Improved Private Set Intersection for Sets with Small Entries

Dung Bui, Geoffroy Couteau

Improved Private Set Intersection for Sets with Small Entries

Dung Bui[1] and Geoffroy Couteau[2]

[1] IRIF, Université de Paris
bui@irif.fr
[2] CNRS, IRIF, Université de Paris
couteau@irif.fr

Abstract. We introduce new protocols for private set intersection (PSI), building upon recent constructions of pseudorandom correlation generators, such as vector-OLE and ring-OLE. Our new constructions improve over the state of the art on several aspects, and perform especially well in the setting where the parties have databases with small entries. We obtain three main contributions:

1. We introduce a new semi-honest PSI protocol that combines subfield vector-OLE with hash-based PSI. Our protocol is the first PSI protocol to achieve communication complexity *independent* of the computational security parameter κ, and has communication lower than all previous known protocols for input sizes ℓ below 70 bits.

2. We enhance the security of our protocol to the malicious setting, using two different approaches. In particular, we show that applying the *dual execution technique* yields a malicious PSI whose communication remains independent of κ, and improves over all known PSI protocols for small values of ℓ.

3. As most previous protocols, our above protocols are in the random oracle model. We introduce a third protocol which relies on subfield ring-OLE to achieve maliciously secure PSI in the *standard model*, under the ring-LPN assumption. Our protocol enjoys extremely low communication, reasonable computation, and standard model security. Furthermore, it is *batchable*: the message of a client can be reused to compute the intersection of their set with that of multiple servers, yielding further reduction in the overall amortized communication.[3]

1 Introduction

Private Set Intersection (PSI) is a cryptographic primitive that allows parties to jointly compute the set of all common elements between their datasets, without leaking any value outside of the intersection. It is a special case of secure multi-party computation (MPC). PSI enjoys a wide array of real-life applications; it is perhaps the most actively researched concrete functionality in secure computation, and has been the target of a tremendous number of works, see [PSZ14, PSSZ15, KKRT16, RR17, KRTW19, PSWW18, PRTY19, PRTY20, CM20, RS21, GPR+21, RT21a] and references therein for a sample. As a consequence of this intense research effort, modern PSI protocols now achieve impressive efficiency features, communicating only a few hundred bits per database items, and processing millions of items in seconds.

Improving PSI with pseudorandom correlation generators. Pseudorandom correlation generators (PCG) have been introduced in the works of [BCG+17, BCGI18, BCG+19b] and have been the subject of a long and fruitful line of work [BCG+17, BCGI18, BCG+19b, BCG+19a, SGRR19, BCG+20b, BCG+20a, YWL+20, CRR21, WYKW21]. At a high level, a PCG allows two parties to securely stretch long pseudorandom correlated strings from short, correlated seeds. Securely sharing correlated random strings is a crucial component in most modern secure computation protocols, which operate in the preprocessing model; PCG allows to realize this functionality with almost no communication. Among their many applications, PCGs allow to construct *silent oblivious transfer extension* protocols [BCG+19a], which can realize (pseudorandom) OT extension with minimal (logarithmic) communication.

Since the top-performing PSI protocols rely on efficient OT extension, using PCG-based techniques to improve their efficiency is a natural idea. And indeed, this was done recently for OKVS-based PSI

[3] This paper appeared at PKC 2023, this is the full version.

in [RS21], leading to the most efficient PSI protocol known to date (OKVS stands for oblivious key-value store [GPR+21]; the use of OKVS is the leading paradigm for the design of PSI protocols). To give a single datapoint, computing the intersection between two databases of size $n = 2^{20}$ with the protocol of [RS21] communicates as little as $426n$ bits in total. In addition, some of the tools used in [RS21] have been significantly improved since: replacing their OKVS (which is the PaXoS OKVS of [PRTY20]) by the more recent 3H-GCT OKVS of [GPR+21], and replacing their PCG (which is the one from [WYKW21]) by the recent PCG of [CRR21], the cost goes down to an impressive $247n$ bits of total communication. In comparison, even the *insecure* approach of exchanging the hashes of all items in the databases already requires $160n$ bits of communication. OKVS-based PSI protocols are now firmly established as the leading paradigm in the field, and the use of PCGs to reduce their communication overhead even more seems to further widen the gap with the other paradigms.

1.1 Our Contributions

We thoroughly investigate how the use pseudorandom correlation generators can reduce communication in PSI protocols. We obtain several contributions:

- A new family of semi-honest hash-based PSI protocols. Our protocols can be instantiated using several hashing techniques, and achieve very low communication, especially for databases whose entries have a small bitlength.
- New maliciously secure hash-based PSI protocols. Here, interestingly, we revive the dual execution technique, which had been used previously to design malicious PSI protocols in [RR17], but was considered outdated. We show that, combined with our new approach, it leads to very competitive protocols, which achieve lower communication than all known alternatives for databases with small entries.
- Eventually, we design a new maliciously secure polynomial-based PSI protocol. Our protocol enjoys several powerful features: competitive communication, security in the standard model under the ring-LPN assumption (in contrast, other maliciously secure PSI use the ROM), and the possibility for a client to publish a single encoding of its database, and later retrieve the intersection of its database with that of multiple servers independently, with a single server-to-client message, plus minimal (database-independent) additional communication.

Below, we elaborate on each of our contributions.

Low communication PSI for databases with small entries. Modern PSI protocols have communication $O(\kappa \cdot n)$, where n is the database size, and κ is a computational security parameter. More precisely, the receiver-to-sender communication is $O(\kappa \dot{n})$, while the sender-to-receiver communication is $O(\lambda \cdot n)$, where λ is a *statistical* security parameter (typically, $\kappa = 128$ and $\lambda = 40$). We introduce a new protocol, that combines hashing techniques (e.g. Cuckoo hashing or its variants, as initially used in [KKRT16]) with a new PCG-based oblivious pseudorandom function (OPRF). In contrast to all previous works, our work avoid the $O(\kappa \cdot n)$ overhead: it reduces the receiver-to-sender communication to be roughly $\ell \cdot n$ (where ℓ is the bitsize of the database items), leading to a significant reduction in the overall communication. To our knowledge, our protocol is the first to achieve communication independent of κ (up to low order terms). To give a datapoint, for $n = 2^{20}$, with 64-bit entries, our protocol communicates $210n$ bits, and with 32-bit entries, it communicates only $148n$ bits. For the same parameters, the leading OKVS-based PSI of [RS21] communicates $197n$ bits, even after improving it with all relevant optimization (such as using the 3H-GCT OKVS of [GPR+21], and the recent PCG of [CRR21]). We provide further datapoints and comparisons to the state of the art on Table 1, when instantiating our protocols with various hashing methods.

Fast maliciously-secure PSI for small entries. We then turn our attention to maliciously secure PSI. We provide two alternative protocols which achieve malicious security; both use standard paradigms for upgrading PSI to malicious security. The first protocol combines our new PCG-based OPRF with simple hashing, and applies the standard paradigm used in most previous OKVS-based PSI to achieve malicious security (e.g. [RS21]). This requires to increase the sender-to-receiver message length, from $O(\lambda \cdot n)$ to $O(\kappa \cdot n)$ (λ is a statistical security parameter, κ is a computational security parameter; typically, $\lambda = 40$ and $\kappa = 128$) to allow for extraction of the sender input. Along the way, we also notice a small mistake in the parameter choices of [RS21]: they devise a new ROM-based extraction strategy in the malicious setting, and prove that a Q-query adversary will make

Table 1. Comparison of the communication cost of several PSI protocols in the semi-honest setting and in the malicious setting, for various choices of the database size n (we assume that both parties have a database of the same size). ℓ denote the bit-length of the inputs in the database; we set the computational security parameter κ to 128 and the statistical security parameter λ to 40 (for usual applications) or 30 (which can be suitable for lower risk applications). For all protocols, we take into account the optimization of [TLP+17] which reduces the costs of sending n elements of bitlength $\lambda + 2 \cdot \log n$ to $n \cdot (\lambda + \log n)$. GCH stands for Generalized Cuckoo hashing (here, with 2 hash functions and 3 items per bin), 2CH for 2-choice hashing, and SH for simple hashing (N is the number of bins).

	$n = 2^{14}$	$n = 2^{16}$	$n = 2^{20}$	$n = 2^{24}$
Semi-honest setting				
KKRT16 [KKRT16]	$930n$	$936n$	$948n$	$960n$
PRTY19 [PRTY19] low*	$491n$	$493n$	$493n$	$494n$
PRTY19 [PRTY19] fast*	$560n$	$571n$	$579n$	$587n$
CM20 [CM20]	$668n$	$662n$	$674n$	$676n$
PRTY20 [PRTY20]	$1244n$	$1192n$	$1248n$	$1278n$
RS21 [RS21]	$2024n$	$898n$	$406n$	$374n$
RS21 [RS21] enhanced**	$280n$	$260n$	$263n$	$275n$
Ours ($\ell = 64$, GCH)	$246n$	$220n$	$210n$	$209n$
Ours ($\ell = 48$, GCH)	$215n$	$189n$	$179n$	$178n$
Ours ($\ell = 32$, GCH)	$184n$	$158n$	$148n$	$147n$
Ours ($\ell = 64$, 2CH)	$214n$	$190n$	$183n$	$185n$
Ours ($\ell = 48$, 2CH)	$193n$	$169n$	$162n$	$164n$
Ours ($\ell = 32$, 2CH)	$171n$	$148n$	$141n$	$142n$
Ours ($\ell = 64$, SH, $N = n/10$)	$332n$	$302n$	$284n$	$276n$
Ours ($\ell = 48$, SH, $N = n/10$)	$261n$	$230n$	$209n$	$198n$
Ours ($\ell = 32$, SH, $N = n/10$)	$191n$	$158n$	$133n$	$120n$
Ours ($\ell = 64$, SH, $N = 1$) ***	$154n$	$131n$	$125n$	$128n$
Ours ($\ell = 48$, SH, $N = 1$) ***	$138n$	$115n$	$109n$	$112n$
Ours ($\ell = 32$, SH, $N = 1$) ***	$122n$	$99n$	$93n$	$96n$
Malicious setting				
RS21 [RS21] enhanced**	$343n$	$320n$	$315n$	$318n$
Ours ($\ell = 48$, SH, $N = n/10$)	$430n$	$393n$	$356n$	$332n$
Ours ($\ell = 40$, SH, $N = n/10$)	$359n$	$321n$	$281n$	$253n$
Ours ($\ell = 32$, SH, $N = n/10$)	$289n$	$249n$	$205n$	$175n$

* PRTY19 has two variants, SpOT-low (lowest communication, higher computation) and SpOT-fast (higher communication, better computation). Both use expensive polynomial interpolation and require significantly more computation compared to all other protocols in this table.

** Using the 3H-GCT OKVS of [GPR+21] instead of PaXoS, and the VOLE of [CRR21] instead of the one from [WYKW21]. Setting κ_{RS21} to $\kappa + \log n$ to achieve κ bits of security.

*** Using $N = 1$ requires an expensive degree-n polynomial interpolation.

extraction fail with probability bounded $Q \cdot n/2^\kappa$ (this is the probability that one of the Q queries of the malicious receiver collides with an element of the sender set). This implies that, to target 128 bits of computational security, one must set $\kappa = 128 + \log n$. However, the numbers reported in [RS21] correspond to choosing $\kappa = 128$ at the 128-bit security level. We took this minor inconsistency into account in our tables.

More interestingly, our second protocol applies *dual execution* [RR17] to our PCG-based protocol with simple hashing. We observe that, in our context, this allows to achieve malicious security without having to increase the length of the sender-to-receiver message, at the cost of increasing the receiver-to-sender communication by a factor 2. Since our approach makes this communication as low as $O(\ell \cdot n)$, this turns out to be an excellent tradeoff whenever the database entries are not too large. Therefore, our results show that the landscape of maliciously secure PSI is more subtle than previously thought: for large entries, the standard approach still dominates, but for smaller entries (e.g. $\ell \leq 40$), the dual execution technique leads to better performances. This revives the dual execution technique, which was previously considered obsolete compared to the modern alternatives.

Efficient PSI in the standard model. Eventually, our last contribution is a new "polynomial-based" PSI protocol that does not rely on the random oracle model, following the high level structure of previous works [KS05, GS19, GN19]. To this end, we introduce the notion of PCG for the *subfield ring-OLE* correlation, and show how a simple variant of the recent PCG for ring-OLE of [BCG+20b] leads to efficient instantiations of this primitive. Then, we describe a new PSI protocol built on top of this PCG, which enjoys a number of very interesting features.

Security features. Our PSI protocol is in the standard model: unlike our first protocol, it does not require the random oracle model, or any tailor-made correlation-robustness assumptions. We rely solely on the (relatively well-established) ring-LPN assumption over polynomial rings with irreducible polynomials. To our knowledge, our protocol is the first standard model protocol which offers competitive performances compared to protocols using the random oracle heuristic or tailored assumptions. Furthermore, our PSI protocol enjoys full malicious security (for both parties) *almost for free*. This stems from the use of PCGs, which allows to confine the "price" of achieving malicious security to the distributed seed generation only, which has logarithmic communication and computation (in the set size n).

We note that, though malicious security comes for free communication- and computation-wise, the tweaks used to guarantee malicious security in our protocol are not straightforward. In fact, achieving malicious security efficiently in polynomial-based PSI protocols is known to be complex and error prone. For example, previous works [GN19] used a superficially similar approach and claimed malicious security, but their protocol was found to be insecure in a recent preprint, which described powerful concrete attacks on this proposal [AMZ21]. Leveraging the specific structure of our protocol, we manage to get around these nontrivial subtleties with careful structural checks, for a minimal cost (independent of the database size).

Efficiency features. Our PSI protocol enjoys a very low communication, considerably lower than all previous PSI protocols in the standard model which we are aware of (excluding iO- or FHE-based protocol, which can have very low communication but poor concrete efficiency). In fact, communication-wise, our PSI protocol is even on par with the best *ROM-based* PSI protocols of previous works. Concretely, for sets of size n with ℓ-bit entries, our protocol communicates $(2\ell + 3\lambda + 3\log n) \cdot n + o(n)$ bits. To give a single datapoint, for $\ell = 32$ and $n = 2^{20}$, we estimate the total communication to be $278n$ bits. This is on par with the best maliciously secure protocol [RS21], which communicates $279n$ bits in the same setting, with comparable computation (it also uses polynomial interpolation), but without standard model security.
On Table 1.1, we compare our protocol to the current fastest maliciously secure PSI protocols [PRTY20, RT21b, RS21]. As the table shows, the communication of our protocol is almost on par with that of the best protocol (the protocol of [RS21], enhanced with the latest VOLE protocol) for small-ish input size, and large enough set sizes. Yet, our protocol is in the standard model under the ring-LPN assumption, while [RS21] is only proven secure in the ROM.

Batch non-interactive PSI. On top of these security and efficiency features, the structure of our protocol allows to obtain a powerful interaction pattern: it leads to a batch non-interactive PSI, where after a short interaction with each server, a client C with set X can broadcast a *single* encoding of its database, and receive afterwards at anytime a single message from each server S_i with set X_i (plus, in

Table 2. Comparison of the communication cost of several PSI protocols in the malicious model, for various choices of the database size n (we assume that both parties have a database of the same size) and statistical security parameter $\lambda = 40$, using the encoding technique of [TLP+17]. ℓ denote the bit-length of the inputs in the database; we set the computational security parameter κ to 128. For fairness of comparison, since our standard model PSI uses interpolation, we compare it to RS21 with an interpolation-based OKVS (which has better communication), and we compare our other PSIs with RS21 instantiated with (computationally) efficient OKVS.

Protocol	Communication					Hardness Assumption	Standard Model
	$n = 2^{16}$	$n = 2^{18}$	$n = 2^{20}$	$n = 2^{22}$	$n = 2^{24}$		
Our Standard PSI						Ring-LPN	✓
$\ell = 64$	$724n$	$423n$	$342n$	$324n$	$323n$	+ OT	
$\ell = 48$	$692n$	$391n$	$310n$	$292n$	$291n$		
$\ell = 32$	$660n$	$359n$	$278n$	$260n$	$259n$		
RS21 [RS21] enhanced*	$318n$	$286n$	$279n$	$279n$	$280n$	LPN + OT	✗
Our Direct PSI						LPN + OT	✗
$\ell = 64$	$421n$	$385n$	$374n$	$369n$	$365n$		
$\ell = 48$	$348n$	$311n$	$298n$	$292n$	$286n$		
$\ell = 32$	$277n$	$237n$	$223n$	$215n$	$208n$		
Our Dual PSI							
$\ell = 64$	$609n$	$535n$	$511n$	$499n$	$489n$		
$\ell = 48$	$465n$	$388n$	$361n$	$345n$	$333n$		
$\ell = 32$	$321n$	$240n$	$210n$	$192n$	$176n$		
PRTY20 [PRTY20]	$1766n$					OT	✗
RT21 [RT21b]	$512n$					DH	✗
RS21 [RS21] enhanced**	$320n$	$315n$	$315n$	$317n$	$318n$	LPN + OT	✗

* Using interpolation instead of PaXoS, and the VOLE of [CRR21] instead of the one from [WYKW21]. Sets κ_{RS21} to $\kappa + \log n$ to achieve κ bits of security.

** Using the new OKVS of [GPR+21] instead of PaXoS, and the VOLE of [CRR21] instead of the one from [WYKW21]. Sets κ_{RS21} to $\kappa + \log n$ to achieve κ bits of security.

the malicious setting, a small database-size-independent 2-round structural check), from which they can decode $X \cap X_i$. To achieve this feature, we build upon the fact that the PCG for subfield ring-OLE correlations is *programmable*, which means that we can enforce that a target party will receive the same pseudorandom string across executions with many different parties. Concretely, we achieve the following form of *batch non-interactive PSI* between a client C with database X and multiple servers S_i with datasets X_i (all of size n):

1. In a preprocessing phase, C interacts with each of the servers, using $O(\log n)$ communication *and* computation in each interaction, in a small constant number of rounds.
2. Then, C performs a single $\tilde{O}(n)$ cost local computation, and broadcasts a single $2\ell n$-size *encoding* E_X of X.
3. Each server S_i can, at any time, send a single message $M_i = m(X_i, E_X)$, of length $3(\lambda + \log n)n$, using $\tilde{O}(n)$ computation.
4. Eventually, given X and M_i, the client C can run a $\tilde{O}(n)$ cost decoding procedure and recover $X \cap X_i$, without further interaction.

When the number of servers becomes large, our batch PSI protocol leads to strong savings for the client compared to executing a PSI protocol individually with each server. Furthermore, in this setting, the amortized communication (per PSI instance) is reduced to $(2\ell/N_S + 3\lambda + \log n) \cdot n + o(n)$, where N_S denotes the number of servers. Even for relatively small number of servers, the amortized communication quickly outperforms that of even the best ROM-based maliciously secure

PSI protocols. For example, for $n = 2^{24}$ and $\ell = 32$, the amortized communication per secure set intersection approaches $195n$ bits with our protocol, versus $280n$ for [RS21].

