle z_j \rangle)$.
   - For $i \in [n]$, P_i computes $[u_j]_i = [z_j]_i + [r_j]_i$.
   - $P_1, \cdots, P_n$ invoke the following multiparty functionalities.
     - $[v_j] \leftarrow \mathsf{MultF}^{n,t}([u_j], [s_j])$.
     - $v_j \leftarrow \mathsf{Reveal}^{n,t}([v_j])$.
5. **Output:** P_1 computes the intersection as $Y = \bigcup_{j\in[\beta]:v_j=0} \mathsf{Table}_1[j]$, permutes its elements and announces to all parties.

Fig. 4: MULTIPARTY PSI PROTOCOL

3.1 Multiparty PSI Protocol

Building blocks: Our protocol uses the weak-PSM functionality $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ (Section 2.4) and the multiparty functionalities from Section 2.5 (with n parties and corruption threshold t) as building blocks. We describe our protocol formally in Figure 4 and provide an overview below.

Protocol Overview: As discussed in protocol blueprint from Section 1.1, our mPSI protocol proceeds in two main phases. In the first phase (steps 2 and 3 in Figure 4), P_1 and P_i (for each $i \in [n]\backslash\{1\}$) execute a protocol such that for each element in P_1's set, they receive as output the same random value, if the element belongs to P_i's set, and otherwise each receive independent random values. In the second phase, all the parties execute a secure multiparty computation (steps 1 and 4 in Figure 4) such that for every element in the intersection, P_1 obtains a 0 value and otherwise learns a random value. We now explain the details of each phase below.

On input X_i from party P_i, for each $i \in [n]$, the protocol proceeds in the following steps. First, is the input independent *Pre-processing* step. Here, the parties generate the randomness required in the *Evaluation* step using the functionalities in Section 2.5. Note that the size of this randomness only depends on the size of the input sets and hence, can be generated independent of the inputs. In the second *Hashing* step, the parties store their input sets in their respective tables as follows: Let h_1, h_2, h_3 be the hash functions used to map elements into $\beta = 1.28m$ bins. Party P_1 hashes its elements into Table_1 using Cuckoo hashing with h_1, h_2, h_3 (see Section 2.2). Also, P_1 inserts a dummy element in empty bins. With this, note that each bin of Table_1 has exactly one element. Parties P_i for $i \in \{2, \ldots, n\}$ do simple hashing of X_i into Table_i, i.e., insert each element of X_i into three locations corresponding to h_1, h_2 and

h_3. If for some element these three locations are not distinct (due to collision of the hash values), dummy element is inserted into any bin (may be randomly picked). Each bin in Table_i can have arbitrary number of elements and in total (including dummies) each Table_i has $3m$ elements. To avoid false positives in the final intersection due to dummies being inserted, we set it up so that dummy elements are different from real elements and the dummy element of P_1 is different from dummy elements inserted by P_i for $i \in \{2, \ldots, n\}$.

In the third step, for each $i = 2, \cdots, n$, P_1 and P_i invoke the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality for $N = 3m$ with P_1 acting as a receiver with queries Table_1, and P_i acting as the sender with input sets Table_i. By the definition of $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$, for query j, P_1 and P_i receive the same random element if P_1's query, i.e., $\mathsf{Table}_1[j]$ belongs to P_i's bin/set, i.e., $\mathsf{Table}_i[j]$ and different random elements, otherwise. In the *Evaluation* step, all parties evaluate a circuit for each bin such that P_1's output for bin j is 0 if and only if $\mathsf{Table}_1[j]$ belongs to the intersection. The circuit is as follows: For each $j \in [\beta]$, P_1 adds the negation of the query outputs from its interaction with each P_i (for each $i = 2, \cdots, n$) in step 3 to get its additive share $\langle a_j \rangle_1$ and for each $i = 2, \cdots, n$, P_i sets its additive share $\langle a_j \rangle_i$ as its response from the same interaction of Step (3). Observe that, $a_j = 0$ if and only if P_1's element $\mathsf{Table}_1[j]$ belongs to the intersection (except with a small error probability as explained later). The next goal is to reveal $v_j = s_j \cdot a_j$ to P_1, where $s_j \in \mathbb{F}_p$ is uniformly random. This ensures that if a_j is 0 then v_j is still 0, else v_j is a uniform random element in $\mathbb{F}_p$ (except with small probability when $s_j = 0$) and hides a_j. To realize this, the parties convert the additive shares of a_j to $(n,t)-$ shares of a_j (denoted by $[u_j]$) and then invoke the multiplication functionality to multiply with a random s_j that is generated during the *Pre-processing* step. The values v_j are revealed to P_1 for each $j \in [\beta]$. In the final step P_1 sets $Y = \bigcup_{j \in [\beta]: v_j = 0} \mathsf{Table}_1[j]$, permutes the elements in Y (to hide the relative ordering of elements in Table_1) and sends it to all the other parties.

3.2 Correctness and Security Proof

Theorem 1. *The protocol in Figure 4 securely realizes* $\mathcal{F}^{n,m}_{\mathsf{PSI}}$ *in the* $\mathcal{F}$*-hybrid model, where* $\mathcal{F} = (\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$, $\mathsf{DoubleRandomF}^{n,t}$, $\mathsf{RandomF}^{n,t}, \mathsf{RevealInReshareF}^{n}, \mathsf{MultF}^{n,t}, \mathsf{Reveal}^{n,t})$, *against a semi-honest adversary corrupting* $t < n/2$ *parties.*

Proof. **Correctness.** Let $Y^* = \cap_{i \in [n]} X_i$ and the output of the protocol is denoted by Y. To prove correctness, we wish to show that $Y = Y^*$, with all but negligible probability. For the rest of the proof we assume that the Cuckoo hashing by P_1 succeeds, i.e., all elements in X_1 get inserted successfully in Table_1. For $\beta = 1.28m$, this happens with probability at least $1 - 2^{-41}$, as discussed in Section 2.2. Now, we prove the following two lemmata.

Lemma 1. $Y^* \subseteq Y$

Proof. Let $e = \mathsf{Table}_1[j] \in Y^*$. By the property of simple hashing, $e \in \mathsf{Table}_i[j]$ for all $i \in \{2, \cdots, n\}$. Now, by correctness of $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$, $y_{ij} = w_{ij}$ for all $i \in \{2, \cdots, n\}$. Then, using the reconstruction of additive secret sharing, $a_j = 0$, and $z_j = a_j - r_j$. Finally, by the correctness of the multiparty functionalities from Section 2.5, we have $u_j = 0 = v_j$. Hence, $e_j \in Y$.

Lemma 2. $Y \subseteq Y^*$*, with probability at least* $1 - 2^{-\kappa-1}$.

Proof. Suppose for some $j \in [\beta]$, let $e = \mathsf{Table}_1[j]$ be such that $e \in Y$ and $e \notin Y^*$. Since $e \in Y$, it holds that $v_j = 0$. Hence, by correctness of $\mathsf{MultF}^{n,t}$, either $u_j = 0$ or $s_j = 0$. The latter happens with probability $F_1 = p^{-1} < 2^{-\sigma}$. If $u_j = 0$, then it holds that $a_j = 0$. There are following two disjoint and exhaustive cases for e.

Case 1: $e \in X_1$: Since $e \notin Y^*$, there exists $i \in \{2, \ldots, n\}$ such that $e \notin X_i$. Using the fact that dummy elements are different from real elements, it implies that $e \notin \mathsf{Table}_i[j]$. Now, the probability that $a_j = 0$ when $e \notin \mathsf{Table}_i[j]$ for some i is bounded by $F_2 = 2^{-\sigma}$.

Case 2: $e \notin X_1$: That is, e is a dummy element inserted by P_1. Now, since dummy elements are different from real elements and are disjoint for P_1 and P_i for all $i \in \{2, \dots, n\}$, it holds that $e \notin \mathsf{Table}_i[j]$ for all $i \in \{2, \dots, n\}$. Hence, same as case 1, the probability that $a_j = 0$ is bounded by $F_2 = 2^{-\sigma}$.

Thus, the probability of false positive happening at bin j is upper bounded by $F = F_1 + F_2 < 2 \cdot 2^{-\sigma}$. Hence, taking a union bound on all bins, $Y \nsubseteq Y^*$ with probability at most $\beta \cdot F < \beta(2 \cdot 2^{-\sigma}) < 2^{-\kappa-1}$.

Hence, our protocol gives the correct output with probabililty at least $(1 - 2^{-41} - 2^{-\kappa-1}) \geq 1 - 2^{-\kappa}$ for $\kappa = 40$.

Security Proof. Let $C \subset [n]$ be the set of corrupted parties ($|C| = t < n/2$). We show how to simulate the view of C in the ideal world, given the input sets $X_C = \{X_j : j \in C\}$ and the output $Y = \cap_{j=1}^{n} X_j$. We consider two cases based on party P_1 being honest or corrupt.

- **Case 1** ($\mathbf{P_1 \notin C}$): In the pre-processing step, the parties run the functionalities $\mathsf{DoubleRandomF}^{n,t}$ and $\mathsf{RandomF}^{n,t}$ from [20]. The simulator can pick random r_j's and s_j's, generate their shares and give their t shares to the corrupted parties. The hashing step is local, and can be executed by the simulator using the inputs of the corrupted parties. In step 3, where the $\mathcal{F}_{\mathsf{w-PSM}}^{\beta,\sigma,N}$ functionality is executed by P_1 and P_i for each $i \in [n] \backslash \{1\}$, the corrupted parties C, only see the sender's views (since $P_1 \notin C$), $\{w_{ij}\}_{i \in C, j \in [\beta]}$, which can all be picked at random by the simulator (by the definition of $\mathcal{F}_{\mathsf{w-PSM}}^{\beta,\sigma,N}$). In step 4, besides the local computations, which can all be executed by the simulator, the parties call the functionalities $\mathsf{RevealInReshareF}^{n}$, $\mathsf{MultF}^{n,t}$ and $\mathsf{Reveal}^{n,t}$. The corrupted parties get at most t shares for values z_j, u_j, and v_j, for each $j \in [\beta]$. The simulator can pick the t shares of the z_j's as shares of some random value (by the security of secret sharing). Then, it adds the t shares of r_j's (from the pre-processing step) and the corresponding shares of z_j's to get the t shares of u_j's. Finally, it sets the t shares of the v_j's as shares of some random value (by the security of secret sharing) and sends the output Y to the corrupted parties.
- **Case 2** ($\mathbf{P_1 \in C}$): The simulation of the pre-processing step and the hashing step is exactly same as in Case 1. In step 3, where the $\mathcal{F}_{\mathsf{w-PSM}}^{\beta,\sigma,N}$ functionality is executed by P_1 and P_i for each $i \in [n]$, since $P_1 \in C$, the corrupted parties get the receiver's view, $\{y_{ij} : i \in \{2, \cdots, n\}, j \in [\beta]\}$, in addition to the sender's views, $\{w_{ij}\}_{i \in C, j \in [\beta]}$. For a corrupted P_i, the simulator picks a random $y_{ij} = w_{ij}$, if $\mathsf{Table}_1[j] \in \mathsf{Table}_i[j]$, else picks a random y_{ij} and w_{ij} independently (by the definition of $\mathcal{F}_{\mathsf{w-PSM}}^{\beta,\sigma,N}$ and since the simulator has both Table_1 and Table_i). For an honest P_i, the simulator can pick all y_{ij}'s at random (again by the definition of $\mathcal{F}_{\mathsf{w-PSM}}^{\beta,\sigma,N}$). Step 4 is simulated as follows: For all $j \in [\beta]$, give random z_j's and t shares of random value as shares of z_j to the adversary (using uniform randomness of r_j, z_j are random). Next, add t shares of r_j and t shares of z_j to compute t shares of u_j. Now, the simulator sets v_j to be 0 for all $j \in [\beta]$ such that $\mathsf{Table}_1[j] \in Y$, and v_j is uniformly random otherwise (since s_j are unformly random given t shares of the corrupt parties). It gives t shares of v_j as output of $\mathsf{MultF}^{n,t}$ and v_j as output of $\mathsf{Reveal}^{n,t}, \forall j \in [\beta]$.

3.3 Complexity

First, we note that our protocol makes $n-1$ invocations of weak-PSM functionality. With this and using linear complexity of n-party functionalities from Section 2.5, our total communication is linear in n (irrespective of the specific instantiation of weak-PSM used). In contrast, Kolesnikov *et al.* [43] makes nt calls to OPPRF functionality (which is a primitive stronger than $\mathcal{F}_{\mathsf{w-PSM}}^{\beta,\sigma,N}$ as shown in the instantiation of the same).

Concretely, instantiating using n-party functionalities in Section 2.5 and polynomial based instantiation of weak-PSM (that has the least communication), our protocol requires at most $m(n-1)(4.5\lambda + 35(\kappa + \lceil \log m \rceil) + 140)$ bits of communication. Its round complexity is 10. On the other hand, [43] requires communication of $m(nt + 2n - 1)(4.5\lambda + 46(\kappa + \lceil \log m \rceil))$ and 8 rounds.

In our protocol, as well as in [43], we can see that the communication cost of P_1, the leader, and P_i, for $i \in \{2, \cdots, n\}$, the clients, are different. Specifically, the client communication complexity of our protocol is $m(4.5\lambda + 64(\kappa + \lceil \log m \rceil) + 256)$. In comparison, [43] client communication complexity is $m(2t+3)(4.5\lambda + 46(\kappa + \lceil \log m \rceil))$.

For instance, consider a setting where $m = 2^{20}$, $\lambda = 128$ and $\kappa = 40$. For this setting our total communication cost and per- client communication cost are $6(t+2)/5$ times and $7(2t+3)/10$ times better than the corresponding costs of [43] respectively.

4 Multiparty Circuit PSI

The goal of multiparty *Circuit* PSI is to evaluate a symmetric function on the private set intersection of n parties. We formally define this functionality, $\mathcal{F}^{n,m,f}_{\mathsf{C-PSI}}$ in Figure 5.

There are n parties $P_1, \cdots, P_n$ and a function f. **Inputs**: For each $i \in [n]$, P_i has a set X_i of size m. **Output**: Return $f(\cap_{i=1}^{n} X_i)$ to each P_i.

