# Efficient Linear Multiparty PSI and Extensions to Circuit/Quorum PSI

Nishanth Chandran
Microsoft Research
nichandr@microsoft.com

Nishka Dasgupta*
Aarhus University
nishka@post.au.dk

Divya Gupta
Microsoft Research
divya.gupta@microsoft.com

Sai Lakshmi Bhavana Obbattu
Microsoft Research
t-saobb@microsoft.com

Sruthi Sekar
Indian Institute of Science
sruthisekar@iisc.ac.in

Akash Shah*
UCLA
akashsha08@ucla.edu

## ABSTRACT

Multiparty Private Set Intersection (mPSI), enables $n$ parties, each holding private sets (each of size $m$) to securely compute the intersection of these private sets. While several protocols are known for this task, 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 $O(nmt\lambda)$, where $t < n$ is the number of corrupt parties and $\lambda$ 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 $O(nm\lambda)$.

– Second, concretely, our protocol has $6(t+2)/5$ times lesser communication than KMPRT and is up to 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.

## CCS CONCEPTS

• **Security and privacy → Cryptography**; **Privacy-preserving protocols**.

## KEYWORDS

Private Set Intersection; secure multi-party computation

**ACM Reference Format:**
Nishanth Chandran, Nishka Dasgupta, Divya Gupta, Sai Lakshmi Bhavana Obbattu, Sruthi Sekar, and Akash Shah. 2021. Efficient Linear Multiparty PSI and Extensions to Circuit/Quorum PSI. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 24 pages. https://doi.org/10.1145/3460120.3484591

*Work done at Microsoft Research.


*CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea*

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484591

## 1 INTRODUCTION

**Multiparty PSI.** Private set intersection (PSI) [53, 74] 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 [4, 5, 39, 76] have proven to be inefficient to solve this problem and hence several works have focused on obtaining concretely efficient specialized protocols [14, 18, 19, 24, 43, 46, 50, 51, 59, 62–65, 67, 69, 70]. The problem of *Multiparty PSI* (mPSI) was first introduced in [31] 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 [44], the first and only known practical realization was provided in [51]. This protocol is secure in the semi-honest dishonest majority setting[1] (i.e., the adversary can corrupt up to $n-1$ parties and follows the protocol specification faithfully) and its total communication complexity is $O(nmt\lambda)$, where $n$ is the number of parties, $t < n$ is the corruption threshold, $m$ is the set size of each party and $\lambda$ 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 [2, 7, 17, 21, 52, 55, 78]. Hence, it is important to explore the concrete efficiency of mPSI protocols in this setting. Unfortunately, even under this relaxation (also considered in [15]), the best known protocol [51] is no better and has complexity $O(nmt\lambda)$.

### 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 $O(nm\lambda)$, thus obtaining an $O(t)$-factor improvement over [51]. While theoretically, this matches the complexity of the protocol from [44] based on homomorphic encryption[2], concretely, our protocol is approximately $6(t+2)/5$ times more communication frugal

[1] The works of [47, 51] also build concretely efficient mPSI in a weaker augmented semi-honest model (see Section 1.2); here we focus on standard semi-honest security.
[2] Although the HE-based mPSI of [44] achieves a communication of $O(nm\lambda)$ in the semi-honest dishonest majority setting, we compute a rough lower bound on its concrete complexity that is much higher than ours (see Sections 1.2 and 6.2.1).

than [51]. This amounts to more than an order of magnitude lesser communication than [51] when the number of parties $> 15$ and $t \approx n/2$; even when $t = 1$, our protocol has nearly 4× lesser communication than [51]. We also implement our protocol and show it to be up to 5× and 6.2× faster than [51] 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 $m = 2^{20}$, our protocol executes in under 40s in LAN and 245s in WAN settings).

Next, we consider 2 important variants of the mPSI problem – circuit PSI and quorum PSI – and provide concretely efficient semi-honest secure protocols in the honest majority setting.

**Circuit PSI.** The problem of circuit PSI was introduced in the 2 party setting [45] 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 such as cardinality, set intersection sum [48, 77], and threshold cardinality/intersection [3, 10, 31, 35, 36, 42, 66, 79, 80]. Circuit PSI has received a lot of attention and has also shown to be practically feasible in the 2-party context [13, 16, 26, 64–66]. 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 circuit PSI protocol achieving a communication of approximately $O(mn(\lambda\kappa + \log^2 n))$. Concretely, its communication is only $\approx 4\times$ the cost of mPSI.

**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 [27, 28] and checking if a list of entities is present in multiple blacklists. We provide an efficient qPSI protocol in the semi-honest honest majority setting achieving a communication cost of $O(nm\kappa(\lambda + \kappa \log n))$.

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 between $P_1$ and $P_i$ on their individual private sets[3] 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 element is in $P_i$'s set, and outputs independent random values, otherwise[4]. 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 [21, 52] in the second phase, resulting in the total communication complexity being *linear in n*. In contrast, in [51], each party interacts in $2t$ instances of a wPSM-like functionality, incurring an additional multiplicative $t$ overhead.

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 wPSM, whereas $P_1$ engages in $n-1$ instances of the same. We show that 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 a client-server setting, where only one party needs to be computationally heavy and is played by the server. Unlike prior works in the client-server setting [1, 54], 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, the protocol in [51] also had an asymmetry between load on different parties, and our clients require $7(2t+3)/10$ times less communication than clients in [51].

To summarize our contributions:

- We give the first concretely efficient protocol for mPSI, with communication complexity of $O(nm\lambda)$ and constant rounds.

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

[4] This resembles the two-party *oblivious programmable pseudorandom function (OPPRF)* functionality [51], and we indeed show that it can be instantiated using an OPPRF.

- 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× and 6.2× faster than prior state-of-the-art [51] in LAN and WAN settings, respectively, even for 15 parties.

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

### 1.2 Related Work

*HE-based mPSI.* The state-of-the-art work on HE-based mPSI protocols is that of [44] who provide a threshold additive homomorphic encryption (HE) based protocol with an asymptotic communication complexity of $O(nm\lambda)$, thus matching ours. While no implementation is provided in this work, in Section 6.2.1, we estimate a lower bound on its concrete computation and communication costs, and show that its expected run-times are much worse than ours.

*Threshold PSI.* The works of [3, 10, 36] propose protocols for the problem of *multiparty threshold PSI* (where the parties learn the intersection only if its size is greater than a threshold) using HE schemes. The works of [3] and [10] further use their respective multiparty threshold PSI protocols to obtain mPSI protocols with complexities $O(n\lambda T \log T)$ and $O(n\lambda T^2 \log T)$ respectively, where $T = m -$ (size of intersection). As mentioned by the authors themselves, these protocols are more suitable for settings when the intersection size is large and close to $m$. While this leads to sub-linear in $m$ protocols when $T$ is sub-linear in $m$, for the general problem of mPSI, when $T$ can be arbitrary (and even $O(m)$), the complexity of their protocol is super-linear in $m$, and concretely worse than that of [44] (and hence our protocol).

*Other Related Works.* The works of [1, 54] build mPSI protocols in the server-aided model (which assumes the existence of a server that does not collude with the clients). Further, [44] as well as the works of [15, 31, 35, 44, 49, 71, 72] also provide theoretical protocols for the malicious setting, whose complexities are naturally much worse than semi-honest protocols.

*Augmented semi-honest security.* An mPSI protocol in the augmented semi-honest model was proposed in [51], whose complexity matches our semi-honest protocol's complexity. However, augmented semi-honest security is much weaker than standard semi-honest security. In particular, the augmented semi-honest mPSI protocol of [51] completely leaks the intersection of the honest parties' sets to the adversary even in the honest majority setting, which is clearly disallowed by standard semi-honest security (see Appendix A for more details).

### 1.3 Organization

We discuss the formal security model and cryptographic primitives in Section 2. Then, we describe our mPSI protocol in Section 3, our circuit PSI protocol in Section 4, and our qPSI protocol in Section 5. Finally, we present our empirical evaluation results in Section 6.

## 2 PRELIMINARIES

*Notations.* Let $\kappa$ and $\lambda$ 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 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 $x \in \{0,1\}^\ell$, 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 [6, 73] 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 [73]. 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 $b \in \{0,1\}$ between 2 parties, we use additive secret sharing scheme over the field $\mathbb{F}_2$. If a bit $b$ is shared amongst two parties $P_i$ and $P_j$, the shares are denoted by $\langle b \rangle_i^B$ and $\langle b \rangle_j^B$ respectively. We note that both Shamir secret sharing and additive sharing are linear schemes. 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 $\langle a + c \rangle$.

### 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 [11, 37, 40], 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 [11], 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 [11], if we have any semi-honest secure protocol $\pi$ 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 $\pi$.

### 2.2 Cuckoo Hashing

Cuckoo hashing [61] uses $K$ random hash functions $h_1, \cdots, h_K : \{0,1\}^\sigma \to [\beta]$ to map $m$ elements into $\beta$ 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.* [67] showed that for $K = 3, 4$ and $5$ and $\beta = 1.27m, 1.09m$ and $1.05m$ respectively, the failure probability is at most $2^{-40}$, by extrapolating their experimental analysis for the failure probability $2^{-30}$. When considering stash-less Cuckoo hashing, to upper bound the overall failure probability of our protocols to $2^{-40}$, we require an analysis of the parameters for the failure probabilities $2^{-41}/2^{-42}/2^{-46}$. Extrapolating, similar to [67], we get $\beta = 1.28m/1.28m/1.31m$ to ensure that the failure probability in stash-less Cuckoo hashing is at most $2^{-41}/2^{-42}/2^{-46}$ respectively, for $K = 3$. Similar to prior works that use Cuckoo hashing [13, 51, 56, 65, 67, 70] that includes the state-of-the-art in mPSI, the main description of all our protocols assumes this stash-less setting. However, our protocols can be extended to the setting with stash. We describe this extension for mPSI in Section 3.4 while similar techniques can also be applied for the case of circuit PSI and qPSI.

## 2.3 Two-party Functionalities

*2.3.1 Equality Test.* We use a two-party equality test functionality $\mathcal{F}_{\text{EQ}}^{\ell}$. Here, 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 use the protocol from [13] that builds on the ideas of [23, 32, 68] to realize this functionality. The concrete communication complexity of this protocol is at most $3\ell\lambda/4 + 8\ell$ and round complexity is $\log \ell$.

*2.3.2 Boolean to Arithmetic Share Conversion.* We also use a two-party functionality $\mathcal{F}_{\text{B2A}}^{\mathbb{F}}$, 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 $\langle b \rangle_1^B$ and $\langle b \rangle_2^B$ respectively and outputs the additive shares $\langle x \rangle_1$ and $\langle x \rangle_2$ of $x \in \mathbb{F}$ for $x = b$ to $P_1$ and $P_2$ respectively. We instantiate this functionality with the share conversion protocol given in [68] that uses one correlated OT and has total communication of $\lambda + \lceil \log |\mathbb{F}| \rceil$ bits and takes 2 rounds.

We remark here that OT extension using the recent line of work on SilentOT [9, 75] 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

We define a 2-party functionality, $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$, called *weak private set membership (wPSM)* that allows a clean exposition of our protocols. We note that this functionality is similar in spirit to the batch oblivious programmable PRF (OPPRF) considered in [65] and as we discuss later, that is indeed one way to realize this functionality efficiently. In a single instance of the wPSM, 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. Similar to [65], we consider a batch version of this functionality, where the parties do multiple instances of wPSM together as a batch. We define the functionality $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$ formally in Figure 1, where $\beta$ is the batch size, $\sigma$ is length of input and output elements, and $N$ is the total size of all sets input by the second party.

$P_1$ 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 $i \in \left[|X_j|\right]$ 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 $P_2$.

**Figure 1: Weak PSM Functionality $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$**

We consider three instantiations of this functionality using primitives considered in the line of OPPRFs [51]. We provide details on instantiations in Appendix B and summarize their costs below.

- **Polynomial-based batch-OPPRF** [65]: Instantiating using the polynomial-based OPPRF from [65] has the concrete communication cost of $3.5\lambda\beta + N\sigma$ and round complexity of 2.
- **Table-based OPPRF** [51]: The instantiation using table-based OPPRF [51] assumes an upper-bound on the size of the individual 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 2.
- **Relaxed batch OPPRF**: We can instantiate $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$ functionality by invoking relaxed batch OPPRF [13] followed[5] by an invocation of table-based OPPRF [51]. The concrete communication of this case is $(8\lambda + 4\sigma)\beta + 1.31N\sigma$ and round complexity is 4.

*Execution Cost:* Instantiations of the $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$ functionality using the above 3 approaches provide trade-offs between computation and communication [13, 51, 65]. 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 (described below) in the honest majority setting. The protocols from [21, 52] can be used to realize these functionalities. Let $\mathbb{F}(+,\cdot)$ be a finite field. Let $n$ be the number of parties and $t < n/2$ be the corruption threshold. Recall that for an element $a \in \mathbb{F}$, $[a]$ denotes its $(n,t)$-

[5]Relaxed batch OPPRF doesn't directly realize batch OPPRF functionality and can't be used as a standalone primitive to instantiate wPSM. Refer Appendix B for details.

| Functionality | Communication | Rounds |
|---|---|---|
| $\mathsf{RandomF}^{n,t}(\ell)$ | $\lceil \frac{\ell}{n-t} \rceil n(n-1)\lceil \log \lvert\mathbb{F}\rvert \rceil$ $< 2\ell(n-1)\lceil \log \lvert\mathbb{F}\rvert \rceil$ | 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 |
| $\mathsf{ConvertShares}^{n,t}(\langle a\rangle)$ (amortized cost) | $2(\frac{n}{n-t}+1)(n-1)\lceil \log \lvert\mathbb{F}\rvert \rceil$ $< 6(n-1)\lceil \log \lvert\mathbb{F}\rvert \rceil$ | 3 |

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

linear secret sharing and $\langle a\rangle$ denotes its additive secret sharing. Since these are linear secret sharing schemes, as discussed, scalar multiplications and additions can be done by local operations.

- $\mathsf{RandomF}^{n,t}(\ell)$ : 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 [21].

- $\mathsf{Reveal}^{n,t}([a])$ : 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 $a$.
- $\mathsf{ConvertShares}^{n,t}(\langle a\rangle)$ : Takes $\langle a\rangle$ where $a \in \mathbb{F}$ and outputs $[a]$. We show how to realize this functionality in Appendix C.

We summarize the communication and round complexity of realizing the above functionalities as per [21] in Table 1. In our results we invoke $\mathsf{RandomF}^{n,t}$ 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 $t < n/2$.

## 2.6 Weak Comparison Functionality

We define a weak form of multiparty comparison functionality, $\mathcal{F}^{p,k,n,t}_{\text{w-CMP}}$ (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 $0 \le a < n$ and the functionality outputs the indicator bit comp, which is 1 iff $a \ge k$, to the leader $P_1$ and the other parties receive no output. We formally describe this functionality in Figure 2. We provide two instantiations of this functionality, which offer different trade-offs to our communication costs, in Section 5.2.