1.2 Concurrent work

In a concurrent and independent work, recently accepted at CCS'22, Rindal and Raghuraman [RR22] introduced a new PSI protocol, using an approach similar to ours: the authors also leveraged subfield-VOLE to achieve communication independent of the computational security parameter κ. Our results have been obtained independently of theirs, around the same time period. Although their main result bears similarities to our first two contributions, we highlight some important distinctions between our work and theirs:

- The work of [RR22] uses an OKVS-based construction, and achieves a receiver-to-sender communication of $(\lambda + 2\log n) \cdot n$. In contrast, we use a hash-based protocol, and achieve an $(\ell - \log n) \cdot n$ receiver-to-sender communication. Therefore, we get smaller communication overall in the setting where the databases have small entries, but a slightly larger computation.
- For malicious security, the work of [RR22] only considers the standard paradigm of previous works (e.g. [RS21]), hence having a $O(\kappa \cdot n)$ receiver-to-sender (and overall) communication. In contrast, we give two protocols, including one based on dual execution which achieves communication independent of κ (and smaller concrete communication for databases with small entries).
- Eventually, our last contribution, a "batchable" ring-OLE-based malicious PSI in the standard model with low communication, is unique to our work.

1.3 Structure of the Paper

We provide preliminaries in Section 2 (additional preliminaries are given in Appendix A of the Supplementary Material), and a detailed technical overview of our contributions in Section 3. Section 4 covers our ROM-based semi-honest and malicious protocols. Due to space limitation, our second malicious protocol, based on dual execution, is deferred to Appendix B of the Supplementary Material. Section 5 covers our standard model PSI and Appendix C provides a detailed comparison of our protocols to a PSI protocol of [PSZ14].

2 Preliminaries

Notation. Throughout the paper we use the following notations: we let κ, λ denote the computational and statistical security parameters, respectively. We write $[1, m]$ to denote a set $\{1, 2, \ldots, m\}$. For a vector $\mathbf{x}$ we define by x_i its i-th coordinate. Given distribution ensembles $\{X_n\}, \{Y_n\}$, we write $X_n \approx Y_n$ to denote that X_n is computationally indistinguishable to Y_n.
We typically write $\mathbb{F}_q$ to denote a field with and arbitrary subfield $\mathbb{F}_p$, where p is a prime power and $q = p^t$. We use $\mathcal{R}_{\mathsf{p}} = \mathbb{F}_{\mathsf{p}}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$ for the ring over the field $\mathbb{F}_p$ where $F(x)$ is some polynomial, and also denote $\mathcal{R}_{\mathsf{q}} = \mathbb{F}_{\mathsf{p}^t}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$. Note that all operations in our paper are field/ring operations not modular arithmetic.

PSI functionality. A private set intersection (PSI) protocol allows two parties to compute the intersection of their input sets while concealing all other information. We typically denote by n the input set sizes. For completeness, the ideal functionalities for PSI (in the semi-honest and in the malicious settings) are given in Appendix A of the Supplementary Material.

Pseudorandom correlation generators (PCG). Pseudorandom correlations generators have been introduced in a recent line of work [BCGI18,BCG+19b,BCG+19a]. A PCG allows to compress long correlations into short, correlated seeds that can later be locally expanded into pseudorandom instances of the target correlation. Slightly more formally, a PCG for a target correlation C (which samples pairs of long correlated strings (y_0, y_1)) is a pair $(\mathsf{Gen}, \mathsf{Expand})$ of algorithms such that $\mathsf{Gen}(1^\lambda)$ outputs a pair of short, correlated keys $(\mathsf{k}_0, \mathsf{k}_1)$ and $\mathsf{Expand}(\sigma, \mathsf{k}_\sigma)$ outputs a long string $\tilde{y}_\sigma$. Correctness states that $(\tilde{y}_0, \tilde{y}_1)$ are indistinguishable from a random sample from C, while security states that given $\mathsf{k}_{1-\sigma}$, $\tilde{y}_\sigma$ looks like a random sample from C conditioned on satisfying the target correlation with $\mathsf{Expand}(1-\sigma, \mathsf{k}_{1-\sigma})$, for $\sigma = 0, 1$.

A PCG does not in itself provide a protocol to efficiently generate long pseudorandom correlations. To get the latter, one must combine a PCG with a *distributed key generation* protocol, which allows

two parties to obliviously run $\mathsf{Gen}(1^\lambda)$ such that each party gets one of the keys. Fortunately, for most PCGs of interest (and in particular, for all PCGs we use in this work), there exists very efficient low-communication distributed setup protocols [BCG+19a, BCG+20b]. Combining a PCG with a distributed setup protocols allows to securely instantiate (with low communication) functionalities that distribute instances of the target correlation. In this work, we will directly rely in a black-box way on such functionalities, and use known protocols to instantiate them. We now expand on the two main functionalities we use in this work.

PARAMETERS:

- 2 parties, a sender and receiver, an integer n, the size of the output vector.
- A finite field $\mathbb{F}_q$ where $q = p^r$, p is a power of prime, r an integer.

FUNCTIONALITY:

- Depending on the parties:
  - If the sender is corrupted then wait for $\mathcal{A}$ to send 2 vectors $\mathbf{u} \in \mathbb{F}_p^n, \mathbf{v} \in \mathbb{F}_q^n$; samples $\Delta \leftarrow_r \mathbb{F}_q$ and computes $\mathbf{w} := \Delta \cdot \mathbf{u} + \mathbf{v}$.
  - If the receiver is corrupted then wait for $\mathcal{A}$ to send $\mathbf{w} \in \mathbb{F}_q^n$, $\Delta \in \mathbb{F}_q$; samples $\mathbf{u} \leftarrow_r \mathbb{F}_p^n$ and computes $\mathbf{v} := \mathbf{w} - \Delta \cdot \mathbf{u}$.
  - Otherwise, samples $\mathbf{u} \in \mathbb{F}_p^n, \mathbf{v} \in \mathbb{F}_q^n, \Delta \leftarrow_r \mathbb{F}_q$ and computes $\mathbf{w} := \Delta \cdot \mathbf{u} + \mathbf{v}$.
- The functionality sends $\mathbf{u} \in \mathbb{F}_p^n$, $\mathbf{v} \in \mathbb{F}_q^n$ to sender and $\Delta \in \mathbb{F}_q$, $\mathbf{w} := \Delta \cdot \mathbf{u} + \mathbf{v}$ to receiver.

Fig. 1. Ideal functionality $(n, p, q) - \mathcal{F}_{\mathsf{svole}}$ of subfield vector-OLE

Subfield Vector-OLE. We described the subfield vector-OLE correlation in the technical overview (see Section A.4). We represent on Figure 1 the ideal functionality that distributes a subfield VOLE correlation. In our concrete instantiations, we will instantiate this functionality using the efficient protocol of [BCG+19a]. The latter provides a general template which can be instantiated under various flavors of the LPN assumption, and provides a conservative choice under LPN for quasi-cyclic choice. A variant of LPN that leads to a considerably more efficient protocol, when plugged in the template of [BCG+19a], was recently put forth in the work [CRR21] (we note that our communications estimate are oblivious to the underlying variant: only the computational costs depends on the LPN flavor).

Subfield Ring-OLE. Recently, a new PCG construction was described in [BCG+20b] for the *ring-OLE* correlation. The ring-OLE correlation over a ring $\mathcal{R}_\mathsf{q}$ is the following correlation: $\{((x_0, z_0), (x_1, z_1)) \mid x_0, x_1, z_0 \leftarrow_r \mathcal{R}_\mathsf{q}, z_1 \leftarrow x_0.x_1 - z_0\}$. In this work, we rely on a slight variant of the ring-OLE correlation, where x_0 is instead sampled from a subring $\mathcal{R}_\mathsf{p}$ of $\mathcal{R}_\mathsf{q}$. We represent the corresponding variant of the ideal functionality on Figure 9. We note that the protocol of [BCG+20b] to instantiate the ring-OLE functionality can be adapted to handle the subfield ring-OLE functionality in a straightforward way.

3 Technical Overview

Our starting point is the classical KKRT protocol [KKRT16], which combines Cuckoo hashing with a batch related-key oblivious pseudorandom function (BaRK-OPRF). We assume some familiarity with the KKRT protocol in this technical overview. For completeness, we provide a high level overview of KKRT, the notion of BaRK-OPRF (batch related-key oblivious pseudorandom function), and its communication costs in Appendix A.7 of the Supplementary Material. Our construction will also rely on a functionality that distributes *subfield vector-OLE* correlation (the sVOLE functionality): Alice gets $(\mathbf{u}, \mathbf{v})$, and Bob gets $(\Delta, \mathbf{w} = \Delta\mathbf{u} + \mathbf{v})$. Such correlation can be distributed with very low communication using pseudorandom correlation generators; we provide more details in Appendix A.4 of the Supplementary Material.

3.1 A New sVOLE-Based PSI for Databases with Small Entries

Subfield-VOLE leads to a simple and natural construction of BaRK-OPRF. Let ℓ be the bitlength of Alice's inputs, and let $\mathbf{x} = (x_1, \cdots, x_n)$ be the inputs of Alice, viewed as elements of $\mathbb{F}_{2^\ell}$. We assume for simplicity that ℓ divides κ, the computational security parameter. Alice and Bob use an sVOLE protocol (e.g. [CRR21]) over the field $\mathbb{F}_{2^\kappa}$, with subfield $\mathbb{F}_{2^\ell}$; let $(\mathbf{u}, \mathbf{v})$ be the output of Alice, and

$(\Delta, \mathbf{w})$ be the output of Bob. Recall that $\mathbf{w} = \Delta \cdot \mathbf{u} + \mathbf{v}$. Alice sends $\mathbf{z} = \mathbf{x} - \mathbf{u}$ to Bob, who defines the BaRK-OPRF keys to be Δ and $(K_1, \cdots, K_n) = \Delta \cdot \mathbf{z} + \mathbf{w}$. The BaRK-OPRF is defined as follows: $F_{\Delta,K_i}(y) = H(i, K_i - \Delta \cdot y)$ (all operations are over $\mathbb{F}_{2^\kappa}$). Eventually, Alice outputs $(H(i, v_i))_{i \leq n}$. Observe that

$$\begin{aligned} H(i, v_i) &= H(i, w_i - \Delta u_i) = H(i, K_i - \Delta(z_i + u_i)) \\ &= H(i, K_i - \Delta \cdot x_i) = F_{\Delta,K_i}(x_i) \end{aligned}$$

The use of sVOLE, rather than OT extension as in the original KKRT BaRK-OPRF, has two main advantages: first, the bitwise AND is now replaced by a field multiplication. In particular, this means that we do not need anymore to use error-correcting codes, and that $y \cdot \Delta$ retains the entire entropy of Δ. In other words, it suffices for Δ to be κ-bit long to achieve κ bits of security for the construction (in contrast, KKRT had to use around 5κ bits). Second, and most importantly, the use of *subfield* VOLE allows us to completely decorrelate the size of $\mathbf{u}$ from that of Δ, something which can fundamentally not be achieved with the INKP OT extension. Concretely, this means that $\mathbf{u}$ only needs to mask the input vector $\mathbf{x}$ of Alice. If $\mathbf{x} \in \mathbb{F}_{2^\ell}^n$, then so do $\mathbf{u}$ and $\mathbf{z}$: the communication now depends solely on the input size.

In total, our BaRK-OPRF communicates $\ell \cdot n$ bits, plus the cost of distributing the seeds for the sVOLE generator. Using the protocol of [BCG+19a] to distribute the seeds[4], the cost is logarithmic in n, hence its effect on the overall communication vanishes for large enough n.

Combining the new OPRF with permutation-based hashing. Plugging our new BaRK-OPRF into KKRT, and using the same parameters for Cuckoo hashing, leads to a protocol with total communication $(1.3 \cdot \ell + 3 \cdot (\lambda + 2\log n))n + o(n)$ bits (where the $o(n)$ terms capture the costs of distributing the PCG seeds). Concretely, for $n = 2^{20}$ and $\ell = 32$ (resp. 64), this already brings the cost down, from $1008n$ bits to $282n$ bits (resp. $324n$ bits). However, this can be further improved using the well-established notion of *permutation-based* hashing [PSSZ15]. Concretely, in *permutation-based* hashing, an item x is written as $x_L || x_R$, where x_L is $\log(1.3n)$-bit long. The item x is inserted by mapping x_R to the bin $x_L \oplus f(x_R)$, where f is a k-wise independent hash function, for some large enough k. This guarantees that no collision occurs, because if two items x, x' end up mapping the same value to the same bin, this means that $x_R = x'_R$ and $x_L \oplus f(x_R) = x'_L \oplus f'(x'_R)$, hence $x = x'$. When multiple hash functions are used, as in Cuckoo hashing, the index of the hash function must be appended to x_R.

Interestingly, our use of sVOLE is crucial to enabling a permutation-hashing-based optimization: the latter only provides savings when the communication involves a $O(\ell \cdot n)$ component (which neither KKRT nor any modern OKVS-based PSI has). In our protocol, however, it further reduces the communication to $(1.3 \cdot (\ell - \log(1.3n) + 1) + 3 \cdot (\lambda + 2\log n))n + o(n)$ bits, which gives $275n$ bits for $n = 2^{20}$ and 32-bit items, or $317n$ bits for 64-bit items. In itself, this is a really small communication improvement. However, it has an important consequence: it implies that the Alice-to-Bob communication is now completely dominated by the Bob-to-Alice communication. Concretely, this means that we can easily afford to use a much higher number of bins (which is $1.3n$ currently) if it can allow us to reduce the number of hash functions (which is 3). This brings us to our last optimization.

Packing multiple items per bin with generalized Cuckoo hashing. In this last optimization, our goal is to reduce the number of hash functions used in the Cuckoo hashing protocol, from 3 to 2, by increasing the number of bins to compensate. Unfortunately, this does not work directly with standard cuckoo hashing even while using a reasonably small stash since the cost of handling the stash is high, and nullifies all communication benefits of using two hash functions in the first place. Instead, we use a different approach: we add one degree of freedom to the Cuckoo hashing parameters, *by allowing bins to contain multiple items*. This generalization of Cuckoo hashing is not new: it has been studied in details in several works [DW07, W+17], because it comes with a much nicer cache-friendliness than standard Cuckoo hashing.

In (d, k)-Cuckoo hashing, n items are mapped to $(1+\varepsilon) \cdot n$ bins using k hash functions, and each bin is allowed to contain up to d items. Allowing more items per bins significantly improves the efficiency; for example, $(3, 2)$-Cuckoo hashing is known to perform strictly better than standard $(1, 3)$-Cuckoo hashing in terms of occupancy (i.e., the total number of slots $N = d \cdot (1 + \varepsilon) \cdot n$ which must be used

[4] This protocol uses a length-t reverse VOLE protocol as a blackbox, which we instantiate with the construction of [ADI+17].

to guarantee a $o(1)$ failure probability). Based on existing analysis of this variant [W+17], it seems reasonable to expect that $(3,2)$-Cuckoo hashing already achieves a strictly smaller failure probability compared to $(1,3)$-Cuckoo hashing, with a smaller number of bins.

We relied on extensive computer simulations on small values of n (from 256 to 2048) to select parameters, and extrapolated from these results parameters for larger values of n. More precisely, we ran 10^7 experiments with $(3,2)$-Cuckoo hashing for $n \in \{2^8, 2^9, 2^{10}\}$ (we also experimented with 2^{11}, but with a smaller number of experiments) with $c \cdot n$ bins for various values of c. Even for a value as low as $c = 0.65$ and values of n as low as 2^9, our experiments never reported any insertion failure, indicating that the empirical failure probability should already be way below 2^{-20}. Since the theoretical failure probability is known to scale as $O(1/n^\delta)$ for some constant δ with reasonably small constant factors, we extrapolate that for large enough values of n, e.g. $n \geq 2^{18}$, the failure probability should be well below 2^{-40}.

Alternative hashing variants. Alternatively, when allowing multiple items per bins, we can consider other hashing variants. Two natural choices are two-choice hashing [PRTY19], where each bin can have up two d items and each item is placed in the least-full of two bins, and simple hashing, where a single hash function is used to map the items to bins (standard results show that, when hashing n items to $O(n)$ bins this way, the maximum load with be of the order of $\log n / \log \log n$ with high probability). As we will see, these choices of hashing lead to various communication versus computation tradeoffs in our protocols, and the optimal choice also depends on the database size.

A membership BaRK-OPRF. There remains a non-trivial task: to use some of the above hashing variants, we need a protocol to handle hashing with up to d items per bins. Intuitively, denoting $\mathbf{x_i} = (x_i^{(1)}, \cdots, x_i^{(d)})$ the d entries of the bin i, we want to construct a new kind of *membership* OPRF (similar in spirit to the notion of multi-point OPRF in the literature), where Bob obtains $F_{\Delta,K_i}(y)$ and Alice obtains the set $F_{\Delta,K_i}(\mathbf{x_i}) = \{F_{\Delta,K_i}(x_i^{(j)})\}_{j \leq d}$. This implies that $F_{\Delta,K_i}(y) \in F_{\Delta,K_i}(\mathbf{x_i})$ if and only if y is equal to any entry of $\mathbf{x_i}$, and $F_{\Delta,K_i}(y)$ looks pseudorandom to Alice otherwise.

Going back to the BaRK-OPRF, recall that for a bin i where Alice placed x_i and Bob placed y_i, Alice computes $H(i, v_i)$ and Bob computes $H(i, K_i - \Delta y_i) = H(i, \Delta \cdot (x_i - y_i) + v_i)$. Here, we view the $x_i - y_i$ term as $P_{x_i}(y_i)$, where $P_{x_i} = X - x_i$ is a degree-1 polynomial with root x_i. This view suggests a natural generalization of this approach, where the P_{x_i} polynomials are replaced by higher degree polynomials. Define $P_{\mathbf{x_i}}$ to be the polynomial $\prod_{j=1}^{d}(X - x_i^{(j)})$, and let $(c_{j,i})_{0 \leq j \leq d-1}$ denote its coefficients: $P_{\mathbf{x_i}}(X) = X^d + \sum_{j=0}^{d-1} c_{j,i} \cdot X^j$. Our new *membership* BaRK-OPRF is a direct generalization of the BaRK-OPRF from Section 3.1, which we sketch below.

Our construction. Let m be the bitlength of Alice's inputs inside the bins, and let $(\mathbf{x_1}, \cdots, \mathbf{x_N})$ be the inputs of Alice in each of the N bins, where the inputs in each bin are viewed as length-d vectors of elements of $\mathbb{F}_{2^m}$. We assume for simplicity that m divides κ, the computational security parameter. Alice and Bob use d sVOLE protocol (e.g. [CRR21]) over the field $\mathbb{F}_{2^\kappa}$, with subfield $\mathbb{F}_{2^m}$, *with the same value* Δ.[5] Let $(\mathbf{u_j}, \mathbf{v_j})_{j \leq d}$ be the outputs of Alice, and $(\Delta, (\mathbf{w_j})_{j \leq d})$ be the output of Bob. Recall that $\mathbf{w_j} = \Delta \cdot \mathbf{u_j} + \mathbf{v_j}$.