Fig. 5: Circuit PSI Functionality $\mathcal{F}^{n,m,f}_{\mathsf{C-PSI}}$

4.1 Circuit PSI Protocol

Building blocks: Our protocol uses the two-party functionalities weak private set membership $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ (Sec. 2.4), equality test $\mathcal{F}^{\sigma}_{\mathsf{EQ}}$ (Sec. 2.3), boolean to arithmetic share conversion $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}$ (Sec. 2.3), and the n-party functionalities (from Sec. 2.5).

We consider standard multiparty functionality $\mathcal{F}_{\mathsf{MPC}}$ that is parameterized by a circuit C. The circuit C takes as inputs I_i from each P_i, for $i \in [n]$ and the functionality computes the circuit C on these inputs and returns $C(I_1, \cdots, I_n)$. In our construction, to evaluate a symmetric function f, we consider the circuit $C_{\beta,\sigma,p}$, which takes as inputs $\{[c_j]_i\}_{j\in[\beta]}$ from P_i for each $i \in [n]$ such that $c_j \in \mathbb{F}_p$ and $a_1, \ldots, a_\beta \in \{0,1\}^\sigma$ from P_1, computes $\{c_j\}_{j\in[\beta]}$ by reconstructing the shares, and computes $T = f(\bigcup_{j\in[\beta]:c_j=1} a_j)$.

We describe our protocol formally in Figure 6 and provide an overview below.

Protocol Overview. On input X_i from party P_i, for each $i \in [n]$, the protocol proceeds in eight steps: The first three steps of the protocol, namely the *Pre-processing, Hashing* and *Invoking the* $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ *functionality*, are the same as the first three steps of our multiparty PSI protocol (Figure 4). At the end of these steps, P_1 holds Table_1 of β bins containing one element each and other parties P_i's hold Table_i with β bins of arbitrary size. Moreover, for each $i \in \{2, \ldots, n\}$ and $j \in [\beta]$, P_1 holds $y_{ij} \in \{0,1\}^\sigma$ and P_i holds $w_{ij} \in \sigma$ such that $y_{ij} = w_{ij}$ if $\mathsf{Table}_1[j] \in \mathsf{Table}_i[j]$ (except with negligible probability). Now, in the next step, the parties check whether this equality holds or not. Formally, in the fourth step, for each $i \in \{2, \cdots, n\}$, parties P_1 and P_i invoke the $\mathcal{F}^{\sigma}_{\mathsf{EQ}}$ functionality with inputs y_{ij} and w_{ij}, respectively and receive as outputs, the boolean shares[7].

Rest of the steps are executed for each bin j independently. In the fifth step, for each $i \in \{2, \cdots, n\}$, parties P_1 and P_i invoke the $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}$ functionality to convert the boolean shares to additive shares over $\mathbb{F}_p$, where $p > n$ is a prime. Next, in step (6), parties convert these additive shares between P_1 and

[7] We note here that these four steps of our protocol together follow the blueprint of executing a circuit PSI protocol [54, 55, 38, 15, 24, 56] between P_1 and P_i (for each $i \in \{2, \cdots, n\}$), while ensuring a consistent mapping of elements of P_1 (via Cuckoo hashing into Table_1) across all instantiations. To explicitly spell out this consistent hashing and for ease of exposition, we make a whitebox use of the circuit-PSI blueprint from [55] and describe these steps as well.

Parameters: n parties $P_1, \ldots, P_n$ with private sets of size m. Let $\beta = 1.28m, \sigma = \kappa + \lceil \log m \rceil + 2$. Additions and multiplications in the protocol are over $\mathbb{F}_p$, where $p > n$ is a prime. Let $d = \lceil \log p \rceil - 1$ and $b_d b_{d-1} \cdots b_1 b_0$ denote the binary representation of $p - 1$. Let $\mathcal{S} = \{i \in (\{0\} \cup [d]) : b_i = 1\}$ and $\mathsf{ind}_k, \ldots, \mathsf{ind}_1, \mathsf{ind}_0$ be the ascending order of elements in S, where $k = |\mathcal{S}| - 1$.
Input: Each party P_i has input set $X_i = \{x_{i1}, \cdots, x_{im}\}$, where $x_{ij} \in \{0,1\}^\sigma$. Note that element size can always be made σ bits by first hashing the elements using an appropriate universal hash function.

Protocol:

1. **Pre-processing**: $P_1, \cdots, P_n$ invoke $\mathsf{DoubleRandomF}^{n,t}(\beta)$ to get $([r_1], \cdots, [r_\beta], \langle r_1 \rangle, \cdots, \langle r_\beta \rangle)$.
2. **Hashing**: Parties agree on hash functions $h_1, h_2, h_3 : \{0,1\}^\sigma \to [\beta]$.
   P_1 does stashless cuckoo hashing on X_1 using h_1, h_2, h_3 to generate Table_1 and inserts random elements into empty bins.
   For $i \in \{2, \cdots, n\}$, P_i does simple hashing of X_i using h_1, h_2, h_3 into Table_i, i.e., stores each $x \in X_i$ at locations $h_1(x), h_2(x)$ and $h_3(x)$. If the three locations are not distinct, random dummy values are inserted in bin with collision.
3. **Invoking the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality:** For each $i \in \{2, \cdots, n\}$, P_1 and P_i invoke the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality for $N = 3m$ as follows:
   - P_i is the sender with inputs $\{\mathsf{Table}_i[j]\}_{j \in [\beta]}$ and P_1 is the receiver with inputs $\{\mathsf{Table}_1[j]\}_{j \in [\beta]}$.
   - P_i receives the outputs $\{w_{ij}\}_{j \in [\beta]}$ and P_1 receives $\{y_{ij}\}_{j \in [\beta]}$.
4. **Invoking the $\mathcal{F}^{\sigma}_{\mathsf{EQ}}$ functionality:** For each $i \in \{2, \cdots, n\}$ and for each $j \in [\beta]$, P_1 and P_i invoke the $\mathcal{F}^{\sigma}_{\mathsf{EQ}}$ functionality as follows: P_1 and P_i send their inputs y_{ij} and w_{ij}, resp., and receive boolean shares $\langle eq_{ij} \rangle^B_1$ and $\langle eq_{ij} \rangle^B_i$ resp., as outputs.
5. **Invoking the $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}$ functionality:** For each $i \in \{2, \cdots, n\}$ and for each $j \in [\beta]$, P_1 and P_i invoke the $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}$ functionality as follows: P_1 and P_i send their inputs $\langle eq_{ij} \rangle^B_1$ and $\langle eq_{ij} \rangle^B_i$, resp., and receive the additive shares $\langle f_{ij} \rangle_1$ and $\langle f_{ij} \rangle_i$ resp., as outputs.
6. **Converting to (n, t) shares:** For each $j \in [\beta]$,
   - P_1 computes $\langle a_j \rangle_1 = \sum_{i=2}^n \langle f_{ij} \rangle_1$ and for each $i \in \{2, \cdots, n\}$, P_i sets $\langle a_j \rangle_i = \langle f_{ij} \rangle_i$.
   - Compute $\langle z_j \rangle = \langle a_j \rangle - \langle r_j \rangle$
   - $[z_j] \leftarrow \mathsf{RevealInReshareF}^n(t, \langle z_j \rangle)$
   - Compute $[u_j] = [z_j] + [r_j]$.
7. **Computing shares of intersection:** For each $j \in [\beta]$,
   - Compute $[v_j^{(0)}] = [u_j] - n + 1$.
   - For each $i \in [d]$, compute $[v_j^{(i)}] \leftarrow \mathsf{MultF}^{n,t}([v_j^{(i-1)}], [v_j^{(i-1)}])$.
   - Let $[q_j^{(0)}] = [v_j^{(\mathsf{ind}_0)}]$.
   - For $i \in [k]$, compute $[q_j^{(i)}] \leftarrow \mathsf{MultF}^{n,t}([q_j^{(i-1)}], [v_j^{(\mathsf{ind}_i)}])$.
   - Compute $[c_j] = 1 - [q_j^{(k)}]$.
8. **Computing the circuit $C_{\beta,\sigma,p}$:** The parties invoke the $\mathcal{F}_{\mathsf{MPC}}$ functionality parameterized $C_{\beta,\sigma,p}$ by as follows:
   - P_1 inputs $\{[c_j]_1\}_{j \in [\beta]}$ and Table_1. For $i \in \{2, \cdots, n\}$, P_i inputs $\{[c_j]_i\}_{j \in [\beta]}$.
   - All parties receive the output T.

Fig. 6: CIRCUIT PSI PROTOCOL

P_i for $i \in [n] \setminus \{1\}$ to (n,t)-shares of values u_j such that u_j denotes the number of parties in $[n]\setminus\{1\}$ that have the element stored at $\mathsf{Table}_1[j]$. In Step (7), the task is to securely compute shares of whether $u_j = n-1$ or not. Let $v_j = u_j - (n-1)$. Now, using property of fields with prime order, $v_j = 0$ (and hence, $u_j = n-1$) if and only if $v_j^{p-1} = 0$. For this, parties first compute shares of $v_j^{2^i}$ for $i \in \{0\} \cup [d]$ where $d = \lceil \log p \rceil - 1$ (requiring d calls to $\mathsf{MultF}^{n,t}$) and then multiply shares of appropriate powers of v_j (requiring at most d calls to $\mathsf{MultF}^{n,t}$). Then, parties locally compute shares of $c_j = 1 - v_j^{p-1}$. It holds that c_j is 1 if and only if $u_j = n-1$.

Finally, parties invoke $\mathcal{F}_{\mathsf{MPC}}$ functionality for circuit $C_{\beta,\sigma,p}$ (described above) with shares of c_j and $\mathsf{Table}_1[j]$, for all $j \in [\beta]$.

4.2 Correctness and Security Proof

Theorem 2. *The protocol in Figure 6 securely realizes $\mathcal{F}^{n,m,f}_{\mathsf{C-PSI}}$ in the $\mathcal{F}$-hybrid model, where $\mathcal{F} = (\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}, \mathsf{DoubleRandomF}^{n,t}, \mathsf{RandomF}^{n,t}, \mathsf{RevealInReshareF}^{n}, \mathsf{MultF}^{n,t}, \mathsf{Reveal}^{n,t})$, against a semi-honest adversary corrupting $t < n/2$ parties.*

Proof. **Correctness:** Let $Y = \bigcup_{j\in[\beta]:c_j=1} \mathsf{Table}_1[j]$ and $Y^* = \cap_{i=1}^{n} X_i$. For statistical correctness, we need to show that $T = f(Y^*)$ with all but negligible probability in κ. By correctness of the $\mathcal{F}_{\mathsf{MPC}}$ (parameterized by the circuit $C_{\beta,\sigma,p}$) functionality, whenever $Y = Y^*$ we have $T = C(\mathsf{Table}_1, \{c_j\}_{j\in[\beta]}) = f(Y) = f(Y^*)$. So it suffices to upper bound the probability of $Y^* \neq Y$. For the rest of the proof we assume that cuckoo hashing by P_1 succeeds which happens with probability atmost $1 - 2^{-41}$.

As we will see later, steps 4–7 do not lead to correctness error of our protocol. We make a few observations about these steps below, that will be used in both lemmata that follow. For each $j \in [\beta]$,

- (Step 4) By correctness of $\mathcal{F}^{\sigma}_{\mathsf{EQ}}$, for each $i \in [n] \setminus \{1\}$, eq_{ij} equals 1 when $y_{ij} = w_{ij}$ and 0 otherwise.
- (Step 5) By correctness of $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}$, for each $i \in [n] \setminus \{1\}$, $f_{ij} = eq_{ij}$.
- (Step 6) Using reconstruction of additive secret sharing, $a_j = \sum_{i=2}^{n} f_{ij} < n$. Moreover, by linearity of (n,t)-secret sharing and correctness of $\mathsf{RevealInReshareF}^{n}$, $u_j = a_j$.
- (Step 7) First, $v_j^{(0)} = u_j - (n-1)$. Also, let $v_j = v_j^{(0)}$. Next, by correctness of $\mathsf{MultF}^{n,t}$ for every $i \in [d]$, it holds that $v_j^{(i)} \equiv (v_j)^{2^i}$ and $q_j^{(k)} = v_j^{p-1}$. Finally, $c_j = 1 - q_j^{(k)}$.

Now, using the property of finite fields, we get that $q_j^{(k)} = 0$, and consequently, $c_j = 1$, if and only if $v_j = 0$. Using above properties, we get that $c_j = 1$ if and only if $u_j = n-1 = a_j$. This in turn implies that $eq_{ij} = 1$ for all $i \in \{2, \ldots, n\}$. To conclude, we have shown that $c_j = 1$ if and only if $eq_{ij} = 1$ for all $i \in \{2, \ldots, n\}$ and this is what we use below. We now prove the following two lemmata.

Lemma 3. *$Y^* \subseteq Y$.*

Proof. Let $e = \mathsf{Table}_1[j] \in Y^*$. Therefore, for each $i \in \{2, \cdots, n\}$, by the definition of simple hashing $e \in \mathsf{Table}_i[j]$. Hence by correctness of $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ guarantees that $y_{ij} = w_{ij}$ (and hence $eq_{ij} = 1$) for each $i \in \{2, \cdots, n\}$. Using what we show above, we get that in this case $c_j = 1$ and hence, $e \in Y$.

Lemma 4. *$Y \not\subseteq Y^*$ with probability atmost $1 - 2^{-\kappa-1}$.*

Proof. Suppose $e = \mathsf{Table}_1[j] \notin Y^*$. Since $e \notin Y^*$, let $i^* \in \{2, \cdots, n\}$ be such that $e \notin X_{i^*}$. We now show that $e \notin \mathsf{Table}_{i}*[j]$ with the following disjoint and exhaustive scenarios.

- Suppose $e \in X_1$. Since $e \notin X_{i^*}$ and any dummy elements inserted by P_{i^*} are distinct from real elements, it holds that $e \notin \mathsf{Table}_{i^*}[j]$.

- Suppose $e \notin X_1$. Since dummy elements of P_i^* and real elements are distinct from dummy elements of P_1, it holds that $e \notin \mathsf{Table}_{i^*}[j]$.