$n$ parties $P_1, \cdots, P_n$. Prime field $\mathbb{F}_p$ such that $1 \le k < n < p$.
**Inputs**: For each $i \in [n]$, $P_i$ holds $[a]_i$ such that $0 \le a < n$.
**Output**: If $a \ge k$, set comp = 1, else set comp = 0. Send comp to $P_1$. Other parties receive no output.

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

# 3 MULTIPARTY PSI

We begin by formally defining the multiparty private set intersection functionality, $\mathcal{F}^{n,m}_{\text{PSI}}$ 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 $P_i$.

**Figure 3: Private Set Intersection Functionality $\mathcal{F}^{n,m}_{\text{PSI}}$**

## 3.1 Multiparty PSI Protocol

**Building blocks:** Our protocol uses the weak PSM functionality $\mathcal{F}^{\beta,\sigma,N}_{\text{wPSM}}$ (Section 2.4) and the multiparty functionalities from Section 2.5 (with $n$ parties and corruption threshold $t < n/2$) 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 that uses 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 $\text{Table}_1$ 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, each bin of $\text{Table}_1$ has exactly one element. Parties $P_i$ for $i \in \{2, \ldots, n\}$ do simple hashing of $X_i$ into $\text{Table}_i$, 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 $\text{Table}_i$ can have arbitrary number of elements and in total (including dummies) each $\text{Table}_i$ 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 $\mathcal{F}^{\beta,\sigma,N}_{\text{wPSM}}$ functionality for $N = 3m$ with $P_1$ acting as a receiver with queries $\text{Table}_1$, and $P_i$ acting as the sender with input sets $\text{Table}_i$. By the definition of $\mathcal{F}^{\beta,\sigma,N}_{\text{wPSM}}$, for query $j$, $P_1$ and $P_i$ receive the same random element if $P_1$'s query, i.e., $\text{Table}_1[j]$ belongs to $P_i$'s bin/set, i.e., $\text{Table}_i[j]$ and different random elements, otherwise. In the *Evaluation* step, all parties evaluate a circuit for each bin

**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 the field $\mathbb{F}_p$. Let $t < n/2$ be the corruption threshold.
**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 $\sigma$ bits by first hashing the elements using an appropriate universal hash function.
**Protocol**:

(1) **Pre-processing** (*Randomness generation required for Step (4)*): $P_1, \cdots, P_n$ compute $([s_1], \cdots, [s_\beta]) \leftarrow \mathsf{RandomF}^{n,t}(\beta)$.
(2) **Hashing**: Parties agree on hash functions $h_1, h_2, h_3 : \{0,1\}^\sigma \rightarrow [\beta]$.
$P_1$ does stash-less cuckoo hashing on $X_1$ using $h_1, h_2, h_3$ to generate $\mathsf{Table}_1$ 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 $\mathsf{Table}_i$, 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, insert dummy elements in $\mathsf{Table}_i$.
(3) **Invoking the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{wPSM}}$ functionality**: For each $i \in \{2, \cdots, n\}$ , $P_1$ and $P_i$ invoke the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{wPSM}}$ 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 $\langle a_j \rangle_i = (w_{ij} \mod p)$.
- $P_1, \cdots, P_n$ compute $[a_j] \leftarrow \mathsf{ConvertShares}^{n,t}(\langle a_j \rangle)$.
- $P_1, \cdots, P_n$ invoke the following multiparty functionalities:
  - $[v_j] \leftarrow \mathsf{MultF}^{n,t}([a_j], [s_j])$.
  - $v_j \leftarrow \mathsf{Reveal}^{n,t}([v_j])$.

(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.

**Figure 4: MULTIPARTY PSI PROTOCOL**

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 $a_j$, using $\mathsf{ConvertShares}^{n,t}$, 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 $\mathsf{Table}_1$) and sends it to all the other parties.

## 3.2 Correctness and Security Proof

Theorem 3.1. *The protocol in Figure 4 securely realizes $\mathcal{F}^{n,m}_{\mathsf{PSI}}$ in the $\mathcal{F}$-hybrid model, where $\mathcal{F} = (\mathcal{F}^{\beta,\sigma,N}_{\mathsf{wPSM}}, \mathsf{ConvertShares}^{n,t}, \mathsf{RandomF}^{n,t}, \mathsf{MultF}^{n,t}, \mathsf{Reveal}^{n,t})$, 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 $\mathsf{Table}_1$. 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 3.2. $Y^* \subseteq Y$.

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 $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{wPSM}}$, $y_{ij} = w_{ij}$ for all $i \in \{2, \cdots, n\}$. Finally, by the correctness of the multiparty functionalities from Section 2.5, we have $a_j = 0 = v_j$. Hence, $e \in Y$. □

Lemma 3.3. *$Y \subseteq Y^*$, 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 $a_j = 0$ or $s_j = 0$. The latter happens with probability $F_1 = p^{-1} < 2^{-\sigma}$. If $a_j = 0$, 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 $i \in \{2, \ldots, n\}$, it holds that $e \notin \mathsf{Table}_i[j]$ for all $i \in \{2, \ldots, n\}$. 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 probability at least $(1 - 2^{-41} - 2^{-\kappa-1}) \geq 1 - 2^{-\kappa}$ for $\kappa = 40$.

**Security.** Let $C \subset [n]$ be the set of corrupt 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 ($P_1$ is honest)**: To simulate the output of $\mathsf{RandomF}^{n,t}$ in the pre-processing step, the simulator can pick random $s_j$'s, generate their shares and give $t$ shares to the corrupt parties. The hashing step is local, and can be executed by the simulator using $\{X_i\}_{i \in C}$. In step 3, where the $\mathcal{F}_{\mathsf{wPSM}}^{\beta,\sigma,N}$ 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 $\{w_{ij}\}_{i \in C, j \in [\beta]}$ , which can all be picked at random by the simulator (by the definition of $\mathcal{F}_{\mathsf{wPSM}}^{\beta,\sigma,N}$). In step 4, besides the local computations, which can all be executed by the simulator, the parties call functionalities $\mathsf{ConvertShares}^{n,t}$, $\mathsf{MultF}^{n,t}$ and $\mathsf{Reveal}^{n,t}$. The corrupted parties get at most $t$ shares for the values $a_j$ and $v_j$, for each $j \in [\beta]$. The simulator can generate $t$ shares of random values (by security of $(n,t)$-secret sharing), and finally, send the output $Y^*$ to the corrupted parties.
- **Case 2 ($P_1$ is corrupt)**: The simulation of the pre-processing step and the hashing step is exactly same as in Case 1. In step 3, where the $\mathcal{F}_{\mathsf{wPSM}}^{\beta,\sigma,N}$ 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, $\{w_{ij}\}_{i \in C\backslash\{1\}, j \in [\beta]}$. 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 $\mathcal{F}_{\mathsf{wPSM}}^{\beta,\sigma,N}$ and since the simulator has both $\mathsf{Table}_1$ and $\mathsf{Table}_i$). For an honest $P_i$, the simulator can pick all $y_{ij}$'s at random (again by the definition of $\mathcal{F}_{\mathsf{wPSM}}^{\beta,\sigma,N}$). Step 4 is simulated as follows: The simulator picks the $t$ shares of the $a_j$'s as shares of some random value. Further, for each $j \in [\beta]$, it sets $v_j$ to $0 \forall j \in [\beta]$ such that $\mathsf{Table}_1[j] \in Y^*$, and picks $v_j$ uniformly at random otherwise (since $s_j$ are uniformly 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

For all our protocols, both theoretical and empirical communication/round complexity calculations, consider the cost of end-to-end protocol execution, which includes the pre-processing phase. First, our protocol makes $n-1$ invocations of wPSM 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 wPSM used). In contrast, Kolesnikov *et al.* [51] makes $nt$ calls to OPPRF functionality (which is a primitive stronger than $\mathcal{F}_{\mathsf{wPSM}}^{\beta,\sigma,N}$ as shown before).

Concretely, instantiating wPSM using polynomial-based construction (that has the least communication) and using complexities of $n$-party functionalities from Table 1, our protocol requires at most $m(n-1)(4.5\lambda + 35(\kappa + \lceil \log m \rceil) + 140)$ bits of communication. Its round complexity[6] is 8. On the other hand, [51] requires communication of $m(nt + 2n - 1)(4.5\lambda + 46(\kappa + \lceil \log m \rceil))$ and 4 rounds.

In our protocol, as well as in [51], we can see that the communication load of $P_1$, the leader, and $P_i$, for $i \in \{2, \cdots, n\}$, the clients, are different. Specifically, the client communication complexity (the total number of bits sent and received by the client) of our protocol is $m(4.5\lambda + 64(\kappa + \lceil \log m \rceil) + 256)$. In comparison, [51] 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 [51] respectively.

## 3.4 Handling stash

The state-of-the-art work in mPSI [51] propose Cuckoo hashing based protocols *only* in the stash-less setting, a setting that has been considered in several Cuckoo hashing based PSI protocols [13, 65, 67, 70]. Nonetheless, in this section, we discuss how our protocol above can be adapted to work in the setting when Cuckoo hashing results in a stash at $P_1$. Let $m_s = O(\log m)$ be the bound on the stash size for input set of size $m$ [41]. Now, for each element in stash, $P_1$ checks for its existence separately in sets of $P_i$ for all $i \in \{2, \cdots, n\}$ using a private set membership protocol (PSM) [13, 30] whose complexity is $O(m\lambda)$. Then, we have a procedure to combine the results from these individual PSMs to compute whether the element lies in the intersection of all parties or not, using ideas similar to our mPSI protocol without stash. We provide a formal description in Appendix D. In the stash setting, our protocol has cost $O(nm\lambda \log m)$, i.e., we pay an additional multiplicative $\log m$ to handle stash. Finally, we note that similar ideas can naturally be used to handle the stash in both circuit PSI and qPSI if needed.

# 4 MULTIPARTY CIRCUIT PSI

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

There are $n$ parties $P_1, \cdots, P_n$ and a symmetric 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 $P_i$.

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

## 4.1 Circuit PSI Protocol

**Building blocks:** Our protocol uses the two-party functionalities weak private set membership $\mathcal{F}_{\mathsf{wPSM}}^{\beta,\sigma,N}$ (Section 2.4), equality test $\mathcal{F}_{\mathsf{EQ}}^{\sigma}$

[6] In this paper, while calculating the round complexity of any protocol, we take into account any parallelization possible amongst different steps of the protocols and optimize the total number of rounds. For example, consider the mPSI protocol. Step 2 is only a local computation. The Steps 1 and 3 have no mutual dependence and can run in parallel contributing to a total of 2 rounds. $\mathsf{ConvertShares}^{n,t}$ executes in 3 rounds, where the first round is independent of its input. So this round can be executed in parallel with Step 2. Hence up to completion of $\mathsf{ConvertShares}^{n,t}$ execution the protocol takes 4 rounds. The $\mathsf{MultF}^{n,t}$ in Step 4 takes 5 total rounds, out of which the first 3 are independent of input to $\mathsf{MultF}^{n,t}$ and hence can start with the first round of the protocol. Therefore up until completion of $\mathsf{MultF}^{n,t}$ our protocol takes 6 rounds. Finally, $\mathsf{Reveal}^{n,t}$ and intersection announcement by $P_1$ take one round each.

**Parameters**: $n$ parties $P_1, \ldots, P_n$ with private sets of size $m$. Let $t < n/2$ be the corruption threshold, $\beta = 1.28m$, $\sigma = \kappa + \lceil \log m \rceil + 2$. Additions and multiplications are over the field $\mathbb{F}_p$, 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 $\sigma$ bits by first hashing the elements using an appropriate universal hash function.

**Protocol**:

(1) **Hashing**: Parties agree on hash functions $h_1, h_2, h_3 : \{0,1\}^\sigma \to [\beta]$.
$P_1$ does stash-less cuckoo hashing on $X_1$ using $h_1, h_2, h_3$ to generate $\mathsf{Table}_1$ 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 $\mathsf{Table}_i$, 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.

(2) **Invoking the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{wPSM}}$ functionality:** For each $i \in \{2, \cdots, n\}$ , $P_1$ and $P_i$ invoke the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{wPSM}}$ 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 $\{y_{ij}\}_{j\in[\beta]}$.

(3) **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.

(4) **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.

(5) **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 $\langle a_j\rangle_i = \langle f_{ij}\rangle_i$.
- $P_1, \cdots, P_n$ compute $[a_j] \leftarrow \mathsf{ConvertShares}^{n,t}(\langle a_j\rangle)$.

(6) **Computing shares of intersection:** For each $j \in [\beta]$,
- Compute $[v^{(0)}_j] = [a_j] - n + 1$.
- For each $i \in [d]$, compute $[v^{(i)}_j] \leftarrow \mathsf{MultF}^{n,t}([v^{(i-1)}_j], [v^{(i-1)}_j])$.
- Let $[q^{(0)}_j] = [v^{(\mathsf{ind}_0)}_j]$.
- For $i \in [k]$, compute $[q^{(i)}_j] \leftarrow \mathsf{MultF}^{n,t}([q^{(i-1)}_j], [v^{(\mathsf{ind}_i)}_j])$.
- Compute $[c_j] = 1 - [q^{(k)}_j]$.

(7) **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 $\mathsf{Table}_1$. For $i \in \{2, \cdots, n\}$, $P_i$ inputs $\{[c_j]_i\}_{j\in[\beta]}$.
- All parties receive the output $T$.

**Figure 6: CIRCUIT PSI PROTOCOL**

(Sec. 2.3.1), boolean to arithmetic share conversion $\mathcal{F}^{\mathbb{F}_p}_{\mathsf{B2A}}$ (Section 2.3.2), and the $n$-party functionalities from Section 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 $T = f\left(\bigcup_{j\in[\beta]:c_j=1} a_j\right)$. We set things up such that $c_j = 1$, if $a_j \in \cap_{i=1}^n X_i$; else $c_j = 0$. Next, give an overview and describe the protocol formally in Figure 6.

**Protocol Overview.** On input $X_i$ from party $P_i$, for each $i \in [n]$, the protocol proceeds in seven steps: The first two steps of the protocol, namely the *Hashing* and *Invoking the $\mathcal{F}^{\beta,\sigma,N}_{\mathsf{wPSM}}$ functionality*, are same as Steps 2 and 3 of our mPSI protocol (Figure 4). At the end of these steps, $P_1$ holds $\mathsf{Table}_1$ of $\beta$ bins containing one element each and other parties $P_i$'s hold $\mathsf{Table}_i$ with $\beta$ 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 $w_{ij} \in \{0,1\}^\sigma$ such that $y_{ij} = w_{ij}$ iff $\mathsf{Table}_1[j] \in \mathsf{Table}_i[j]$ (except with negligible probability). Now, in the next step, the parties check whether this equality holds or not. Formally, in Step 3, 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 Step 4, 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 5, parties convert these additive shares between $P_1$ and $P_i$ for $i \in [n]\setminus\{1\}$ to $(n,t)$-shares of values $a_j$ such that $a_j$ denotes the number of parties in $[n]\setminus\{1\}$ that have the element stored at $\mathsf{Table}_1[j]$. In Step 6, the task is to securely compute shares of whether $a_j = n-1$ or not. Let $v_j = a_j - (n-1)$.