For each $\mathbf{x_i}$, let $(c_{0,i}, \cdots, c_{d-1,i})$ be the coefficients of the polynomial $P_{\mathbf{x_i}}$ (omitting the coefficient of X^d, which is always 1). Let $\mathbf{c_j}$ denote the vector $(c_{j,i})_{i \leq N}$ for $j = 0$ to $d-1$. Alice sends $\mathbf{z_j} = \mathbf{c_j} - \mathbf{u_j}$ for $j = 0$ to $d-1$ to Bob, who defines the membership BaRK-OPRF keys to be Δ and $K_i = (k_{j,i})_{0 \leq j \leq d-1} = (\Delta \cdot z_{j,i} + w_{j,i})_{0 \leq j \leq d-1}$ for $i = 1$ to N. Define the following degree-d polynomial P_{Δ,K_i} over $\mathbb{F}_q$: $P_{\Delta,K_i}(X) = \Delta \cdot X^d + \sum_{j=0}^{d-1} k_{j,i} \cdot X^j$. The OPRF is defined as follows: $F_{\Delta,K_i}(y) = H(i, P_{\Delta,K_i}(y))$ (all operations are over $\mathbb{F}_{2^\kappa}$). Eventually, for each bin i, Alice sets her d tuple of outputs to be $F_{\Delta,K_i}(\mathbf{x_i}) = \{H(i, \sum_{j=0}^{d-1} v_{j,i} \cdot (x_i^{(k)})^j\}_{k \leq d}$. Observe that, since $k_{j,i} = \Delta z_{j,i} + w_{j,i} = \Delta c_{j,i} + v_{j,i}$ for all i, j, we have $H(i, P_{\Delta,K_i}(y)) = H\left(i, \Delta \cdot \left(y^d + \sum_{j=0}^{d-1} c_{j,i} y^j\right) + \sum_{j=0}^{d-1} v_{j,i} y^j\right)$, which is equal to $H\left(i, \Delta \cdot P_{\mathbf{x_i}}(y) + \sum_{j=0}^{d-1} v_{j,i} y^j\right)$. Therefore, if there exists $k \in \{1, \cdots, d\}$ such that $y = x_i^{(k)}$, we have $P_{\mathbf{x_i}}(y) = 0$, and $H(i, P_{\Delta,K_i}(y)) = H(i, \sum_{j=0}^{d-1} v_{j,i} \cdot (x_i^{(k)})^j) \in F_{\Delta,K_i}(\mathbf{x_i})$. On the other hand, whenever $P_{\mathbf{x_i}}(y) \neq 0$, then the $\Delta \cdot P_{\mathbf{x_i}}(y)$ term in the hash makes the output pseudorandom from the viewpoint of Alice, under the correlation robustness of the hash function.

[5] Note that all known sVOLE protocols allow Bob to choose the value of Δ, hence Bob can enforce the use of the same Δ across all instances.

Tying up loose ends. Using the new construction from the previous Section, together with $(3,2)$-Cuckoo hashing, leads to a total communication of $(0.65\cdot 3(\ell-\log(0.65n)+1)+2\cdot(\lambda+2\log n))n+o(n)$ bits, where the $o(n)$ corresponds to the cost of setting up the PCG seeds. For $n=2^{20}$ and 32 bits items, this gives $148n$ bits of communication. We mention a few remaining details. First, in the construction of membership BaRK-OPRF, Alice and Bob need to invoke $d=3$ length-N sVOLE. In fact, it suffices to invoke a single length-$3N$ sVOLE, and to cut the output in three equal length parts, to obtain the necessary correlation. This means that the concrete cost of distributing the sVOLE seeds remains that of generating a single sVOLE (e.g. $\approx 0.7n$ bits for $n=2^{20}$).

Second, in the above, we overlooked an important subtlety: a bin can possibly contain less than d items. In KKRT, this was handled by adding dummy items to empty bins. We use instead a more efficient approach with a negligible extra cost called a *variant* of our OPRF (details in section 4).

3.2 Malicious Security

We then turn our attention to maliciously secure PSI. Here, it is well known that Cuckoo hashing and two-choice hashing are not usable. Consequently, we focus on simple hashing as our choice of the underlying hash technique. Using maliciously secure subfield-VOLE, which can be implemented very efficiently [BCG+19a, CRR21], we enhance our membership BaRK-OPRF to the malicious setting, with a minimal overhead. Then, we apply two standard methods to achieve security against malicious adversaries in our PSI protocol:

First method: direct approach. The first method increases the PRF output length to κ. Using the analysis of [RS21], this suffices to allow for extracting the input of a malicious sender. However, this makes the communication depend linearly on κ, which severely harms communication complexity.

Second method: dual execution. To recover a κ-independent communication complexity, we then turn our attention to the dual execution technique [RR17]. Here, the idea is simple: the parties will invoke the malicious BaRK-OPRF twice, exchanging their roles. Then, the sender sends, for each entry x of his database, a value of the form $\mathsf{PRF}_A(x)\oplus\mathsf{PRF}_B(x)$, where $\mathsf{PRF}_A(x)$ is obtained by the sender when invoking the BaRK-OPRF functionality as sender, and $\mathsf{PRF}_B(x)$ is the PRF output obtained when invoking the functionality as receiver. Here, it becomes possible to extract the input set of each party simply from its call as receiver to the BaRK-OPRF functionality, which does not require to increase the output length of the OPRF. The price to pay is that the protocol now uses two calls to the BaRK-OPRF. Concretely, the total communication becomes $(2\cdot N\cdot d(\ell-\log(N))+(\lambda+\log n))n+o(n)$, where N is the number of bins, d the maximum load of a bin, and ℓ the input size (e.g. for $n=2^{20}$, one can choose $N=n/10$ and $d=47$, see [RR17, Figure 5]). For small database entries, this outperforms all known malicious PSI protocols.

3.3 An Efficient PSI in the Standard Model

In our last construction, we use a different functionality: we rely on the subfield ring-OLE functionality (given on Figure 9), that generates a subfield ring-OLE correlation over the rings $\mathcal{R}_{\mathsf{p}}=\mathbb{F}_{\mathsf{p}}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$, $\mathcal{R}_{\mathsf{q}}=\mathbb{F}_{\mathsf{p}^t}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$, and $F(X)$ is some polynomial of degree $2n+1$ (more generally, when the two parties have sets of different size n and m, F will be of degree $n+m+1$). At a high level, the functionality $\mathcal{F}_{\mathsf{sole}}$ distributes to Alice $(a,s_A)\in\mathcal{R}_{\mathsf{p}}\times\mathcal{R}_{\mathsf{q}}$ and $(b,s_B)\in(\mathcal{R}_{\mathsf{q}})^2$ to Bob such that $ab=s_A+s_B$. Our protocol makes a single black-box call to this functionality. Consider two parties, a sender Alice and a receiver Bob, where Alice has a set $A=\{x_1,x_2,\ldots,x_n\}\in\mathbb{F}_p^n$ and Bob has a set $B=\{y_1,y_2,\ldots,y_n\}\in\mathbb{F}_p^n$. Define $p_A:=\prod_{i=1}^n(X-x_i)\in\mathcal{R}_p$ and $p_B:=\prod_{i=1}^n(X-y_i)\in\mathcal{R}_p$. Let $I:=A\cap B$ denote the target output. The protocol computes the common roots of p_A and p_B, *i.e.*, $\gcd(p_A,p_B)$.

By revealing appropriate linear combination of their shares and their input polynomials, Alice and Bob will "derandomize" this correlation, allowing Alice to learn the polynomial $u=p_Ab_0+p_Bb'_0$, where b_0,b'_0 are two uniformly random degree-n polynomials known by Bob (this also requires revealing the high-order coefficients of b, to reduce the degree-$2n$ random polynomial b to a degree-n random polynomial b_0). Using some standard lemmas about polynomials (Lemma 8 and Lemma 9), the polynomial u can be factored as $\gcd(p_A,p_B)\cdot p_R$, where with high probability, p_R has no common root with p_A. This allows Alice to compute the intersection $I=A\cap B$ as $I=\{x_i\in A\ :\ u(x_i)=0\}$. Concretely:

- Alice computes and sends $t_A=a-p_A$ to Bob.

- Bob sets $s'_B \leftarrow s_B - t_A b$. Then, Bob decomposes b as $b = b_0 + b_1 \cdot X^n$ (where b_0, b_1 are degree-n polynomials), sets $s'_B \leftarrow s_B - t_A b$, and picks a random degree-n polynomial b'_0 over $\mathcal{R}_q$. He sends b_1 and $t_B \leftarrow s'_B + p_B b'_0$ to Alice.
- **(Output)** Alice sets $u \leftarrow t_B - p_A b_1 \cdot X^n + s_A$; note that $u = p_A b_0 + p_B b'_0$. Alice outputs the set $I = \{x \in A \mid u(x) = 0\}$.

We prove that this construction achieves "augmented semi-honest security", a strengthening of honest-but-curious corruption where the adversary is allowed to change the corrupted parties' inputs. Furthermore, we securely realize the functionality $\mathcal{F}_{\mathsf{sole}}$ using the PCG-based protocol of [BCG$^+$20b], which is secure under the ring-LPN assumption. Instantiating the subfield ring OLE this way allows to import a powerful feature of the PCG of [BCG$^+$20b], which is its *programmability*: when generating a ring-OLE correlation, the receiver can ensure that her output a remains *identical* across multiple instances of the protocol with different parties. Using this programmability feature, we show that our protocol can be *batched*: a single $O(\ell \cdot n)$-size client message encoding her database A can be reused with N different servers with databases B_i, allowing her to learn $A \cap B_i$ using a single message from each server afterwards.

Achieving malicious security. We then turn our attention to security against malicious adversaries. Our upgrade introduces only a minimal communication overhead to the protocol, independent of the set sizes n. At a high level, the main issues that can occur in the malicious setting is when Alice sets $p_A = 0$, or when Bob sets $p_B b'_0 = 0$. Indeed, since Alice gets $u = p_A b_0 + p_B b'_0$, if $p_A = 0$, she can learn Bob's entire input set p_B. On the other hand, if $p_B b'_0 = 0$, Bob forces the output to be A.

We handle both issues separately. The second issue is intuitively simpler to handle, since when Bob carries out this attack, Alice will notice that her output is exactly her set A. This suggests a simple way around: if Alice notice at the end of the protocol that the output is equal to A, she aborts the protocol. Of course, a honest Bob could have an input B with $A \subseteq B$, in which case this modification would harm correctness. But there is a simple way around: prior to the protocol, Alice and Bob can just agree on a reserved dummy item d (we will pick $d = 1$ in the protocol, but this choice is arbitrary), which is guaranteed to be in neither databases. If database entries are elements of a field $\mathbb{F}_{\mathsf{p}'}$, this can simply be done by choosing any slightly larger field $\mathbb{F}_{\mathsf{p}}$ of size $|\mathbb{F}_{\mathsf{p}}| \geq |\mathbb{F}_{\mathsf{p}'}| + 1$, reserving one element of $\mathbb{F}_{\mathsf{p}}$ to encode d, and mapping the elements of $\mathbb{F}_{\mathsf{p}'}$ to the remaining elements. Then, Alice and Bob execute the protocol on inputs $A \cup \{1\}$ and B, which guarantees that B does not contain A.

For the first issue, Bob must check before sending $t_B = s'_B + p_B b'_0$ that Alice did not set p_A to be 0 when computing $t_A = a - p_A$. Intuitively, this will be done by letting Bob check that $p_A(x) \neq 0$, for an appropriate input x. This, however, must be done with some care, since learning $p_A(x)$ could leak information to a corrupted Bob. We handle this issue by reserving a second element of $\mathbb{F}_{\mathsf{p}}$ (hence we now need $|\mathbb{F}_{\mathsf{p}}| \geq |\mathbb{F}_{\mathsf{p}'}| + 2$), which we assume w.l.o.g. to be 0, which should again be in neither set. Then, Alice will define the encoding of her set to be the degree-n polynomial p_A such that $p_A(\mathsf{map}(a)) = 0$ for every $a \in A$, and $p_A(0) = 1$. Then, we let Bob first send b_1, without sending t_B. Afterwards, Bob computes $s'_B \leftarrow s_B - t_A b$ and Alice computes $s'_A \leftarrow s_A - p_A b_1 \cdot X^n$. Observe that if both parties behave honestly, $s'_a + s'_b = ab - t_A b - p_A b_1 \cdot X^n = ab - ab + p_A b - p_A b_1 \cdot X^n = p_A b_0$. To enforce $p_A \neq 0$, we will check that the above equation holds for some nonzero p_A. Crucially, since both p_A and b_0 have degree at most n, no reduction modulo $\mathsf{F}(X)$ occurs in the right hand side of the equation. This implies that we can simply check that the equation holds for the reserved input $x = 0$ (since a honest p_A is guaranteed to satisfy $p_A(0) = 1 \neq 0$). To check this, we let Alice send $s'_A(0)$ to Bob, who checks that $s'_A(0) = b_0(0) - s'_B(0)$; if the check fails, Bob aborts the protocol.

4 PSI from Subfield-VOLE

4.1 A new membership batched OPRF

Our BaRK-OPRF allows the sender to hold a set of keys $(\mathbf{k_i})_{i \leq N}$ such that each key is assigned with a tuple of d input elements of the receiver and then the receiver learns a PRF output on each element in this tuple corresponding with the same key. More formally, denoting $\mathbf{x_i} = (x_i^{(1)}, \cdots, x_i^{(d)})$ consisting of d entries, the sender gets $F(i, y)$ and the receiver obtains a set $\{F(i, x_i^{(j)})\}_{j \leq d}$ such that $F(i, y) \in \{F(i, x_i^{(j)})\}_{j \leq d}$ if and only if y is equal to any entry of $\mathbf{x_i}$, and $F(i, y)$ looks pseudorandom to the receiver otherwise.

PARAMETERS:

$\mathbb{F}_p$ is a finite field. There are 2 parties, a sender and a receiver with input set $X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\} \subseteq \mathbb{F}_p$ where $\mathbf{x_i} = (x_i^{(1)}, \cdots, x_i^{(d)})$.

FUNCTIONALITY:

- Wait for input $(\mathsf{sender}, \mathsf{id})$ from the sender and $(\mathsf{receiver}, \mathsf{id}, \mathsf{X})$ from the receiver. The functionality samples a PRF F then $\forall x \in \mathbf{x_i}$ outputs $F(i, x)$ to the receiver for $i \in [1, N]$.
- When the sender inputs any $(i, y) \in [1, N] \times \mathbb{F}_p$, functionality gives $F(i, y)$ to the sender.

Fig. 2. Ideal functionality $\mathcal{F}_{\mathsf{oprf}}$

Main construction. Assume that the receiver inputs the set of $n = Nd$ elements: $X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\} \subseteq \mathbb{F}_p$ where $\mathbf{x_i} = (x_i^{(1)}, \cdots, x_i^{(d)})$. First, the sender and the receiver invoke the $\mathcal{F}_{\mathsf{svole}}$ protocol of dimension n, with their roles reversed, to get a random sVOLE correlation. Specifically, the receiver learns a pair of vectors $(\mathbf{u}, \mathbf{v})$ where $\mathbf{u} \in \mathbb{F}_p^n$, $\mathbf{v} \in \mathbb{F}_q^n$, the sender gets $\Delta \in \mathbb{F}_q$ and $\mathbf{w} := \Delta \cdot \mathbf{u} + \mathbf{v}$. Denoting $\mathbf{u} = (\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_N})$ where $(u_{j,i})_{1 \le j \le d}$ are d entries of vector $\mathbf{u_i}$. This notation is the same for $\mathbf{v}, \mathbf{w}$. Consider $\mathbf{x_i}$ and its associated polynomial as $P_{\mathbf{x_i}}(X) = \prod_{j=1}^{d}(X - x_i^{(j)}) = X^d + \sum_{j=1}^{d} c_{j,i} \cdot X^{j-1}$ where $c_{j,i} \in \mathbb{F}_p$ for $i \in [1, N]$, $j \in [1, d]$.

Now, the receiver defines $\mathbf{c_i} := (c_{j,i})_{j \le d}$, $\mathbf{c} := (\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_N})$, and then $\forall i \in [1, N]$ sends to the sender $\mathbf{z_i} := \mathbf{c_i} - \mathbf{u_i} \in \mathbb{F}_p^d$. Above, the $\mathbf{u_i}$ are masks for the coefficients $\mathbf{c_i}$ of (the polynomial associated) $\mathbf{x_i}$. Indeed, $\mathbf{u_i}$ are distributed uniformly at random in the subfield $\mathbb{F}_p$, then the vector $\mathbf{z_i}$ is a uniformly random over $\mathbb{F}_p^n$ from the viewpoint of the sender. The two parties will run a coin flipping protocol to get a random value $t \leftarrow \mathbb{F}_q$. For $i \in [1, N]$, the receiver defines the PRF output on each input $x \in \mathbf{x_i}$ as $F(i, x) = \mathsf{H}\left(i|t|x\ ,\ \sum_{j=1}^{d} v_{j,i} \cdot x^{j-1}\right)$.

On the other hand, after receiving the vectors $\mathbf{z_i}$, for $i \in [1, N]$, the sender defines the vector $\mathbf{k_i} := \mathbf{w_i} + \Delta \cdot \mathbf{z_i}$. As a consequence, for any input $(i, y) \in [1, N] \times \mathbb{F}_p$, its PRF output is computed as: $F(i, y) = \mathsf{H}\left(i|t|y\ ,\ \Delta \cdot y^d + \sum_{j=1}^{d} k_{j,i} \cdot y^{j-1}\right)$.

Correctness and Security. To see why PRF output is defined as above. Observe that $\mathbf{k_i} := \mathbf{w_i} + \Delta \cdot \mathbf{z_i} = \mathbf{v_i} + \Delta \cdot \mathbf{c_i}$. Then, we have

$$\Delta \cdot y^d + \sum_{j=1}^{d} k_{j,i} \cdot y^{j-1} = \Delta \cdot y^d + \sum_{j=1}^{d} (v_{j,i} + \Delta \cdot c_{j,i}) \cdot y^{j-1}$$

$$= \Delta \cdot (y^d + \sum_{j=1}^{d} c_{j,i} \cdot y^{j-1}) + \sum_{j=1}^{d} v_{j,i} \cdot y^{j-1} = \Delta \cdot P_{\mathbf{x_i}}(y) + \sum_{j=1}^{d} v_{j,i} \cdot y^{j-1}$$

so if $y \in \mathbf{x_i}$ then $P_{\mathbf{x_i}}(y) = 0$ which leads to $F(i, y) \in \{F(i, x_i^{(j)})\}_{j \le d}$.