Probability that $y_{i^*j} = w_{i^*j}$ (and hence eq_{i^*j}=1) when $e \notin \mathsf{Table}_{i^*}$ is atmost $2^{-\sigma}$. Recall that $c_j \neq 1$ when $eq_{i^*j} \neq 1$. Therefore, probability that $c_j = 1$ is atmost $2^{-\sigma}$. Note that this is the probability that $\mathsf{Table}_1[j] \in Y \backslash Y^*$. By union bound over all bins it holds that with probability atleast $1 - \beta 2^{-\sigma}$ the set $Y \backslash Y^*$ is empty.

Hence, except with failure probability atmost $2^{-\kappa}$ (that includes the probability of cuckoo hashing failure), the output of the protocol is correct, for $\kappa = 40$.

We give a detailed security proof in Appendix B.

4.3 Circuit PSI Complexity

We discuss the communication complexity of our protocol in Figure 6 for $\mathcal{F}^{n,m,f}_{\mathsf{C-PSI}}$ functionality. In the protocol, instantiate $\mathcal{F}^{\sigma}_{\mathsf{EQ}}$, $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}$ and multiparty functionalities as described in sections 2.3, 2.3 and 2.5, respectively. Here, we instantiate the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality with the polynomial-based batch-OPPRF (Sec. 2.4) that gives least concrete communication. We pick the smallest prime $p > n$, and hence, we can assume that $\lceil \log p \rceil \leq \lceil \log n \rceil + 1$. Also, recall that $\beta = 1.28m$.

We split the communication of the protocol into two parts. 1) Steps 2–5 where P_1 interacts with each P_i for $i \in [2, \ldots, n]$ separately. 2) Steps 1, 6, and 7 where parties run n-party functionalities. In the first part, protocol invokes $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$, $\mathcal{F}^{\sigma}_{\mathsf{EQ}}$, $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}$ functionalities $(n-1)$, $\beta(n-1)$, and $\beta(n-1)$ times respectively. Concretely, communication cost of this part is at most $m(n-1)(\lambda\sigma + 5.8\lambda + 14\sigma + 1.28\lceil \log n \rceil)$, where $\sigma = \kappa + \lceil \log m \rceil + 2$. In the second part, the protocol invokes $\mathsf{RandomF}^{n,t}(\beta)$ once and $\mathsf{Reveal}^{n,t}$ β times. It also makes at most $2\beta\lceil \log n \rceil$ calls to the $\mathsf{MultF}^{n,t}$ functionality. The concrete cost of this part is at most $m(n-1)(36(\lceil \log n \rceil)^2 + 40\lceil \log n \rceil)$.

Total cost of our protocol is simply the sum of the cost of both parts and is dominated by $2mn(\lambda\kappa + 36(\log n)^2)$. Round complexity of the protocol is atmost $4\lceil \log n \rceil + \lceil \log \sigma \rceil + 8$.

4.4 A Linear Circuit PSI Protocol

We also show how to obtain a circuit PSI protocol, with asymptotically better communication complexity (linear in n) than the protocol in Figure 6. This protocol follows a similar blueprint to Figure 6, but works over a different field $\mathbb{F}_p$ whose size is independent of n. In particular, it works over a prime field where $p > 2^{\sigma}$.
Parameters: n parties $P_1, \ldots, P_n$ with private sets of size m. Let $\beta = 1.28m, \sigma = \kappa + \lceil \log m \rceil + 3$. Additions and multiplications in the protocol are over $\mathbb{F}_p$, where $p > 2^{\sigma}$ is a prime. Let $d = \lceil \log p \rceil - 1$ and $b_d b_{d-1} \cdots b_1 b_0$ denote the binary representation of $p-1$. Let $\mathcal{S} = \{i \in (\{0\} \cup [d]) : b_i = 1\}$ and $\mathsf{ind}_k, \ldots, \mathsf{ind}_1, \mathsf{ind}_0$ be the ascending order of elements in S, where $k = |\mathcal{S}| - 1$.
Input: Each party P_i has input set $X_i = \{x_{i1}, \cdots, x_{im}\}$, where $x_{ij} \in \{0,1\}^{\sigma}$.
Protocol: The protocol executes steps (1)-(3) of the circuit PSI protocol from Figure 6. Then, it executes the first four sub-steps of step (4) of the mPSI protocol from Figure 4. At the end of step (4), the parties have shares $[u_j]$, for each $j \in [\beta]$. Now, execute step (7) and (8) of the circuit PSI protocol (figure 6), albeit over a different field compared to what is used in Figure 6, while setting $[v_j^{(0)}] = [u_j]$ for each $j \in [\beta]$ in step (7).

The correctness and security proof of the protocol combines the analysis of appropriate steps in mPSI and circuit PSI and is straightforward.

Complexity. The above protocol has a total communication cost of atmost $m(n-1)(4.5\lambda + 36\sigma^2 + 83\sigma + 36)$ and a round complexity of $8 + 2\sigma$, where recall that $\sigma = \kappa + \log m + 3$. Note, that while asymptotically, this solution has lower communication, its concrete communication cost is more than that of the circuit PSI in Figure 6; hence, we choose to only implement that protocol.

5 Quorum Private Set Intersection

The goal of *quorum* private set intersection is to compute the set of all elements which are present in the leader P_1's private set and in at least k other parties' sets, where k denotes the quorum threshold (excluding P_1), and output it to P_1 only. We begin by formally defining the quorum private set intersection functionality $\mathcal{F}^{n,m,k}_{\mathsf{QPSI}}$ in Figure 7. Observe that when $k = n - 1$, (intuitively) this is simply multiparty private set intersection and reduces to the functionality of Figure 3.

There are n parties $P_1, \cdots, P_n$, where P_1 is the leader and $k \in [n-1]$ denotes the quorum threshold.
Input: For each $i \in [n]$, P_i inputs a set X_i of size m.
Output: For each $x \in X_1$, let $q_x = |\{i : x \in X_i \text{ for } i \in \{2, \cdots, n\}\}|$. Then, output $Y^* = \{x \in X_1 : q_x \geq k\}$ to P_1.

Fig. 7: Quorum PSI Functionality $\mathcal{F}^{n,m,k}_{\mathsf{QPSI}}$

5.1 Quorum PSI Protocol

Building blocks: Our protocol uses the two-party functionalities weak private set membership $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ (Section 2.4), equality test $\mathcal{F}^{\sigma}_{\mathsf{EQ}}$ (Sec. 2.3), boolean to arithmetic share conversion $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}$ (Section 2.3), the n-party functionalities from Section 2.5, and the weak comparison functionality $\mathcal{F}^{p,k,n,t}_{\mathsf{w\text{-}CMP}}$ (Section 2.6) in the honest majority setting. In Section 5.2 we provide two weak comparison protocols that realize $\mathcal{F}^{p,k,n,t}_{\mathsf{w\text{-}CMP}}$ and discuss their trade-offs.

Overview. Since the protocol follows most of the steps of circuit-PSI protocol from Section 4, we provide an in-text description of the quorum PSI protocol highlighting only the changes (with full description in Figure 9). At a high level, for each $j \in [\beta]$, after obtaining (n,t)-shares of value u_j that denotes the number of P_i's that contain the element of P_1 stored at $\mathsf{Table}_1[j]$, they invoke an n-party weak comparison protocol that compares the value of u_j with k and outputs the result to P_1. We now provide more details.

Parameters: There are n parties $P_1, \ldots, P_n$ with private sets of size m and $1 < k \leq n-1$ is the quorum. Let $\beta = 1.28m, \sigma = \kappa + \lceil \log m \rceil + \lceil \log n \rceil + 2$. Additions and multiplications in the protocol are over $\mathbb{F}_p$, where p is a prime (larger than n) that depends on specific instantiation of $\mathcal{F}_{\mathsf{w\text{-}CMP}}$.

Input: Each party P_i has input set $X_i = \{x_{i1}, \cdots, x_{im}\}$, where $x_{ij} \in \{0,1\}^\sigma$. Element size can always be made σ bits by first hashing the elements using an appropriate universal hash function.

Protocol: The protocol executes steps (1)-(6) of the circuit-PSI protocol from Figure 6. After this step, for each $j \in [\beta]$, parties hold $[u_j]$. Then, parties $P_1, \cdots, P_n$ invoke $\mathcal{F}^{p,k,n,t}_{\mathsf{w\text{-}CMP}}$ with P_i's input being $[u_j]_i$ for $i \in [n]$ and P_1 learns c_j as output.

P_1 computes the quorum intersection as $Y = \bigcup_{j \in [\beta] : c_j = 1} \mathsf{Table}_1[j]$.

Complexity. Based on the two intantiations of $\mathcal{F}^{p,k,n,t}_{\mathsf{w\text{-}CMP}}$ described in the next section, we have two protocols for quorum PSI and we discuss their complexities in Section 5.3.

Theorem 3. *The protocol given above securely realizes $\mathcal{F}^{n,m,k}_{\mathsf{QPSI}}$ in the $\mathcal{F}$-hybrid model, where $\mathcal{F} = (\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}, \mathcal{F}^{\sigma}_{\mathsf{EQ}}, \mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}, \mathcal{F}^{p,k,n,t}_{\mathsf{w\text{-}CMP}}, \mathsf{DoubleRandomF}^{n,t}, \mathsf{RevealnReshareF}^{n}, \mathsf{MultF}^{n,t}, \mathsf{Reveal}^{n,t})$, against a semi-honest adversary corrupting $t < n/2$ parties.*

We give a complete proof of the above theorem in Appendix C.

5.2 Weak Comparison Protocols

In this section, we describe two protocols realizing the weak comparison functionality $\mathcal{F}_{\textsf{w-CMP}}^{p,k,n,t}$ (Section 2.6), w-CMP1 and w-CMP2 and discuss their trade-offs in Section 5.2.

Weak Comparison Protocol w-CMP1 This protocol uses the multiparty functionalities in Section 2.5 (with n parties and corruption threshold t) as building blocks.
On input, the $(n,t)-$ shares $[a]_i$ from each P_i, for $i \in [n]$ (where $0 \le a < n$ and $a \in \mathbb{F}_p$), the protocol proceeds as follows: For $k \ge n/2$, consider the polynomial $\psi(x) = (x-k)\cdot(x-(k+1))\cdots(x-n)$, of degree $n-k+1$, that satisfies the following property: $\psi(x) = 0$ for all $n > x \ge k$. Similarly, for $k < n/2$, consider the polynomial $\psi(x) = x\cdot(x-1)\cdot(x-2)\cdots(x-(k-1))$, of degree k, that satisfies the following property: $\psi(x) = 0$ for all $0 \le x < k$. The protocol takes as input $[a]$ and uses the $\textsf{MultF}^{n,t}$ funcitonality to evaluate $[\psi(a)\cdot s]$, for random $s \in \mathbb{F}_p$. Now, P_1 recovers $\psi(a)\cdot s$, which is 0 whenever $\psi(a) = 0$ and is random, otherwise (hiding $\psi(a) \neq 0$). By the property of ψ, P_1 gets the required comparison bit comp. We formally describe the protocol in Figure 8.

Parameters: There are n parties $P_1, \cdots, P_n$ with $(n,t)-$ shares $[a]$, of $a \in \mathbb{F}_p$ and $a < n$. Here, p, n, t and k are such that p is a prime, $p > n > k$ and $n > 2t$. Additions and multiplications in the protocol are over $\mathbb{F}_p$.
Define the polynomial ψ (publicly known to all parties):

$$\psi(x) = \begin{cases} (x-k)\cdot(x-(k+1))\cdots(x-n)\,, & \text{if } k \ge \frac{n}{2} \\ x\cdot(x-1)\cdot(x-2)\cdots(x-(k-1))\,, & \text{if } k < \frac{n}{2} \end{cases}$$

Input: For each $i \in [n]$, P_i inputs its $(n,t)-$ share $[a]_i$.
Protocol:

1. **Pre-processing:** $P_1, \cdots, P_n$ invoke:
   $[s] \leftarrow \textsf{RandomF}^{n,t}(1)$.
2. **Evaluating the polynomial:** $P_1, \cdots, P_n$ invoke the following functionalities:
   - On input $[a]$, invoke $\textsf{MultF}^{n,t}$ to compute all the required $[a^i]$, followed by scalar multiplications and additions to compute $[\psi(a)]$.
   - $[v] \leftarrow \textsf{MultF}^{n,t}([\psi(a)], [s])$.
   - $v \leftarrow \textsf{Reveal}^{n,t}([v])$.

Output: If $k \ge n/2$, P_1 sets $\textsf{comp} = 1$, if $v = 0$ and $\textsf{comp} = 0$, otherwise. If $k < n/2$, P_1 sets $\textsf{comp} = 0$ if $v = 0$ and $\textsf{comp} = 1$, otherwise. Other parties get no output.

Fig. 8: WEAK COMPARISON PROTOCOL I

Theorem 4. *The protocol in Figure 8 securely realizes $\mathcal{F}_{\textsf{w-CMP}}^{p,k,n,t}$ in the $\mathcal{F}$-hybrid model, where $\mathcal{F} =$ ($\textsf{RandomF}^{n,t}$, $\textsf{MultF}^{n,t}$, $\textsf{Reveal}^{n,t}$), against a semi-honest adversary corrupting $t < n/2$ parties.*

We give a complete proof of Theorem 4 in Appendix D.1.

Weak Comparison Protocol w-CMP2 This protocol is a slight modification of the comparison protocol from [11]. The main idea of their comparison protocol is as follows: For $0 \le a, k < n$, $a \ge k$ iff $\left\lfloor \frac{(a-k)}{2^\gamma} \right\rfloor = 0$ (where $\gamma = \lceil \log n \rceil + 1$). Hence, the protocol takes the $(n,t)-$ shares of a and evaluates the $(n,t)-$ shares of $\left\lfloor \frac{(a-k)}{2^\gamma} \right\rfloor$. This protocol invokes the multiparty functionalities $\textsf{MultF}^{n,t}$, $\textsf{RandomF}^{n,t}$ and $\textsf{Reveal}^{n,t}$. Corresponding to the instantiations of these funcitonalities used in [11], their protocol has an n^2 factor in the communication complexity. Instead, we use the instantiations from [20] for these functionalities, which reduces the communication complexity of their protocol. For completeness, we give the full protocol, which is modified (and simplified) appropriately to instantiate $\mathcal{F}^{p,k,n,t}_{\mathsf{w\text{-}CMP}}$, in Appendix D.2.