[7] We note that these four steps of our protocol together follow the blueprint of executing a circuit PSI protocol [16, 26, 45, 64–66] 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 $\mathsf{Table}_1$) across all instantiations. To explicitly spell out this consistent hashing, we make a whitebox use of the circuit-PSI blueprint from [65].

Now, using property of fields with prime order, $v_j = 0$ (and hence, $a_j = n-1$) 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 $a_j = n-1$.

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 $j \in [\beta]$.

**Remark.** The well-known/standard definition of circuit-PSI [13, 45, 64–66] outputs shares of 0/1 values ($c_j$) and these are given as input to the circuit $C_{\beta,\sigma,p}$. We compute the same in step 6 of our protocol because it offers the flexibility to compute arbitrary functions. However, circuits for certain functions might themselves involve computing equality checks on $c_j$'s. For such functions, one can consider an optimization wherein $C_{\beta,\sigma,p}$ takes shares of $a_j$, for all $j \in [\beta]$ (computed in step 5 of the protocol) as input and $C_{\beta,\sigma,p}$ checks if $a_j = n-1$ to determine if the corresponding element is in the intersection. On the other hand, many functions, such as cardinality of intersection, can be computed by adding the $c_j$ values and no additional equality checks are needed in $C_{\beta,\sigma,p}$.

## 4.2 Correctness and Security Proof

Theorem 4.1. *The protocol in Figure 6 securely realizes $\mathcal{F}_{\mathsf{C-PSI}}^{n,m,f}$ in the $\mathcal{F}$-hybrid model, where $\mathcal{F} = (\mathcal{F}_{\mathsf{wPSM}}^{\beta,\sigma,N}, \mathsf{ConvertShares}^{n,t}, \mathcal{F}_{\mathsf{EQ}}^{\sigma}, \mathcal{F}_{\mathsf{B2A}}^{\mathbb{F}_p}, \mathsf{MultF}^{n,t})$, against a semi-honest adversary corrupting $t < n/2$ parties. Our protocol has total communication at most $2mn(\lambda\kappa + 36(\log n)^2)$ with at most $4\lceil \log n \rceil + \lceil \log \sigma \rceil + 6$ rounds.*

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 $\kappa$. 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 at most $1 - 2^{-41}$.

As we will see later, steps 3–6 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 3) By correctness of $\mathcal{F}_{\mathsf{EQ}}^{\sigma}$, for each $i \in [n] \setminus \{1\}$, $eq_{ij}$ equals 1 when $y_{ij} = w_{ij}$ and 0 otherwise.
- (Step 4) By correctness of $\mathcal{F}_{\mathsf{B2A}}^{\mathbb{F}_p}$, for each $i \in [n] \setminus \{1\}$, $f_{ij} = eq_{ij}$.
- (Step 5) By correctness of $\mathsf{ConvertShares}^{n,t}$, $a_j = \sum_{i=2}^{n} f_{ij} < n$.
- (Step 6) First, $v_j^{(0)} = a_j - (n-1)$. 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 $v_j = 0$. Hence, $c_j = 1$ if and only if $a_j = n-1$. 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 $i \in \{2, \ldots, n\}$. We now prove the following two lemmata.

Lemma 4.2. $Y^* \subseteq Y$.

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 $\mathcal{F}_{\mathsf{wPSM}}^{\beta,\sigma,N}$ 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 4.3. *$Y \subseteq Y^*$ with probability at least $1 - 2^{-\kappa-1}$.*

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.

- $e \in X_1$: Since $e \notin X_{i^*}$ and any dummy elements inserted by $P_{i^*}$ are distinct from real elements, it holds that $e \notin \mathsf{Table}_{i^*}[j]$.
- $e \notin X_1$: Then, $e$ is a dummy element inserted by $P_1$. Since dummy elements of $P_i^*$ are distinct from dummy elements of $P_1$, it holds that $e \notin \mathsf{Table}_{i^*}[j]$.

Since $e \in Y$, it holds that $c_j = 1$ and hence, $eq_{i^*j} = 1$, i.e., $y_{i^*j} = w_{i^*j}$. Now, probability that $y_{i^*j} = w_{i^*j}$ when $e \notin \mathsf{Table}_{i^*}$ is at most $2^{-\sigma}$. Note that this is the probability that $\mathsf{Table}_1[j] \in Y \setminus Y^*$. By union bound over all bins it holds that with probability at least $1 - \beta 2^{-\sigma}$ the set $Y \setminus Y^*$ is empty. □

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

**Security.** 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 ($P_1$ is honest)**: The hashing step is local, and can be executed by the simulator using the inputs of the corrupted parties. In Step 2, $P_1$ and $P_i$ (for each $i \in \{2, \cdots, n\}$) invoke the $\mathcal{F}_{\mathsf{wPSM}}^{\beta,\sigma,N}$ 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 $\mathcal{F}_{\mathsf{wPSM}}^{\beta,\sigma,N}$). In Steps 3 and 4, 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 only one of the two boolean and additive shares, $\{\langle eq_{ij} \rangle_i^B\}_{i \in C, j \in [\beta]}$ 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 5, besides the local computations, which the simulator can do, the corrupted parties get at most $t$ shares of $a_j$, for each $j \in [\beta]$, which can be picked as shares of a random value by the simulator (by the security of secret sharing). In Step 6, 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 7, the simulator can set the output as $T$.

- **Case 2 ($P_1$ is corrupt)**: The simulation of the hashing step is exactly the same as in Case 1. In Step 2, $P_1$ and $P_i$ (for each $i \in \{2, \cdots, n\}$) invoke the $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$ 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 $\{w_{ij}\}_{i \in C \setminus \{1\}, j \in [\beta]}$. 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 $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$ and since the simulator has both $\mathsf{Table}_1$ and $\mathsf{Table}_i$). For each $i \notin C$, the simulator picks $y_{ij}$'s at random. In Steps 3 and 4, for each $i \in \{2, \cdots, n\}$, parties $P_1$ and $P_i$ invoke the $\mathcal{F}_{\text{EQ}}^{\sigma}$ and $\mathcal{F}_{\text{B2A}}^{\mathbb{F}_p}$ functionalities and the corrupted parties see both the boolean and additive shares for $i \in C$, $\{\langle eq_{ij}\rangle_1^B, \langle eq_{ij}\rangle_i^B\}_{i \in C \setminus \{1\}, j \in [\beta]}$ and $\{\langle f_{ij}\rangle_1, \langle f_{ij}\rangle_i\}_{i \in C \setminus \{1\}, j \in [\beta]}$, 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 $i \in C \setminus \{1\}$ and $j \in [\beta]$, 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. It then generates the boolean and arithmetic shares of the $eq_{ij}$'s and $f_{ij}$'s, respectively. For each $i \in [n] \setminus C$, 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 5 and 6 is exactly as in Case 1. Again, for Step 7, the simulator sets the output as $T$.

**Complexity.** We instantiate $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$ using the polynomial-based batch-OPPRF and set $p$ to be the smallest prime greater than $n$, i.e., $\lceil \log p \rceil \leq \lceil \log n \rceil + 1$. We split the communication of the protocol into two parts. 1) Steps 1–4 where $P_1$ interacts with each $P_i$ for $i \in [2, \ldots, n]$ separately. 2) Steps 5 and 6 where parties run $n$-party functionalities. In the first part, protocol invokes $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$, $\mathcal{F}_{\text{EQ}}^{\sigma}$, $\mathcal{F}_{\text{B2A}}^{\mathbb{F}_p}$ 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 $\mathsf{ConvertShares}^{n,t}$ and $\mathsf{MultF}^{n,t}$ functionalities $\beta$ and (at most) $2\beta\lceil \log n \rceil$ times respectively. The concrete cost of this part is at most $m(n-1)(36(\lceil \log n \rceil)^2 + 40\lceil \log n \rceil)$. Summing up both gives the total complexity of our 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 $\mathcal{F}_{\text{QPSI}}^{n,m,k}$ 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 $P_1$.

**Figure 7: Quorum PSI Functionality $\mathcal{F}_{\text{QPSI}}^{n,m,k}$**

## 5.1 Quorum PSI Protocol

**Building blocks:** Our protocol uses the two-party functionalities weak private set membership $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$ (Section 2.4), equality test $\mathcal{F}_{\text{EQ}}^{\sigma}$ (Sec. 2.3.1), boolean to arithmetic share conversion $\mathcal{F}_{\text{B2A}}^{\mathbb{F}_p}$ (Section 2.3.2), the $n$-party functionalities from Section 2.5, and the weak comparison functionality $\mathcal{F}_{\text{w-CMP}}^{p,k,n,t}$ (Section 2.6) in the honest majority setting. In Section 5.2 we provide two weak comparison protocols that realize $\mathcal{F}_{\text{w-CMP}}^{p,k,n,t}$ 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 16). At a high level, for each $j \in [\beta]$, after obtaining $(n,t)$-shares of value $a_j$ 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 $a_j$ 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 the field $\mathbb{F}_p$, where $p$ is a prime (larger than $n$) that depends on specific instantiation of $\mathcal{F}_{\text{w-CMP}}$.
**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 $\sigma$ bits by first hashing the elements using an appropriate universal hash function.
**Protocol:** The protocol executes Steps 1–5 of the circuit-PSI protocol from Figure 6. After this step, for each $j \in [\beta]$, parties hold $[a_j]$. Then, parties $P_1, \cdots, P_n$ invoke $\mathcal{F}_{\text{w-CMP}}^{p,k,n,t}$ with $P_i$'s input being $[a_j]_i$ 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 instantiations of $\mathcal{F}_{\text{w-CMP}}^{p,k,n,t}$ described in the next section, we have two protocols for quorum PSI, Quorum-I and Quorum-II.

Theorem 5.1. *The protocol given above securely realizes $\mathcal{F}_{\text{QPSI}}^{n,m,k}$ in the $\mathcal{F}$-hybrid model, where $\mathcal{F} = (\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}, \mathcal{F}_{\text{EQ}}^{\sigma}, \mathcal{F}_{\text{B2A}}^{\mathbb{F}_p}, \mathcal{F}_{\text{w-CMP}}^{p,k,n,t}, \mathsf{ConvertShares}^{n,t}, \mathsf{MultF}^{n,t})$, against a semi-honest adversary corrupting $t < n/2$ parties. The communication complexities of the two quorum PSI protocols, Quorum-I and Quorum-II, are at most $m(n-1)(\lambda\sigma + 5.8\lambda + 14\sigma + 18k'(\lceil \log n \rceil + 1) + 22\tau + 10\lceil \log n \rceil)$, where $k' = \min\{k-1, n-k\}$ and $\tau = \kappa + \lceil \log m \rceil + 3$, and $m(n-1)(\lambda\sigma + 5.8\lambda + 14\sigma + 27(\lceil \log n \rceil + 1)(\kappa + \lceil \log n \rceil + 1)^2)$, respectively, and their round complexities are at most $10 + \lceil \log \sigma \rceil + 2k'$ and $8 + \lceil \log \sigma \rceil + 2\lceil \log n \rceil$, respectively.*

We give a complete proof of the correctness and security in Appendix F.1 and discuss the complexities in Appendix F.2.

## 5.2 Weak Comparison Protocols

In this section, we describe two protocols realizing the weak comparison functionality $\mathcal{F}_{\text{w-CMP}}^{p,k,n,t}$ (Section 2.6), w-CMP1 and w-CMP2 and discuss their trade-offs in Appendix F.2.1.

*5.2.1 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 $0 \leq a < n$ and $a \in \mathbb{F}_p$), the protocol proceeds as follows: For $k \geq n/2$, 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 $n > x \geq k$. 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 $0 \leq x < k$. The protocol takes as input $[a]$ and for $J$ values of random $s_j$'s in $\mathbb{F}_p$, evaluates $[\psi(a)\cdot s_j]$ using the $\mathsf{MultF}^{n,t}$ functionality. Subsequently, $P_1$ recovers $\psi(a)\cdot s_j$, for each $j \in [J]$. If $\psi(a)\cdot s_j = 0$, for each $j \in [J]$, then $\psi(a) = 0$ with probability $1 - 2^{-\tau}$, where $\tau$ is a configurable parameter for correctness of the construction. Hence, by the property of $\psi$, $P_1$ 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, $p, n, t, \tau, J$ and $k$ are such that $p$ is a prime, $p > n > k$, $n > 2t$, $\tau$ is a configurable parameter for correctness of construction and $J = \lceil \frac{\tau}{\log|\mathbb{F}_p|} \rceil$. Additions and multiplications are over the field $\mathbb{F}_p$.

Define the polynomial $\psi$ (publicly known to all parties):

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

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

**Protocol**:

(1) **Pre-processing:** $P_1, \cdots, P_n$ run: $[s_1], \cdots, [s_J] \leftarrow \mathsf{RandomF}^{n,t}(J)$.

(2) **Evaluating the polynomial:** $P_1, \cdots, P_n$ do:
- On input $[a]$, invoke $\mathsf{MultF}^{n,t}$ to compute all the required $[a^i]$, followed by scalar multiplications and additions to compute $[\psi(a)]$.
- For each $j \in [J]$, do:
  - $[v_j] \leftarrow \mathsf{MultF}^{n,t}([\psi(a)], [s_j])$.
  - $v_j \leftarrow \mathsf{Reveal}^{n,t}([v_j])$.

**Output:** If $k \geq n/2$, if $v_j = 0, \forall j \in [J]$, $P_1$ sets comp = 1, else sets comp = 0. If $k < n/2$, if $v_j = 0, \forall j \in [J]$, $P_1$ sets comp = 0 else sets comp = 1. Other parties get no output.

**Figure 8: WEAK COMPARISON PROTOCOL I**

Theorem 5.2. *The protocol in Figure 8 securely realizes $\mathcal{F}_{\text{w-CMP}}^{p,k,n,t}$ 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. The total amortized communication cost of the protocol is at most $14k'(n-1)(\lceil \log n \rceil + 1) + 17\tau(n-1)$ and the round complexity is $6 + 2k'$, where $k' = \min\{k-1, n-k\}$.*

We give a complete proof of Theorem 5.2 in Appendix E.1 and discuss the complexities in Appendix F.2.1.