Theorem 1. *The protocol Π_{oprf} (Figure 3) instantiated with random oracles H, H', securely realizes the ideal functionality of $\mathcal{F}_{\mathsf{oprf}}$ (Figure 2) against a malicious setting in the $\mathcal{F}_{\mathsf{svole}}$ hybrid model.*

Proof. **Corrupted sender.** The Sim interacts with the sender as follows:

- Sim emulates $\mathcal{F}_{\mathsf{svole}}$, waits for sender to send $\Delta, \mathbf{w}$.
- Sim samples uniformly $h_r \leftarrow \mathbb{F}_q$ and then sends h_r to $\mathcal{A}$.
- After receiving h_s, Sim samples uniformly vectors $(\mathbf{z_i})_{1 \le i \le N}$ instead of $\mathbf{z_i} := \mathbf{c_i} - \mathbf{u_i}$.
- Sim samples uniformly $t_r \leftarrow \mathbb{F}_q$ and programs $\mathsf{H}'(t_r) := h_r$.
- On behalf of receiver, Sim sends $\mathbf{z_i}$ and t_r to $\mathcal{A}$.
- Sim computes $\mathbf{k_i} = \mathbf{w_i} + \Delta \mathbf{z_i}$. Whenever $\mathcal{A}$ queries $\mathsf{H}(i|t|y, q)$ where $q = \Delta \cdot y^d + \sum_{j=1}^{d} k_{j,i} \cdot y^{j-1}$ and $\mathsf{H}(i|t|y, q)$ has not been previously queried, Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ with (i, y) being the input of sender, Sim samples uniformly a value as $F(i, y)$ and programs

$$\mathsf{H}\left(i|t|y, \Delta \cdot y^d + \sum_{j=1}^{d} k_{j,i} \cdot y^{j-1}\right) := F(i, y)$$

This simulation is indistinguishable from the real world by the following hybrids:

- *Hybrid 0.* The same as the real protocol with a honest receiver and $\mathcal{F}_{\mathsf{svole}}$ is executed honestly.

PARAMETERS:

- Given $\mathbb{F}_p \subseteq \mathbb{F}_q$ where $\mathbb{F}_q \approx O(2^\kappa)$, $\mathsf{H} : \{0,1\}^* \times \mathbb{F}_q \to \{0,1\}^v$ and $\mathsf{H}' : \mathbb{F}_q \to \mathbb{F}_q$ are random oracles.
- The sender has no input and the receiver inputs a set $X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\} \subseteq \mathbb{F}_p$ where $\mathbf{x_i} = (x_i^{(1)}, \cdots, x_i^{(d)})$ and $n = Nd$.

PROTOCOL:

1. The sender and the receiver invoke to the $\mathcal{F}_{\mathsf{svole}}$ of dimension n in the $\mathbb{F}_q$ over the $\mathbb{F}_p$ with the inverse role. The receiver gets two random vectors $\mathbf{u} \in \mathbb{F}_p^n, \mathbf{v} \in \mathbb{F}_q^n$ and the sender receives $\Delta \in \mathbb{F}_q$, $\mathbf{w} := \Delta\mathbf{u} + \mathbf{v} \in \mathbb{F}_q^n$. Denoting $\mathbf{u} = (\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_N})$ where $\mathbf{u_i} = (c_{j,i})_{1 \le j \le d}$. This denotation is the same for $\mathbf{v}, \mathbf{w}$.
2. The receiver samples $t_r \leftarrow \mathbb{F}_q$ and sends $h_r := \mathsf{H}'(t_r)$ to the sender.
3. The sender samples $t_s \leftarrow \mathbb{F}_q$ and sends $h_s := \mathsf{H}'(t_s)$ to the receiver.
4. The receiver determines the associated polynomial for each $\mathbf{x_i}$ as

$$P_{\mathbf{x_i}}(X) = \prod_{j=1}^{d}(X - x_i^{(j)}) = X^d + \sum_{j=1}^{d} c_{j,i} \cdot X^{j-1}$$

   where $c_{j,i} \in \mathbb{F}_p$ for $i \in [1, N]$, $j \in [1, d]$.
5. Denoting $\mathbf{c_i} := (c_{j,i})_{1 \le j \le d}$; $\mathbf{c} := (\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_N})$, the receiver computes $\mathbf{z_i} := \mathbf{c_i} - \mathbf{u_i} \in \mathbb{F}_p^d$, and then sends $\mathbf{z_i}$ and t_r to the sender.
6. The sender aborts if $\mathsf{H}'(t_r) \neq h_r$.
7. The sender sends t_s to the receiver, the receiver aborts if $\mathsf{H}'(t_s) \neq h_s$ and both parties define $t = t_s \oplus t_r$.
8. The receiver outputs the PRF values on the input $x \in \mathbf{x_i}$ for $i \in [1, N]$ as

$$F(i, x) = \mathsf{H}\left(i|t|x\ ,\ \sum_{j=1}^{d} v_{j,i} \cdot x^{j-1}\right)$$

9. For $i \in [1, N]$, the sender defines $\mathbf{k_i} = \mathbf{w_i} + \Delta\mathbf{z_i}$. For any input $(i, y) \in [1, N] \times \mathbb{F}_p$, the sender computes the PRF output by below formula

$$F(i, y) = \mathsf{H}\left(i|t|y\ ,\ \Delta \cdot y^d + \sum_{j=1}^{d} k_{j,i} \cdot y^{j-1}\right)$$

Fig. 3. Our batch BaRK-OPRF Π_{oprf} based on subVOLE

- *Hybrid 1.* The same as hybrid 0 except the Sim emulates $\mathcal{F}_{\mathsf{svole}}$, receives $\Delta, \mathbf{w}$ from the $\mathcal{A}$.
- *Hybrid 2.* On behalf of receiver, Sim samples $h_r \leftarrow \mathbb{F}_q$ and sends it to $\mathcal{A}$ instead of $h_r := \mathsf{H}'(t_r)$ where t_r is sampled in $\mathbb{F}_q$. Then before sending the value t_r to $\mathcal{A}$, Sim samples $t_r \leftarrow \mathbb{F}_q$ and programs $\mathsf{H}'(t_r) := h_r$. The probability of abort when $\mathsf{H}'(t_r)$ has been queried previous is negligible $O(1/\mathbb{F}_q) = O(2^{-\kappa})$ since t_r is sampled uniformly. This hybrid has an identical distribution.
- *Hybrid 3.* Sim samples uniformly vectors $(\mathbf{z_i})_{1 \le i \le N}$ from $\mathbb{F}_p$ as opposed to $\mathbf{z_i} := \mathbf{c_i} - \mathbf{u_i}$. Since $\mathbf{u_i}$ is distributed uniformly at random and $\mathbf{c_i}$ is arbitrary vector in $\mathbb{F}_p$ then this hybrid is indistinguishable from the previous hybrid.
- *Hybrid 4.* After receiving t_s from $\mathcal{A}$. Sim computes $\mathbf{k_i} = \mathbf{w_i} + \Delta\mathbf{z_i}$ and $t := t_r \oplus t_s$. Whenever $\mathcal{A}$ queries $\mathsf{H}(i|t|y, q)$ where $q = \Delta \cdot y^d + \sum_{j=1}^{d} k_{j,i} \cdot y^{j-1}$ and $\mathsf{H}(i|t|y, q)$ has not been previously queried, Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ with (i, y) being the input of sender, Sim samples uniformly a value as $F(i, y)$ and programs

$$\mathsf{H}\left(i|t|y, \Delta \cdot y^d + \sum_{j=1}^{d} k_{j,i} \cdot y^{j-1}\right) := F(i, y)$$

  Sim will abort if there exists (i, y) such that $\mathsf{H}(i|t|y, q)$ has been previously queried. Since t_s being an arbitrary value fixed by $\mathcal{A}$ from the beginning and t_r is uniformly sampled before sending then t have an uniformly distribution over $\mathbb{F}_q \approx O(2^\kappa)$ which leads to a negligible probability of abort.

Corrupted receiver. The Sim interacts with the receiver as follows:

- Sim emulates $\mathcal{F}_{\mathsf{svole}}$ functionality, waits for the receiver to send $\mathbf{u}, \mathbf{v}$.

- Sim samples uniformly $h_s \leftarrow \mathbb{F}_q$ and then sends h_s to $\mathcal{A}$.
- After corrupted receiver sends $(\mathbf{z_i})_{1\leq i\leq N}$, Sim defines $\mathbf{c_i} := \mathbf{z_i} + \mathbf{u_i}$. From $\mathbf{c_i}$, Sim extracts the set $X = \{\mathbf{x_i}\}_{i\leq N}$.
- After receiving t_r from $\mathcal{A}$, Sim samples uniformly $t \in \mathbb{F}_q$, defines $t_s := t - t_r$ and programs $\mathsf{H}'(t_s) := h_s$ then sends t_s to $\mathcal{A}$.
- Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ with X being the input of receiver then Sim samples a sequence of uniformly random values which defined as $\{F(i,x) \mid \forall x \in \mathbf{x_i}\}_{i\leq N}$. Sim programs

$$\mathsf{H}\left(i|t|x\ ,\ \sum_{j=1}^{d} v_{j,i}\cdot x^{j-1}\right) := F\left(i,x\right)$$

This simulation is indistinguishable from the real world by the following hybrids:

- *Hybrid 0.* The same as the real protocol with a honest sender and $\mathcal{F}_{\mathsf{svole}}$ is executed honestly.
- *Hybrid 1.* The same as hybrid 0 except the Sim emulates $\mathcal{F}_{\mathsf{svole}}$, receiving $\mathbf{u}, \mathbf{v}$ from the $\mathcal{A}$.
- *Hybrid 2.* On behalf of sender, Sim samples $h_s \leftarrow \mathbb{F}_q$ and sends it to $\mathcal{A}$ instead of $h_s := \mathsf{H}'(t_s)$ where t_s is sampled in $\mathbb{F}_q$. After receiving t_r from $\mathcal{A}$, Sim samples $t \leftarrow \mathbb{F}_q$ and defines $t_s := t \oplus t_r$. Sim programs $\mathsf{H}'(t_s) := h_s$ then sends t_s to $\mathcal{A}$. The probability of abort when $\mathsf{H}'(t_s)$ has been queried previous is negligible $O(1/\mathbb{F}_q) = O(2^{-\kappa})$ since t is sampled uniformly and t_r is an arbitrary value. This hybrid has an identical distribution.
- *Hybrid 3.* After receiving $(\mathbf{z_i})_{i\leq N}$ from $\mathcal{A}$, Sim computes $\mathbf{c_i} = \mathbf{z_i} + \mathbf{u_i}$ for all $i \in [1, N]$.
  Now for each $i \in [1, N]$, Sim defines the polynomial $P_{\mathbf{x_i}}(X) = X^d + \sum_{j=1}^{d} c_{j,i}\cdot X^{j-1}$ and then extracts a set $\mathbf{x_i}$ which includes all the root of $P_{\mathbf{x_i}}$. Formally,

$$\mathbf{x_i} = \{x \in \mathbb{F}_p \mid P_{\mathbf{x_i}}(x) = 0\}$$

  This hybrid can not be distinguished with the previous since Sim only extracts set.
- *Hybrid 4.* Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ with $X = \{\mathbf{x_i}\}_{i\leq N}$ being the input of receiver then Sim samples a sequence of uniformly random values over $\{0,1\}^v$ which defined as $\{F(i,x) \mid \forall x \in \mathbf{x_i}\}_{i\leq N}$.
  For each $x \in \mathbf{x_i}$, Sim aborts if any $\mathsf{H}(i|t|x,q)$ where $q := \sum_{j=1}^{d} v_{j,i}\cdot x^{j-1}$ has been made by $\mathcal{A}$. The probability of aborting is negligible since t is uniformly distributed over $\mathbb{F}_q \approx O(2^{\kappa})$. Sim programs

$$\mathsf{H}\left(i||t||x\ ,\ \sum_{k=1}^{d} v_{(i-1)d+k}\cdot x^{k-1}\right) := F\left(i,x\right)$$

  Since the $F(i,x)$ is sampled uniform then this hybrid is indistinguishable with previous one.
- *Hybrid 5.* Sim will abort protocol if corrupted receiver is able to learn the PRF value on a element which is not in any set $\mathbf{x_i}$ for $i \in [1, N]$. This means that the $\mathcal{A}$ made a query $\mathsf{H}\left(i|t|x,h\right)$ such that $x \in \mathbb{F}_p \setminus \mathbf{x_i}$ for $i \in [1, N]$ and $h = \Delta\cdot y^d + \sum_{j=1}^{d} k_{j,i}\cdot x^{j-1}$. Observe that

$$\Delta\cdot y^d + \sum_{j=1}^{d} k_{j,i}\cdot x^{j-1} = \Delta\cdot P_{\mathbf{x_i}}(x) + v_{j,i}\cdot x^{j-1}$$

  Since Δ is distributed uniformly over $\mathbb{F}_q$ in the viewpoint of $\mathcal{A}$ and $P_{\mathbf{x_i}}(x) \neq 0$ for $x \notin \mathbf{x_i}$ then the probability of aborting is negligible at most $O(1/2^q)$. This concludes the proof.

Note that the output v of H is chosen depending on the concrete structure of PSI and the target setting (semi-honest or malicious). This parameter is detailed in the section 4.2 for a semi-honest setting and the section 4.3 for a malicious setting.

4.2 A new semi-honest PSI from mOPRF

A variant of BaRK-OPRF. We now propose a variant of our BaRK-OPRF to deal with the case when the size of each tuple input is not necessarily equal to d. This means that the receiver now can divide the input set to N tuples $\mathbf{x_i}$ and each tuple has less than or equal to d items. Meanwhile, the sender is not allowed to learn about how many exactly items are in each tuple. This functionality can be obtained from our BaRK-OPRF plus a small extra cost, i.e, a *subfield* VOLE of length N over the subfield $\mathbb{F}_2$.

The idea is as follows. The receiver's input set $X = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\} \subseteq \mathbb{F}_p$ where $\mathbf{x_i} = (x_i^{(1)}, \cdots, x_i^{(j_i)})$, $j_i \leq d$. The polynomial associated to $\{\mathbf{x_i}\}_{i \leq N}$ will be expressed as a polynomial of degree d: $P_{\mathbf{x_i}}(X) = \prod_{j=1}^{j_i}(X - x_i^{(j)}) = \sum_{j=1}^{d+1} c_{j,i} \cdot X^{j-1}$ where $c_{j,i} \in \mathbb{F}_p$.

As a result, the set of the coefficients of $P_{\mathbf{x_i}}(X) = (c_{1,i}, c_{2,i}, \ldots, c_{d+1,i}\}$. We remark that, compared to the associated polynomial in our original BaRK-OPRF which has a constant coefficient of degree d of 1, in our variant version this coefficient will equal 0 or 1 since the degree of $P_{\mathbf{x_i}}(X)$ is *less* than or equal to d. So, it requires $(d+1)$ masks for this polynomial instead of d, but the mask for the coefficient of degree d only needs to be in $\mathbb{F}_2$. For each tuple, we require an additional value $u_i \in \mathbb{F}_2$, so in total we need an additional subfield VOLE of length N over the subfield $\mathbb{F}_2$.

More formally, the sender and receiver invoke a subfield VOLE of length n over the subfield $\mathbb{F}_p$ as before (all the notations in figure 3 are reused), and additionally invoke another subfield VOLE instance over the subfield $\mathbb{F}_2$ of length N with an inverse role, while the receiver gets $\mathbf{u'} \in \mathbb{F}_2^N$, and $\mathbf{v'} \in \mathbb{F}_q^N$ the sender holds $\Delta \in \mathbb{F}_q$ (Δ is the same for each time invoking *subfield* VOLE) and $\mathbf{w'} := \Delta \cdot \mathbf{u'} + \mathbf{v'}$. The receiver sends to the sender vectors $\mathbf{z_i}$ as before, and an extra vector $\mathbf{z'}$ defined as $z'_i := c_{d+1,i} - u'_i$ for $i \in [1, N]$. The receiver outputs on input $x \in \mathbf{x_i}$ are computed as $F(i, x) = \mathsf{H}(i|t|x\ ,\ v'_i \cdot x^d + \sum_{j=1}^{d} v_{j,i} \cdot x^{j-1})$. On the other hand, the sender defines their PRF values on input (i, y) where $i \in [1, N]$, $y \in \mathbb{F}_p$ as $F(i, y) = \mathsf{H}(i|t|y\ ,\ (w'_i + \Delta z'_i) \cdot y^d + \sum_{j=1}^{d} k_{j,i} \cdot y^{j-1})$.

Main construction of a new PSI. The sender and the receiver have two input sets $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$. Assume that all of these elements have the bit-length ℓ. Intuitively, our BaRK-OPRF is constructed from $\mathsf{subVOLE}$ to handle the case when having multiple items per bin. Then this specialized BaRK-OPRF can combine with some hashing techniques to form an efficient PSI protocol. In the next part 4.2, we discuss these types of hashing. Our PSI protocol is described in Figure 4; it builds upon the protocol of [KKRT16] using GCH and BaRK-OPRF. For simplicity, we describe our protocol directly with generalized Cuckoo hashing; adapting the protocol to other variants is immediate. We elaborate on our protocol below. In our protocol, the receiver first uses (d, k)-Cuckoo hashing to map his input set Y to a table with N bins, note that the bit-length of the values stored in a bin is $\ell - \log N$ insted of ℓ. Depending on the size of n, we use one of two approaches to handle the bins which are not full (the threshold was chosen empirically to optimize communication).

- If $n \geq 2^{20}$, the variant of our BaRK-OPRF (using an additional subfield VOLE over $\mathbb{F}_2$) is used; for such sizes, the concrete cost of implementing the additional sVOLE vanishes.
- Otherwise, when $n < 2^{20}$, the receiver adds dummy items to bins such that each bin contains *exactly* d items. To avoid collisions between the dummy items and the elements in the same bin of the sender, we pad an extra bit to all items in the following way: $i|x|b$ where i is the index of hash function corresponding with the stored value x while $b = 1$ if x is a dummy item added and $b = 0$ otherwise.

In both case, the sender computes $k \cdot n$ PRF evaluations and sends (shuffled) to the receiver, who compares them with his OPRF outputs, and outputs the intersection set. To reduce the computational cost in this step, the sender can send separately each set H_i ($i \in [1, k]$) which contains the PRF outputs of each $x \in X$ with the related bin $h_i(x)$. Then for each element, the receiver only needs to search for one set (among k sets H_i) of n items instead of $k \cdot n$.

Alternative hashing methods. There are two hashing schemes that can be fit into our PSI structure.

2-choice hashing [PRTY19] is a variant of Cuckoo hashing where one item x is assigned to one of two bins $h_1(x)$ or $h_2(x)$. However, there is no restriction on the number of items per bin and an item is put in a bin which already has fewer items. [PRTY19] proposes both theoretical references and heuristic parameters for 2-choice hashing, which require only a small number of dummy items. Let us assume we have n items and 2 hash functions; using 2-choice hashing allows to map n items to N bins in time $O(n \log n)$ where each bin contains at most $L = \lceil n/N \rceil + 1$ items with a probability $1 - O(1/N)^{L-1}$.