Trade-offs between w-CMP1 and w-CMP2 We first discuss the communication complexity and rounds of both protocols. Multiparty functionalities in both the protocols are instantiated as referred in Sec. 2.5. Since these instantiations provide good amortized complexities, we give amortized costs of both the protocols.

The amortized communication cost of w-CMP1 is atmost $(14k' + 3)(n-1)(\lceil \log n \rceil + 1)$ and the round complexity is $4 + 2k'$, when we set $\lceil \log p \rceil = \lceil \log n \rceil + 1$ and $k' = \min\{k, n-k+1\}$.While for w-CMP2, the (expected[8]) communication complexity is $20(n-1)\lceil \log 2n \rceil(\kappa + \lceil \log 2n \rceil)^2$, when we set $\lceil \log p \rceil = \kappa + \lceil \log n \rceil + 2$. The expected round complexity is $9 + 2\lceil \log n \rceil$.

We now discuss trade-offs between the two comparison protocols. Complexity of w-CMP2 protocol is independent of k, in contrast to w-CMP1 protocol's dependence on k. Hence, theoretically, for large values of k', the communication complexity and round complexity of w-CMP2 is better than w-CMP1. However, for practical setting of $k' < n < 512$, the concrete communication of w-CMP1 is better than that of w-CMP2. For any $\lceil \log n \rceil + 5 < k'$, the round complexity of w-CMP2 is better than that of w-CMP1.

5.3 Quorum PSI Complexity

We instantiate the $\mathcal{F}^{\sigma}_{\mathsf{EQ}}, \mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}, \mathsf{DoubleRandomF}^{n,t}, \mathsf{Reveal}^{n,t}$ functionalities as specified in sections 2.3, 2.3 and 2.5. We instantiate the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality using the polynomial-based batch OPPRF. Let Quorum-I and Quorum-II denote instantiations of $\mathcal{F}^{n,m,k}_{\mathsf{QPSI}}$ when $\mathcal{F}^{p,k,n,t}_{\mathsf{w\text{-}CMP}}$ is instantiated with w-CMP1 and w-CMP2 respectively.

Our protocol, in total, calls the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$, $\mathcal{F}^{\sigma}_{\mathsf{EQ}}$, $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}$, $\mathsf{RandomF}^{n,t}$, $\mathsf{Reveal}^{n,t}$ and $\mathcal{F}^{p,k,n,t}_{\mathsf{w\text{-}CMP}}$ functionalities $(n-1)$, $\beta(n-1)$, $\beta(n-1)$, 1, β and β times respectively, where $\beta = 1.28m$. Let $k' = \min\{k, n-k+1\}$. Recall that $\sigma = \kappa + \lceil \log m \rceil + \lceil \log n \rceil + 2$. We first give the costs of the steps common to Quorum-I and Quorum-II.

- Steps (2)-(5) cost less than $m(n-1)(\lambda\sigma + 5.8\lambda + 14\sigma + 1.28\lceil \log p \rceil)$.
- Steps (1) and (6) contribute atmost $8m(n-1)\lceil \log p \rceil$.

The total cost of w-CMP1 executions by Quorum-1 is atmost $m(n-1)(18k'(\lceil \log n \rceil + 1) + 4\lceil \log n \rceil)$. Therefore, the concrete communication of Quorum-I is atmost $m(n-1)(\lambda\sigma + 5.8\lambda + 14\sigma + 18k'(\lceil \log n \rceil + 1) + 14\lceil \log n \rceil)$, when we set $\lceil \log p \rceil = \lceil \log n \rceil + 1$. The round complexity of Quorum-I is atmost $10 + \lceil \log \sigma \rceil + 2k'$.

The (expected) total cost of w-CMP2 executions by Quorum-II is atmost $26m(n-1)(\lceil \log n \rceil + 1)(\kappa + \lceil \log n \rceil + 1)^2$. Therefore, (expected) concrete communication of Quorum-II is atmost $m(n-1)(\lambda\sigma + 5.8\lambda + 14\sigma + 27(\lceil \log n \rceil + 1)(\kappa + \lceil \log n \rceil + 1)^2)$, when we set $p = \kappa + \lceil \log n \rceil + 2$. The (expected) round complexity of Quorum-II is atmost $10 + \lceil \log \sigma \rceil + 2\lceil \log n \rceil$.

6 Implementation and Performance

In this section, we discuss the performance of our mPSI (multiparty PSI) protocols when instantiated using the three different instantiations of weak-PSM (see Section 2.4), as well as the Circuit PSI and qPSI

[8] One of the underlying sub-protocol uses rejection sampling for randomness that incurs repeated executions with small probability, namely, $1/p$.

n, t	4, 1			5, 2			10, 4			15, 7		
m	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
KMPRT	7.2	114.1	2057.7	13.4	211.2	3805.4	44.7	706.2	12730.4	103.4	1635.4	29487.9
Protocol A	3.2	49.4	790.2	4.6	72.7	1162.8	12.3	192.4	3077.2	22.5	353.4	5652.9

Table 2: Total communication in MB of mPSI protocols: KMPRT [43] and Protocol A.

n, t	4, 1			5, 2			10, 4			15, 7		
m	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
KMPRT	3.3	51.9	935.2	4.9	77.8	1402.0	8.3	131.7	2373.5	13.1	207.5	3741.0
Protocol A	1.3	19.9	318.0	1.5	23.3	372.6	2.0	30.8	492.1	2.4	38.8	620.1

Table 3: Client communication in MB of mPSI protocols: KMPRT [43] and Protocol A.

(quorum PSI) protocols when instantiated using relaxed batch OPPRF [12]. Let Protocol A, Protocol B and Protocol C denote our mPSI protocol when instantiated with polynomial-based batch OPPRF [55], table-based OPPRF [43] and relaxed batch OPPRF [12] respectively. We compare the performance of our mPSI protocols with the state-of-the-art mPSI protocol in literature [43].

Protocol Parameters. We set statistical security parameter κ=40 and computational security parameter λ=128. From the correctness analysis in proof of Theorem 1, we note that we need failure probability of atmost 2^{-41} in Cuckoo hashing at step 2 of mPSI protocol in Figure 4. Similar to [43, 57, 55, 12], we use the empirical analysis to instantiate the parameters of Cuckoo hashing scheme in the stashless setting. Based on the analysis given in Section 2.2, we instantiate cuckoo hashing with $\beta = 1.28m$ for $K = 3$. Based on Theorem 1, we set size of elements $\sigma = \kappa + \lceil \log m \rceil + 3$ to achieve statistical security of κ bits. Hence, the minimum element size σ required in mPSI protocol to ensure that the failure probability of the overall protocol is at most 2^{-40} is 55, 59 and 63 for input set size 2^{12}, 2^{16} and 2^{20} respectively. In the implementation of step 4 (see Figure 4) of mPSI protocol for input set size 2^{12} and 2^{16}, we perform arithmetic over prime field where the prime is the Mersenne prime $2^{61} - 1$. For input set size 2^{20}, we choose the prime field with Mersenne prime $2^{127} - 1$ for the LAN setting; for WAN setting we choose the Galois Field over an irreducible polynomial where each element is represented in 64 bits. This is due to compute vs communication trade-offs between the two fields.

Based on correctness analysis, we set $\sigma = \kappa + \lceil \log m \rceil + \lceil \log n \rceil + 2$ for our Circuit PSI and qPSI protocols, i.e., the maximum of the minimum element size required by these two protocols.

LAN Setting												
n, t	4, 1			5, 2			10, 4			15, 7		
m	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
KMPRT	0.28	2.47	41.30	0.39	4.03	65.43	0.67	6.77	98.04	1.40	13.32	193.90
Ours	0.23 (**B**)	1.60 (**B**)	23.80 (**C**)	0.23 (**B**)	1.66 (**B**)	25.48 (**C**)	0.31 (**B**)	2.48 (**B**)	31.45 (**C**)	0.44 (**B**)	3.27 (**C**)	39.45 (**C**)

WAN Setting												
n, t	4, 1			5, 2			10, 4			15, 7		
m	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
KMPRT	2.5	10.3	108.2	3.7	14.4	196.2	4.2	37.6	615.4	6.8	87.6	1524.5
Ours	1.9 (**A**)	7.0 (**A**)	69.6 (**C**)	2.2 (**A**)	7.6 (**A**)	86.3 (**C**)	3.0 (**A**)	10.4 (**C**)	153.9 (**C**)	3.3 (**A**)	15.4 (**C**)	244.8 (**C**)

Table 4: Total run-time in seconds of mPSI protocols: KMPRT [43] and Ours. For our protocols, we report the best time among the three protocols and the label in parenthesis denotes the name of this protocol.

Implementation Details. We make use of the implementation of polynomial-based batch OPPRF [55] and table-based OPPRF [43] available at [23] and [50] respectively. For implementation of relaxed batch OPPRF [12] and equality test functionality $\mathcal{F}^{\ell}_{\mathsf{EQ}}$ [12, 29, 21, 58], we use the code of [12]. For Boolean to Arithmetic Share Conversion functionality $\mathcal{F}^{\mathbb{F}}_{\mathsf{B2A}}$ [58], we use the implementation of correlated OTs available at [48]. Finally, we use the code available at [19] for multiparty functionalities [20, 44] (see Section 2.5).

Experimental Setup. Similar to [43], we ran our experiments on a single machine with 64-core Intel Xeon 2.6GHz CPU and 256GB RAM, and simulated the network environment using linux *tc* command. We configure a LAN connection with bandwidth 10 Gbps and round-trip latency of 0.06ms. For WAN setting, we set the network bandwidth to 200 Mbps and round-trip latency to 96ms.

In this section, n, t and m denote the number of parties, corruption threshold and the size of the input sets respectively. In our experiments, we consider the following values of (n, t): $(4, 1)$, $(5, 2)$, $(10, 4)$ and $(15, 7)$. We note that among these, three settings, namely, $(4, 1)$, $(5, 2)$, and $(15, 7)$ were considered explicitly in the experimental analysis of KMPRT [43, Section 7]. We compare the performance of our protocols with the implementation of KMPRT protocol provided at [50].

6.1 Communication Comparison of mPSI

In this section, we compare the concrete communication cost of our most communication frugal mPSI protocol $\mathsf{Protocol\ A}$ with that of KMPRT protocol [43]. Table 2 summarizes the overall communication cost of $\mathsf{Protocol\ A}$ and KMPRT protocol [43]. As can be observed from the table, $\mathsf{Protocol\ A}$ is $2.3-5.2\times$ more communication efficient than KMPRT protocol[9].

Further, as noted earlier, the clients (parties $P_2, \ldots, P_n$) in our protocol are much lighter compared to KMPRT protocol as is illustrated by Table 3. The concrete communication cost of a client in $\mathsf{Protocol\ A}$ is $2.6-6\times$ less than that of KMPRT protocol. Recall that a client in KMPRT is involved in $2t+3$ calls to OPPRF functionality whereas in our protocol a client only makes a single call to weak-PSM functionality followed by the interaction in Evaluation phase (step 4 in Figure 4).

n	4			5			10			15		
m	2^{12}	2^{16}	2^{18}	2^{12}	2^{16}	2^{18}	2^{12}	2^{16}	2^{18}	2^{12}	2^{16}	2^{18}
Run-time LAN (s)	1.46	2.91	9.32	1.62	3.10	9.49	2.19	4.12	11.27	2.26	4.54	13.12
Run-time WAN (s)	7.10	13.74	34.04	6.98	15.44	39.34	7.88	23.08	74.02	8.14	31.28	108.36
Total Communication (MB)	16.98	209.86	874.23	24.64	290.68	1166.28	55.44	667.73	2627.01	86.24	1038.68	4086.45
Client Communication (MB)	5.66	69.95	291.41	6.16	72.67	291.57	6.16	74.19	291.9	6.16	74.19	291.89

Table 5: Run-time in seconds and communication in MB for steps 2–5 of our Circuit PSI and qPSI protocols.

6.2 Run-time Comparision of mPSI

In this section, we compare the run-times of our mPSI protocols with that of KMPRT Protocol [43]. In Table 4, we report the run-time of KMPRT protocol along with the run-time of our best performing

[9] $\mathsf{Protocol\ A}$'s implementation depends on the Polynomial based Batch OPPRF [55], which is implemented in [23] over prime field with Mersenne prime $2^{61}-1$. We remark here that this only gives statistical security of 38 bits for input sets of size 2^{20}. To obtain statistical security of 40 bits, one can implement $\mathsf{Protocol\ A}$ over a field with at least 2^{63} elements, i.e., each element is represented using 64 bits. However, since an element over prime field with Mersenne Prime $2^{61}-1$ is communicated using 64 bits in the implementation, the communication with 40 bits of security would remain the same if $\mathsf{Protocol\ A}$ were to be implemented over a field where an element is represented using 64 bits. Hence, the communication obtained from the current implementation [23] of $\mathsf{Protocol\ A}$ gives us a correct bound on the communication of $\mathsf{Protocol\ A}$.

protocol (i.e., Protocol A, Protocol B, or Protocol C as discussed above). For each entry in Table 4, we report the median value across 5 executions. As can be observed from Table 4, our best protocol achieves a speedup of $1.2-4.9\times$ and $1.3-6.2\times$ over KMPRT protocol [43] in LAN and WAN setting respectively. This is because KMPRT protocol involves execution of $n(t+2)-1$ instances of OPPRF protocol whereas our protocols involve execution of just $n-1$ weak-PSM protocols followed by a very efficient Evaluation phase (step 4 in Figure 4).

In the LAN Setting, Protocol A is the least efficient of the three instantiations of our mPSI protocol. This is because Protocol A involves expensive computation of polynomial interpolation in contrast to Protocol B and Protocol C which involve inexpensive hashing computations. Between Protocol B and Protocol C, there is a trade-off between compute and communication. Protocol B has non-linear (in set-size m) communication that starts to dominate as m increases. Protocol C has higher fixed compute but linear communication in m. Hence, Protocol C is slower than Protocol B for smaller set size but is faster as the set size increases.