*5.2.2 Weak Comparison Protocol* w-CMP2. This protocol is a slight modification of the comparison protocol from [12]. The main idea of their comparison protocol is as follows: For $0 \leq a, k < n$, $a \geq k$ 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 $\mathsf{MultF}^{n,t}$, $\mathsf{RandomF}^{n,t}$ and $\mathsf{Reveal}^{n,t}$. Corresponding to the instantiations of these functionalities used in [12], their protocol has an $n^2$ factor in the communication complexity. Instead, we use the instantiations from [21] 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 $\mathcal{F}_{\text{w-CMP}}^{p,k,n,t}$, in Appendix E.2.

## 6 IMPLEMENTATION AND EVALUATION

In this section, we discuss the performance of our mPSI (multiparty PSI) protocols[8]. Let Protocol A, Protocol B and Protocol C denote our mPSI protocols when instantiated with polynomial-based batch OPPRF [65], table-based OPPRF [51] and relaxed batch OPPRF [13] respectively. We compare the performance of our mPSI protocols with the state-of-the-art mPSI protocol in literature [51].

*Protocol Parameters.* We set statistical security parameter $\kappa$=40 and computational security parameter $\lambda$=128. Correctness of Theorem 3.1 requires Cuckoo hashing failure in Step 2 (Figure 4) to be at most $2^{-41}$. Similar to [13, 51, 65, 67], we use the empirical analysis to instantiate the parameters of Cuckoo hashing scheme in the stash-less setting as $\beta = 1.28m$ for $K = 3$ (see Section 2.2). Based on Theorem 3.1, we set size of elements $\sigma = \kappa + \lceil \log m \rceil + 3$ to achieve statistical security of $\kappa$ bits. Hence, the minimum element size $\sigma$ required in mPSI protocol to ensure that the failure probability is at most $2^{-40}$ is 55, 59 and 63 for input set sizes $2^{12}$, $2^{16}$ 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., larger of the element size required by these two protocols.

*Implementation Details.* We make use of the implementation of polynomial-based batch OPPRF [65] and table-based OPPRF [51] available at [25] and [60] respectively. For implementation of relaxed batch OPPRF [13] and equality test functionality $\mathcal{F}_{\text{EQ}}^{\ell}$ [13, 23, 32, 68], we use the code available at [58]. For Boolean to Arithmetic share conversion functionality $\mathcal{F}_{\text{B2A}}^{\mathbb{F}}$ [68], we use the implementation of correlated OTs available at [57]. Finally, we use the code available at [20] for multiparty functionalities [21, 52] (see Section 2.5).

*Experimental Setup.* Similar to [51], 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 the Linux *tc* command. We configure a LAN connection with bandwidth 10

[8] Code available at https://aka.ms/PQC-mPSI

| $n, t$ | 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 [51] 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 [51] and** Protocol A**.**

| 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 [51] 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.**

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 [51, Section 7]. We compare the performance with the implementation of KMPRT provided at [60].

## 6.1 Communication Comparison of mPSI

In this section, we compare the concrete communication cost of our most communication frugal mPSI protocol Protocol A with KMPRT [51]. Table 2 summarizes the overall communication cost of of both of these protocols. As can be observed from the table, 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 more light-weight compared to clients in KMPRT as illustrated by Table 3. The concrete communication cost of a client in Protocol A is $2.6 - 6\times$ less than that of KMPRT protocol.

[9]Protocol A's implementation builds on the Polynomial based Batch OPPRF's [65] implementation at [25] that uses Mersenne prime $2^{61} - 1$. We note 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 Protocol A over a field with at least $2^{63}$ elements, i.e., each element is represented using 64 bits. However, in the implementation, since an element over prime field with Mersenne Prime $2^{61} - 1$ is communicated using 64 bits, the number in the table gives a correct bound on communication of Protocol A with 40 bits of security.

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 wPSM functionality followed by the interaction in Evaluation phase (step 4 in Figure 4).

## 6.2 Run-time Comparison of mPSI

In this section, we compare the run-times of our mPSI protocols with that of KMPRT [51]. In Table 4, we report the run-time of KMPRT along with the run-time of our best performing 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. Our best protocol achieves a speedup of $1.2 - 4.9\times$ and $1.3 - 6.2\times$ over KMPRT in LAN and WAN settings respectively. This is because KMPRT protocol involves execution of $n(t + 2) - 1$ instances of OPPRF protocol whereas our protocols involve execution of just $n - 1$ wPSM protocols followed by a very efficient Evaluation phase.

In the LAN Setting, Protocol A is the least efficient of our three mPSI protocols. 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 set sizes. But as

| $n$ | 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 1–4 of our Circuit PSI and qPSI protocols.**

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.

*6.2.1 Cost Estimation of HE-based mPSI.* [44] gave mPSI schemes based on threshold additively homomorphic encryption (AHE) schemes and we estimate the cost of their most efficient variant that uses hashing to significantly reduce the computation cost. The protocol has four main steps: 1) distributed key generation phase for an AHE scheme; 2) encryption phase where each client (parties $P_2, \cdots, P_n$) generates $B\zeta$ ciphertexts (using the AHE encryption) and sends them to the leader $P_1$, where $B = m/\log m$ is the number of bins after hashing, and each bin has $\zeta = m/B + \sqrt{m \log B/B}$ values; 3) homomorphic evaluation phase where the leader $P_1$ performs $B\zeta n + \zeta m$ homomorphic additions and $m\zeta$ homomorphic scalar multiplications (for plaintext size of 32 bits); 4) threshold decryption phase where parties together do a decryption of $m$ ciphertexts. As the end-to-end protocol would require concretely efficient protocols for distributed key generation (Step 1), we focus only on AHE schemes for which such concretely efficient protocols are known (and hence do not consider lattice based AHE schemes [8, 33, 34]). In particular, as suggested by [44], we use El Gamal and Paillier based AHE schemes and estimate the cost of Steps 2 and 3 that is clearly a strict lower bound for the overall cost.

As suggested by [44] we use the costs of microbenchmarking provided in [29] for both El Gamal and Paillier encryption schemes. For $m = 2^{20}$ and $n = 15$, this gives us a computation cost lower bound of 729s for the El Gamal instantiation, and 7973s for the Paillier instantiation in a similar setting as ours. In contrast, our end-to-end protocol takes only $\approx$ 40s and hence, is at least 18× and 200× faster than El Gamal and Paillier based schemes, respectively. Communication estimate for these steps is similar to ours in case of El Gamal and is much higher for the Paillier based scheme. Similarly, for $m = 2^{12}$ and $n = 5$, these two steps take at least 4s with El Gamal and 35s with Paillier, and communication of both schemes is much worse than our scheme. In comparison, our protocol executes in 0.23s and is 17× and 150× faster than the El Gamal and Paillier based schemes, respectively.

## 6.3 Performance of Circuit PSI and qPSI

**Circuit PSI.** As discussed in Section 4, in steps 5,6 (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 up to 30 parties and also for all the settings we consider. However, the smallest prime $p$ for which the implementation of protocols for multiparty functionalities from Section 2.5 is available (at [20]) is for the Mersenne prime $2^{31} - 1$, which is an overkill for our implementations. Based on the concrete communication analysis discussed in Section 4.2, we observe that the communication in steps 5,6 using Mersenne prime 31 is $< 8.2\%$ of the communication involved in steps 1 – 4 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 1–4 of the protocol is a strong indicator of its overall performance.

We illustrate the performance of steps 1–4 in Table 5 when wPSM is instantiated using relaxed-batch OPPRF [13]. These numbers can be extrapolated to estimate the overall run-time of the protocol. For instance, we estimate our Circuit PSI protocol to take 12.19$s$ and 80.09$s$ 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 Theorem 5.1). The aforementioned discussion in the context of Circuit PSI protocol also holds for protocol Quorum-I. From the concrete communication analysis in Appendix F.2, for the values of $n, t, m$ considered in experiments, the communication in step 5 (see Figure 16) using Mersenne prime 31 is $< 21\%$ of the communication involved in steps 1 – 4 for all values of $k \leq n-1$. Hence, for instance, the run-time of Quorum-I protocol can be estimated to be 5.49$s$ and 37.85$s$ in LAN and WAN setting respectively for 15 parties with $t = 7$, $m = 2^{16}$ and any $k \leq 14$.

## REFERENCES


[1] Aydin Abadi, Sotirios Terzis, and Changyu Dong. 2015. O-PSI: Delegated Private Set Intersection on Outsourced Datasets. In *ICT Systems Security and Privacy Protection - 30th IFIP TC 11 International Conference, SEC 2015, Hamburg, Germany, May 26-28, 2015, Proceedings (IFIP Advances in Information and Communication Technology, Vol. 455)*, Hannes Federrath and Dieter Gollmann (Eds.). Springer, 3–17. https://doi.org/10.1007/978-3-319-18467-8_1

[2] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. 2016. High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM, 805–817. https://doi.org/10.1145/2976749.2978331

[3] Saikrishna Badrinarayanan, Peihan Miao, and Peter Rindal. 2020. Multi-Party Threshold Private Set Intersection with Sublinear Communication. *IACR Cryptol. ePrint Arch.* 2020 (2020), 600. https://eprint.iacr.org/2020/600

[4] Donald Beaver, Silvio Micali, and Phillip Rogaway. 1990. The Round Complexity of Secure Protocols (Extended Abstract). In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, Harriet Ortiz (Ed.). ACM, 503–513. https://doi.org/10.1145/100216.100287

[5] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, Janos Simon (Ed.). ACM, 1–10. https://doi.org/10.1145/62212.62213

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

[7] Dan Bogdanov, Sven Laur, and Jan Willemson. 2008. Sharemind: A Framework for Fast Privacy-Preserving Computations. In *Computer Security - ESORICS 2008, 13th*

*European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings (Lecture Notes in Computer Science, Vol. 5283)*, Sushil Jajodia and Javier López (Eds.). Springer, 192–206. https://doi.org/10.1007/978-3-540-88313-5_13

[8] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. 2018. Threshold Cryptosystems from Threshold Fully Homomorphic Encryption. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 10991)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, 565–596. https://doi.org/10.1007/978-3-319-96884-1_19

[9] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. 2019. Efficient Pseudorandom Correlation Generators: Silent OT Extension and More. *Advances in Cryptology—Crypto 2019*, Part III, LNCS, pages 489–518. Springer,.

[10] Pedro Branco, Nico Döttling, and Sihang Pu. 2020. Multiparty Cardinality Testing for Threshold Private Set Intersection. *IACR Cryptol. ePrint Arch.* 2020 (2020), 1307. https://eprint.iacr.org/2020/1307

[11] Ran Canetti. 2001. 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*. IEEE Computer Society, 136–145. https://doi.org/10.1109/SFCS.2001.959888

[12] Octavian Catrina and Sebastiaan de Hoogh. 2010. Improved Primitives for Secure Multiparty Integer Computation. In *Security and Cryptography for Networks, 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6280)*, Juan A. Garay and Roberto De Prisco (Eds.). Springer, 182–199. https://doi.org/10.1007/978-3-642-15317-4_13

[13] Nishanth Chandran, Divya Gupta, and Akash Shah. 2022. Circuit-PSI with Linear Complexity via Relaxed Batch OPPRF. *Proc. Priv. Enhancing Technol.* 2022, 1 (2022).

[14] Melissa Chase and Peihan Miao. 2020. Private Set Intersection in the Internet Setting from Lightweight Oblivious PRF. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 12172)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, 34–63. https://doi.org/10.1007/978-3-030-56877-1_2

[15] Jung Hee Cheon, Stanislaw Jarecki, and Jae Hong Seo. 2012. Multi-Party Privacy-Preserving Set Intersection with Quasi-Linear Complexity. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* 95-A, 8 (2012), 1366–1378. https://doi.org/10.1587/transfun.E95.A.1366

[16] Michele Ciampi and Claudio Orlandi. 2018. Combining Private Set-Intersection with Secure Two-Party Computation. In *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11035)*, Dario Catalano and Roberto De Prisco (Eds.). Springer, 464–482. https://doi.org/10.1007/978-3-319-98113-0_25

[17] Ronald Cramer, Ivan Damgård, and Ueli M. Maurer. 2000. General Secure Multi-party Computation from any Linear Secret-Sharing Scheme. In *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding (Lecture Notes in Computer Science, Vol. 1807)*, Bart Preneel (Ed.). Springer, 316–334. https://doi.org/10.1007/3-540-45539-6_22

[18] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. 2010. Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6477)*, Masayuki Abe (Ed.). Springer, 213–231. https://doi.org/10.1007/978-3-642-17373-8_13

[19] Emiliano De Cristofaro and Gene Tsudik. 2010. Practical Private Set Intersection Protocols with Linear Complexity. In *Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, Spain, January 25-28, 2010, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 6052)*, Radu Sion (Ed.). Springer, 143–159. https://doi.org/10.1007/978-3-642-14577-3_13

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

[21] Ivan Damgård and Jesper Buus Nielsen. 2007. Scalable and Unconditionally Secure Multiparty Computation. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings (Lecture Notes in Computer Science, Vol. 4622)*, Alfred Menezes (Ed.). Springer, 572–590. https://doi.org/10.1007/978-3-540-74143-5_32

[22] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society. https://www.ndss-symposium.org/ndss2015/aby---framework-efficient-mixed-protocol-secure-two-party-computation

[23] Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. 2017. 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. https://www.ndss-symposium.org/ndss2017/ndss-2017-programme/pushing-communication-barrier-secure-computation-using-lookup-tables/

[24] Changyu Dong, Liqun Chen, and Zikai Wen. 2013. When private set intersection meets big data: an efficient and scalable protocol. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung (Eds.). ACM, 789–800. https://doi.org/10.1145/2508859.2516701

[25] encryptogroup. 2020. OPPRF-PSI. https://github.com/encryptogroup/OPPRF-PSI Accessed: 2020-08-31.

[26] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. 2019. Private Set Intersection with Linear Communication from General Assumptions. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society, WPES@CCS 2019, London, UK, November 11, 2019*, Lorenzo Cavallaro, Johannes Kinder, and Josep Domingo-Ferrer (Eds.). ACM, 14–25. https://doi.org/10.1145/3338498.3358645

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

[28] Financial Action Task Force. 2018. Professional Money Laundering. https://www.fatf-gafi.org/media/fatf/documents/Professional-Money-Laundering.pdf.