Simple hashing uses one hash function h to map an item x to bin $h(x)$. For security, the number of items per bin can leak some information then it requires padding each bin with dummy items until having an equal number of items per bin. With very high probability, for $N = O(n \log n)$ bins, the maximum possible items per bin is $O(\log n)$. The percentage of the occupation of dummy items is higher than others. However, simple hashing avoids ambiguities about where an item can be placed, a property which is crucial in the malicious setting.

PARAMETERS:

- The sender and the receiver have respectively input sets $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$, all elements of bit-length ℓ.
- A (d, k)-generalized Cuckoo hashing (GCH) scheme mapping n items to N bins by k hash functions $h_1, h_2, \ldots, h_k : \{0,1\}^* \to [N]$ where $Nd > n$ and $d = O(1)$ (see Section 4.2).

PROTOCOL:

1. The receiver uses (d, k)-Cuckoo hashing with k hash functions to map the elements in Y to the table $\mathcal{B}$ consisting of N bins, where each bin i has $j_i \leq d$ items.
   Denote $y_{j,i}$ is an element in Y assigned to position j of bin i and its stored value in table $\mathcal{B}$ is $y'_{j,i}$.
2. Depending on the size of n, there are two alternatives:
   (a) $n \geq 2^{20}$, the sender and receiver invoke our variant of Π_{oprf} where the receiver uses the input set $Y_{\mathcal{B}} = \{\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_N}\}$ defined as follows:
      - $\mathbf{y_i} = \{r_{1,i}, r_{2,i}, \ldots, r_{j_i,i}\}$.
      - $r_{j,i} = t \parallel y'_{j,i}$ where t is index of a hash function such that $h_t(y_{j,i}) = i$.
   (b) $n < 2^{20}$, the sender and receiver directly invoke the Π_{oprf} where the receiver uses the input set $Y_{\mathcal{B}} = \{\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_N}\}$ defined as follows:
      - $\mathbf{y_i} = \{r_{1,i}, r_{2,i}, \ldots, r_{d,i}\}$.
        - For $j \leq j_i$: $r_{j,i} = t \parallel y'_{j,i} \parallel 1$ where t is index of hash function such that $h_t(y_{j,i}) = i$.
        - Otherwise, $r_{j,i} = t \parallel$ dummy value $\parallel 0$ where $t \leftarrow_r [1, k]$.
3. The receiver obtains n instances OPRF:
$$Y' = \{\mathsf{PRF}(i\ ,\ r_{i,j}) \mid i \in [1, N]\ ,\ j \leq j_i\}$$
4. The sender uses the k hash functions to map the n element in X to the N bins. Let x_t denote the value stored at bin $h_t(x)$ when mapping x for $t \in [1, k]$.
5. The sender computes the sets of $k \cdot n$ PRF outputs:
   (a) For $n \geq 2^{20}$: $H_t = \{\mathsf{PRF}(h_t(x)\ ,\ t \parallel x_t) \mid x \in X\}$ for $t \in [1, k]$.
   (b) For $n < 2^{20}$: $H_t = \{\mathsf{PRF}(h_t(x)\ ,\ t \parallel x_t \parallel 1) \mid x \in X\}$ for $t \in [1, k]$.
   Then the sender randomly permutes and sends each set to the receiver.
6. The receiver finds the intersection:
   - if $y \in Y$ is mapped to the position j of bin i by function h_t then check whether $\mathsf{PRF}(i, r_{i,j}) \in H_t$ ($r_{i,j}$ is defined depending on n).
   - Outputs the intersection set.

Fig. 4. Our new semi-honest PSI protocol from BaRK-OPRF

Parameters. In this section, we discuss concrete parameters used in our new PSI semi-honest protocol. We use $\kappa = 128$ and $\lambda = 40$. The protocol contains several parameters:

The length of OPRF output. The output domain of PRF would be $\{0,1\}^v$ where $v = \lambda + 2\log_2(n)$ guarantees a $2^{-\lambda}$ bound on the collision probability of PRF outputs among the two size-n sets. Furthermore, communicating the hashes can be reduced to communicating only $\approx \lambda + \log n$ bits per hash, using a heuristic technique of [TLP+17] that directly leads to an optimization of our PSI protocol.

The size of $\mathbb{F}_p$ and $\mathbb{F}_q$ in BaRK-OPRF. After using permutation-based hashing, each element is mapped to a bin with a stored value in this bin, the bit-length reduces from ℓ to $\ell - \log N$. The input set of BaRK-OPRF in PSI protocol constructs from stored values concatenating with some extra bits. Then the bit-length of an input element of BaRK-OPRF is computed as $\ell - \log N + 1$ if $n \geq 2^{20}$ or $\ell - \log N + 2$ otherwise, i.e, the size of $q = 2^{\ell - \log N + 1}$ or $q = 2^{\ell - \log N + 2}$ respectively.

Generalized Cuckoo hashing. We use a (d, k)-general cuckoo hashing scheme without stash. The parameters are chosen such that the failure probability is $2^{-\lambda}$. When $d = 1, k = 3$ these parameters are identical with KKRT except for the number of bins increases slightly to $N = 1.3n$ which is a trade-off to obtain no stash. Even with the higher number of bins, our PSI protocol significantly outperforms KKRT.
To minimize the overall communication, we set $k = 2$ to reduce the cost of sending $k \cdot n$ PRF outputs. We used a Python script to simulate randomly assigning n values to $N = c \cdot n$ bins using $(d, 2)$-Cuckoo hashing, for several values of d and c, and for $n = 2^9, 2^{10}, 2^{11}, 2^{12}$. For a value of c as low as 0.65, we never observed any insertion failure over 10^7 trials for each values of n (for $n = 2^{12}$, we could only do 10^6 trials), when using $d = 3$ items per bins. For $d = 2$, the failure probability became

noticeable already for $c \approx 1$. Based on known theoretical analysis of (d, k)-Cuckoo hashing, the failure probability is known to scale inverse polynomially with n. Therefore, we expect that for reasonably large values of n (e.g. $n \geq 2^{18}$), our parameters should guarantee a failure probability significantly below 2^{-40}.

2-choice hashing. Following the analysis of [PRTY19], we set the number N of bins to $n/3$, and the maximum load $d = L + 1$ to 4. This guarantees a failure probability which we empirically estimate to be $1/N^{L-1}$, which is below 2^{-40} for all values of n above 2^{14}.

Simple hashing. Eventually, for simple hashing, we set arbitrarily the number of bins N to $n/10$, and derive the corresponding value of d from Figure 5 in [RR17]. We note that the parameters for simple hashing are much less heuristic that the other two, in that concrete bound can actually be achieved which are relatively close to the heuristic (computer-estimated) bounds. For example, [PRTY19] experimentally observes that for a 2^{-40} failure probability, setting $d = 47$ suffices when using $N = n/10$ bins. Using a standard Chernoff bound, it is in fact straightforward to prove formally that $d = 49$ already suffices to reach this failure probability, which is very close to the experimental bound. In contrast, experimental bounds in more complex hashing variants are typically much more distant from provable bounds. The choice of $N = n/10$ is entirely arbitrary: any smaller N leads to better communication, but requires using higher values of d, leading to worse computation (due to the need to perform N polynomial interpolations with degree-d polynomial). This allows for a smooth tradeoff between communication and computation, where better computational power can be used to further reduce the communication. At the extreme end of the spectrum, using $N = 1$ and $d = n$ requires one expensive degree-n polynomial interpolation, but can achieve extremely low communications, e.g. $93n$ bits of communication for $\ell = 32$ and $n = 2^{20}$.

Efficiency. We compare the communication of our protocols, using three hashing methods, on Table 1. Regarding computation, we provide a breakdown of the computation costs of our protocols in Appendix C of supplementary material. Briefly, though, compared to the protocol of [RS21], and when using a standard choice of parameters for our protocol (e.g. $n = 2^{20}$, and using generalized Cuckoo hashing with $d = 3$ and $N = 0.65n$), our protocol requires essentially a length-$1.9n$ VOLE (with a small subfield), $0.65n$ degree-3 polynomial interpolations (roughly $3n$ multiplications over a small field), and computing n hashes. In contrast, the enhanced version of [RS21] (using the OKVS of [GPR+21] and the VOLE of [CRR21]) will require solving a linear system to set up an OKVS (this requires on the order of $(1.3 \log n + \lambda)^3$ multiplications over $\mathbb{F}_{2^{128}}$, plus $O(\lambda n)$ operations), computing a length-$1.3n$ VOLE (over $\mathbb{F}_{2^{128}}$), and computing $2n$ hashes. The cost of the VOLE dominates that of performing n hashes, so for sufficiently large set sizes ($n \gg 2^{20}$), the protocol of [RS21] should become roughly 30% more efficient than our protocol computation-wise. For smaller sets (e.g. $n \approx 2^{16}$), the cost of setting up the OKVS becomes more significant, requiring around $20n$ field multiplications over $\mathbb{F}_{2^{128}}$, hence the computational efficiency of our protocol becomes roughly on par with that of [RS21]. Of course, real runtimes can vary due to e.g. cache misses, so these estimations should only be viewed as a first order approximation indicating that the computational efficiency of our protocols is close to that of [RS21] (but likely slightly larger).

In terms of computation, the main computational overhead comes from performing N polynomial interpolations of *only degree-d* polynomials. Based on our analysis, to achieve $2^{-\lambda} = 2^{-40}$ probability of insertion failure, the following parameters can be chosen:

- $N = 0.65n$ and $d = 3$ for generalized Cuckoo hashing (GCH),
- $N = 0.33n$ and $d = 4$ for two-choice hashing,
- $N = n/10$ and $d \approx 46$ for simple hashing.

As the above illustrates, the cost of performing N polynomial interpolations will be very small for GCH, two-choice hashing, but becomes higher for simple hashing (though performing $n/10$ degree-46 interpolations remains reasonably fast).

4.3 A malicious PSI from mOPRF

In this section, we propose a maliciously secure PSI protocol based on our BaRK-OPRF (section 4.1) and simple hashing combining a permutation-based hash function. The PSI protocol is shown in figure 5 and its security against a corrupted adversary is proven in theorem 2. The estimated overhead communication cost of this PSI is $Nd(\ell - \log N) + (\kappa + \log n)n + o(n)$. Observe that the PSI protocol in section 4.2 is insecure against malicious settings since the general hashing scheme does not allow the simulation in ideal world. To handle this we use simple hashing schemes with only

one permutation-based hash function. This protocol is constructed from the natural approach used recently in [PRTY19, PRTY20, CM20, RS21], i.e, Alice (a sender) and Bob (a receiver) invoke the $\mathcal{F}_{\mathsf{oprf}}$ then Bob gets the PRF values on his input and Alice enables to compute the PRF on any input so Alice computes on her input after that she sends these PRF values to Bob; Bob compares and outputs the intersection.

PARAMETERS:

- Alice (sender) and Bob (receiver) have respectively input set $X = \{x_1, x_2, \ldots, x_n\} \in \mathbb{F}_p$ and $Y = \{y_1, y_2, \ldots, y_n\} \in \mathbb{F}_p$, all elements of bit-length ℓ.
- A random hash functions $h : \{0,1\}^* \to [N]$.
- A Permutation-based hashing $\mathsf{Per}_{h,X}$ maps a set X to table $\mathcal{B}_X$ consisting of N bins such that each bin has d slots where $Nd > |X|$, and $d = O(1)$. Denote $\mathsf{Per}(x) := (i, x')$ where x' is the stored value of x in bin i which defined by h and x then $\mathsf{Per}^{-1}(i, x') = x$.

PROTOCOL:

1. Bob uses Per to map Y to $\mathcal{B}_Y$, for each empty slot in each bin $\mathcal{B}_Y[i]$, put here a dummy item of length $\ell - \log N$.
2. Alice sends $(\mathsf{sender}, \mathsf{id})$ and Bob sends $(\mathsf{receiver}, \mathsf{id}, \mathcal{B}_Y)$ to $\mathcal{F}_{\mathsf{oprf}}$ then
   - Bob receives the $Y' = \{F(i, y') \mid y' \in \mathcal{B}_Y[i]\}_{i \le N}$.
3. For each $x \in X$, Alice queries x to $\mathcal{F}_{\mathsf{oprf}}$ with corresponding input (i, x') such that $\mathsf{Per}(x) = (i, x')$, then Alice gets $F(i, x')$. Alice sends to Bob
$$U = \{F(i, x') \mid x \in X \wedge \mathsf{Per}(x) = (i, x')\}$$
4. Now for each $y \in Y$, $\mathsf{Per}(y) = (i, y')$, if $F(i, y') \in U$ then Bob outputs y as an element in the intersection.

Fig. 5. Our malicious PSI protocol based on $\mathcal{F}_{\mathsf{oprf}}$

Intuitively, in a malicious setting, when the sender is corrupted, the simulation needs to extract the sender's input set X from the queries to $\mathcal{F}_{\mathsf{oprf}}$ and the set U. Denote $F(y) := F(i, y')$ where $\mathsf{Per}(y) = (i, y')$ and the set of all elements queried to $\mathcal{F}_{\mathsf{oprf}}$ is X' where $n' = |X'|$. The extraction procedure is that $X = \{x \in \mathbb{F}_p \mid x \in X' \wedge F(x) \in U\}$. Observe that if there exist two distinct elements x_1, $x_2 \in X'$ such that $F(x_1) = F(x_2) \in U$ then more than one element is extracted to X. The probability of existing collision is $2^{-v+2\log n'}$ then one approach to avoid collision is choosing $v = 2\kappa$. However, when $v = 2\kappa$, the overhead communication cost significantly increases.
Therefore, another approach is that Sim only extracts elements $x \in X'$ if its PRF is distinct and appears in U, i.e, $x \in X'$ such that $F(x) \in U$ and $\nexists x' \in X'$ where $F(x) = F(x')$. [RS21] proposed this simulation and claimed that if the output domain of PRF $v = \kappa$ then this simulation is correct and can not be distinguishable from the real protocol. We point out the proof of [RS21] has a gap and show that the output of PRF should be $\kappa + \log n$.

Indeed, if there exist some $x_1, x_2 \in X'$ such that $F(x_1) = F(x_2)$ then Sim only needs to extract x_1, x_2 when one of them is in Y. Let assume $x_1 \in Y$, the probability of $F(x_2) = F(y)$ for some $y \in Y$ is $2^{-v+\log(n_Y)}$ since Y is first fixed before the function F is sampled. [RS21] shows $n_Y = O(\kappa)$ then the security can hold if $v = \kappa$. However, this should be $v = \kappa + \log n_Y$ since $n_Y = O(\mathrm{poly}(\kappa))$ instead of $O(\kappa)$. In particular, PSI protocols in [RS21] are targeted on large input set because of the usage of vector OLE.

Theorem 2. *The PSI protocol on Figure 5 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{psi}}$ (figure 8) over the field $\mathbb{F}_p$ for set size n and malicious set size $n_X = n$, $n_Y = Nd$ with statistical security against malicious adversaries in $\mathcal{F}_{\mathsf{oprf}}$ hybrid model.*

Proof. **Alice is corrupted.** Sim interacts with Alice as below:

- Sim emulates $\mathcal{F}_{\mathsf{oprf}}$, extracts the set X' containing all elements queried to $\mathcal{F}_{\mathsf{oprf}}$. Then Sim defines a set
$$X^* = \{x \in X' \mid \nexists x' \in X', x \neq x' : F(x) = F(x')\}$$
- From the set U, Sim defines
$$X := \{x \in X^* \mid F(x) \in U\}$$
then inputs X to $\mathcal{F}_{\mathsf{psi}}$ and obtains the set $X \cap Y$.

The simulation is indistinguishable from the real protocol by following hybrids:

- *Hybrid 0.* The same as real protocol. Bob is honest with his input Y and $\mathcal{F}_{\mathsf{oprf}}$ is executed perfectly.
- *Hybrid 1.* Sim emulates the functionality $\mathcal{F}_{\mathsf{oprf}}$, receiving the queries (i, x') from $\mathcal{A}$. For each query (i, x'), Sim determines an element x such that $\mathsf{Per}(x) = (i, x')$ then let X' be the set containing of all such elements. Sim defines

$$X^* = \{x \in X' \mid \nexists x' \in X, x \neq x' : F(x) = F(x')\}$$

- *Hybrid 2.* After $\mathcal{A}$ sends the set U, on behalf of Bob, Sim gets U then defines the set

$$X := \{x \in X^* \mid F(x) \in U\}$$

  Sim will abort if there exist two distinct values $x_1, x_2 \in X'$ such that $F(x_1) = F(x_2)$ and one of them being in Y. Since Y is first fixed and then the function F is sampled then w.l.o.g assume $x_1 \in Y$, the probability of $F(x_2) = F(y)$ for some $y \in Y$ is $n_Y/2^v$ which is negligible when $v = \kappa + \log(n_Y)$.
  Observe that X^* can have more than n elements but $|X|$ is always less than n since $|U| = n$ and X^* contains only elements with distinct PRF values.
- *Hybrid 3.* Sim inputs X to $\mathcal{F}_{\mathsf{psi}}$ and obtains the set $X \cap Y$. Sim outputs it as the output of a honest Bob.

Bob is corrupted. Sim interacts with Bob as below:

- Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ functionality. Sim gets $\mathcal{B}_Y$ being the input set of receiver to $\mathcal{F}_{\mathsf{oprf}}$ and then Sim samples a uniformly random sequence $Z = \{Z_{i,y'}\}_{i \leq N}$ as the output PRF value of the set $\mathcal{B}_Y$, where $Z_{i,y'}$ corresponds to $y' \in \mathcal{B}_Y[i]$.
- From $\mathcal{B}_Y$, Sim extracts Y, inputs Y being the receiver's input to $\mathcal{F}_{\mathsf{psi}}$, receiving $I = X \cap Y$.
- Sim sends to Bob a set U containing of
  - The set of $|I|$ values: $\{Z_{i,y'} | y \in I \wedge \mathsf{Per}(y) = (i, y')\}$.
  - $n - |I|$ uniformly random values in $\{0,1\}^v \setminus Z$.

  The simulation can not distinguish from the real protocol by following hybrids:

- *Hybrid 0.* The same as real protocol except Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ functionality. Sim gets a set $\mathcal{B}_Y$ as the input set of Bob to $\mathcal{F}_{\mathsf{oprf}}$ and then Sim samples a uniformly random sequence $Z = \{Z_{i,y'}\}_{i \leq N}$ as the output PRF value of the set $\mathcal{B}_Y$, where $Z_{i,y'} \in \{0,1\}^v$ corresponds to $y' \in \mathcal{B}_Y[i]$.
- *Hybrid 1.* Sim computes the set

$$Y = \{\mathsf{Per}^{-1}(i, y') \mid \forall y' \in \mathcal{B}_Y[i], i \in [1, N]\}$$

  Note that $|Y| \leq Nd$ since corrupted Bob is only allowed to input up to Nd elements to $\mathcal{F}_{\mathsf{oprf}}$. Sim inputs Y to $\mathcal{F}_{\mathsf{psi}}$ on behalf of receiver, receiving $I = X \cap Y$.
- *Hybrid 2.* Sim sends to Bob the set U of n elements consisting of
  - The set of $|I|$ values: $\{Z_{i,y'} | y \in I \wedge \mathsf{Per}(y) = (i, y')\}$.
  - $n - |I|$ uniformly random values in $\{0,1\}^v \setminus Z$.