In the WAN Setting, Protocol A owing to its least concrete communication cost, is the most efficient for small sized input sets. But as the set size increases, the non-linear compute starts to become a bottleneck and it loses to Protocol C. Note that Protocol C enjoys much more light-weight compute and linear communication complexity. Since Protocol B communicates more, it is inefficient when compared to the other two protocols in the WAN setting (due to lower bandwidth).

6.3 Performance of Circuit PSI and qPSI

Circuit PSI. As discussed in Section 4, in steps 1,6,7 (Figure 6), we need to work over a prime field $\mathbb{F}_p$ such that $p > n$. Hence, the Mersenne prime $2^5 - 1$ suffices for upto 30 parties and also for all the settings we consider. However, the smallest prime p for which the implementation of these protocols is available (at [19]) is for the Mersenne prime $2^{31} - 1$, which is an overkill for our implementations. Based on the concrete communication analysis discussed in Section 4.3, we observe that the communication in steps 1,6,7 using Mersenne prime 31 is $< 8.2\%$ of the communication involved in steps $2-5$ of the protocol for the values of n, t and m considered in our experiments. Moreover, the computation done in these steps are arithmetic operations over the small field $\mathbb{F}_{31}$. Hence, performance of the steps 2–5 of the protocol is a strong indicator of its overall performance.

We illustrate the performance of steps 2–5 in Table 5 when weak-PSM is instantiated using relaxed-batch OPPRF [12]. These numbers can be extrapolated to estimate the overall run-time of the protocol. For instance, we estimate our Circuit PSI protocol to take $12.19s$ and $80.09s$ in LAN and WAN setting respectively for 10 parties with $t = 4$ and input set size 2^{18}.

qPSI. Protocol Quorum-I convincingly outperforms Quorum-II for the values of n, t and m that we consider in our experiments (see Section 5.3). The aforementioned discussion in the context of Circuit PSI protocol also holds for protocol Quorum-I. From the concrete communication analysis in Section 5.3, for the values of n, t, m considered in experiments, the communication in steps 1,6 (see Figure 9) using Mersenne prime 31 is $< 8.2\%$ of the communication involved in steps $2-5$ for all values of $k \leq n-1$. Hence, for instance, the run-time of Quorum-I protocol can be estimated to be $4.91s$ and $33.84s$ in LAN and WAN setting respectively for 15 parties with $t = 7$, $m = 2^{16}$ and any $k \leq 14$.

Bibliography

[1] Aydin Abadi, Sotirios Terzis, and Changyu Dong. O-PSI: delegated private set intersection on outsourced datasets. In Hannes Federrath and Dieter Gollmann, editors, *ICT Systems Security and Privacy Protection - 30th IFIP TC 11 International Conference, SEC 2015, Hamburg, Germany, May 26-28, 2015, Proceedings*, volume 455 of *IFIP Advances in Information and Communication Technology*, pages 3–17. Springer, 2015.

[2] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 805–817. ACM, 2016.

[3] Saikrishna Badrinarayanan, Peihan Miao, and Peter Rindal. Multi-party threshold private set intersection with sublinear communication. *IACR Cryptol. ePrint Arch.*, 2020:600, 2020.

[4] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In Harriet Ortiz, editor, *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 503–513. ACM, 1990.

[5] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988.

[6] G.R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the 1979 AFIPS National Computer Conference*, pages 313–317, Monval, NJ, USA, 1979. AFIPS Press.

[7] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In Sushil Jajodia and Javier López, editors, *Computer Security - ESORICS 2008, 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings*, volume 5283 of *Lecture Notes in Computer Science*, pages 192–206. Springer, 2008.

[8] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent ot extension and more. 2019.

[9] Pedro Branco, Nico Döttling, and Sihang Pu. Multiparty cardinality testing for threshold private set intersection. *IACR Cryptol. ePrint Arch.*, 2020:1307, 2020.

[10] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.

[11] Octavian Catrina and Sebastiaan de Hoogh. Improved primitives for secure multiparty integer computation. In Juan A. Garay and Roberto De Prisco, editors, *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings*, volume 6280 of *Lecture Notes in Computer Science*, pages 182–199. Springer, 2010.

[12] Nishanth Chandran, Divya Gupta, and Akash Shah. Circuit-psi with linear complexity via relaxed batch opprf. Cryptology ePrint Archive, Report 2021/034, 2021. `https://eprint.iacr.org/2021/034`.

[13] Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 34–63. Springer, 2020.

[14] Jung Hee Cheon, Stanislaw Jarecki, and Jae Hong Seo. Multi-party privacy-preserving set intersection with quasi-linear complexity. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, 95-A(8):1366–1378, 2012.

[15] Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In Dario Catalano and Roberto De Prisco, editors, *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, volume 11035 of *Lecture Notes in Computer Science*, pages 464–482. Springer, 2018.

[16] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 316–334. Springer, 2000.

[17] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, volume 6477 of *Lecture Notes in Computer Science*, pages 213–231. Springer, 2010.

[18] 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.

[19] cryptobiu. Mpchonestmajority. `https://github.com/cryptobiu/MPC-Benchmark/tree/master/MPCHonestMajority`, 2019. Accessed: 2020-08-31.

[20] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In Alfred Menezes, editor, *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, volume 4622 of *Lecture Notes in Computer Science*, pages 572–590. Springer, 2007.

[21] Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. Pushing the communication barrier in secure computation using lookup tables. In *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. The Internet Society, 2017.

[22] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 789–800. ACM, 2013.

[23] encryptogroup. Opprf-psi, 2020. Accessed: 2020-08-31.

[24] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Private set intersection with linear communication from general assumptions. In Lorenzo Cavallaro, Johannes Kinder, and Josep Domingo-Ferrer, editors, *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society, WPES@CCS 2019, London, UK, November 11, 2019*, pages 14–25. ACM, 2019.

[25] Financial Action Task Force. Concealment of beneficial ownership. `http://www.fatf-gafi.org/media/fatf/documents/reports/FATF-Egmont-Concealment-beneficial-ownership.pdf`, 2018.

[26] Financial Action Task Force. Professional money laundering. `https://www.fatf-gafi.org/media/fatf/documents/Professional-Money-Laundering.pdf`, 2018.

[27] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In Joe Kilian, editor, *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*, pages 303–324. Springer, 2005.

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

[29] Juan A. Garay, Berry Schoenmakers, and José Villegas. Practical and secure solutions for integer comparison. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*, volume 4450 of *Lecture Notes in Computer Science*, pages 330–342. Springer, 2007.

[30] Satrajit Ghosh and Mark Simkin. The communication complexity of threshold private set intersection. *IACR Cryptol. ePrint Arch.*, 2019:175, 2019.

[31] Oded Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications.* Cambridge University Press, 2004.

[32] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th Annual Symposium on Foundations of Computer Science, West Palm Beach, Florida, USA, 24-26 October 1984*, pages 464–479. IEEE Computer Society, 1984.

[33] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.

[34] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987.

[35] Per A. Hallgren, Claudio Orlandi, and Andrei Sabelfeld. Privatepool: Privacy-preserving ridesharing. In *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*, pages 276–291. IEEE Computer Society, 2017.

[36] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings*, volume 6056 of *Lecture Notes in Computer Science*, pages 312–331. Springer, 2010.

[37] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Scalable multi-party private set-intersection. In Serge Fehr, editor, *Public-Key Cryptography - PKC 2017 - 20th IACR International Conference on Practice and Theory in Public-Key Cryptography, Amsterdam, The Netherlands, March 28-31, 2017, Proceedings, Part I*, volume 10174 of *Lecture Notes in Computer Science*, pages 175–203. Springer, 2017.

[38] Yan Huang, David Evans, and Jonathan Katz. Private set intersection: Are garbled circuits better than custom protocols? In *19th Annual Network and Distributed System Security Symposium, NDSS, 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012.

[39] Bernardo A. Huberman, Matthew K. Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In Stuart I. Feldman and Michael P. Wellman, editors, *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*, pages 78–86. ACM, 1999.

[40] Roi Inbar, Eran Omri, and Benny Pinkas. Efficient scalable multiparty private set-intersection via garbled bloom filters. In Dario Catalano and Roberto De Prisco, editors, *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings*, volume 11035 of *Lecture Notes in Computer Science*, pages 235–252. Springer, 2018.

[41] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In Victor Shoup, editor, *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, volume 3621 of *Lecture Notes in Computer Science*, pages 241–257. Springer, 2005.

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

[43] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1257–1272. ACM, 2017.

[44] Yehuda Lindell and Ariel Nof. A framework for constructing fast MPC over arithmetic circuits with malicious adversaries and an honest-majority. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer*

and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017, pages 259–276. ACM, 2017.

[45] Catherine A. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *Proceedings of the 1986 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 7-9, 1986*, pages 134–137. IEEE Computer Society, 1986.

[46] Atsuko Miyaji and Shohei Nishida. A scalable multiparty private set intersection. In Meikang Qiu, Shouhuai Xu, Moti Yung, and Haibo Zhang, editors, *Network and System Security - 9th International Conference, NSS 2015, New York, NY, USA, November 3-5, 2015, Proceedings*, volume 9408 of *Lecture Notes in Computer Science*, pages 376–385. Springer, 2015.

[47] Payman Mohassel and Peter Rindal. Aby^3: A mixed protocol framework for machine learning. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 35–52. ACM, 2018.

[48] mpc msri. Ezpc, 2020. Accessed: 2021-01-17.

[49] Michele Orrù, Emmanuela Orsini, and Peter Scholl. Actively secure 1-out-of-n OT extension with application to private set intersection. In Helena Handschuh, editor, *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings*, volume 10159 of *Lecture Notes in Computer Science*, pages 381–396. Springer, 2017.

[50] osu crypto. Multipartypsi. `https://github.com/osu-crypto/MultipartyPSI`, 2020. Accessed: 2020-06-30.

[51] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In Friedhelm Meyer auf der Heide, editor, *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, volume 2161 of *Lecture Notes in Computer Science*, pages 121–133. Springer, 2001.

[52] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse OT extension. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 401–431. Springer, 2019.

[53] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from paxos: Fast, malicious private set intersection. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 739–767. Springer, 2020.

[54] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In Jaeyeon Jung and Thorsten Holz, editors, *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, pages 515–530. USENIX Association, 2015.

[55] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 122–153. Springer, 2019.

[56] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 125–157. Springer, 2018.

[57] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, 21(2):7:1–7:35, 2018.

[58] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. In Jay Ligatti, Xinming

Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 325–342. ACM, 2020.

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

[60] Peter Rindal and Mike Rosulek. Malicious-secure private set intersection via dual execution. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1229–1242. ACM, 2017.

[61] Yingpeng Sang and Hong Shen. Privacy preserving set intersection protocol secure against malicious behaviors. In David S. Munro, Hong Shen, Quan Z. Sheng, Henry Detmold, Katrina E. Falkner, Cruz Izu, Paul D. Coddington, Bradley Alexander, and Si-Qing Zheng, editors, *Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2007), 3-6 December 2007, Adelaide, Australia*, pages 461–468. IEEE Computer Society, 2007.

[62] Yingpeng Sang and Hong Shen. Privacy preserving set intersection based on bilinear groups. In Gillian Dobbie and Bernard Mans, editors, *Computer Science 2008, Thirty-First Australasian Computer Science Conference (ACSC2008), Wollongong, NSW, Australia, January 22-25, 2008*, volume 74 of *CRPIT*, pages 47–54. Australian Computer Society, 2008.

[63] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[64] Adi Shamir. On the power of commutativity in cryptography. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 582–595. Springer, 1980.

[65] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated ot with small communication. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 1607–1626, New York, NY, USA, 2020. Association for Computing Machinery.

[66] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167. IEEE Computer Society, 1986.

[67] Moti Yung. From mental poker to core business: Why and how to deploy secure computation protocols? In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1–2. ACM, 2015.

[68] Yihua Zhang, Aaron Steele, and Marina Blanton. PICCO: a general-purpose compiler for private distributed computation. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 813–826. ACM, 2013.

Appendices

A Instantiations of the Weak-PSM functionality

The weak-PSM functionality from Section 2.4 can be instantiated using an oblivious programmable pseudorandom function (OPPRF), which was first introduced in [43]. More specifcally, we can instantiate this functionality using any of the three OPPRF: polynomial-based batch OPPRF, table-based OPPRF and relaxed-batch OPPRF, each of which offer a different trade-off in parameters. We infomally describe these variants below, and explain how they can be used to realize the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality. We refer the reader to [43, 12] for detailed definitions.

Batch PPRF. [43] Informally, a pseudorandom function (PRF) [32], sampled with a key from a function family, is guaranteed to be computationally indistinguishable from a uniformly random function, to an adversary (who does not have the key), given oracle access to the function. In a programmable PRF (PPRF), the PRF function outputs "programmed" values on a set of "programmed" input points. A "hint", which is also given to the adversary, helps in encoding such programmed inputs and outputs. The guarantee is that the hint leaks no information about the programmed values (but can leak the number of programmed points). When β instances of a PPRF are used, then the corresponding β hints can be combined into a single hint, that hides all the programmed values (but not the number of programmed points). This variant of PPRF is called as a Batch PPRF [55].

OPRF and Batch OPPRF. An oblivious PRF (OPRF) functionality [27] is a two-party functionality, where the sender learns a PRF key k and the receiver learns the PRF outputs on its queries $q_1, \cdots, q_t$. An oblivious PPRF (OPPRF) is a two-party functionality, $\mathcal{F}_{\mathsf{opprf}}$, similar to the OPRF, where now the sender specifies the programmed inputs/outputs, the receiver specifies the evaluation points $q_1, \cdots, q_t$, and the sender gets the PPRF key k and the hint, while the receiver gets the hint and the PPRF outputs on $q_1, \cdots, q_t$. The OPPRF functionality defined with respect to a Batch PPRF is called a Batch OPPRF, denoted by $\mathcal{F}_{\mathsf{b-opprf}}$.

Relaxed Batch OPPRF. [12] A relaxed batch PPRF is a variant of PPRF, where now the function outputs a set of d pseudorandom values corresponding to every input point, with the constraint that for a programmed input, the programmed output is one of these d elements. The corresponding relaxed batch OPPRF functionality, denoted by $\mathcal{F}^{d}_{\mathsf{rb-opprf}}$, uses the relaxed batch PPRF to respond to the sender and receiver. The sender inputs the programmed inputs/outputs and gets the relaxed batch PPRF keys and the hint, while the receiver inputs the evaluation points and gets the hint and the relaxed batch PPRF outputs on its queries.