[29] Michael J. Freedman, Carmit Hazay, Kobbi Nissim, and Benny Pinkas. 2016. Efficient Set Intersection with Simulation-Based Security. *J. Cryptol.* 29, 1 (2016), 115–155. https://doi.org/10.1007/s00145-014-9190-0

[30] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. 2005. Keyword Search and Oblivious Pseudorandom Functions. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3378)*, Joe Kilian (Ed.). Springer, 303–324. https://doi.org/10.1007/978-3-540-30576-7_17

[31] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. 2004. Efficient Private Matching and Set Intersection. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings (Lecture Notes in Computer Science, Vol. 3027)*, Christian Cachin and Jan Camenisch (Eds.). Springer, 1–19. https://doi.org/10.1007/978-3-540-24676-3_1

[32] Juan A. Garay, Berry Schoenmakers, and José Villegas. 2007. Practical and Secure Solutions for Integer Comparison. In *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings (Lecture Notes in Computer Science, Vol. 4450)*, Tatsuaki Okamoto and Xiaoyun Wang (Eds.). Springer, 330–342. https://doi.org/10.1007/978-3-540-71677-8_22

[33] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, Michael Mitzenmacher (Ed.). ACM, 169–178. https://doi.org/10.1145/1536414.1536440

[34] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I (Lecture Notes in Computer Science, Vol. 8042)*, Ran Canetti and Juan A. Garay (Eds.). Springer, 75–92. https://doi.org/10.1007/978-3-642-40041-4_5

[35] Satrajit Ghosh and Tobias Nilges. 2019. An Algebraic Approach to Maliciously Secure Private Set Intersection. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 11478)*, Yuval Ishai and Vincent Rijmen (Eds.). Springer, 154–185. https://doi.org/10.1007/978-3-030-17659-4_6

[36] Satrajit Ghosh and Mark Simkin. 2019. The Communication Complexity of Threshold Private Set Intersection. *IACR Cryptol. ePrint Arch.* 2019 (2019), 175. https://eprint.iacr.org/2019/175

[37] Oded Goldreich. 2004. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press. https://doi.org/10.1017/CBO9780511721656

[38] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. 1984. 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*. IEEE Computer Society, 464–479. https://doi.org/10.1109/SFCS.1984.715949

[39] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, Alfred V. Aho (Ed.). ACM, 218–229. https://doi.org/10.1145/28395.28420

[40] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, Alfred V. Aho (Ed.). ACM, 218–229. https:

//doi.org/10.1145/28395.28420

[41] Michael T. Goodrich and Michael Mitzenmacher. 2011. Privacy-Preserving Access of Outsourced Data via Oblivious RAM Simulation. In *Automata, Languages and Programming - 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 6756)*, Luca Aceto, Monika Henzinger, and Jirí Sgall (Eds.). Springer, 576–587. https://doi.org/10.1007/978-3-642-22012-8_46

[42] Per A. Hallgren, Claudio Orlandi, and Andrei Sabelfeld. 2017. PrivatePool: Privacy-Preserving Ridesharing. In *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*. IEEE Computer Society, 276–291. https://doi.org/10.1109/CSF.2017.24

[43] Carmit Hazay and Kobbi Nissim. 2010. Efficient Set Operations in the Presence of Malicious Adversaries. In *Public Key Cryptography - PKC 2010, 13th International Conference on Practice and Theory in Public Key Cryptography, Paris, France, May 26-28, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6056)*, Phong Q. Nguyen and David Pointcheval (Eds.). Springer, 312–331. https://doi.org/10.1007/978-3-642-13013-7_19

[44] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. 2017. Scalable Multi-party Private Set-Intersection. In *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 (Lecture Notes in Computer Science, Vol. 10174)*, Serge Fehr (Ed.). Springer, 175–203. https://doi.org/10.1007/978-3-662-54365-8_8

[45] Yan Huang, David Evans, and Jonathan Katz. 2012. 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. https://www.ndss-symposium.org/ndss2012/private-set-intersection-are-garbled-circuits-better-custom-protocols

[46] Bernardo A. Huberman, Matthew K. Franklin, and Tad Hogg. 1999. Enhancing privacy and trust in electronic communities. In *Proceedings of the First ACM Conference on Electronic Commerce (EC-99), Denver, CO, USA, November 3-5, 1999*, Stuart I. Feldman and Michael P. Wellman (Eds.). ACM, 78–86. https://doi.org/10.1145/336992.337012

[47] Roi Inbar, Eran Omri, and Benny Pinkas. 2018. Efficient Scalable Multiparty Private Set-Intersection via Garbled Bloom Filters. In *Security and Cryptography for Networks - 11th International Conference, SCN 2018, Amalfi, Italy, September 5-7, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11035)*, Dario Catalano and Roberto De Prisco (Eds.). Springer, 235–252. https://doi.org/10.1007/978-3-319-98113-0_13

[48] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. 2020. On Deploying Secure Computing: Private Intersection-Sum-with-Cardinality. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*. IEEE, 370–389. https://doi.org/10.1109/EuroSP48549.2020.00031

[49] Lea Kissner and Dawn Xiaodong Song. 2005. Privacy-Preserving Set Operations. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3621)*, Victor Shoup (Ed.). Springer, 241–257. https://doi.org/10.1007/11535218_15

[50] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. 2016. Efficient Batched Oblivious PRF with Applications to Private Set Intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM, 818–829. https://doi.org/10.1145/2976749.2978381

[51] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. 2017. Practical Multi-party Private Set Intersection from Symmetric-Key Techniques. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 1257–1272. https://doi.org/10.1145/3133956.3134065

[52] Yehuda Lindell and Ariel Nof. 2017. A Framework for Constructing Fast MPC over Arithmetic Circuits with Malicious Adversaries and an Honest-Majority. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 259–276. https://doi.org/10.1145/3133956.3133999

[53] Catherine A. Meadows. 1986. 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*. IEEE Computer Society, 134–137. https://doi.org/10.1109/SP.1986.10022

[54] Atsuko Miyaji and Shohei Nishida. 2015. A Scalable Multiparty Private Set Intersection. In *Network and System Security - 9th International Conference, NSS 2015, New York, NY, USA, November 3-5, 2015, Proceedings (Lecture Notes in Computer Science, Vol. 9408)*, Meikang Qiu, Shouhuai Xu, Moti Yung, and Haibo Zhang (Eds.). Springer, 376–385. https://doi.org/10.1007/978-3-319-25645-0_26

[55] Payman Mohassel and Peter Rindal. 2018. ABY$^3$: A Mixed Protocol Framework for Machine Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 35–52. https://doi.org/10.1145/3243734.3243760

[56] Payman Mohassel, Peter Rindal, and Mike Rosulek. 2020. Fast Database Joins and PSI for Secret Shared Data. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM, 1271–1287. https://doi.org/10.1145/3372297.3423358

[57] mpc msri. 2020. EzPC. https://github.com/mpc-msri/EzPC Accessed: 2021-01-17.

[58] mpc msri. 2021. 2PC-Circuit-PSI. https://aka.ms/2PC-Circuit-PSI

[59] Michele Orrù, Emmanuela Orsini, and Peter Scholl. 2017. Actively Secure 1-out-of-N OT Extension with Application to Private Set Intersection. In *Topics in Cryptology - CT-RSA 2017 - The Cryptographers' Track at the RSA Conference 2017, San Francisco, CA, USA, February 14-17, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10159)*, Helena Handschuh (Ed.). Springer, 381–396. https://doi.org/10.1007/978-3-319-52153-4_22

[60] osu crypto. 2020. MultipartyPSI. https://github.com/osu-crypto/MultipartyPSI. Accessed: 2020-06-30.

[61] Rasmus Pagh and Flemming Friche Rodler. 2001. Cuckoo Hashing. In *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings (Lecture Notes in Computer Science, Vol. 2161)*, Friedhelm Meyer auf der Heide (Ed.). Springer, 121–133. https://doi.org/10.1007/3-540-44676-1_10

[62] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2019. SpOT-Light: Lightweight Private Set Intersection from Sparse OT Extension. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 11694)*, Alexandra Boldyreva and Daniele Micciancio (Eds.). Springer, 401–431. https://doi.org/10.1007/978-3-030-26954-8_13

[63] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2020. PSI from PaXoS: Fast, Malicious Private Set Intersection. In *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 (Lecture Notes in Computer Science, Vol. 12106)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, 739–767. https://doi.org/10.1007/978-3-030-45724-2_25

[64] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. 2015. Phasing: Private Set Intersection Using Permutation-based Hashing. In *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*, Jaeyeon Jung and Thorsten Holz (Eds.). USENIX Association, 515–530. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/pinkas

[65] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. 2019. Efficient Circuit-Based PSI with Linear Communication. In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 11478)*, Yuval Ishai and Vincent Rijmen (Eds.). Springer, 122–153. https://doi.org/10.1007/978-3-030-17659-4_5

[66] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. 2018. Efficient Circuit-Based PSI via Cuckoo Hashing. In *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 (Lecture Notes in Computer Science, Vol. 10822)*, Jesper Buus Nielsen and Vincent Rijmen (Eds.). Springer, 125–157. https://doi.org/10.1007/978-3-319-78372-7_5

[67] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2018. Scalable Private Set Intersection Based on OT Extension. *ACM Trans. Priv. Secur.* 21, 2 (2018), 7:1–7:35. https://doi.org/10.1145/3154794

[68] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow2: Practical 2-Party Secure Inference. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM, 325–342. https://doi.org/10.1145/3372297.3417274

[69] Peter Rindal and Mike Rosulek. 2017. Improved Private Set Intersection Against Malicious Adversaries. In *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 (Lecture Notes in Computer Science, Vol. 10210)*, Jean-Sébastien Coron and Jesper Buus Nielsen (Eds.). 235–259. https://doi.org/10.1007/978-3-319-56620-7_9

[70] Peter Rindal and Mike Rosulek. 2017. Malicious-Secure Private Set Intersection via Dual Execution. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 1229–1242. https://doi.org/10.1145/3133956.3134044

[71] Yingpeng Sang and Hong Shen. 2007. Privacy Preserving Set Intersection Protocol Secure against Malicious Behaviors. In *Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT 2007), 3-6 December 2007, Adelaide, Australia*, David S. Munro, Hong Shen, Quan Z. Sheng, Henry Detmold, Katrina E. Falkner, Cruz Izu, Paul D. Coddington, Bradley Alexander, and Si-Qing Zheng (Eds.). IEEE Computer Society, 461–468. https://doi.org/10.1109/PDCAT.2007.59

[72] Yingpeng Sang and Hong Shen. 2008. Privacy preserving set intersection based on bilinear groups. In *Computer Science 2008, Thirty-First Australasian Computer Science Conference (ACSC2008), Wollongong, NSW, Australia, January 22-25, 2008 (CRPIT, Vol. 74)*, Gillian Dobbie and Bernard Mans (Eds.). Australian Computer Society, 47–54. https://dl.acm.org/citation.cfm?id=1378290

[73] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (1979), 612–613.

[74] Adi Shamir. 1980. On the Power of Commutativity in Cryptography. In *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherlands, July 14-18, 1980, Proceedings (Lecture Notes in Computer Science, Vol. 85)*, J. W. de Bakker and Jan van Leeuwen (Eds.). Springer, 582–595. https://doi.org/10.1007/3-540-10003-2_100

[75] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. 2020. Ferret: Fast Extension for Correlated OT with Small Communication. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, USA) *(CCS '20)*. Association for Computing Machinery, New York, NY, USA, 1607–1626. https://doi.org/10.1145/3372297.3417276

[76] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets (Extended Abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*. IEEE Computer Society, 162–167. https://doi.org/10.1109/SFCS.1986.25

[77] Moti Yung. 2015. From Mental Poker to Core Business: Why and How to Deploy Secure Computation Protocols?. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM, 1–2. https://doi.org/10.1145/2810103.2812701

[78] Yihua Zhang, Aaron Steele, and Marina Blanton. 2013. PICCO: a general-purpose compiler for private distributed computation. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung (Eds.). ACM, 813–826. https://doi.org/10.1145/2508859.2516752

[79] Yongjun Zhao and Sherman S. M. Chow. 2017. Are you The One to Share? Secret Transfer with Access Structure. *Proc. Priv. Enhancing Technol.* 2017, 1 (2017), 149–169. https://doi.org/10.1515/popets-2017-0010

[80] Yongjun Zhao and Sherman S. M. Chow. 2018. Can You Find The One for Me?. In *Proceedings of the 2018 Workshop on Privacy in the Electronic Society, WPES@CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, and Aaron Johnson (Eds.). ACM, 54–65. https://doi.org/10.1145/3267323.3268965

# A REMARK ON AUGMENTED SEMI-HONEST MPSI PROTOCOL OF [51]

Here, we describe how the augmented semi-honest mPSI protocol of [51] leaks intersection of the honest parties' sets to the adversary. Consider the following setting: Let $P_1$ and $P_2$ be the two corrupt parties and let element $x$ be present in the input set of all parties except $P_2$. In augmented semi-honest protocol of [51], the relation of zero-sharing phase with OPPRF outputs received by $P_1$ will leak that $x$ belongs to the intersection of honest party sets, even when it clearly doesn't belong to the intersection. Note that, as mentioned before, such leakage is disallowed in standard semi-honest security.

# B INSTANTIATIONS OF THE WPSM FUNCTIONALITY

The wPSM functionality from Section 2.4 can be instantiated using an oblivious programmable pseudorandom function (OPPRF), which was first introduced in [51]. More specifically, 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 informally describe these variants below, and explain how they can be used to realize the $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$ functionality. We refer the reader to [13, 51, 65] for detailed definitions.

*Batch PPRF.* [51] Informally, a pseudorandom function (PRF) [38], 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 $\beta$ instances of a PPRF are used, then the corresponding $\beta$ 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 a Batch PPRF [65].

*OPRF and Batch OPPRF.* An oblivious PRF (OPRF) functionality [30] 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, $\mathcal{F}_{\text{opprf}}$, 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 $\mathcal{F}_{\text{b-opprf}}$.

*Relaxed Batch OPPRF.* [13] 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 $\mathcal{F}_{\text{rb-opprf}}^{d}$, 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 $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$ functionality:

- **Using the Batch OPPRF functionality** [65]: 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 $\mathcal{F}_{\text{b-opprf}}$ 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 $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$ functionality in the $\mathcal{F}_{\text{b-opprf}}$-hybrid model. Specifically, the polynomial-based batch-OPPRF from [65] can be used to instantiate $\mathcal{F}_{\text{b-opprf}}$ in the above construction, which gives a concrete communication cost of $3.5\lambda\beta + N\sigma$ and has a round complexity of 2.