  The simulation can not be distinguishable from the real protocol since the output of F is pseudorandom over $\{0,1\}^v$ while $\{0,1\}^v \setminus Z = O(2^\kappa)$. This concludes the proof.

In general, the malicious PSI (figure 5) has a communication cost that depends on the security parameter κ and is dominated by κn. We now present a new PSI protocol that is secure in malicious setting via a dual execution while its communication cost only depends on the statistic parameter λ and the set size n. The idea of using a dual execution has been used in [RR17] but when combining this with our BaRK-OPRF it achieves efficient results, i.e, the total communication cost is only $2Nd(\ell - \log N) + n(\lambda + \log n) + o(n)$. The detailed construction of dual PSI is shown in the appendix B of supplementary material.

5 A standard PSI from subfield-ring OLE

In this section, we describe a new PSI protocol, which builds upon a (simple variant of) a pseudorandom correlation generator for the ring-OLE correlation [BCG+20b]. Our protocol enjoys a number of important features: it is in the *standard model*, achieves *malicious security* at essentially no cost, has *low communication* (competitive even with the best maliciously secure PSI protocols in the random oracle model), and reasonable computation (albeit superlinear in n). Our protocol can also be generalized to a powerful notion of *batch non-interactive PSI*, where (after a small logarithmic-cost preprocessing step with each server) a client can broadcast a single encoding of his database, and then obtain the intersection with any of the server databases at any time after receiving a single message from this server. We believe that this functionality itself is of independent interest.

5.1 Semi-Honest Batch Non-Interactive PSI from Subfield Ring-OLE

We describe a new PSI scheme in the semi-honest model. Our protocol enjoys two interesting features: (1) it is in the standard model, and (2) it is a *batch non-interactive* protocol, a useful communication pattern which we describe afterwards. The full construction is represented on Figure 5.1.

Theorem 3. *The PSI protocol on Figure 5.1 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{psi}}$ (Figure 8) over the field $\mathbb{F}_{\mathsf{p}}$ with set size n and malicious set size $n' = n_X = n_Y = 2n$, with statistical security against augmented semi-honest adversaries in the $\mathcal{F}_{\mathsf{sole}}$ hybrid model.*

Proof. We first show that the protocol is correct with probability at least $1 - n^2/q = 1 - 2^{-\lambda}$ using $q = \lambda + 2\log n$. It follows from the description of the protocol that $u = p_A b_0 + p_B b'_0$, where b_0, b'_0 are uniformly random degree-n polynomials over $\mathcal{R}_q$. Then $x \in A \cap B$ implies $p_A(x) = 0 \ \wedge \ p_B(x) = 0$, which implies $u(x) = 0$. In the other direction, it holds by Lemma 8 that the probability that p_A and b'_0 share a common root (i.e. $\gcd(p_A, b'_0) \neq 1$) is at most n^2/q. Then,

$$\begin{aligned} x \in I \quad &\implies \quad p_A(x) = 0 \wedge U(x) = 0 \\ &\implies \quad p_A(x) = 0 \wedge p_B(x) \cdot b'_0(x) = u(x) - p_A(x) \cdot b_0(x) = 0 \\ &\implies \quad p_A(x) = 0 \wedge p_B(x) = 0 \text{ (since } \gcd(p_A, r_B^0) = 1 \text{ and } b'_0 \neq 0 \text{ w.h.p)}, \end{aligned}$$

hence $I \subseteq A \cap B$, which concludes the proof.

We now turn our attention to security. We use the following fact: given any set S, denoting by $p_S \in \mathcal{R}_p$ the polynomial whose set of roots is S, it holds that

$$u = p_A \cdot b_0 + p_B \cdot b'_0 = p_{A \cap B} \cdot \left(p_{A \setminus B} \cdot b_0 + p_{B \setminus A} \cdot b'_0\right),$$

where $p_{A \setminus B}, p_{B \setminus A} \in \mathcal{R}_p^2$ are two polynomials of degree at most n and $\gcd(p_{A \setminus B}, p_{B \setminus A}) = 1$.

Alice is corrupted. We describe a simulator Sim which emulates Bob:

- (Setting up the correlation) Sim emulates the functionality $\mathcal{F}_{\mathsf{sole}}$: when Alice queries $\mathcal{F}_{\mathsf{sole}}$, Sim samples and sends two random polynomials $(a, s_A) \leftarrow_r \mathcal{R}_p \times \mathcal{R}_q$.
- (Client set encoding) upon receiving t_A, Sim defines $p_{\tilde{A}} = a - t_A$ and obtains the roots $\tilde{A}$ of $p_{\tilde{A}}$ (note that Bob is 'augmented semi-honest', so $p_{\tilde{A}}$ is well-formed, but $\tilde{A}$ might differ from A). Sim queries the PSI functionality $\mathcal{F}_{\mathsf{psi}}$ on behalf of Alice with input set $\tilde{A}$ and obtains $\tilde{A} \cap B$.
- (Server-to-client message) Sim generates a random degree-n polynomial $b_1 \in \mathcal{R}_q$ and picks $v \leftarrow_r \mathcal{R}_q$. Sim sets $u \leftarrow p_{\tilde{A} \cap B} \cdot v$ and $t_B \leftarrow u + p_{\tilde{A}} b_1 \cdot X^n - s_A$. Sim sends (b_1, t_B) to Alice.

We prove that the simulated protocol is indistinguishable from an honest execution through a sequence of hybrids: in the first game, Sim simulates $\mathcal{F}_{\mathsf{sole}}$ honestly, and behaves as a honest Bob (using p_B) otherwise. Sim also extracts $p_{\tilde{A}}$ from $t_A = a - p_{\tilde{A}}$ and queries A to the PSI functionality on behalf of Alice in the ideal world, obtaining $\tilde{A} \cap B$. This game is perfectly indistinguishable from an honest execution of the protocol. Then, in the second game, Sim computes t_B as $p_{\tilde{A} \cap B} \cdot v + p_{\tilde{A}} b_1 \cdot X^n - s_A$, where v, b_1 are uniformly random degree-n polynomials over $\mathcal{R}_q$.

It remains to show that the second game is indistinguishable from the first game. Recall that $u = p_{\tilde{A} \cap B} \cdot \left(p_{\tilde{A} \setminus B} \cdot b_0 + p_{B \setminus \tilde{A}} \cdot b'_0\right)$ when the parties play honestly, but Alice uses input $\tilde{A}$. From the viewpoint of Alice, both b_0 and b'_0 are distributed as uniformly random degree-n polynomials over $\mathcal{R}_q$. By Lemma 9, this implies that $p_{\tilde{A} \setminus B} \cdot b_0 + p_{B \setminus \tilde{A}} \cdot b'_0$ is distributed as a uniformly random degree-n polynomial $v \in \mathcal{R}_q$. By construction, $u = t_B - p_{\tilde{A}} b_1 \cdot X^n + s_A$, hence t_B is distributed as $p_{\tilde{A} \cap B} \cdot v + p_{\tilde{A}} b_1 \cdot X^n - s_A$ where v is a random degree-n polynomial over $\mathcal{R}_q$. This concludes the proof.

Bob is corrupted. We describe a simulator Sim which emulates Bob:

- (Setting up the correlation) Sim emulates the functionality $\mathcal{F}_{\mathsf{sole}}$: when Bob queries $\mathcal{F}_{\mathsf{sole}}$, Sim samples and sends two random polynomials $(b, s_B) \leftarrow_r \mathcal{R}_q^2$.
- (Client set encoding) Sim sends a uniformly random polynomial $t_A \leftarrow_r \mathcal{R}_q$.
- (Server-to-client message) Upon receiving (b_1, t_B), Sim computes $w \leftarrow t_B - s_B + t_A b$ and defines $\bar{B}$ to be the set of roots of w (since Bob is augmented semi-honest, $w = p_B b'_0$ has at least n roots). Sim queries $\bar{B}$ to $\mathcal{F}_{\mathsf{psi}}$ on behalf of Bob in the ideal world, and gets $\bar{B} \cap A$. Sim outputs $\bar{I} \leftarrow \bar{B} \cap A$.

t_A is distributed exactly as in the honest protocol by construction. It remains to show that Sim's simulated output $\bar{I}$ is the same as the honest output $\tilde{I} = A \cap \tilde{B}$ with overwhelming probability, where $\tilde{B}$ is the input used by Bob when computing $w = p_{\tilde{B}} b'_0$ (which can differ from Bob's real input B). This follows from Lemma 8: since Bob is (augmented) semi-honest, $w = p_{\tilde{B}} b'_0$ where b'_0 is a uniformly random degree-n polynomial, hence by Lemma 8, $\gcd(p_A, b'_0) = 1$ with probability at least $1 - n^2/q$. Therefore, with probability at least $1 - n^2/q = 1 - 2^{-\lambda}$ (using $q = \lambda + 2\log n$), it holds that $\bar{I} = A \cap \bar{B} = A \cap (\tilde{B} \cup \mathsf{roots}(b'_0)) = A \cap \tilde{B} = \tilde{I}$. This concludes the proof.

Above, "augmented semi-honest security" refers to a strengthening of honest-but-curious corruption where the adversary is allowed to change the corrupted parties' inputs. This is a standard strengthening of semi-honest security, which has been argued to better capture real-world security [HL10]. It will also facilitate upgrading security to the malicious setting later on.

Batch non-interactivity. To securely realize the functionality $\mathcal{F}_{\mathsf{sole}}$, we rely on the PCG-based protocol of [BCG+20b] (using a straightforward adaptation to the subfield setting), which is secure under the ring-LPN assumption. Interestingly, instantiating the subfield ring OLE this way allows to import a powerful feature of the PCG of [BCG+20b], which is its *programmability*: when generating a ring-OLE correlation, the receiver can ensure that her output a remains *identical* across multiple instances of the protocol with different parties.

This feature enables the following communication structure: after a short (logarithmic-communication) interaction with N servers, a client, playing the role of Alice with input set A, can broadcast a single compact encoding of her dataset to all the servers (with input sets $B_1 \cdots B_N$). Afterwards, each server B_i can at any time send a single message m_i to Alice, from which she can recover $A \cap B_i$ without further interaction. To our knowledge, this batch non-interactive communication pattern was never achieved by any prior proposal; we believe that it can make our protocol appealing in realistic scenarios.

More concretely, after a logarithmic-communication preprocessing phase where Alice sets up PCG seeds with each of servers, Alice broadcasts the value $t_A = a - p_A$ to everyone, which communicates $2n \log p \approx 2\ell n$ bits. This message can be seen as a compact public encoding of her dataset (it is only twice as large as Alice's set). Afterwards, each server can complete the protocol of Figure 5.1 by sending a single message (b_1, t_B) to the receiver, of length $3n \log q \approx 3(\lambda + 2\log n)n$, from which the receiver can locally recover $X \cap X_i$. Furthermore, using the encoding technique of [TLP+17], the $\lambda + 2\log n$ term can be reduced to $\lambda + \log n$ (the improvement is based on the observation that for an appropriate ordering, n random elements of a set of size $2^{\lambda + 2\log n}$ are on average at distance $2^{\lambda + \log n}$ for each other, hence the cost of transmitting them can be reduced to essentially $\lambda + \log n$ per element by sending the distance between consecutive elements instead).

Efficiency. The communication cost of protocol (Figure 5.1) is $n \cdot (2\log p + 3\log q) + o(n)$ bits of communication. Here, the size of the subfield $\mathbb{F}_p$ depends only on the bitsize ℓ of the items in the sets A and B, hence we can set $\log p = \ell$. As we will see in the analysis, $\log q$ must be set to $\log q \approx \lambda + 2\log n$ to guarantee λ bits of statistical security. This leads to a total communication of $n \cdot (2\ell + 3\lambda + 6\log n) + o(n)$ bits, which is reduced to $n \cdot (2\ell + 3\lambda + 3\log n) + o(n)$ with the encoding of [TLP+17]. The $o(n)$ term above captures the cost of distributing the PCG seeds of the subfield ring-OLE (we discuss the concrete value of $o(n)$ later on, for our maliciously secure version of the protocol).

Regarding computation, the computational cost scales as $O(n \log^2 n)$ due to the fast polynomial interpolations, or as $O(n \log n)$ when using cyclotomic rings. We provide a concrete analysis of the computational cost of the maliciously secure version of our protocol in Section 5.2.

PARAMETERS:

- Two rings $\mathcal{R}_\mathsf{p} = \mathbb{F}_\mathsf{p}[\mathsf{X}]/\mathsf{F}(\mathsf{X}) \subseteq \mathcal{R}_\mathsf{q} = \mathbb{F}_{\mathsf{p}^\mathsf{r}}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$, where $F(X)$ has degree $2n+1$.
- The sender (Alice) and receiver (Bob) have respective input sets $A = \{a_1, a_2, \ldots, a_n\} \subset \mathbb{F}_p$ and $B = \{b_1, b_2, \ldots, b_n\} \subset \mathbb{F}_p$.
- A subfield ring-OLE in the ring $\mathcal{R}_q$ over the subring $\mathcal{R}_p$.

PROTOCOL:

1. **(Setting up the correlation)** Alice and Bob encode their sets to $p_A = \prod_{i=1}^n (X - a_i)$, $p_B = \prod_{i=1}^n (X - b_i)$ respectively, and invoke $\mathcal{F}_\mathsf{sole}$ to generate a subfield ring-OLE correlation over $\mathcal{R}_p, \mathcal{R}_q$: Alice receives $(a, s_A) \in \mathcal{R}_p \times \mathcal{R}_q$ and Bob receives $(b, s_B) \in \mathcal{R}_q^2$ such that $s_A + s_B = ab$.
2. **(Broadcasting the client set encoding)** Alice computes and sends $t_A = a - p_A$ to Bob.
3. **(Server-to-client message)** Bob sets $s'_B \leftarrow s_B - t_A b$. Then, Bob decomposes b as $b = b_0 + b_1 \cdot X^n$ (where b_0, b_1 are degree-n polynomials), sets $s'_B \leftarrow s_B - t_A b$, and picks a random degree-n polynomial b'_0 over $\mathcal{R}_q$. He sends b_1 and $t_B \leftarrow s'_B + p_B b'_0$ to Alice.
4. **(Output)** Alice sets $u \leftarrow t_B - p_A b_1 \cdot X^n + s_A$; note that $u = p_A b_0 + p_B b'_0$. Alice outputs the set $I = \{x \in A \mid u(x) = 0\}$.

Fig. 6. Augmented semi-honest PSI protocol based on ring-OLE

5.2 Maliciously Secure PSI in the Standard Model

In this section, we upgrade the security of our protocol to the malicious setting. Our upgrade introduces only a minimal communication overhead to the protocol, independent of the set sizes n. The full protocol is represented on Figure 5.2.

Theorem 4. *The PSI protocol on Figure 5.2 securely realizes the ideal functionality $\mathcal{F}_\mathsf{psi}$ (Figure 8) over the field $\mathbb{F}_\mathsf{p}$ with set size n and malicious set size $n' = n_X = n_Y = 2n$, with statistical security against malicious adversaries in the $\mathcal{F}_\mathsf{sole}$-hybrid model.*

Proof. We first consider the case where Alice is corrupted. The simulator Sim behaves as follows:

- He waits for the adversary to send (a, s_A) to $\mathcal{F}_\mathsf{sole}$ and receives t_A from Alice. Sim defines $p_A \leftarrow a - t_A$ and computes $\tilde{A} = \{x \in \mathbb{F}_{\mathsf{p}'} \mid p_A(\mathsf{map}(x)) = 0\}$, and inputs $\tilde{A}$ to the PSI functionality on behalf of Alice, receiving $\tilde{A} \cap B$.
- He picks a uniformly random degree-n polynomial b_1 over $\mathcal{R}_q$ and sends b_1 to Alice.
- Upon receiving y from Alice, Sim computes $s'_A \leftarrow s_A - p_A b_1 \cdot X^n$. Then, if either (1) $p_A(0) \neq 1$ or (2) $y \neq s'_A(0)$, Sim aborts on behalf of Bob. Otherwise, Sim simulates t_B as in the augmented semi-honest model, picking $v \leftarrow_r \mathcal{R}_q$ and setting $t_B \leftarrow p_{\tilde{A} \cap B} \cdot v + p_A b_1 \cdot X^n - s_A$. Sim sends t_B to Alice.

The analysis of this simulator is similar to the analysis in the augmented semi-honest model, up to two distinctions: first, the extracted polynomial p_A is not guaranteed to be of degree n anymore – but this corresponds to a malicious adversary using a set of larger size $n' \leq 2n$, which is allowed by the functionality. Second, we must show that Sim correctly emulates the behavior of an honest Bob when deciding to abort based on the checks (1) and (2). We show that the simulation is statistically close to the behavior of an honest Bob. To do so, we consider three cases:

Case 1: $y = s'_A(0)$ and $p_A(0) = 1$. In this case, Sim does not abort. We show that an honest Bob would not abort either:

$$\begin{aligned} s'_A(0) + s'_B(0) &= s_A(0) + s_B(0) - (p_A(0)(b_1 \cdot X^n)(0) + t_A(0)b(0)) \\ &= s_A(0) + s_B(0) - (p_A(0)(b_1 \cdot X^n)(0) + (a(0) - p_A(0))b(0)) \\ &= a(0)b(0) - (p_A(0)(b_1 \cdot X^n)(0) + (a(0) - p_A(0))b(0)) \\ &= -(b_1 \cdot X^n)(0) + b(0) \\ &= b_0(0), \end{aligned}$$

hence Bob does not abort.

Case 2: $p_A(0) \neq 1$. In this case, check (1) of Sim causes an abort. We show that when this is the case, Bob would abort as well with high probability, *i.e.*, that $y \neq b_0(0) - s'_B(0)$. Indeed,

$$b_0(0) - s'_B(0) = b_0(0) - s_B(0) + t_A(0)b(0)$$

$$\begin{aligned}
&= b_0(0) - a(0)b(0) + s_A(0) + t_A(0)b(0) \\
&= b_0(0) - a(0)b(0) + s_A(0) - p_A(0)b(0) + a(0)b(0) \\
&= b_0(0) \cdot (1 - p_A(0)) + s_A(0),
\end{aligned}$$

Hence if Alice manages to send $y = b_0(0) - s'_B(0)$, it implies that $(y - s_A(0)) \cdot (1 - p_A(0))^{-1} = b_0(0)$. The left hand side is a value known to Alice, but the right hand side is the constant coefficient of b_0, which is a uniformly random independent element of $\mathbb{F}_{\mathsf{q}}$. The probability that Alice does not cause an abort is therefore bounded by $1/q < 1/2^\lambda$, hence Bob aborts with probability at least $1 - 1/2^\lambda$.