We now describe the three variants of OPPRFs, which can be used to instantiate the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality:

- **Using the Batch OPPRF functionality** [55]: On sender's inputs $\{X_j\}_{j\in[\beta]}$ and receiver's input $q_1, \cdots, q_\beta$, the protocol proceeds as follows: the sender picks w_j at random for each $j \in [\beta]$, sets T_j as a set of size $|X_j|$, all equal to w_j, and the sender and receiver invoke the $\mathcal{F}_{\mathsf{b-opprf}}$ functionality on inputs $\{(X_j, T_j)\}_{j\in[\beta]}$ and $\{q_j\}_{j\in[\beta]}$, respectively. The receiver gets its output $\{y_j\}_{j\in[\beta]}$ from the OPPRF functionality and the sender sets its output as $\{w_j\}_{j\in[\beta]}$ (and ignores its output from the OPPRF functionality). By the property of the batch OPPRF, it is guaranteed that $y_j = w_j$ for each $j \in [\beta]$ such that $q_j \in X_j$ and y_j is random otherwise. Hence, this protocol securely realizes the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality in the $\mathcal{F}_{\mathsf{b-opprf}}$-hybrid model.
  Specifically, the polynomial-based batch-OPPRF from [55] can be used to instantiate $\mathcal{F}_{\mathsf{b-opprf}}$ in the above construction, which gives a concrete communication cost of $3.5\lambda\beta + N\sigma$ and has a round complexity of 4.

- **Using the OPPRF functionality** [43]: On sender's inputs $\{X_j\}_{j\in[\beta]}$ and receiver's input $q_1, \cdots, q_\beta$, the protocol proceeds as follows: the sender picks w_j at random for each $j \in [\beta]$, sets T_j as a set of size $|X_j|$, all equal to w_j. Let max_β be the application specific upper-bound on the size of the input sets. The sender pads set X_j with dummy elements and set T_j with random elements, upto the upper-bound max_β, $\forall j \in [\beta]$. The sender and receiver invoke the $\mathcal{F}_{\mathsf{opprf}}$ functionality on inputs (X_j, T_j) and q_j respectively, $\forall j \in [\beta]$. The receiver gets output y_j from j^{th} OPPRF functionality invocation. The sender sets its output as $\{w_j\}_{j\in[\beta]}$ (and ignores its output from the invocations of OPPRF functionalities). By the property of OPPRF, it is guaranteed that $y_j = w_j$ for each $j \in [\beta]$ such that $q_j \in X_j$ and y_j is random otherwise. Hence, this protocol securely realizes the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality in the $\mathcal{F}_{\mathsf{opprf}}$-hybrid model.
  Specifically, the table-based OPPRF from [43] can be used to instantiate $\mathcal{F}_{\mathsf{opprf}}$ in the above construction, which gives a concrete communication cost of $(4.5\lambda + 2^{\lceil \log(\mathsf{max}_\beta) \rceil}\sigma)\beta$ and a round complexity of 4. For the application of PSI, max_β is $O(\log m / \log\log m)$.
- **Using the Relaxed Batch OPPRF functionality** [12]: Fix $d = 3$ in the relaxed batch OPPRF functionality, $\mathcal{F}^{d}_{\mathsf{rb-opprf}}$. On sender's inputs $\{X_j\}_{j\in[\beta]}$ and receiver's input $q_1, \cdots, q_\beta$, the protocol

proceeds as follows: the sender picks w_j at random for each $j \in [\beta]$, sets T_j as a set of size $|X_j|$, all equal to w_j, and the sender and receiver invoke the $\mathcal{F}_{\mathsf{rb-opprf}}$ functionality on inputs $\{(X_j, T_j)\}_{j\in[\beta]}$ and $\{q_j\}_{j\in[\beta]}$ respectively. The receiver gets its output $\{W_j\}_{j\in[\beta]}$ from the relaxed batch OPPRF functionality. By the property of relaxed batch OPPRF, it is guaranteed that $w_j \in W_j$ and the other elements in W_j are random if $q_j \in X_j$, else W_j is completely random. Observe that $|W_j| = 3, \forall j \in [\beta]$. In the next phase, the receiver of $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality picks v_j at random for each $j \in [\beta]$, sets target set V_j as a set of size $|W_j|$, all equal to v_j. The sender and receiver of $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality invoke β many instances of OPPRF functionality, where the receiver of $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality plays the role of sender with inputs (W_j, U_j) and the sender of $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality plays the role of receiver with input w_j in the j^{th} OPPRF instance. The sender of $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality gets output y_j from j^{th} OPPRF functionality invocation. The receiver of $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality sets output as $\{v_j\}_{j\in[\beta]}$. By the property of OPPRF, it is guaranteed that $y_j = v_j$ for each $j \in [\beta]$ such that $w_j \in W_j$ and y_j is random otherwise. Transitively, this implies that $y_j = v_j$ for each $j \in [\beta]$ such that $q_j \in X_j$ and y_j is random otherwise. Hence, this protocol securely realizes the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality in the $(\mathcal{F}_{\mathsf{rb-opprf}}, \mathcal{F}_{\mathsf{opprf}})$-hybrid model.
Specifically, using the solution proposed in [12] to instantiate $\mathcal{F}_{\mathsf{rb-opprf}}$ and table-based OPPRF [43] to instantiate $\mathcal{F}_{\mathsf{opprf}}$ gives a concrete communication cost of $(8\lambda+4\sigma)\beta+1.31N\sigma$ and a round complexity of 8.

B Security Proof of Circuit PSI

We complete the proof of Theorem 2, by giving a security proof for our Circuit PSI protocol (Figure 6) below.
Security Proof. Let $C \subset [n]$ be the set of corrupted parties ($|C| = t < n/2$). We show how to simulate the view of C in the ideal world, given the input sets $X_C = \{X_j : j \in C\}$ and the output, $T = f(\cap_{i=1}^{n} X_i)$. We consider two cases based on party P_1 being corrupt or not.

- **Case 1** ($\mathbf{P_1 \notin C}$): In the pre-processing step, the parties run the functionality $\mathsf{DoubleRandomF}^{n,t}$ from [20] and the corrupted parties get up to t shares of the random r_j's, which can all be picked as shares of some random strings by the simulator. The hashing step is local, and can be executed by the simulator using the inputs of the corrupted parties. In step 3, P_1 and P_i (for each $i \in \{2, \cdots, n\}$) invoke the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality and the corrupted parties only see the sender's views (since $P_1 \notin C$), $\{w_{ij}\}_{i\in C, j\in[\beta]}$, which can all be picked at random by the simulator (by the definition of $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$). In steps 4 and 5, for each $i \in \{2, \cdots, n\}$, parties P_1 and P_i invoke the $\mathcal{F}^{\sigma}_{\mathsf{EQ}}$ and $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}$ functionalities and the corrupted parties see only one of the two boolean and additive shares, $\{\langle eq_{ij}\rangle^B_i\}_{i\in C, j\in[\beta]}$ and $\{\langle f_{ij}\rangle_i\}_{i\in C, j\in[\beta]}$, respectively, which can be generated as corresponding shares of some random bit (by the security of secret sharing). In step 6, besides the local computations, which the simulator can do, the corrupted parties see at most t shares of the values z_j and u_j. Here, the simulator can pick shares of some random values as the t shares of the z_j's (by the security of secret sharing) and add them with the t shares of the r_j's (from the pre-processing step), to get the t-shares of the u_j's. In step 7, besides the local computations, the parties invoke the $\mathsf{MultF}^{n,t}$ functionality. The view of corrupted parties includes: at most t shares of the values $\{v_j^{(i)}\}_{i\in[d], j\in[\beta]}$, $\{q_j^{(i)}\}_{i\in[k], j\in[\beta]}$ and $\{c_j\}_{j\in[\beta]}$. Each of the t shares of the $v_j^{(i)}$'s and the $q_j^{(i)}$'s can be picked as shares of random values (by the security of secret sharing) and the t shares of c_j's can be obtained by local computation. Finally, for step 8, the simulator can set the output as T.
- **Case 2** ($\mathbf{P_1 \in C}$): The simulation of the pre-processing step and the hashing step is exactly the same as in Case 1. In step 3, P_1 and P_i (for each $i \in \{2, \cdots, n\}$) invoke the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality and the corrupted parties see both the receiver's view $\{y_{ij} : i \in \{2, \cdots, n\}, j \in [\beta]\}$, and the sender's views $\{w_{ij}\}_{i\in C, j\in[\beta]}$. For each $i \in C$, the simulator picks a random $y_{ij} = w_{ij}$, if $\mathsf{Table}_1[j] \in \mathsf{Table}_i[j]$, else picks a random y_{ij} and w_{ij} independently, for each $j \in [\beta]$ (the faithfulness of this step of simulation follows from the definition of $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ and since the simulator has both Table_1 and Table_i). For each $i \notin C$,

the simulator picks y_{ij}'s at random. In steps 4 and 5, for each $i \in \{2, \cdots, n\}$, parties P_1 and P_i invoke the $\mathcal{F}_{\mathsf{EQ}}^{\sigma}$ and $\mathcal{F}_{\mathsf{B2A}}^{\mathbb{F}_p}$ functionalities and the corrupted parties see both the boolean and additive shares for $i \in C$, $\{\langle eq_{ij}\rangle_1^B, \langle eq_{ij}\rangle_i^B\}_{i\in C, j\in[\beta]}$ and $\{\langle f_{ij}\rangle_1, \langle f_{ij}\rangle_i\}_{i\in C, j\in[\beta]}$, and only one of the two shares for $i \notin C$, $\{\langle eq_{ij}\rangle_1^B\}_{i\in[n]\setminus C, j\in[\beta]}$ and $\{\langle f_{ij}\rangle_1\}_{i\in[n]\setminus C, j\in[\beta]}$. For each $i \in C$, the simulator sets $eq_{ij} = f_{ij} = 1$, if $\mathsf{Table}_1[j] \in \mathsf{Table}_i[j]$ and sets $eq_{ij} = f_{ij} = 0$, otherwise, for each $j \in [\beta]$. It then generates the boolean and arithemetic shares of the eq_{ij}'s and f_{ij}'s, respectively. For each $i \notin C$, the simulator generates both the boolean and additive shares as shares of some random bit (by the security of secret sharing). The simulation of steps 6 and 7 is exactly as in Case 1, with the only addition of giving the random z_j's along with their shares (since $P_1 \in C$). Again, for step 8, the simulator sets the output as T.

C Correctness and Security of Quorum PSI

We recall the Quorum PSI protocol from Section 5.1 in Figure 9.

Parameters: There are n parties $P_1, \ldots, P_n$ with private sets of size m and $1 < k \leq n-1$ is quorum. Let $\beta = 1.28m, \sigma = \kappa + \lceil \log m \rceil + \lceil \log n \rceil + 2$. Additions and multiplications in the protocol are over $\mathbb{F}_p$, where p is a prime (larger than n) that depends on specific instantiation of $\mathcal{F}_{\mathsf{w\text{-}CMP}}$.
Input: Each party P_i has input set $X_i = \{x_{i1}, \cdots, x_{im}\}$, where $x_{ij} \in \{0,1\}^\sigma$. Note that element size can always be made σ bits by first hashing the elements using an appropriate universal hash function.
Protocol:

1. **Pre-processing**: $P_1, \cdots, P_n$ invoke $\mathsf{DoubleRandomF}^{n,t}(\beta)$ to get $([r_1], \cdots, [r_\beta], \langle r_1\rangle, \cdots, \langle r_\beta\rangle)$.
2. **Hashing**: Parties agree on hash functions $h_1, h_2, h_3 : \{0,1\}^\sigma \to [\beta]$.
   P_1 does stashless cuckoo hashing on X_1 using h_1, h_2, h_3 to generate Table_1 and inserts random elements into empty bins.
   For $i \in \{2, \cdots, n\}$, P_i does simple hashing of X_i using h_1, h_2, h_3 into Table_i, i.e., stores each $x \in X_i$ at locations $h_1(x), h_2(x)$ and $h_3(x)$. If the three locations are not distinct, random dummy values are inserted in bin with collision.
3. **Invoking the $\mathcal{F}_{\mathsf{w-PSM}}^{\beta,\sigma,N}$ functionality:** For each $i \in \{2, \cdots, n\}$, P_1 and P_i invoke the $\mathcal{F}_{\mathsf{w-PSM}}^{\beta,\sigma,N}$ functionality for $N = 3m$ as follows:
   - P_i is the sender with inputs $\{\mathsf{Table}_i[j]\}_{j\in[\beta]}$ and P_1 is the receiver with inputs $\{\mathsf{Table}_1[j]\}_{j\in[\beta]}$.
   - P_i receives the outputs $\{w_{ij}\}_{j\in[\beta]}$ and P_1 receives $\{y_{ij}\}_{j\in[\beta]}$.
4. **Invoking the $\mathcal{F}_{\mathsf{EQ}}^{\sigma}$ functionality:** For each $i \in \{2, \cdots, n\}$ and for each $j \in [\beta]$, P_1 and P_i invoke the $\mathcal{F}_{\mathsf{EQ}}^{\sigma}$ functionality as follows: P_1 and P_i send their inputs y_{ij} and w_{ij}, resp., and receive boolean shares $\langle eq_{ij}\rangle_1^B$ and $\langle eq_{ij}\rangle_i^B$ resp., as outputs.
5. **Invoking the $\mathcal{F}_{\mathsf{B2A}}^{\mathbb{F}_p}$ functionality:** For each $i \in \{2, \cdots, n\}$ and for each $j \in [\beta]$, P_1 and P_i invoke the $\mathcal{F}_{\mathsf{B2A}}^{\mathbb{F}_p}$ functionality as follows: P_1 and P_i send their inputs $\langle eq_{ij}\rangle_1^B$ and $\langle eq_{ij}\rangle_i^B$, resp., and receive the additive shares $\langle f_{ij}\rangle_1$ and $\langle f_{ij}\rangle_i$ resp., as outputs.
6. **Invoking n-party functionalities:** For each $j \in [\beta]$,
   - P_1 computes $\langle a_j\rangle_1 = \sum_{i=2}^{n} \langle f_{ij}\rangle_1$ and for each $i \in \{2, \cdots, n\}$, P_i sets $\langle a_j\rangle_i = \langle f_{ij}\rangle_i$.
   - For $i \in [n]$, P_i computes $\langle z_j\rangle_i = \langle a_j\rangle_i - \langle r_j\rangle_i$
   - $[z_j] \leftarrow \mathsf{RevealInReshareF}^n(t, \langle z_j\rangle)$
   - For each $i \in [n]$, P_i computes $[u_j]_i = [z_j]_i + [r_j]_i$.
   - Parties invoke $\mathcal{F}_{\mathsf{w\text{-}CMP}}^{p,k,n,t}$ with P_i's input being $[u_j]_i$ for $i \in [n]$ and P_1 learns c_j as output.
7. **Output:** P_1 computes the quorum intersection as $Y = \bigcup_{j\in[\beta]:c_j=1} \mathsf{Table}_1[j]$.