- **Using the OPPRF functionality** [51]: 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 $\max_\beta$ 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, up to the upper-bound $\max_\beta, \forall j \in [\beta]$. The sender and receiver invoke the $\mathcal{F}_{\text{opprf}}$ functionality on inputs $(X_j, T_j)$ and $q_j$ respectively, $\forall j \in [\beta]$. The receiver gets output $y_j$ from $j^{\text{th}}$ 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 $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$ functionality in the $\mathcal{F}_{\text{opprf}}$-hybrid model.
  Specifically, the table-based OPPRF from [51] can be used to instantiate $\mathcal{F}_{\text{opprf}}$ in the above construction, which gives a concrete communication cost of $(4.5\lambda + 2^{\lceil\log(\max_\beta)\rceil}\sigma)\beta$ and a round complexity of 2. For the application of PSI, $\max_\beta$ is $O(\log m/\log\log m)$, where $m$ denotes the size of the input sets.
- **Using the Relaxed Batch OPPRF functionality** [13]: Fix $d = 3$ in the relaxed batch OPPRF functionality, $\mathcal{F}_{\text{rb-opprf}}^{d}$. Let $P_1$ and $P_2$ be the sender and receiver of $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$ functionality respectively. On $P_1$'s input $\{X_j\}_{j\in[\beta]}$ and $P_2$'s input $q_1, \cdots, q_\beta$, the protocol proceeds as follows: $P_1$ 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 $P_1$ and $P_2$ invoke the $\mathcal{F}_{\text{rb-opprf}}^{d}$ functionality with $P_1$ as sender with inputs $\{(X_j, T_j)\}_{j\in[\beta]}$ and $P_2$ as receiver with inputs $\{q_j\}_{j\in[\beta]}$. $P_2$ 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, $P_2$ 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 $v_j$. $P_1$ and $P_2$ invoke $\beta$ many instances of OPPRF functionality, where $P_2$ plays the role of sender with inputs $(W_j, V_j)$ and $P_1$ plays the role of receiver with input $w_j$ in the $j^{\text{th}}$ OPPRF instance. $P_1$ gets output $y_j$ from $j^{\text{th}}$ OPPRF functionality invocation. $P_2$ 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. Using transitivity of implication, 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 $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$ functionality in the $(\mathcal{F}_{\text{rb-opprf}}^{d}, \mathcal{F}_{\text{opprf}})$-hybrid model.
  Specifically, using the solution proposed in [13] to instantiate $\mathcal{F}_{\text{rb-opprf}}^{d}$ and table-based OPPRF [51] to instantiate $\mathcal{F}_{\text{opprf}}$ gives a concrete communication cost of $(8\lambda + 4\sigma)\beta + 1.31N\sigma$ and a round complexity of 4.

# C INSTANTIATION OF THE CONVERTSHARES FUNCTIONALITY

To instantiate $\text{ConvertShares}^{n,t}$, we make use of a functionality $\text{DoubleRandomF}^{n,t}(\ell)$ which generates $(n,t)$-shares and additive shares of uniform field elements $r_1, \cdots, r_\ell$. We give an instantiation of this functionality in Appendix C.1. Now, we describe the protocol which instantiates the $\text{ConvertShares}^{n,t}$ functionality in Figure 9.

**Parameters:** $P_1, \cdots, P_n$ are $n$ parties. All additions and multiplications are considered in $\mathbb{F}$. Let $t < n/2$ denote the corruption threshold.
**Input:** Additive shares, $\langle a\rangle$ of $a \in \mathbb{F}$.
**Protocol:**

(1) *Randomness Generation*:
$P_1, \cdots, P_n$ compute $([r], \langle r\rangle) \leftarrow \text{DoubleRandomF}^{n,t}(1)$.
(2) For each $i \in [n]$: $P_i$ computes $\langle z\rangle_i = \langle a\rangle_i - \langle r\rangle_i$. Each $P_i$, for $i \in \{2, \cdots, n\}$, sends $\langle z\rangle_i$ to $P_1$.
(3) $P_1$ reconstructs $z$, computes and sends $(n,t)$-shares, $[z]_i$ to $P_i$, for each $i \in \{2, \cdots, n\}$.
(4) For $i \in [n]$, $P_i$ computes $[u]_i = [z]_i + [r]_i$.
(5) Output $[u]$.

**Figure 9:** $\text{ConvertShares}^{n,t}(\langle a\rangle)$ **Protocol**

Theorem C.1. *The protocol given in Figure 9 securely realizes* $\text{ConvertShares}^{n,t}$ *in the* $\text{DoubleRandomF}^{n,t}$*–hybrid model against a semi-honest adversary corrupting* $t < n/2$ *parties.*

Proof. **Correctness.** The correctness of the protocol follows from the correctness of $\text{DoubleRandomF}^{n,t}(1)$ and since the local computations on the shares guarantee that $u = z+r = (a-r)+r = a$.
**Security.** Let $C \subset [n]$ be the set of corrupted parties. We show how to simulate the view of $C$ in the ideal world, given the input shares $\{\langle a\rangle_i\}_{i\in C}$ and the output shares $\{[u]_i\}_{i\in C}$. We consider two cases based on $P_1$ being corrupt or not.

- **Case 1** ($\mathbf{P_1 \notin C}$): In this case, the simulator can pick an $r$ at random and generate its $t$ additive and $(n,t)$–shares corresponding to the output of $\text{DoubleRandomF}^{n,t}$. For simulating the $t$ additive shares of $z$, the simulator locally computes $\langle a\rangle_i - \langle r\rangle_i$, for each $i \in C$. For the $t$, $(n,t)$–shares of $z$, it locally computes $[u]_i - [r]_i$, for each $i \in C$.
- **Case 2** ($\mathbf{P_1 \in C}$): The simulation of the $t$ additive and $(n,t)$–shares of $r$ and $z$ are exactly as in Case 1. In addition, since $P_1 \in C$, the corrupted parties also get $z$ in clear, which the simulator picks at random (since $r$ is random, $z$ looks random as well).

□

**Complexity.** The instantiation of $\text{DoubleRandomF}^{n,t}$ from (Appendix C.1) gives us a communication cost upper bound of $6(n-1)\lceil\log|\mathbb{F}|\rceil$ and a round complexity of 3.

## C.1 Instantiation of the DoubleRandom Functionality

From [21], we have an instantiation of a functionality ("DOUBLE-RANDOM($\ell$)") which outputs $(n,t)$-shares and $(n,2t)$-shares of uniform field elements $r_1, \cdots, r_\ell$ for $2t < n$. Slight modification to the techniques from [21] would result in an instantiation of $\text{DoubleRandomF}^{n,t}(\ell)$, which we describe below for any $\ell \le n-t$. To generate $\ell(> n-t)$ pairs of double random sharings, the below protocol can be executed $\lceil\ell/(n-t)\rceil$ times.

**Parameters:** $P_1, \cdots, P_n$ are $n$ parties. $t$ is the corruption threshold. $\ell \leq n - t$. All additions and multiplications are considered in $\mathbb{F}$. Let $\alpha_1, \cdots, \alpha_n$ be distinct non-zero elements in $\mathbb{F}$. Let $M$ be a $\ell \times n$ matrix, where for $i \in [\ell]$ and $j \in [n]$, the $ij^{th}$ element of the matrix $m_{ij} = (\alpha_j)^{i-1}$.

**Protocol:**

(1) For each $i \in [n]$,
- $P_i$ chooses an $s^{(i)} \in \mathbb{F}$ uniformly.
- $P_i$ computes an $(n,t)$-sharing $[s^{(i)}]$ and an additive sharing $\langle s^{(i)} \rangle$ and sends $([s^{(i)}]_j, \langle s^{(i)} \rangle_j)$ to $P_j$ for each $j \in [n] \backslash \{i\}$.
- $P_i$ locally computes

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

(2) Output $([r_1], \cdots, [r_\ell])$ and $(\langle r_1 \rangle, \cdots, \langle r_\ell \rangle)$.

**Figure 10:** DoubleRandomF$^{n,t}(\ell)$ **Protocol**

The security proof of the protocol is verbatim similar to the proof of DOUBLE-RANDOM($\ell$) from [21]. We refer the reader to [21] for security proof ideas of the same. For generating shares of $\ell$ random values, the communication cost of this protocol is $2\lceil \frac{\ell}{n-t} \rceil n(n-1)\lceil \log |\mathbb{F}| \rceil$ and the round complexity is 1. In this paper we invoke DoubleRandomF$^{n,t}$ for the setting $t < n/2$ and (implicitly)[10] for $\ell \gg (n-t)$. Hence, where ever we invoke DoubleRandomF$^{n,t}$ we consider the amortised cost of a single ($\ell = 1$) double sharing as $4(n-1)\lceil \log |\mathbb{F}| \rceil$ for the sake of simplicity.

# D MPSI PROTOCOL WITH STASH

Figure 11 gives a formal description of the sub-procedure to handle stash in our mPSI protocol. In step 2 of our mPSI protocol without stash (see Figure 4), let $S$ denote the stash observed at $P_1$'s end on performing Cuckoo hashing. Let $m_s = O(\log m)$ denote the upper-bound on the stash size for input set of size $m$ [41]. $P_1$ pads the stash $S$ with dummy elements until its size is $m_s$. Let $S = \{y_1, \cdots, y_{m_s}\}$. Party $P_1$ participates in steps 3 and 4 of Figure 4 with cuckoo hash table Table$_1$ excluding the stash $S$. After the execution of these steps, $P_1$ computes the set $Y$ as in step 5 of Figure 4. Next, for each element $y_j \in S$ and for all $i \in \{2, \cdots, n\}$, $P_1$ and $P_i$ invoke the private set membership (PSM) functionality $\mathcal{F}_{\mathsf{PSM}}$ [13, 16, 22, 30, 67] which takes as input element $y_j$ from $P_1$ and set $X_i$ from $P_i$ and outputs shares $\langle g_{ij} \rangle_1^B$ and $\langle g_{ij} \rangle_i^B$ to $P_1$ and $P_i$ respectively s.t. $g_{ij} = 1$ if $y_j \in X_i$, $g_{ij} = 0$ otherwise. We make use of the protocol proposed in [13] which has a computation and communication complexity of $O(m\lambda)$. Party $P_i$ sets $\langle g_{ij} \rangle_i^B = \neg \langle g_{ij} \rangle_i^B$, for all $i \in \{2, \cdots, n\}$. Party $P_1$ and $P_i$ invoke the $\mathcal{F}_{\mathsf{B2A}}^{\mathbb{F}_p}$ on shares of $g_{ij}$ to obtain the corresponding additive shares, $\langle f_{ij} \rangle_1$ and $\langle f_{ij} \rangle_i$. $P_1$ then sets $\langle b_j \rangle_1 = \sum_{i=2}^n \langle f_{ij} \rangle_1$ and for $i \in \{2, \cdots, n\}$, $P_i$ sets $\langle b_j \rangle_i = \langle f_{ij} \rangle_i$. Observe that, $b_j = 0$ iff $y_j$ belongs to the intersection. Finally, the parties perform computation similar to step 4 of Figure 4 to reveal $q_j = s_j \cdot b_j$ to $P_1$, where $s_j \in \mathbb{F}_p$ is uniformly random. Finally, $P_1$ computes the set $Y_s = \bigcup_{j \in [m_s] : q_j = 0} y_j$. $P_1$ permutes the elements in set $Y \cup Y_s$ and announces to all parties. From Theorem D.1, we get that the sub-procedure securely computes $Y_s$ such that $Y_s = S \cap (\cap_{i=2}^n X_i)$. The correctness and security of mPSI protocol with stash follows from that of the mPSI protocol without stash (Figure 4) and the sub-procedure to handle stash in mPSI protocol.

[10] For example, although ConvertShares$^{n,t}$ invokes DoubleRandomF$^{n,t}$ on $\ell = 1$, in all our protocols ConvertShares$^{n,t}$ itself will be invoked large number of times and the DoubleRandomF$^{n,t}$ calls of all these instances can be batched together into a single DoubleRandomF$^{n,t}$ call on a large $l$.

**Correctness and Security Proof.**

Theorem D.1. *The protocol in Figure 11 securely computes the intersection $Y_s^* = S \cap (\cap_{i=2}^n X_i)$, in the $\mathcal{F}$-hybrid model, where $\mathcal{F} = (\mathcal{F}_{\mathsf{PSM}}, \mathcal{F}_{\mathsf{B2A}}^{\mathbb{F}_p}, \mathsf{RandomF}^{n,t}, \mathsf{ConvertShares}^{n,t}, \mathsf{MultF}^{n,t}, \mathsf{Reveal}^{n,t})$, against a semi-honest adversary corrupting $t < n/2$ parties.*

Proof. **Correctness.** Let $Y_s^* = S \cap (\cap_{i=2}^n X_i)$ and $Y_s$ is the output of the protocol. In order to prove correctness, we need to show that $Y_s = Y_s^*$, with all but negligible probability in $\kappa$.

Lemma D.2. $Y_s^* \subseteq Y_s$.

Proof. Let $e = y_j \in Y_s^*$. From the correctness of $\mathcal{F}_{\mathsf{PSM}}$ functionality, we have $\langle g_{ij} \rangle_1^B \oplus \langle g_{ij} \rangle_i^B = 1$, for all $i \in \{2, \cdots, n\}$, in step 2 of the protocol. This implies that $\langle g_{ij} \rangle_1^B \oplus \langle g_{ij} \rangle_i^B = 0$, for all $i \in \{2, \cdots, n\}$, after step 3 of the protocol. From the correctness of $\mathcal{F}_{\mathsf{B2A}}^{\mathbb{F}_p}$, we have that $\langle f_{ij} \rangle_1 + \langle f_{ij} \rangle_i = 0$, for all $i \in \{2, \cdots, n\}$. Finally, by the correctness of the multiparty functionalities from Section 2.5, we have $b_j = 0 = q_j$. Hence, $e \in Y_s$. □

Lemma D.3. *$Y_s \subseteq Y_s^*$, with probability at least $1 - 2^{-\kappa-2}$.*

Proof. Let $e = y_j \in S$ s.t. $e \in Y_s$ and $e \notin Y_s^*$. Since $e \in Y_s$, $q_j = 0$. From the correctness of MultF$^{n,t}$, then we have that either $b_j = 0$ or $s_j = 0$. The probability that $s_j = 0$ is $p^{-1} < 2^{-\sigma}$. Let $S'$ denote the stash at $P_1$'s end before padding it with dummy elements. If $b_j = 0$, consider the following disjoint and exhaustive cases for $e$.

- $e \in S'$: Since, $e \notin Y_s^*$, there exists $i \in \{2, \cdots, n\}$ such that $e \notin X_i$. Thus, $\langle g_{ij} \rangle_1^B \oplus \langle g_{ij} \rangle_i^B = 0$ in step 2 of the protocol or $\langle g_{ij} \rangle_1^B \oplus \langle g_{ij} \rangle_i^B = 1$ after step 3 of the protocol. From the correctness of $\mathcal{F}_{\mathsf{B2A}}^{\mathbb{F}_p}$, we have that $\langle f_{ij} \rangle_1 + \langle f_{ij} \rangle_i = 1$. This contradicts the fact that $b_j = 0$. Hence, such an $i$ does not exist.
- $e \notin S'$: That is, $e$ is a dummy element inserted by $P_1$. Since dummy elements are different from real elements, $\langle g_{ij} \rangle_1^B \oplus \langle g_{ij} \rangle_i^B = 0$, for all $i \in \{2, \cdots, n\}$, in step 2 of the protocol. This implies that $\langle g_{ij} \rangle_1^B \oplus \langle g_{ij} \rangle_i^B = 1$, for all $i \in \{2, \cdots, n\}$, after step 3 of the protocol. From the correctness of $\mathcal{F}_{\mathsf{B2A}}^{\mathbb{F}_p}$, we have that $\langle f_{ij} \rangle_1 + \langle f_{ij} \rangle_i = 1$, for all $i \in \{2, \cdots, n\}$. This contradicts the fact that $b_j = 0$.