Case 3: $p_A(0) = 1$ but $y \neq s'_A(0)$. In this case, check (2) of Sim causes an abort. As we saw in Case 1, it holds that $s'_A(0) = b_0(0) - s'_B(0)$, hence $y \neq b_0(0) - s'_B(0)$, hence Bob necessarily aborts.

Overall, Sim's emulation of Bob in the p_A check phase is $1/2^\lambda$-close to the honest game, which concludes the proof.

We now turn our attention to the case where Bob is corrupted. The simulator Sim behaves as follows:

- He waits for the adversary to send (b, s_B) to $\mathcal{F}_{\mathsf{sole}}$ and sends a uniformly random $t_A \leftarrow_r \mathcal{R}_p$ on behalf of Alice.
- Upon receiving b_1 from Bob, Sim defines $b_0 \leftarrow b - (b_1 \cdot X^n)$ (note that b_0 might not be a degree-n polynomial) and sets $s'_B \leftarrow s_B - t_A b$. He sends $y \leftarrow b_0(0) - s'_B(0)$ to Bob.
- Upon receiving t_B from Bob, Sim computes $w \leftarrow t_B - s_B + t_A b$ and defines the set $\bar{B} = \{x \in \mathbb{F}_{\mathsf{p}'} \mid w(\mathsf{map}(x)) = 0\}$. Sim queries $\bar{B}$ to $\mathcal{F}_{\mathsf{psi}}$ on behalf of Bob in the ideal world, and gets $\bar{B} \cap A$. Sim checks whether $t_B(1) - s'_B(1) \neq 0$. It outputs $\bar{I} \leftarrow \bar{B} \cap A$ if this holds, and aborts otherwise.

The analysis is identical to that of the augmented semi-honest model (note that $|\bar{B}| \leq 2n$ by construction), up to Sim's check that $t_B(1) - s'_B(1) \neq 0$. We show that with overwhelming probability, Sim aborts if and only if the honest Bob aborts at this stage. Recall that Bob aborts if $u(1) = 0$. Then

$$\begin{aligned}
u(1) &= t_B(1) - p_A(1)(b_1 \cdot X^n)(1) + s_A(1) \\
&= t_B(1) + s_A(1) \text{ since } p_A(1) = 0 \\
&= t_B(1) + a(1)b(1) - s_B(1) \\
&= t_B(1) + a(1)b(1) - s'_B(1) - t_A(1)b(1) \\
&= t_B(1) + p_A(1)b(1) - s'_B(1) \\
&= t_B(1) - s'_B(1) \text{ since } p_A(1) = 0,
\end{aligned}$$

hence Sim aborts iff Bob aborts. This concludes the proof.

Efficiency. Our malicious protocol has minimal communication overhead over our augmented semi-honest protocol. The main overhead stems from starting from a slightly larger field in which two elements can be "reserved elements". If p' is a prime power and $\ell \approx \log \mathsf{p}'$, the price to pay is therefore increasing ℓ to $\log \mathsf{p}$ where p is the smallest prime power above $\mathsf{p}' + 2$. While an exact expression would be rather tedious, for any reasonable input size this cost should be negligible (the simplest strategy is to pick $\mathsf{p}' = 2^\ell$ and $\mathsf{p} = 2^{\ell+1}$, in which case ℓ is increased by one bit, but much better encoding methods exist). Therefore, the communication remains $n \cdot (2\ell + 3\lambda + 6\log n) + o(n)$ bits, or $n \cdot (2\ell + 3\lambda + 3\log n) + o(n)$ with the encoding of [TLP+17]. We provide a more concrete analysis of the $o(n)$ term (setting up the ring-OLE) in the malicious setting in the appendix A.5.

Computation cost. Note that our standard model protocol shares with our other protocols the feature of having a communication independent of κ. Our protocol requires more computation compared to the best ROM-based protocols, due to its use of polynomial interpolation. However, it still allows for very fast PSI computation (we estimate a few seconds to compute the intersection between databases of size 2^{20}, on one core of a standard laptop). Concretely, the protocol requires only

- a single degree-n polynomial interpolation, one FFT over a polynomial ring with degree-$2n$ polynomials, and 3 multiplications of degree-n polynomials for the receiver, and
- a single degree-n polynomial interpolation, one FFT as above, 2 multiplications of degree-n polynomials, and a single n-multipoint polynomial evaluation for the sender.

PARAMETERS:

- A field $\mathbb{F}_{\mathsf{p}'}$ and two rings $\mathcal{R}_{\mathsf{p}} = \mathbb{F}_{\mathsf{p}}[\mathsf{X}]/\mathsf{F}(\mathsf{X}) \subseteq \mathcal{R}_{\mathsf{q}} = \mathbb{F}_{\mathsf{p}^t}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$, where $F(X)$ has degree $2n+1$ and $|\mathbb{F}_{\mathsf{p}'}| \leq |\mathbb{F}_{\mathsf{p}}| - 2$. map is an efficient (and efficiently invertible) injective mapping, with $\mathsf{map}(\mathbb{F}_{\mathsf{p}'}) \subseteq \mathbb{F}_{\mathsf{p}} \setminus \{0, 1\}$.
- The sender (Alice) and receiver (Bob) have respective input sets $A = \{a_1, a_2, \ldots, a_n\} \subset \mathbb{F}_{\mathsf{p}'}$ and $B = \{b_1, b_2, \ldots, b_n\} \subset \mathbb{F}_{\mathsf{p}'}$.
- A subfield ring-OLE in the ring $\mathcal{R}_q$ over the subring $\mathcal{R}_p$.

PROTOCOL:

1. **(Setting up the correlation)** Alice and Bob encode their sets to $p_A = c \cdot (X-1) \cdot \prod_{i=1}^{n}(X - \mathsf{map}(a_i))$ with $c = -(\prod_{i=1}^{n}(-\mathsf{map}(a_i)))^{-1}$ (note that this guarantees $p_A(0) = 1$ and $p_A(1) = 0$) and $p_B = \prod_{i=1}^{n}(X - \mathsf{map}(b_i))$ respectively. Alice and Bob invoke $\mathcal{F}_{\mathsf{sole}}$ to generate a subfield ring-OLE correlation over $\mathcal{R}_p, \mathcal{R}_q$: Alice receives $(a, s_A) \in \mathcal{R}_p \times \mathcal{R}_q$ and Bob receives $(b, s_B) \in \mathcal{R}_q^2$ such that $s_A + s_B = ab$.
2. **(Broadcasting the client set encoding)** Alice computes and sends $t_A = a - p_A$ to Bob.
3. **(Server-to-client message)** Bob sets $s'_B \leftarrow s_B - t_A b$. Then, Bob decomposes b as $b = b_0 + b_1 \cdot X^n$ (where b_0, b_1 are degree-n polynomials), and sets $s'_B \leftarrow s_B - t_A b$. He sends b_1 to Alice.
4. **(Checking p_A)** Alice computes $s'_A \leftarrow s_A - p_A b_1 \cdot X^n$. Alice sends $y \leftarrow s'_A(0)$ to Bob. If $y \neq b_0(0) - s'_B(0)$, Bob aborts. Else, Bob picks a random degree-n polynomial b'_0 over $\mathcal{R}_q$ and sends $t_B \leftarrow s'_B + p_B b'_0$ to Alice.
5. **(Output)** Alice sets $u \leftarrow t_B - p_A b_1 \cdot X^n + s_A$; note that $u = p_A b_0 + p_B b'_0$. If $u(1) = 0$, Alice aborts; otherwise, Alice computes the set $I = \{x \in A \mid u(\mathsf{map}(x)) = 0\}$ and outputs I.

Fig. 7. Maliciously secure PSI protocol in the $\mathcal{F}_{\mathsf{sole}}$-hybrid model

Furthermore, both polynomial interpolations only have to be performed over a field $\mathbb{F}$, of size $|\mathbb{F}| \approx 2^{\ell}$ where ℓ is the bit size of the set items (e.g. 32 or 64 bits), and the multipoint evaluation is over a field of size $\lambda + 2\log n$ bits. This stands in stark contrasts with previous state of the art protocols [PRTY19] that relied on polynomial interpolation (*on top* of using the ROM), where the interpolations and multipoint evaluations had to be performed over a very large field $\mathbb{F}$ of size $|\mathbb{F}| \approx 2^{400}$. By using a cyclotomic ring, the FFTs and polynomial multiplications are much faster than the interpolations. On Table 1.1, we compare our protocol to the current fastest maliciously secure PSI protocols [PRTY20, RT21b, RS21].

On the attacks of [AMZ21]. We note that constructing maliciously secure PSI protocols using an algebraic approach, along the lines of our protocol, is known to be non-trivial and error prone. Indeed, previous works [GN19] used a similar approach based on polynomial manipulation, OLEs, and the lemmas 8, 9, to build a malicious PSI protocol. However, their protocol was found to be insecure in a recent preprint, which described powerful concrete attacks on this proposal [AMZ21]. Intuitively, the key technical difficulties revolve in both cases around how to handle null polynomials ($p_A = 0$ or $p_B = 0$). In our specific context, it turns out that our direct use of ring-OLE enables relatively elegant and simple (in hindsight) strategies to enforce nonzero polynomials. Our modification has almost no impact on the communication or the computation of our protocol, essentially giving us malicious security for free (though we note that we still require an additional round of communication). It is not, however, completely clear how to adapt our strategy to the setting of OLE-based algebraic PSI in [GN19]. We believe that this provides further support for the intuition that ring-OLE is the right primitive to build PSI protocols using this algebraic approach (beyond its direct advantage in terms of communication efficiency).

References

ADI+17. B. Applebaum, I. Damgård, Y. Ishai, M. Nielsen, and L. Zichron. Secure arithmetic computation with constant computational overhead. In *CRYPTO 2017, Part I*, *LNCS* 10401, pages 223–254. Springer, Heidelberg, August 2017.

AMZ21. A. Abadi, S. J. Murdoch, and T. Zacharias. Polynomial representation is tricky: Maliciously secure private set intersection revisited. Cryptology ePrint Archive, Report 2021/1009, 2021. `https://ia.cr/2021/1009`.

BCG+17. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, and M. Orrù. Homomorphic secret sharing: Optimizations and applications. In *ACM CCS 2017*, pages 2105–2122. ACM Press, October / November 2017.

BCG+19a. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

BCG+19b. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO 2019, Part III*, *LNCS* 11694, pages 489–518. Springer, Heidelberg, August 2019.

BCG+20a. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Correlated pseudorandom functions from variable-density LPN. In *61st FOCS*, pages 1069–1080. IEEE Computer Society Press, November 2020.

BCG+20b. E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators from ring-LPN. In *CRYPTO 2020, Part II*, *LNCS* 12171, pages 387–416. Springer, Heidelberg, August 2020.

BCGI18. E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai. Compressing vector OLE. In *ACM CCS 2018*, pages 896–912. ACM Press, October 2018.

BMR20. S. Badrinarayanan, P. Miao, and P. Rindal. Multi-party threshold private set intersection with sublinear communication. Cryptology ePrint Archive, Report 2020/600, 2020. `https://eprint.iacr.org/2020/600`.

CM20. M. Chase and P. Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In *CRYPTO 2020, Part III*, *LNCS* 12172, pages 34–63. Springer, Heidelberg, August 2020.

CRR21. G. Couteau, P. Rindal, and S. Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. LNCS, pages 502–534. Springer, Heidelberg, 2021.

DW07. M. Dietzfelbinger and C. Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theoretical Computer Science*, 380(1-2):47–68, 2007.

GN19. S. Ghosh and T. Nilges. An algebraic approach to maliciously secure private set intersection. In *EUROCRYPT 2019, Part III*, *LNCS* 11478, pages 154–185. Springer, Heidelberg, May 2019.

GPR+21. G. Garimella, B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. Oblivious key-value stores and amplification for private set intersection. LNCS, pages 395–425. Springer, Heidelberg, 2021.

GS19. S. Ghosh and M. Simkin. The communication complexity of threshold private set intersection. In *CRYPTO 2019, Part II*, *LNCS* 11693, pages 3–29. Springer, Heidelberg, August 2019.

HKL+12. S. Heyse, E. Kiltz, V. Lyubashevsky, C. Paar, and K. Pietrzak. Lapin: An efficient authentication protocol based on ring-LPN. In *FSE 2012*, *LNCS* 7549, pages 346–365. Springer, Heidelberg, March 2012.

HL10. C. Hazay and Y. Lindell. A note on the relation between the definitions of security for semi-honest and malicious adversaries. Cryptology ePrint Archive, Report 2010/551, 2010. `https://eprint.iacr.org/2010/551`.

IKNP03. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO 2003*, *LNCS* 2729, pages 145–161. Springer, Heidelberg, August 2003.

KKRT16. V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *ACM CCS 2016*, pages 818–829. ACM Press, October 2016.

KRTW19. V. Kolesnikov, M. Rosulek, N. Trieu, and X. Wang. Scalable private set union from symmetric-key techniques. In *ASIACRYPT 2019, Part II*, *LNCS* 11922, pages 636–666. Springer, Heidelberg, December 2019.

KS05. L. Kissner and D. X. Song. Privacy-preserving set operations. In *CRYPTO 2005*, *LNCS* 3621, pages 241–257. Springer, Heidelberg, August 2005.

LP15. H. Lipmaa and K. Pavlyk. Analysis and implementation of an efficient ring-LPN based commitment scheme. In *CANS 15*, LNCS, pages 160–175. Springer, Heidelberg, December 2015.

MB72. R. Moenck and A. Borodin. Fast modular transforms via division. In *Proceedings of the 13th Annual Symposium on Switching and Automata Theory (Swat 1972)*, SWAT '72, page 90–96, USA, 1972. IEEE Computer Society.

PRTY19. B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. SpOT-light: Lightweight private set intersection from sparse OT extension. In *CRYPTO 2019, Part III*, *LNCS* 11694, pages 401–431. Springer, Heidelberg, August 2019.

PRTY20. B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. PSI from PaXoS: Fast, malicious private set intersection. In *EUROCRYPT 2020, Part II*, *LNCS* 12106, pages 739–767. Springer, Heidelberg, May 2020.

PSSZ15. B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security 2015*, pages 515–530. USENIX Association, August 2015.

PSWW18. B. Pinkas, T. Schneider, C. Weinert, and U. Wieder. Efficient circuit-based PSI via cuckoo hashing. In *EUROCRYPT 2018, Part III*, *LNCS* 10822, pages 125–157. Springer, Heidelberg, April / May 2018.

PSZ14. B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on OT extension. In *USENIX Security 2014*, pages 797–812. USENIX Association, August 2014.

PSZ18. B. Pinkas, T. Schneider, and M. Zohner. Scalable private set intersection based on ot extension. *ACM Transactions on Privacy and Security (TOPS)*, 21(2):1–35, 2018.

RR17. P. Rindal and M. Rosulek. Malicious-secure private set intersection via dual execution. In *ACM CCS 2017*, pages 1229–1242. ACM Press, October / November 2017.

RR22. P. Rindal and S. Raghuraman. Blazing fast PSI from improved OKVS and subfield VOLE. *IACR Cryptol. ePrint Arch.*, page 320, 2022.

RS21. P. Rindal and P. Schoppmann. VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. LNCS, pages 901–930. Springer, Heidelberg, 2021.

RT21a. M. Rosulek and N. Trieu. Compact and malicious private set intersection for small sets. Cryptology ePrint Archive, Report 2021/1159, 2021. `https://eprint.iacr.org/2021/1159`.

RT21b. M. Rosulek and N. Trieu. Compact and malicious private set intersection for small sets. Cryptology ePrint Archive, Report 2021/1159, 2021. `https://ia.cr/2021/1159`.

SGRR19. P. Schoppmann, A. Gascón, L. Reichert, and M. Raykova. Distributed vector-OLE: Improved constructions and implementation. In *ACM CCS 2019*, pages 1055–1072. ACM Press, November 2019.

TLP+17. S. Tamrakar, J. Liu, A. Paverd, J.-E. Ekberg, B. Pinkas, and N. Asokan. The circle game: Scalable private membership test using trusted hardware. In *ASIACCS 17*, pages 31–44. ACM Press, April 2017.

W+17. U. Wieder et al. Hashing, load balancing and multiple choice. *Foundations and Trends® in Theoretical Computer Science*, 12(3–4):275–379, 2017.

WYKW21. C. Weng, K. Yang, J. Katz, and X. Wang. Wolverine: fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1074–1091. IEEE, 2021.

YWL+20. K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang. Ferret: Fast extension for correlated OT with small communication. In *ACM CCS 20*, pages 1607–1626. ACM Press, November 2020.

Supplementary Material

A Additional Preliminaries

In this appendix, we provide further preliminaries, including necessary preliminaries for our second construction, described in Section 5.

A.1 Ideal Functionalities and Security Model

The ideal functionality of PSI in the semi-honest and malicious settings are shown on Figure 8. In the semi-honest setting, the size of a corrupted party's input set is fixed to n, while in the malicious setting, the functionality allows a malicious party to use a possibly larger set of bounded size $n' > n$. Since PSI is a special case of secure computation, the security analysis of a two-party PSI protocol is performed via the standard simulation paradigm, recalled below.

PARAMETERS:

- An arbitrary field $\mathbb{F}, n, n_X, n_Y$ public parameters for honest parties and corrupt parties respectively where $n \leq n_X, n_Y$.
- There are two parties, a sender with a input set $X \subseteq \mathbb{F}$ and a receiver with a input set $Y \subseteq \mathbb{F}$.

FUNCTIONALITY:

- Wait for sender to send the input set X, if $|X| > n_X$ and the sender is malicious or if $|X| \neq n$ and the sender is honest then abort.
- Wait for receiver to send the input set Y. If $|Y| > n_Y$ and the receiver is malicious or if $|Y| \neq n$ and the receiver is honest then abort.
- The functionality outputs the intersection $X \cap Y$ to one party.

Fig. 8. Ideal functionality of PSI $\mathcal{F}_{\mathsf{psi}}$

Semi-honest security. Let $\mathrm{view}_1^{\Pi}(X,Y)$ and $\mathrm{view}_2^{\Pi}(X,Y)$ be the view of P_1 and P_2 in the protocol Π, $\mathsf{out}^{\Pi}(X,Y)$ be the output of P_2 in the protocol, and $f(X,Y)$ be the output of P_2 from the ideal functionality. The protocol Π is semi-honest secure if there exist PPT simulators Sim_1 and Sim_2 such that for all inputs X, Y,

$$\left(\mathrm{view}_1^{\Pi}(X,Y), \mathsf{out}^{\Pi}(X,Y)\right) \approx \left(\mathsf{Sim}_1(1^{\kappa}, X, n), f(X,Y)\right);$$
$$\mathrm{view}_2^{\Pi}(X,Y) \approx \mathsf{Sim}_2\left(1^{\kappa}, Y, n, f(X,Y)\right)$$

Malicious Security. Let f be a two-party functionality and Π be a secure protocol for computing f. The protocol Π is said to be secure against malicious adversary if for all non-uniform PPT adversary $\mathcal{A}$ in the real model, there exists a non-uniform PPT adversary $\mathcal{S}$ in the deal model satisfying:

$$\mathrm{IDEAL}_{(f,\mathcal{S},i)}(X,Y) \approx \mathrm{REAL}_{(\Pi,\mathcal{A},i)}(X,Y)$$

where $i \in \{1,2\}$ index of corrupted party. $\mathrm{IDEAL}_{(f,\mathcal{S},i)}(X,Y)$ is the output pair of the honest party and the adversary $\mathcal{S}$ in the ideal model, $\mathrm{REAL}_{(\Pi,\mathcal{A},i)}(X,Y)$ is defined as the output pair of the honest party and the adversary $\mathcal{A}$ from the real execution of Π.