Fig. 9: QUORUM PSI PROTOCOL

We now give a complete proof of Theorem 3, by proving the correctness and security of the protocol in Figure 9.

Correctness. For $x \in X_1$, define $q_x = |\{i \in \{2, \cdots, n\} : x \in X_i\}|$. Let $Y^* = \{x \in X_1 : q_x \geq k\}$ and the output of the protocol is denoted by Y. We now show that $Y = Y^*$, with all but negligible in κ probability. For the rest of the proof we assume that the cuckoo hashing by P_1 succeeds (i.e., all elements of X_1 get inserted successfully in Table_1), which happens with probability atleast $1 - 2^{-41}$ (see Section 2.2). Now, the following two lemmata complete the proof of correctness.

Lemma 5. $Y^* \subseteq Y$.

Proof. Let $e = \mathsf{Table}_1[j] \in Y^*$ and $\mathcal{E} = \{i \in \{2, \cdots, n\} : e \in X_i\}$. By the property of simple hashing, $e \in \mathsf{Table}_i[j]$ for all $i \in \mathcal{E}$. By correctness of $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$, $\mathcal{F}^{\sigma}_{\mathsf{EQ}}$ and $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}$, we have $y_{ij} = w_{ij}$, $eq_{ij} = 1$ and $f_{ij} = 1$ respectively, for all $i \in \mathcal{E}$. For $i \notin \mathcal{E}$, since $\mathcal{F}^{\sigma}_{\mathsf{EQ}}$ gives a boolean output, $eq_{ij} \in \{0, 1\}$, and by correctness of $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}$, we have $f_{ij} \in \{0, 1\}$. By reconstruction of additive secret sharing we get, $a_j = \sum_{i \in \{2,\cdots,n\}} f_{ij} < n < p$. Since $e \in Y^*$, we get $a_j \geq |\mathcal{E}| \geq k$. Using linearity of both additive and (n, t) secret sharing schemes and correctness of multiparty functionalities used, we get $u_j = a_j$. Finally, by correctness of $\mathcal{F}^{p,k,n,t}_{\mathsf{w\text{-}CMP}}$ we will get $c_j = 1$ when invoked on shares of $u_j \geq k$. Therefore, $e \in Y$.

Lemma 6. $Y \subseteq Y^*$*, with probability at least* $1 - 2^{-\kappa-1}$.

Proof. Suppose $Y \nsubseteq Y^*$. Let $e = \mathsf{Table}_1[j] \in Y \backslash Y^*$. First, $e \in Y$ implies $c_j = 1$. Further, by correctness of $\mathcal{F}^{p,k,n,t}_{\mathsf{w\text{-}CMP}}$, and linearity of additive and (n, t) secret sharing schemes, it follows that $a_j = u_j \geq k$ and $a_j = \sum_{i \in \{2,\cdots,n\}} f_{ij}$. Now, for every $i \in \{2, \cdots, n\}$, by correctness of $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}$ $f_{ij} = eq_{ij}$ and by correctness of $\mathcal{F}^{\sigma}_{\mathsf{EQ}}$, eq_{ij} equals 1 if $y_{ij} = w_{ij}$ and 0 otherwise.

Let $\mathcal{E} = \{i \in \{2, \cdots, n\} : e \in X_i\}$, the set of indices of parties (other than P_1) who possess e in their private sets. Let $\mathcal{E}' = \{i \in \{2, \cdots, n\} : eq_{ij} = 1\}$, the set of indices of parties (other than P_1) whom the protocol interprets to have possession of e. We now show that false positive ($Y \nsubseteq Y^*$) implies when $\mathcal{E}' \backslash \mathcal{E}$ and finally prove that the later event occurs with low probability. Since $\sum_{i \in \{2,\cdots,n\}} f_{ij} = a_j \geq k$ and for all $i \in \{2, \cdots, n\}$, $eq_{ij} \in \{0, 1\}$ we have $|\mathcal{E}'| \geq k$. Consider the following disjoint cases.

- Case 1: $e \notin X_1$. By the construction of Table_1, this implies that e is a dummy element inserted by P_1. Then, $|\mathcal{E}| = 0$ since real elements are distinct from e. Therefore, $\mathcal{E}' \backslash \mathcal{E}$ is non-empty. Further, since any dummy elements inserted by parties other than P_1 are distinct from e, for every $i \in \mathcal{E}' \backslash \mathcal{E}$ it holds that $e \notin \mathsf{Table}_i[j]$.
- Case 2: $e \in X_1$. Since $e \notin Y^*$, we have $|\mathcal{E}| < k$ and hence $\mathcal{E}' \backslash \mathcal{E}$ is not a null set. Further, for each $i \in \mathcal{E}' \backslash \mathcal{E}$, since dummy elements added by P_i are distinct from real elements it holds that $e \notin \mathsf{Table}_i[j]$.

Probability that $i \in \mathcal{E}'$ (that is $y_{ij} = w_{ij}$) when $e \notin \mathsf{Table}_i[j]$ is atmost $2^{-\sigma}$. Note that for any $i \in \mathcal{E}$, by correctness of simple hashing $e \in \mathsf{Table}_i[j]$. Therefore, probability that $\mathcal{E}' \backslash \mathcal{E}$ is non-empty (and hence $e \in Y \backslash Y^*$) is atmost $n \cdot 2^{-\sigma}$. By union bound, the probability that there exists $j \in [\beta]$ such that $\mathsf{Table}_1[j] \in Y \backslash Y^*$ is atmost $\beta n \cdot 2^{-\sigma} < 2^{-\kappa-1}$.

Hence with probability atleast $1 - 2^{-41} - 2^{-\kappa-1} > 1 - 2^{-\kappa}$ (for $\kappa = 40$) the protocol's output will be correct.

Security Proof. Let $C \subset [n]$ be the set of corrupted parties ($|C| = t < n/2$). We show how to simulate the view of C in the ideal world, given the input sets $X_C = \{X_j : j \in C\}$ and the output, $Y = \{x \in X_1 : q_x \geq k\}$, where, for each $x \in X_1$, $q_x = |\{i : x \in X_i \text{ for } i \in \{2, \cdots, n\}\}|$, when $P_1 \in C$, and no output, otherwise. We consider two cases based on party P_1 being corrupt or not.

- **Case 1** ($\mathbf{P_1} \notin \mathbf{C}$): In the pre-processing step, the parties run the functionality $\mathsf{DoubleRandomF}^{n,t}$ from [20] and the corrupted parties get up to t shares of the random r_j's, which can all be picked as shares of some random strings by the simulator. The hashing step is local, and can be executed by the simulator using the inputs of the corrupted parties. In step 3, P_1 and P_i (for each $i \in \{2, \cdots, n\}$)

invoke the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality and the corrupted parties only see the sender's views (since $P_1 \notin C$), $\{w_{ij}\}_{i\in C, j\in[\beta]}$, which can all be picked at random by the simulator (by the definition of $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$). In steps 4 and 5, for each $i \in \{2, \cdots, n\}$, parties P_1 and P_i invoke the $\mathcal{F}^{\sigma}_{\mathsf{EQ}}$ and $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}$ functionalities and the corrupted parties see only one of the two boolean and additive shares, $\{\langle eq_{ij}\rangle^B_i\}_{i\in C, j\in[\beta]}$ and $\{\langle f_{ij}\rangle_i\}_{i\in C, j\in[\beta]}$, respectively, which can be generated as corresponding shares of some random bit (by the security of secret sharing). In step 6, besides the local computations, the parties invoke the functionalities $\mathsf{RevealInReshareF}^n$ and $\mathcal{F}^{p,k,n,t}_{\mathsf{w\text{-}CMP}}$. The view of the corrupted parties in this step includes: at most t shares of the values z_j and u_j, for each $j \in [\beta]$. Here, the simulator can pick shares of some random values as the t shares of the z_j's (by the security of secret sharing) and add them with the t shares of the r_j's (from the pre-processing step), to get the t-shares of the u_j's. Note that, the corrupted parties get no output from the $\mathcal{F}_{\mathsf{w\text{-}CMP}}$ functionality (since $P_1 \notin C$), and also no output from the protocol.

- **Case 2 ($\mathbf{P_1 \in C}$)**: The simulation of the pre-processing step and the hashing step is exactly the same as in Case 1. In step 3, P_1 and P_i (for each $i \in \{2, \cdots, n\}$) invoke the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ functionality and the corrupted parties see both the receiver's view $\{y_{ij} : i \in \{2, \cdots, n\}, j \in [\beta]\}$, and the sender's views $\{w_{ij}\}_{i\in C, j\in[\beta]}$. For each $i \in C$, the simulator picks a random $y_{ij} = w_{ij}$, if $\mathsf{Table}_1[j] \in \mathsf{Table}_i[j]$, else picks a random y_{ij} and w_{ij} independently, for each $j \in [\beta]$ (the faithfulness of this step of simulation follows from the definition of $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{w-PSM}}$ and since the simulator has both Table_1 and Table_i). In steps 4 and 5, for each $i \in \{2, \cdots, n\}$, parties P_1 and P_i invoke the $\mathcal{F}^{\sigma}_{\mathsf{EQ}}$ and $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}$ functionalities and the corrupted parties see both the boolean and additive shares for $i \in C$, $\{\langle eq_{ij}\rangle^B_1, \langle eq_{ij}\rangle^B_i\}_{i\in C, j\in[\beta]}$ and $\{\langle f_{ij}\rangle_1, \langle f_{ij}\rangle_i\}_{i\in C, j\in[\beta]}$, and only one of the two shares for $i \notin C$, $\{\langle eq_{ij}\rangle^B_1\}_{i\in[n]\setminus C, j\in[\beta]}$ and $\{\langle f_{ij}\rangle_1\}_{i\in[n]\setminus C, j\in[\beta]}$. For each $i \in C$, the simulator sets $eq_{ij} = f_{ij} = 1$, if $\mathsf{Table}_1[j] \in \mathsf{Table}_i[j]$ and sets $eq_{ij} = f_{ij} = 0$, otherwise, for each $j \in [\beta]$. It then generates the boolean and arithemetic shares of the eq_{ij}'s and f_{ij}'s, respectively. For each $i \notin C$, the simulator generates both the boolean and additive shares as shares of some random bit (by the security of secret sharing). To simulate steps 6 and 7, the simulator does the following: for all $j \in [\beta]$, give random z_j's and t shares of the random value as shares of z_j's (since r_j's are random, z_j's are random). Next, add t shares of r_j and t shares of z_j to get t shares of u_j. Finally, for each $j \in [\beta]$, set $c_j = 1$ if $\mathsf{Table}_1[j] \in Y$ and set $c_j = 0$, otherwise, and set the final output as Y.

D Weak Comparison Protocols

D.1 Correctness and Security of Weak Comparison Protocol I

We give a complete proof of Theorem 4 by proving the correctness and security of the weak comparison protocol I in Figure 8.

Correctness. The correctness of the protocol directly follows from the correctness of the multiparty functionalities $\mathsf{RandomF}^{n,t}$ and $\mathsf{MultF}^{n,t}$ from [20] and the definition of the polynomial $\psi(x)$.

Security Proof. Let $C \subset [n]$ be the set of corrupted parties ($|C| = t < n/2$). We show how to simulate the view of C in the ideal world, given the input shares $\{[a]_i\}_{i\in C}$ and the output comp, if $P_1 \in C$, and no output, otherwise. We consider two cases based on party P_1 being corrupt or not.

- **Case 1 ($\mathbf{P_1 \notin C}$)**: For the pre-processing step, the simulator can pick a random string s and send t shares of s to parties in C. To simulate step 2, the simulator picks random values and gives their t shares as shares of $[a^i]$. The scalar multiplications and additions are done locally on these shares to obtain t shares of $\psi(a)$. Next, it picks v at random and gives its t shares to corrupted parties. Simulation of this step is correct by security of (n, t)-secret sharing scheme.
- **Case 2 ($\mathbf{P_1 \in C}$)**: The simulation of step 1 and step 2 until generating t shares of v is done exactly as in the previous case. The opened value $v = \psi(a) \cdot s$ is simulated as follows: Since s is uniformly random (as only t shares are known to the corrupted parties), $v = \psi(a) \cdot s$ looks random whenever $v \neq 0$. Hence, if $k \geq n/2$, the simulator sets $v = 0$, if $\mathsf{comp} = 1$ and picks v at random, otherwise, and vice versa, if $k < n/2$. The simulator sends v to the corrupted parties.

D.2 Weak Comparison Protocol II

Building Blocks: The protocol uses the multiparty functionalities in Section 2.5 (with n parties and corruption threshold t) and the $\mathcal{F}_{\mathsf{Mod}}^{p,n,t}$ functionality, which we define in Figure 10, as building blocks. The $\mathcal{F}_{\mathsf{Mod}}^{p,n,t}$ functionality takes as input the $(n,t)-$ shares of some $a \in \mathbb{F}_p$ and outputs the $(n,t)-$ shares of $(a \mod 2)$.

There are n parties $P_1, \cdots, P_n$. t denotes the corruption threshold. All operations and elements are over $\mathbb{F}_p$, such that $n < p$.
Inputs: For each $i \in [n]$, P_i inputs its $(n,t)-$ share $[a]_i$ corresponding to some $0 \leq a < n$ and $[a]_i \in \mathbb{F}_p$.
Output: Reconstruct the shares to get a, evaluate $d = a \mod 2$, generate $(n,t)-$ shares of d and output $[d]_i$ to each P_i.