Thus, the probability of false positive is $m_s \cdot 2^{-\sigma} < 2^{-\kappa-2}$. □

**Security.** Let $C \subset [n]$ be the set of corrupt parties ($|C| = t < n/2$). We show how to simulate the view of $C$ in the ideal world, given the stash $S$ (if $P_1$ is corrupt) and input sets $X_C = \{X_j : j \in C \setminus \{1\}\}$ and the output $Y_s^* = S \cap (\cap_{i=2}^n X_i)$. We consider two cases based on party $P_1$ being corrupt or not.

**Parameters**: Let $\sigma = \kappa + \lceil \log m \rceil + 3$ and $p > 2^\sigma$ is a prime. Additions and multiplications in the protocol are over $\mathbb{F}_p$. Let $t < n/2$ be the corruption threshold.
**Input**: Party $P_1$ has stash $S = \{y_1, \cdots, y_{m_s}\}$, where $y_j \in \{0,1\}^\sigma$ and for all $i \in \{2, \cdots, n\}$, $P_i$ has input set $X_i = \{x_{i1}, \cdots, x_{im}\}$, where $x_{ij} \in \{0,1\}^\sigma$. Let $t < n/2$ be the corruption threshold.
**Protocol**:

(1) **Pre-processing**: $P_1, \cdots, P_n$ invoke the multiparty functionality $\mathsf{RandomF}^{n,t}(m_s)$ to get $([s_1], \cdots, [s_{m_s}])$.
(2) For each $i \in \{2, \cdots, n\}$, $j \in [m_s]$, $P_1$ and $P_i$ invoke $\mathcal{F}_{\mathsf{PSM}}$ functionality as follows: $P_1$ and $P_i$ send inputs $y_j$ and $X_i$, resp., and receive boolean shares $\langle g_{ij} \rangle_1^B$ and $\langle g_{ij} \rangle_i^B$, resp., as outputs.
(3) $P_i$ computes $\langle g_{ij} \rangle_i^B = \neg \langle g_{ij} \rangle_i^B$, for all $i \in \{2, \cdots, n\}$ and $j \in [m_s]$.
(4) For each $i \in \{2, \cdots, n\}$ and $j \in [m_s]$, $P_1$ and $P_i$ invoke $\mathcal{F}_{\mathsf{B2A}}^{\mathbb{F}_p}$ functionality as follows: $P_1$ and $P_i$ send inputs $\langle g_{ij} \rangle_1^B$ and $\langle g_{ij} \rangle_i^B$, resp., and receive additive shares $\langle f_{ij} \rangle_1$ and $\langle f_{ij} \rangle_i$, resp., as outputs.
(5) For $j \in [m_s]$,
   - $P_1$ computes $\langle b_j \rangle_1 = \sum_{i=2}^n \langle f_{ij} \rangle_1$ and for $i \in \{2, \cdots, n\}$, $P_i$ sets $\langle b_j \rangle_i = \langle f_{ij} \rangle_i$.
   - $P_1, \cdots, P_n$ compute $[b_j] \leftarrow \mathsf{ConvertShares}^{n,t}(\langle b_j \rangle)$.
   - $P_1, \cdots, P_n$ invoke the following multiparty functionalities.
     - $[q_j] \leftarrow \mathsf{MultF}^{n,t}([b_j], [s_j])$.
     - $q_j \leftarrow \mathsf{Reveal}^{n,t}([q_j])$.
(6) $P_1$ computes the intersection as $Y_s = \bigcup_{j \in [m_s]: q_j = 0} y_j$ from elements in stash.

**Figure 11: Handling Stash in mPSI Protocol**

- **Case 1 ($P_1$ is honest)**: To simulate the output of $\mathsf{RandomF}^{n,t}$ in step 1, pick random $s_j$'s, generate their shares and give $t$ shares to the corrupted parties. In step 2, $P_1$ and $P_i$ invoke the $\mathcal{F}_{\mathsf{PSM}}$ functionality, for all $i \in \{2, \cdots, n\}$ and $j \in [m_s]$. Since $P_1 \notin C$, the view of corrupted parties comprises of only one of the two boolean shares, i.e., $\{\langle g_{ij} \rangle_i^B\}_{i \in C, j \in [m_s]}$, which can be generated as corresponding shares of some random bit (by definition of $\mathcal{F}_{\mathsf{PSM}}$). Step 3 is local and can be executed by the simulator. In step 4, the corrupted parties see only one of the two additive shares $\{\langle f_{ij} \rangle_i\}_{i \in C, j \in [m_s]}$, which can be generated as shares of a random bit (by definition of $\mathcal{F}_{\mathsf{B2A}}^{\mathbb{F}_p}$). In step 5, apart from the local computations, which can all be executed by the simulator, the parties call functionalities $\mathsf{ConvertShares}^{n,t}$, $\mathsf{MultF}^{n,t}$ and $\mathsf{Reveal}^{n,t}$). The corrupted parties get $t$ shares of $b_j$ and $q_j$, for each $j \in [m_s]$. The simulator can generate $t$ shares of random values (by the security of $(n,t)$-secret sharing), and finally, send the output $Y_s^*$ to the corrupted parties.
- **Case 2 ($P_1$ is corrupt)**: The simulation of step 1 is exactly same as in Case 1. In step 2, $P_1$ and $P_i$ invoke $\mathcal{F}_{\mathsf{PSM}}$ functionality, for all $i \in \{2, \cdots, n\}$ and $j \in [m_s]$. The corrupted parties see both the boolean shares $\{\langle g_{ij} \rangle_1^B, \langle g_{ij} \rangle_i^B\}_{i \in C \setminus \{1\}, j \in [m_s]}$ and only one of the boolean shares $\{\langle g_{ij} \rangle_1^B\}_{i \in [n] \setminus C, j \in [m_s]}$. For each $i \in C \setminus \{1\}$, simulator sets $g_{ij} = 1$ if $y_j \in X_i$ else it sets $g_{ij} = 0$. The simulator then generates boolean shares of $g_{ij}$ to simulate $\{\langle g_{ij} \rangle_1^B, \langle g_{ij} \rangle_i^B\}_{i \in C \setminus \{1\}, j \in [m_s]}$ (by definition of $\mathcal{F}_{\mathsf{PSM}}$). For $i \in [n] \setminus C$, the simulator generates boolean shares of random bits to simulate $\{\langle g_{ij} \rangle_1^B\}_{i \in [n] \setminus C, j \in [m_s]}$. Step 3 is local and can be executed by the simulator. Corrupted parties see both the arithmetic shares $\{\langle f_{ij} \rangle_1, \langle f_{ij} \rangle_i\}_{i \in C \setminus \{1\}, j \in [m_s]}$ and one of the arithmetic shares $\{\langle f_{ij} \rangle_1\}_{i \in [n] \setminus C, j \in [m_s]}$ in step 4. For each $i \in C \setminus \{1\}$, simulator sets $f_{ij} = 0$ if $y_j \in X_i$ else it sets $f_{ij} = 1$. The simulator then generates arithmetic shares of $f_{ij}$ to simulate $\{\langle f_{ij} \rangle_1, \langle f_{ij} \rangle_i\}_{i \in C \setminus \{1\}, j \in [m_s]}$ (by definition of $\mathcal{F}_{\mathsf{B2A}}^{\mathbb{F}_p}$). To simulate the view $\{\langle f_{ij} \rangle_1\}_{i \in [n] \setminus C, j \in [m_s]}$, simulator generates arithmetic shares of random bits. In step 5, apart from the local computations, which can all be executed by the simulator, the parties call functionalities $\mathsf{ConvertShares}^{n,t}$, $\mathsf{MultF}^{n,t}$ and $\mathsf{Reveal}^{n,t}$). The corrupted parties see at most $t$ shares of $b_j$, for each $j \in [m_s]$, which can be simulated by generating $t$ shares of random values (by security of $(n,t)$-secret sharing). Moreover, for all $j \in [m_s]$, simulator sets $q_j = 0$ if $y_j \in Y_s^*$, else it picks $q_j$ uniformly at random from $\mathbb{F}_p$ (since $s_j$ are uniformly random given at most $t$ shares of the corrupted parties). For all $j \in [m_s]$, it gives $t$ shares of $q_j$ as output of $\mathsf{MultF}^{n,t}$ and $q_j$ as output of $\mathsf{Reveal}^{n,t}$.

□

**Complexity.** $P_1$ invokes the PSM protocol for every element $y_j$ in stash and for every party $P_i$, $i \in \{2, \cdots, n\}$. Thus, the computation and communication complexity of this step is $O(nm \log(m) \lambda)$. Notice that, this cost dominates the overall cost of the sub-procedure as the rest of the steps have a complexity of $O(n \log(m) \lambda)$. Hence, we obtain an mPSI protocol with an overall complexity of $O(nm \log(m) \lambda)$ in cuckoo hashing with stash setting.

# E WEAK COMPARISON PROTOCOLS

## E.1 Correctness and Security of Weak Comparison Protocol I

We give a complete proof of Theorem 5.2 by proving the correctness and security of the weak comparison protocol I in Figure 8.
**Correctness.** The correctness of the evaluation of the polynomial $\psi(x)$ directly follows from its definition and from the correctness of the multiparty functionalities $\mathsf{RandomF}^{n,t}$ and $\mathsf{MultF}^{n,t}$ from [21]. We need to show that, for each $a$, except with negligible probability in correctness parameter $(\tau)$, $v_j = 0, \forall j \in [J] \iff \psi(a) = 0$.

Lemma E.1. $\psi(a) = 0 \implies \big(v_j = 0, \forall j \in [J]\big)$.

Proof. This follows directly from the definition of $v_j$. □

Lemma E.2. *Probability that $\psi(a) = 0$ when $(v_j = 0, \forall j \in [J])$ is atleast $1 - 2^{-\tau}$.*

Proof. For any $j \in [J]$, if $v_j = 0$ then either $s_j = 0$ or $\psi(a) = 0$. If $\psi(a) = 0$ then we are done. If $\psi(a) \neq 0$ and $v_j = 0$ then $s_j = 0$, which occurs with probability $2^{-\log|\mathbb{F}_p|}$. Probability that $s_j = 0$ for every $j \in [J]$ is $2^{-\log|\mathbb{F}_p| \cdot J} \leq 2^{-\tau}$ by the definition of $J$. Therefore if $\psi(a) \neq 0$ then at least one $v_j \neq 0$ with probability at least $1 - 2^{-\tau}$. □

Hence except with failure probability at most $2^{-\tau}$ the output of the protocol is correct.

**Security.** 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 $P_1$ is corrupt, and no output, otherwise. We consider two cases based on party $P_1$ being corrupt or not.

- **Case 1 ($P_1$ is honest)**: For the pre-processing step, the simulator can pick random strings $s_1, \cdots, s_J$ and send $t$ shares of each $s_1, \cdots, s_J$ 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 $v_j$ at random and gives its $t$ shares to corrupted parties, for each $j \in [J]$. Simulation of this step is correct by security of $(n, t)$-secret sharing scheme.
- **Case 2 ($P_1$ is corrupt)**: The simulation of step 1 and step 2 until generating $t$ shares of each $v_j$ is done exactly as in the previous case. The opened value $v_j = \psi(a) \cdot s_j$ is simulated as follows: Since $s_j$ is uniformly random (as only $t$ shares are known to the corrupted parties), $v_j = \psi(a) \cdot s_j$ looks random whenever $v_j \neq 0$. Hence, if $k \geq n/2$, the simulator sets all $v_j = 0$ if comp = 1, and picks all $v_j$ at random otherwise, and vice versa, if $k < n/2$. The simulator sends all $v_j$ to the corrupted parties.

## E.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 $\mathcal{F}_{\text{Mod}}^{p,n,t}$ functionality, which we define in Figure 12, as building blocks. The $\mathcal{F}_{\text{Mod}}^{p,n,t}$ 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 $P_i$.

**Figure 12: Mod2 Functionality $\mathcal{F}_{\text{Mod}}^{p,n,t}$**

We instantiate the $\mathcal{F}_{\text{Mod}}^{p,n,t}$ functionality using the protocol from [12], which sets $p > 2^{\kappa+\gamma}$ to be a prime such that $p \mod 4 = 3$ and $\gamma = \lceil \log n \rceil + 1$. The details of this protocol are given in Appendix E.2.1.

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 $\mathcal{F}_{\text{Mod}}^{p,n,t}$ 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 $\text{Reveal}^{n,t}$ 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 13.

**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 multiplications in the protocol are over $\mathbb{F}_p$, where $p$ depends on the specific instantiation of $\mathcal{F}_{\text{Mod}}^{p,n,t}$.
**Input**: For each $i \in [n]$, $P_i$ inputs its $(n, t)-$ share $[a]_i$.
**Protocol**:
(1) $P_1, \cdots, P_n$ locally compute $[b] = [a] - k$.
(2) Let $c_1 = b$. For each $i = 1, \cdots, \gamma$, $P_1, \cdots, P_n$ do the following:
  - Invoke the $\mathcal{F}_{\text{Mod}}^{p,n,t}$ 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) $c_{\gamma+1} \leftarrow \text{Reveal}^{n,t}([c_{\gamma+1}])$.
**Output:** $P_1$ sets comp = 1, if $c_{\gamma+1} = 0$ and comp = 0, otherwise. Other parties get no output.

**Figure 13: WEAK COMPARISON PROTOCOL II**

Theorem E.3. *The protocol given in Figure 13 securely realizes $\mathcal{F}_{\text{w-CMP}}^{p,k,n,t}$ in the $\mathcal{F}-$hybrid model, where $\mathcal{F} = (\mathcal{F}_{\text{Mod}}^{p,n,t}, \text{Reveal}^{n,t})$, against a semi-honest adversary corrupting $t < n/2$ parties.*

Proof. **Correctness.** The correctness of the protocol follows from the correctness of the functionalities $\mathcal{F}_{\text{Mod}}^{p,n,t}$ and $\text{Reveal}^{n,t}$ and the fact that $\left\lfloor \frac{(a-k)}{2^\gamma} \right\rfloor = 0$ iff $a \geq k$.

**Security.** 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 $P_1$ is corrupt, and no output, otherwise. We consider two cases based on party $P_1$ being corrupt or not.

- **Case 1 ($P_1$ is honest)**: 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 ($P_1$ is corrupt)**: 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 comp = 1 and $c_{\gamma+1} = p - 1$, otherwise. Finally, set the output as comp.

□

*E.2.1 The Mod2 Protocol.* We now describe the Mod2 protocol from [12] (using the instantiations from [21]), which we use to instantiate the $\mathcal{F}_{\text{Mod}}^{p,n,t}$ functionality, used in our weak comparison protocol II (Figure 13).

**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 14.

**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 RandBit() sub-protocol (Figure 15) 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) $c \leftarrow \text{Reveal}^{n,t}([c])$.
(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 $[d]_i$.

**Figure 14:** 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 14.

Theorem E.4. *The protocol given in Figure 14 securely realizes $\mathcal{F}_{\text{Mod}}^{p,n,t}$ in the $\mathcal{F}-$ hybrid model, where $\mathcal{F} = (\text{RandomF}^{n,t}, \text{MultF}^{n,t}, \text{Reveal}^{n,t})$, 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 $\text{RandomF}^{n,t}$ and $\text{MultF}^{n,t}$ from [21], 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

**Input**: No input taken.
**Protocol**:

(1) $[r] \leftarrow \text{RandomF}^{n,t}(1)$.
(2) Compute $[u] \leftarrow \text{MultF}^{n,t}([r],[r])$.
(3) $u \leftarrow \text{Reveal}^{n,t}([u])$. 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 $[b]_i$.

**Figure 15:** RandBit **SUB-PROTOCOL**

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 that $s'$ is a single bit).

**Security.** 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 ($P_1$ is honest)**: 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 ($P_1$ is corrupt)**: 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.

# F CORRECTNESS, SECURITY AND COMPLEXITY OF QUORUM PSI

## F.1 Correctness and Security of Quorum PSI

We recall the Quorum PSI protocol from Section 5.1 in Figure 16. We now give a complete proof of Theorem 5.1, by proving the correctness and security of the protocol in Figure 16.

**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 $\mathcal{F}_{\text{w-CMP}}$. Let $t < n/2$ denote the corruption threshold.
**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 $\sigma$ bits by first hashing the elements using an appropriate universal hash function.
**Protocol**:

(1) **Hashing**: Parties agree on hash functions $h_1, h_2, h_3 : \{0,1\}^\sigma \to [\beta]$.
$P_1$ does stash-less cuckoo hashing on $X_1$ using $h_1, h_2, h_3$ to generate $\text{Table}_1$ 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 $\text{Table}_i$, 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.

(2) **Invoking the $\mathcal{F}^{\beta,\sigma,N}_{\text{wPSM}}$ functionality:** For each $i \in \{2, \cdots, n\}$ , $P_1$ and $P_i$ invoke the $\mathcal{F}^{\beta,\sigma,N}_{\text{wPSM}}$ functionality for $N = 3m$ as follows:
- $P_i$ is the sender with inputs $\{\text{Table}_i[j]\}_{j\in[\beta]}$ and $P_1$ is the receiver with inputs $\{\text{Table}_1[j]\}_{j\in[\beta]}$.
- $P_i$ receives the outputs $\{w_{ij}\}_{j\in[\beta]}$ and $P_1$ receives $\{y_{ij}\}_{j\in[\beta]}$.

(3) **Invoking the $\mathcal{F}^{\sigma}_{\text{EQ}}$ functionality:** For each $i \in \{2, \cdots, n\}$ and for each $j \in [\beta]$, $P_1$ and $P_i$ invoke the $\mathcal{F}^{\sigma}_{\text{EQ}}$ 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.

(4) **Invoking the $\mathcal{F}^{\mathbb{F}_p}_{\text{B2A}}$ functionality:** For each $i \in \{2, \cdots, n\}$ and for each $j \in [\beta]$, $P_1$ and $P_i$ invoke the $\mathcal{F}^{\mathbb{F}_p}_{\text{B2A}}$ 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.

(5) **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 $\langle a_j\rangle_i = \langle f_{ij}\rangle_i$.
- $P_1, \cdots, P_n$ compute $[a_j] \leftarrow \text{ConvertShares}^{n,t}(\langle a_j\rangle)$.
- Parties invoke $\mathcal{F}^{p,k,n,t}_{\text{w-CMP}}$ with $P_i$'s input being $[a_j]_i$ for $i \in [n]$ and $P_1$ learns $c_j$ as output.

(6) **Output:** $P_1$ computes the quorum intersection as $Y = \bigcup_{j\in[\beta]:c_j=1} \text{Table}_1[j]$.

**Figure 16: QUORUM PSI PROTOCOL**

**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 $\kappa$ 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 $\text{Table}_1$), which happens with probability at least $1 - 2^{-42}$ (see Section 2.2). Now, the following two lemmata complete the proof of correctness.

Lemma F.1. $Y^* \subseteq Y$.

Proof. Let $e = \text{Table}_1[j] \in Y^*$ and $\mathcal{E} = \{i \in \{2, \cdots, n\} : e \in X_i\}$. By the property of simple hashing, $e \in \text{Table}_i[j]$ for all $i \in \mathcal{E}$. By correctness of $\mathcal{F}^{\beta,\sigma,N}_{\text{wPSM}}$, $\mathcal{F}^{\sigma}_{\text{EQ}}$ and $\mathcal{F}^{\mathbb{F}_p}_{\text{B2A}}$, 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 $\mathcal{F}^{\sigma}_{\text{EQ}}$ gives a boolean output, $eq_{ij} \in \{0,1\}$, and by correctness of $\mathcal{F}^{\mathbb{F}_p}_{\text{B2A}}$, we have $f_{ij} \in \{0,1\}$. By correctness of $\text{ConvertShares}^{n,t}$, we know that $[a_j]$ corresponds to $a_j = \sum_{i\in\{2,\cdots,n\}} f_{ij} < n < p$. Since $e \in Y^*$, we get $a_j \geq |\mathcal{E}| \geq k$. Finally, by correctness of $\mathcal{F}^{p,k,n,t}_{\text{w-CMP}}$ we will get $c_j = 1$ when invoked on shares of $a_j \geq k$. Therefore, $e \in Y$. □

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

Proof. Suppose $Y \not\subseteq Y^*$. Let $e = \text{Table}_1[j] \in Y \backslash Y^*$. First, $e \in Y$ implies $c_j = 1$. Further, by correctness of $\mathcal{F}^{p,k,n,t}_{\text{w-CMP}}$ and $\text{ConvertShares}^{n,t}$, it follows that $a_j \geq k$ and $a_j = \sum_{i\in\{2,\cdots,n\}} f_{ij}$. Now, for every $i \in \{2, \cdots, n\}$, by correctness of $\mathcal{F}^{\mathbb{F}_p}_{\text{B2A}}$ $f_{ij} = eq_{ij}$ and by correctness of $\mathcal{F}^{\sigma}_{\text{EQ}}$, $eq_{ij}$ 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 ($Y \not\subseteq Y^*$) implies that $\mathcal{E}' \backslash \mathcal{E}$ is non-empty 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 $\text{Table}_1$, 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 $e \notin \text{Table}_i[j]$.
- 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 $e \notin \text{Table}_i[j]$.

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

Hence with probability at least $1 - 2^{-42} - 2^{-\kappa-1} > 1 - 2^{-\kappa}$ (for $\kappa = 40$) the protocol's output will be correct.

**Remark.** To ensure the correctness of Quorum-I (instantiated using w-CMP1) with probability at least $1 - 2^{-\kappa}$, we set the parameter $\tau$ of w-CMP1 to be $\kappa + \lceil \log m \rceil + 3$.

**Security.** 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 $P_1$ is corrupt, and no output, otherwise. We consider two cases based on party $P_1$ being corrupt or not.

- **Case 1 ($P_1$ is honest)**: The hashing step is local, and can be executed by the simulator using the inputs of the corrupted parties. In Step 2, $P_1$ and $P_i$ (for each $i \in \{2, \cdots, n\}$) invoke the $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$ functionality and the corrupted parties only see the sender's views (since $P_1$ is honest), $\{w_{ij}\}_{i \in C, j \in [\beta]}$, which can all be picked at random by the simulator (by the definition of $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$). In Steps 3 and 4, for each $i \in \{2, \cdots, n\}$, parties $P_1$ and $P_i$ invoke the $\mathcal{F}_{\text{EQ}}^{\sigma}$ and $\mathcal{F}_{\text{B2A}}^{\mathbb{F}_p}$ functionalities and the corrupted parties see only one of the two boolean and additive shares, $\{\langle eq_{ij}\rangle_i^B\}_{i \in C, j \in [\beta]}$ 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 5, besides the local computations, the parties invoke the functionalities $\text{ConvertShares}^{n,t}$ and $\mathcal{F}_{\text{w-CMP}}^{p,k,n,t}$. The view of the corrupted parties in this step includes: at most $t$ shares of the $a_j$, for each $j \in [\beta]$. Here, the simulator can pick shares of some random values as the $t$ shares of the $a_j$'s (by the security of secret sharing). Note that, the corrupted parties get no output from the $\mathcal{F}_{\text{w-CMP}}$ functionality (since $P_1$ is honest), and also no output from the protocol.
- **Case 2 ($P_1$ is corrupt)**: The simulation of the hashing step is exactly the same as in Case 1. In Step 2, $P_1$ and $P_i$ (for each $i \in \{2, \cdots, n\}$) invoke the $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$ 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 $\{w_{ij}\}_{i \in C \setminus \{1\}, j \in [\beta]}$. For each $i \in C \setminus \{1\}$, the simulator picks a random $y_{ij} = w_{ij}$, if $\text{Table}_1[j] \in \text{Table}_i[j]$, 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 $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$ and since the simulator has both $\text{Table}_1$ and $\text{Table}_i$). In Steps 3 and 4, for each $i \in \{2, \cdots, n\}$, parties $P_1$ and $P_i$ invoke the $\mathcal{F}_{\text{EQ}}^{\sigma}$ and $\mathcal{F}_{\text{B2A}}^{\mathbb{F}_p}$ functionalities and the corrupted parties see both the boolean and additive shares for $i \in C$, $\{\langle eq_{ij}\rangle_1^B, \langle eq_{ij}\rangle_i^B\}_{i \in C \setminus \{1\}, j \in [\beta]}$ and $\{\langle f_{ij}\rangle_1, \langle f_{ij}\rangle_i\}_{i \in C \setminus \{1\}, j \in [\beta]}$, and only one of the two shares for $i \in [n] \setminus C$, $\{\langle eq_{ij}\rangle_1^B\}_{i \in [n] \setminus C, j \in [\beta]}$ and $\{\langle f_{ij}\rangle_1\}_{i \in [n] \setminus C, j \in [\beta]}$. For each $i \in C \setminus \{1\}$, the simulator sets $eq_{ij} = f_{ij} = 1$, if $\text{Table}_1[j] \in \text{Table}_i[j]$ and sets $eq_{ij} = f_{ij} = 0$, otherwise, for each $j \in [\beta]$. It then generates the boolean and arithmetic shares of the $eq_{ij}$'s and $f_{ij}$'s, respectively. For each $i \in [n] \setminus C$, the simulator generates both the boolean and additive shares as shares of some random bit (by the security of secret sharing). To simulate Steps 5 and 6, the simulator does the following: for all $j \in [\beta]$, give $t$ shares of the random values as shares of the $a_j$'s (by the security of secret sharing). Finally, for each $j \in [\beta]$, set $c_j = 1$ if $\text{Table}_1[j] \in Y^*$ and set $c_j = 0$, otherwise, and set the final output as $Y^*$.

## F.2 Quorum PSI Complexity

We instantiate the $\mathcal{F}_{\text{EQ}}^{\sigma}$, $\mathcal{F}_{\text{B2A}}^{\mathbb{F}_p}$, $\text{ConvertShares}^{n,t}$ functionalities as specified in sections 2.3.1, 2.3.2 and 2.5. We instantiate the $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$ functionality using the polynomial-based batch OPPRF. Let Quorum-I and Quorum-II denote instantiations of $\mathcal{F}_{\text{QPSI}}^{n,m,k}$ when $\mathcal{F}_{\text{w-CMP}}^{p,k,n,t}$ is instantiated with w-CMP1 (with $\tau = \kappa + \lceil \log m \rceil + 3$) and w-CMP2 respectively. We first discuss the complexity trade-off between w-CMP1 and w-CMP2 and then discuss the complexities of Quorum-I and Quorum-II

*F.2.1 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 at most $14k'(n-1)(\lceil \log n \rceil + 1) + 17\tau(n-1)$ and the round complexity is $6 + 2k'$, when we set $\lceil \log p \rceil = \lceil \log n \rceil + 1$ and $k' = \min\{k - 1, n - k\}$. While for w-CMP2, the (expected[11]) 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 $\kappa = 40$ and $\tau = \kappa + 23$. For any $\lceil \log n \rceil + 2 < k'$, the round complexity of w-CMP2 is better than that of w-CMP1.

*F.2.2 Complexities of Quorum-I and Quorum-II.* Our protocol, in total, calls the $\mathcal{F}_{\text{wPSM}}^{\beta,\sigma,N}$, $\mathcal{F}_{\text{EQ}}^{\sigma}$, $\mathcal{F}_{\text{B2A}}^{\mathbb{F}_p}$, $\text{ConvertShares}^{n,t}$ and $\mathcal{F}_{\text{w-CMP}}^{p,k,n,t}$ functionalities $(n-1)$, $\beta(n-1)$, $\beta(n-1)$, $\beta$ and $\beta$ times respectively, where $\beta = 1.28m$. Let $k' = \min\{k - 1, n - k\}$. 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 1-4 cost less than $m(n-1)(\lambda\sigma + 5.8\lambda + 14\sigma + 1.28\lceil \log p \rceil)$.
- Excluding the cost of $\mathcal{F}_{\text{w-CMP}}^{p,k,n,t}$, Step 5 contributes at most $8m(n-1)\lceil \log p \rceil$.

The total cost of w-CMP1 executions by Quorum-1 is at most $m(n-1)(18k'(\lceil \log n \rceil + 1) + 22\tau)$, where $\tau = \kappa + \lceil \log m \rceil + 3$ . Therefore, the concrete communication of Quorum-I is at most $m(n-1)(\lambda\sigma + 5.8\lambda + 14\sigma + 18k'(\lceil \log n \rceil + 1) + 22\tau + 10\lceil \log n \rceil)$, when we set $\lceil \log p \rceil = \lceil \log n \rceil + 1$. The round complexity of Quorum-I is at most $10 + \lceil \log \sigma \rceil + 2k'$.

The (expected) total cost of w-CMP2 executions by Quorum-II is at most $26m(n-1)(\lceil \log n \rceil + 1)(\kappa + \lceil \log n \rceil + 1)^2$. Therefore, (expected) concrete communication of Quorum-II is at most $m(n-1)(\lambda\sigma + 5.8\lambda + 14\sigma + 27(\lceil \log n \rceil + 1)(\kappa + \lceil \log n \rceil + 1)^2)$, when we set $p = \kappa + \lceil \log n \rceil + 2$. The (expected) round complexity of Quorum-II is at most $8 + \lceil \log \sigma \rceil + 2\lceil \log n \rceil$.

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