Fig. 10: $\mathsf{Mod2}$ Functionality $\mathcal{F}_{\mathsf{Mod}}^{p,n,t}$

We instantiate the $\mathcal{F}_{\mathsf{Mod}}^{p,n,t}$ functionality using the protocol from [11], which sets $p > 2^{\kappa+\gamma}$ to be a prime such that $p \mod 4 = 3$ and $\gamma = \log n + 1$. The details of this protocol are given in Appendix D.2.

The weak comparison protocol takes as input, the $(n,t)-$ shares $[a]_i$ from each P_i $(i \in [n])$, where $a \in \mathbb{F}_p$ (such that $0 \leq a < n$). For $k \in \mathbb{F}_p$ (with $0 \leq k < n$) and $\gamma = \lceil \log n \rceil + 1$, the protocol proceeds as follows: it first computes the $(n,t)-$ shares of $(a-k)$. Next, by sequentially invoking the $\mathcal{F}_{\mathsf{Mod}}^{p,n,t}$ functionality, the parties $P_1, \cdots, P_n$ receive the $(n,t)-$ shares of $\left\lfloor \frac{(a-k)}{2^\gamma} \right\rfloor$. Finally, by invoking the $\mathsf{Reveal}^{n,t}$ functionality, party P_1 recovers $\left\lfloor \frac{(a-k)}{2^\gamma} \right\rfloor$, which is 0 iff $a \geq k$. A formal description of the protocol is given in Figure 11.

Parameters: There are n parties $P_1, \cdots, P_n$ with $(n,t)-$ shares $[a]$, of $a \in \mathbb{F}_p$ and $a < n$. Let p, n, k, t be such that p is a prime, $p > n > k$ and $n > 2t$. Let $\gamma = \lceil \log n \rceil + 1$. Additions and multiplciations in the protocol are over $\mathbb{F}_p$, where p depends on the specific instantiation of $\mathcal{F}_{\mathsf{Mod}}^{p,n,t}$.
Input: For each $i \in [n]$, P_i inputs its $(n,t)-$ share $[a]_i$.
Protocol:

1. For each $i \in [n]$, P_i computes $[b]_i = [a]_i - k$.
2. Let $c_1 = b$. For each $i = 1, \cdots, \gamma$, $P_1, \cdots, P_n$ do the following:
   - Invoke the $\mathcal{F}_{\mathsf{Mod}}^{p,n,t}$ functionality with the input $[c_i]$ to get the output $[d_i]$.
   - For each $j \in [n]$, P_j sets $[c_{i+1}]_j = ([c_i]_j - [d_i]_j) \cdot 2^{-1}$.
3. $c_{\gamma+1} \leftarrow \mathsf{Reveal}^{n,t}([c_{\gamma+1}])$.

Output: P_1 sets $\mathsf{comp} = 1$, if $c_{\gamma+1} = 0$ and $\mathsf{comp} = 0$, otherwise. Other parties get no output.

Fig. 11: WEAK COMPARISON PROTOCOL II

Theorem 5. *The protocol given in Figure 11 securely realizes $\mathcal{F}_{\mathsf{w\text{-}CMP}}^{p,k,n,t}$ in the $\mathcal{F}-$hybrid model, where $\mathcal{F} = (\mathcal{F}_{\mathsf{Mod}}^{p,n,t}, \mathsf{Reveal}^{n,t})$, against a semi-honest adversary corrupting $t < n/2$ parties.*

Proof. **Correctness.** The correctness of the protocol follows from the correctness of the functionalities $\mathcal{F}_{\mathsf{Mod}}^{p,n,t}$ and $\mathsf{Reveal}^{n,t}$ and the fact that $\left\lfloor \frac{(a-k)}{2^\gamma} \right\rfloor = 0$ iff $a \geq k$.
Security Proof. Let $C \subset [n]$ be the set of corrupted parties $(|C| = t < n/2)$. We show how to simulate the view of C in the ideal world, given the input shares $\{[a]_i\}_{i \in C}$ and the output comp, if $P_1 \in C$, and no output, otherwise. We consider two cases based on party P_1 being corrupt or not.

– **Case 1** ($\mathbf{P_1} \notin \mathbf{C}$): For the first step, the simulator can perform local addition to get t shares of b. For the second step, the corrupted parties get at most t shares of the values, d_i and c_i, for $i = 1, \cdots, \gamma$, and $c_{\gamma+1}$. The simulator picks the t shares of $[d_i]$'s as shares of random value (by the security of secret sharing) and performs the local addition and scalar multiplication to get the t shares of the $[c_i]$'s. Here, the corrupted parties get no output.
– **Case 2** ($\mathbf{P_1} \in \mathbf{C}$): The simulation of the first and second steps is done exactly as in Case 1. For the third step, the simulator sets $c_{\gamma+1} = 0$, if $\mathsf{comp} = 1$ and $c_{\gamma+1} = p - 1$, otherwise. Finally, set the output as comp.

The Mod2 Protocol We now describe the Mod2 protocol from [11] (using the instantiations from [20]), which we use to instatiate the $\mathcal{F}^{p,n,t}_{\mathsf{Mod}}$ functionality, used in our weak comparison protocol II (Figure 11).
Building Blocks: The protocol uses the multiparty functionalities in Section 2.5 (with n parties and corruption threshold t) as building blocks.
The protocol takes as input, the $(n,t)-$ shares $[a]$ from parties $P_1, \cdots, P_n$, where $a \in \mathbb{F}_p$ (such that $0 \leq a < n$), for prime $p > 2^{\kappa+\gamma}$ (where $\gamma = \lceil \log n \rceil + 1$) with $p \mod 4 = 3$ and proceeds as follows: first, in an input-independent *Pre-processing* step, the parties generate $(n,t)-$ shares of a pair of random non-negative integers (s', s''), such that $(2 \cdot s'' + s')$ is of $\gamma + \kappa$ bits, which is required for security reasons as discussed later. Then, they locally compute and get the $(n,t)-$ shares of $c = 2^{\gamma-1} + a + 2s'' + s'$, which is revealed to P_1. P_1 then computes $c_0 = c \mod 2$ and sends it to all parties. Finally, all parties locally compute and get $(n,t)-$ shares of $d = c_0 + s' - 2c_0 s'$, which is the required output. A formal description of the protocol is given in Figure 12.

Parameters: There are n parties $P_1, \cdots, P_n$ with $(n,t)-$ shares $[a]$, of $a \in \mathbb{F}_p$ and $a < n$. Let $\gamma = \lceil \log n \rceil + 1$. Additions and multiplications in the protocol are over $\mathbb{F}_p$, where $p > 2^{\kappa+\gamma}$ is a prime such that $p \mod 4 = 3$.
Input: For each $i \in [n]$, P_i inputs its $(n,t)-$ shares $[a]_i$.
Protocol:

1. **Pre-processing**:
   – For each $i = 1, \cdots, \kappa+\gamma$, $P_1, \cdots, P_n$ use the $\mathsf{RandBit}()$ sub-protocol (Figure 13) to get $[b_i]$.
   – For each $i \in [n]$, P_i sets $[s'']_i = \sum_{j=1}^{\kappa+\gamma-1} 2^{j-1} \cdot [b_j]_i$ and $[s']_i = [b_{\kappa+\gamma}]_i$.
2. For each $i \in [n]$, P_i sets $[c]_i = (2^{\gamma-1} + [a]_i + 2[s'']_i + [s']_i)$.
3. $c \leftarrow \mathsf{Reveal}^{n,t}([c])$.
4. P_1 computes: $c_0 = c \mod 2$ and sends to all parties.
5. For each $i \in [n]$, P_i sets $[d]_i = c_0 + [s']_i - 2 \cdot c_0 \cdot [s']_i$.

Output: For each, $i \in [n]$, P_i gets the output $[d]_i$.

Fig. 12: Mod2 PROTOCOL

We now describe the sub-protocol RandBit used in the pre-processing step of the above protocol, which takes no input and outputs the $(n,t)-$ shares of a random bit b. The parameters of this sub-protocol are as in the main Mod2 protocol of Figure 12.

Theorem 6. *The protocol given in Figure 12 securely realizes $\mathcal{F}^{p,n,t}_{\mathsf{Mod}}$ in the $\mathcal{F}-$ hybrid model, where $\mathcal{F} = (\mathsf{RandomF}^{n,t}, \mathsf{MultF}^{n,t}, \mathsf{Reveal}^{n,t})$, against a semi-honest adversary corrupting $t < n/2$ parties.*

Proof. **Correctness.** We begin by proving the correctness of the RandBit sub-protocol, invoked in the first step. For this, it suffices to show that $b \in \{0,1\}$. By the correctness of the functionalities $\mathsf{RandomF}^{n,t}$ and $\mathsf{MultF}^{n,t}$ from [20], we know that $u = r^2$. If $u \neq 0$, $(vr+1)2^{-1} \mod p = (r^{(1-p)/2} + 1)2^{-1} \mod p$. We know that for any prime order field element r, $r^{(1-p)/2} = \pm 1 \mod p$ and hence $b \in \{0,1\}$. Now, the correctness of the Mod2 protocol follows from the following observations: consider $c = 2^{\gamma-1} + a + 2s'' + s'$, which implies that $c_0 = c \mod 2 = (a + s') \mod 2$. Now, clearly, $d = c_0 + s' - 2c_0 s' = a \mod 2$ (recall

Input: No input taken.
Protocol:

1. $[r] \leftarrow \mathsf{RandomF}^{n,t}(1)$.
2. Compute $[u] \leftarrow \mathsf{MultF}^{n,t}([r],[r])$.
3. $u \leftarrow \mathsf{Reveal}^{n,t}([u])$. If $u = 0$, discard u and repeat step 1. Else, P_1 sends u to all parties.
4. For each $i \in [n]$, P_i sets: $v = u^{-(p+1)/4} \mod p$.
5. For each $i \in [n]$, P_i sets: $[b]_i = (v[r]_i + 1)2^{-1} \mod p$.

Output: For each $i \in [n]$, P_i gets the output $[b]_i$.

Fig. 13: RandBit SUB-PROTOCOL

that s' is a single bit).
Security Proof. Let $C \subset [n]$ be the set of corrupted parties ($|C| = t < n/2$). We show how to simulate the view of C in the ideal world, given the input shares $\{[a]_i\}_{i\in C}$ and the output shares $\{[d]\}_{i\in C}$ (for $d = a \mod 2$). But note that the output is something the simulator can set on its own (by the security of secret sharing). We consider two cases based on party P_1 being corrupt or not.

- **Case 1** ($\mathbf{P_1} \notin \mathbf{C}$): In the pre-processing step, to simulate the view of the corrupted parties in the RandBit sub-protocol, the simulator does the following: it picks the t shares of r as shares of a random value. It picks a random u and sends its t shares to the corrupted parties. Further, it does local computations to get v and the t shares of b. Then, the simulator does local computations to get the t shares of s' and s''. For step 2, the simulator does local computations to get the t shares of c. Finally, it picks c_0 at random (this is because of the following reason: for a random r, $r^{(1-p)/2} = \pm 1 \mod p$, with equal probability and hence, b is a random bit. Thus, s' looks random to the corrupted parties, by the security of secret sharing, which implies that $c_0 = a + s' \mod 2$ looks random to the corrupted parties) and sets the t shares of $[d]$ by doing the local computation.
- **Case 2** ($\mathbf{P_1} \in \mathbf{C}$): The simulation of the pre-processing step and step 2 is exactly as in Case 1. The simulator picks both c and c_0 at random (this is because of the following reason: $c = 2^{\gamma-1} + a + 2r'' + r'$ and $c_0 = c \mod 2$. $(2s'' + s') \mod p$ is a random field element (corresponding to a random integer of length $\kappa + \gamma$) and hence, c looks random in the field $\mathbb{F}_p$, which implies that c_0 also looks random). Finally, the simulator does the local computation to set the t shares of $[d]$.

Complexity. The Mod2 protocol has an expected communication complexity of $19.3n(\lceil \log p \rceil)^2$ and an expected round complexity of 10.

E Instantiation of the DoubleRandom functionality

From [20], we have an instantion of $\mathsf{DoubleRandomF}^{n,t}$, which gives $(n,t)-$ and $(n,2t)-$ shares of random strings. We modify this protocol appropriately to get $(n,t)-$ shares and additive shares of random strings. In Figure 14, we give the protocol to generate $(n,t)-$ and additive shares of ℓ random values $r_1, \cdots, r_\ell$.

In Figure 14, step (2) denotes that, for each $i \in [n]$, P_i computes $([r_1]_i, \cdots, [r_j]_i) = M([s^{(1)}]_i \cdots [s^{(n)}]_i)^T$ and $(\langle r_1\rangle_i, \cdots, \langle r_\ell\rangle_i) = M(\langle s^{(1)}\rangle_i, \cdots, \langle s^{(n)}\rangle_i)^T$. We refer the reader to [20] for a proof of security of the protocol.

Parameters: $P_1, \cdots, P_n$ are n parties. t is the corruption threshold. All additions and multiplications are considered in $\mathbb{F}$. Let $M = (m_{ij})^T_{i \in [n], j \in \{0\} \cup [\ell-1]}$, where $m_{ij} = \alpha_i^j$ for each $i \in [n], j \in \{0\} \cup [\ell - 1]$, and $\alpha_1, \cdots, \alpha_n \in \mathbb{F}$ are distinct, be the Van der Monde matrix.
Protocol:

1. For each $i \in [n]$, P_i picks a uniformly random $s^{(i)} \in \mathbb{F}$ and deals a t-sharing $[s^{(i)}]$ and an additive sharing $\langle s^{(i)} \rangle$.
2. Compute:

$$([r_1], \cdots, [r_\ell]) = M([s^{(1)}], \cdots, [s^{(n)}])^T$$
$$(\langle r_1 \rangle, \cdots, \langle r_\ell \rangle) = M(\langle s^{(1)} \rangle, \cdots, \langle s^{(n)} \rangle)^T$$

Fig. 14: $\mathsf{DoubleRandomF}^{n,t}(\ell)$ Protocol