A.2 Learning Parity with Noise

Our protocols are built on top of pseudorandom correlation generators (PCGs). State of the art constructions of PCGs rely on various flavors of the learning parity with noise (LPN) assumption. Since we make a black-box use of PCGs, our work is essentially oblivious to the underlying assumptions. However, for the sake of completeness, we recall the assumptions which we build upon below. We

note that, for our second contribution, we build upon the PCG of [BCG+20b]. The latter uses a relatively new flavor of the ring-LPN assumption, over a polynomial ring where the polynomial splits completely; however, in this work, we do *not* need this new flavor, and instead rely solely on the (relatively well-established) standard ring-LPN assumption over a polynomial ring with an irreducible polynomial.

We define the LPN assumption over a ring $\mathcal{R}$ with dimension k, number of samples n, w.r.t. a code generation algorithm $\mathbf{C}$, and a noise distribution $\mathcal{D}$:

Definition 5 (Dual LPN). *Let $\mathcal{D}(\mathcal{R}) = \{\mathcal{D}_{k,n}(\mathcal{R})\}_{k,n\in\mathbb{N}}$ denote a family of efficiently sampleable distributions over a ring $\mathcal{R}$, such that for any $k, n \in \mathbb{N}$, $\mathsf{Im}(\mathcal{D}_{k,n}(\mathcal{R})) \subseteq \mathcal{R}^n$. Let $\mathbf{C}$ be a probabilistic code generation algorithm such that $\mathbf{C}(k, n, \mathcal{R})$ outputs a matrix $H \in \mathcal{R}^{k\times n}$. For dimension $k = k(\lambda)$, number of samples (or block length) $n = n(\lambda)$, and ring $\mathcal{R} = \mathcal{R}(\lambda)$, the (dual) $(\mathcal{D}, \mathbf{C}, \mathcal{R})$-$\mathsf{LPN}(k, n)$ assumption states that*

$$\{(H, \mathbf{b}) \mid H \leftarrow_r \mathbf{C}(k, n, \mathcal{R}), \mathbf{e} \leftarrow_r \mathcal{D}_{k,n}(\mathcal{R}), \mathbf{b} \leftarrow H \cdot \mathbf{s}\}$$
$$\stackrel{c}{\approx} \{(H, \mathbf{b}) \mid H \leftarrow_r \mathbf{C}(k, n, \mathcal{R}), \mathbf{b} \leftarrow_r \mathcal{R}^n\}.$$

The dual LPN assumption is also called *syndrome decoding assumption* in the code-based cryptography literature. The dual LPN assumption as written above is equivalent to the *primal* LPN assumption with respect to G (a matrix $G \in \mathcal{R}^{n\times n-k}$ such that $H \cdot G = 0$), which states that $G \cdot \mathbf{s} + \mathbf{e}$ is indistinguishable from random, where $\mathbf{s} \leftarrow_r \mathcal{R}^{n-k}$ and $\mathbf{e} \leftarrow_r \mathcal{D}_{k,n}(\mathcal{R})$; the equivalence follows from the fact that $H(G \cdot \mathbf{s} + \mathbf{e}) = H \cdot \mathbf{e}$.

The standard LPN assumption refers to the case where H is a uniformly random matrix over $\mathbb{F}_2$, and $\mathbf{e}$ is sampled from $\mathsf{Ber}_r(\mathbb{F}_2)$, where r is called the *noise rate*. Other common noise distributions include exact noise (the noise vector $\mathbf{e}$ is a uniformly random weight-rn vector from $\mathbb{F}_2^n$; this is a common choice in concrete LPN-based constructions) and regular noise (the noise vector $\mathbf{e}$ is a concatenation of rn random unit vectors from $\mathbb{F}_2^{1/r}$, widely used in the PCG literature [BCGI18, BCG+19b, BCG+19a]).

Known constructions of subfield-VOLE use various flavors of the dual LPN assumption with regular noise over a finite field. For example, the work of [BCGI18] suggests relying on an LDPC code, while [BCG+19a] uses quasi-cyclic codes, and [CRR21] uses a new family of codes, called Silver codes.

In this section, we recall the Ring-LPN assumption, which was first introduced in [HKL+12] to build efficient authentication protocols. Since then, it has received some attention from the cryptography community [LP15], due to its appealing combination of LPN-like structure, compact parameters, and short running times. Below, we also provide a definition of Module-LPN, which generalizes Ring-LPN in the same way that the more well-known Module-LWE generalizes Ring-LWE.

A.3 Ring-LPN

We now define the Ring-LPN assumption, a variant of the dual LPN assumption over polynomial rings, first introduced in [HKL+12] The assumption has been used in multiple works since. Ring-LPN is the natural "ring analog" of LPN, in the same way that ring-LWE is the ring analog of LWE.

Definition 6 (Ring-LPN). *Let $\mathcal{R} = \mathbb{F}[X]/(F(X))$ for some field $\mathbb{F}$ and degree-N polynomial $F(X) \in \mathbb{Z}[X]$, and let $m, t \in \mathbb{N}$. Let HW_t be the distribution over R_p that is obtained via sampling t noise positions $A \leftarrow [0..N)^t$ as well as t payloads $\mathbf{b} \leftarrow \mathbb{Z}_p^t$ uniformly at random, and outputting $e(X) := \sum_{j=0}^{t-1} \mathbf{b}[j] \cdot X^{A[j]}$. The R-$\mathsf{LPN}_{p,q,t}$ problem is hard if for any PPT adversary $\mathcal{A}$, it holds that*

$$|\Pr[\mathcal{A}((a_i, a_i \cdot s + e_i)_{i=1}^m) = 1] - \Pr[\mathcal{A}((a_i, u_i)_{i=1}^m) = 1]| \leq \mathsf{negl}(\lambda)$$

where the probabilities are taken over the random choices of the values $a_1, \ldots, a_m, u_1, \ldots, u_m \leftarrow \mathcal{R}_p$, $s, e_1, \ldots, e_m \leftarrow \mathsf{HW}_t$ and the randomness of $\mathcal{A}$.

A.4 Pseudorandom Correlation Generators

In our new protocol, we will reuse the KKRT template, but replace the BaRK-OPRF with a subfield-VOLE-based construction. Doing so enables several new optimizations of the scheme, which we describe afterwards. We start by recalling the notion of pseudorandom correlation generator, and that

of subfield vector OLE. A pseudorandom correlation generator is a pair of algorithm $(\mathsf{Gen}, \mathsf{Expand})$, where Gen distributes a pair of small seeds (s_0, s_1), and $\mathsf{Expand}(i, s_i)$ stretches a seed into a long pseudorandom string. Informally, a PCG for a target *correlation* satisfies two properties:

- **Correctness.** Given $(s_0, s_1) \leftarrow_r \mathsf{Gen}(1^\kappa)$, the pair $(x_0, x_1) = ((\mathsf{Expand}(0, s_0), \mathsf{Expand}(1, s_1))$ is indistinguishable from a random sample from the target correlation.
- **Security.** Given s_0, the string $x_1 = \mathsf{Expand}(1, s_1)$ is indistinguishable from a uniformly random string sampled *conditioned on satisfying the right correlation with* $\mathsf{Expand}(0, s_0)$. The converse property holds as well.

In addition, a PCG should satisfy some *shortness* conditions: the seeds (s_0, s_1) should be significantly smaller than a sample from the target correlation.

A subfield vector oblivious linear evaluation generator (sVOLE generator) is a PCG for the following two-party correlation: Alice gets a pair of random vectors $(\mathbf{u}, \mathbf{v})$, and Bob gets a random scalar Δ and the vector $\mathbf{w} = \Delta \cdot \mathbf{u} + \mathbf{v}$, where $\mathbf{v}, \mathbf{w} \in \mathbb{F}^n$ for some field $\mathbb{F}$, $\Delta \in \mathbb{F}$, and $\mathbf{u} \in (\mathbb{F}')^n$, where $\mathbb{F}'$ is a subfield of $\mathbb{F}$. Efficient PCGs for the sVOLE correlation have been designed in [BCG+19b,BCG+19a,CRR21], with seed sizes logarithmic in n, and the latest protocol of [CRR21] achieve extremely impressive efficiency features (around 300ms to generate an sVOLE correlation of length 10^7 on one core of a standard laptop; using this sVOLE to achieve OT extension results in a protocol using 37% less computation and $\sim 1300\times$ less communication than the standard IKNP protocol).

In the paper, we sometimes slightly abuse the notion of PCG, and use formulations such as "Alice and Bob use a PCG protocol". What this means is the following: Alice and Bob use some dedicated two-party computation protocol to distributively and securely generate seeds $(s_0, s_1) \leftarrow_r \mathsf{Gen}(1^\kappa)$, such that Alice gets s_0 and Bob gets s_1; then, Alice and Bob *locally* expand these seeds into a pseudorandom instance of the target correlation. Efficient protocols for distributing PCG seeds have been introduced in [BCG+19a], and typically have communication linear in the seed length – that is, logarithmic in n.

A.5 Subfield Ring-OLE

In a recent work [BCG+20b], a new PCG construction was described for the *ring-OLE* correlation. The ring-OLE correlation over a ring $\mathcal{R}_\mathsf{q}$ is the following correlation: $\{((x_0, z_0), (x_1, z_1)) \mid x_0, x_1, z_0 \leftarrow_r \mathcal{R}_\mathsf{q}, z_1 \leftarrow x_0.x_1 - z_0\}$. The main motivation in [BCG+20b] was that, when the ring is a polynomial ring where the polynomial splits fully into n linear factors, such a correlation can be locally converted into n instances of an OLE correlation over a large field, which is very useful for secure computation of arithmetic circuit. We note that our work will actually directly rely on the ring-OLE correlation over a polynomial ring, and we do not need the polynomial to split. This allows to build the necessary PCG from a much more conservative assumption. We note that the work of [BCG+20b] also describes a maliciously secure protocol to distribute the seed of this PCG which, combined with the PCG, leads to a maliciously secure protocol to instantiate the ideal functionality for malicious ring-OLE correlation.

In this work, we rely on a slight variant of the ring-OLE correlation: given the ring $\mathcal{R}_\mathsf{q} = \mathbb{F}_{\mathsf{p}^\mathsf{t}}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$ for some polynomial $F(X)$, we consider the *subfield* ring-OLE correlation, where x_0 is instead sampled from the ring $\mathcal{R}_\mathsf{p} = \mathbb{F}_\mathsf{p}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$ (that is, the coefficients of x_0 are sampled from the subfield $\mathbb{F}_\mathsf{p}$ instead of the field $\mathbb{F}_\mathsf{q} = \mathbb{F}_{\mathsf{p}^\mathsf{t}}$). We represent the corresponding variant of the ideal functionality on Figure 9. We note that the protocol of [BCG+20b] to instantiate the ring-OLE functionality can be adapted to handle the subfield ring-OLE functionality in a straightforward way.

Theorem 7 ([BCG+20b]). *If the ring-LPN assumption holds over $\mathcal{R} = \mathbb{F}[X]/(F(X))$ where F is a degree-N polynomial, there exists a maliciously secure protocol instantiating the functionality 9 over $\mathcal{R}$, with communication logarithmic in N.*

Cost of ring-OLE. The work of [BCG+20b] provides a concrete formula for computing the cost of ring-OLE. Their protocol assumes a one-time setup procedure, which can be reused indefinitely for any number of ring-OLE setups, similar to the one-time setup of OT extension protocols. As is common in PSI papers, we ignore this one-time cost in our estimations. The total cost of setting up a ring-OLE is computed as $2[ct(\log n/t + \log p) + 34(ct)^2 \log n/t + (ct)^2(2\kappa + 3)\log(2n/t) + 13(ct)^2 \log p]$

PARAMETERS:

- The security parameter κ, the subring $\mathcal{R}_\mathsf{p} = \mathbb{F}_\mathsf{p}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$, the ring $\mathcal{R}_\mathsf{q} = \mathbb{F}_{\mathsf{p}^\mathsf{t}}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$ where $F(x)$ has the degree of n, p is a power prime and t is arbitrary integer.
- There are two parties, a sender and a receiver.

FUNCTIONALITY:

1. If all the parties are honest:
   - Sample uniformly random $x_0 \leftarrow_r \mathcal{R}_p$, $x_1, z_1 \leftarrow_r \mathcal{R}_q$, let $z_0 = x_0 \cdot x_1 + z_1$.
   - Output $(x_0, z_0), (x_1, z_1)$ to the sender and the receiver respectively.
2. One party is corrupt:
   - If the sender is malicious:
     Waiting for the input $x_0 \in \mathcal{R}_p$ and $z_0 \in \mathcal{R}_q$ from the adversary. Sample $x_1 \leftarrow_r \mathcal{R}_q$ and compute $z_1 = x_0 \cdot x_1 - z_0$. Output (x_1, z_1) to the receiver.
   - If the receiver is malicious:
     Waiting for the input $(x_1, z_1) \in \mathcal{R}_\mathsf{p}^2$ from the adversary. Sample $x_0 \leftarrow_r \mathcal{R}_\mathsf{p}$ and compute $z_0 = x_0 \cdot x_1 - z_1$. Output (x_0, z_0) to the sender.

Fig. 9. The ideal functionality $\mathcal{F}_{\mathsf{sole}}$ of a malicious *subfield-ring* OLE of length n in $\mathcal{R}_q$ over the subring $\mathcal{R}_p$

in [BCG+20b], where c, t are parameters related to the underlying ring-LPN assumption and κ is a computational security parameter, which we set to 128. In our setting, using the irreducible ring-LPN assumption and targeting 128 bits of security, we can use e.g. $c = 4$ and $t = 14$ according to the analysis of [BCG+20b].

A.6 Useful Lemmas about Polynomials

Our standard model protocol will rely on encoding the input sets as polynomials: the set $X = \{x_1, x_2, \ldots, x_n\}$ is encoded as the coefficients of $P(X) = \sum_{i=0}^{n}(X - x_i)$. Encoding and decoding can be performed in $O(n^2)$ field operations by Lagrange interpolation and Horner evaluation. For large values of n, the work of [MB72] requires $O(n \log^2 n)$ arithmetic operations where the interpolation and evaluation are reducible to a recursive use of polynomial divisions.

Private set intersection can be reduced to simple arithmetic operations on the polynomial encoding of the sets. This was observed in previous works [KS05,GS19,GN19]. The reduction builds upon simple lemmas, which we state below (we also consider, and prove, a slight generalization of these lemmas).

Lemma 8. *Let $F(X)$ be a degree-n polynomial, $q = p^t$ where p is a prime, $P(x) \in \mathcal{R}_\mathsf{p} = \mathbb{F}_\mathsf{p}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$ be an arbitrary polynomial of degree n and $R(x) \in \mathcal{R}_\mathsf{q} = \mathbb{F}_{\mathsf{p}^\mathsf{t}}[\mathsf{X}]/\mathsf{F}(\mathsf{X})$ be a uniformly random polynomial of degree n. Then*

$$\Pr[\gcd(P(x), R(x)) \neq 1] \leq n^2/q.$$

Proof. $\gcd(P(x), R(x)) = 1$ iff $P(x)$ and $R(x)$ share no common root. A random polynomial over $\mathcal{R}_p$ of degree n has at most n roots, which are distributed uniformly; hence, each root of $R(x)$ is equal to a root of $P(x)$ with probability at most $1 - n/q$. Therefore:

$$\begin{aligned}\Pr[\gcd(P(x), R(x)) \neq 1] &= 1 - \Pr[\gcd(P(x), R(x)) = 1] \\ &= 1 - (1 - n/q)^n \leq n^2/q \text{ (union bound).}\end{aligned}$$

Lemma 9 ([BMR20]). *Given $\mathbb{F}_p$ be a finite field of prime order p. Fix any $p = O(\mathsf{poly}(\lambda))$. Let $P(x), Q(x) \in \mathbb{F}_p[x]$ be two arbitrary polynomials of degrees α_1 and α_2 respectively. Let $R_1(x), R_2(x)$ be two polynomials sampled independently and uniformly at random over $\mathbb{F}_p[x]$, of degrees β_1, and β_2 respectively, where $n = \alpha_1 + \beta_1 = \alpha_2 + \beta_2 \leq \alpha_1 + \alpha_2$. Let $S(x) = P(x) \cdot R_1(x) + Q(x) \cdot R_2(x) \in \mathbb{F}_p$. Then $S(x) = \gcd(P(x), Q(x)) \cdot U(x)$, where $U(x)$ is an uniformly random polynomial of degree at most n over $\mathbb{F}_p[x]$.*

A.7 The KKRT Protocol

In this section, we provide a high level overview of the KKRT protocol [KKRT16]. At a high level, it combines Cuckoo hashing with a *batch related-key oblivious pseudorandom function* (BaRK-OPRF). A BaRK-OPRF is a protocol between a receiver, Alice, and a sender, Bob, where Alice

has inputs $(x_1, \cdots, x_n)$. Bob receives one global key Δ and n local keys $(K_1, \cdots, K_n)$, and Alice learns $(F_{\Delta,K_1}(x_1), \cdots, F_{\Delta,K_n}(x_n))$. In KKRT, the BaRK-OPRF has the following structure: $F_{\Delta,K_i}(x_i) = H(i, K_i \oplus (\Delta \wedge \mathsf{Enc}(x_i)))$, where

- H is a hash function, modeled as a random oracle (or satisfying a tailor-made *Hamming correlation robustness* assumption),
- Enc is the encoding procedure of a suitable good error-correcting code,
- $\Delta \wedge \mathsf{Enc}(x_i)$ denotes the bitwise AND of Δ and $\mathsf{Enc}(x_i)$.

The intuition underlying the security is that for any value $x \neq x_i$, $\mathsf{Enc}(x)$ and $\mathsf{Enc}(x_i)$ differ on many positions (because the code has high minimum distance), hence $K_i \oplus \Delta \wedge \mathsf{Enc}(x)$ contains a lot of entropy, even given $K_i \oplus \Delta \wedge \mathsf{Enc}(x_i)$. Under a suitable assumption on the hash function, $H(K_i \oplus \Delta \wedge \mathsf{Enc}(x))$ therefore looks random even given $F_{\Delta,K_i}(x_i)$, for any $x \neq x_i$. Note that to turn this intuition into a proof, the authors of KKRT introduced a formal notion of Hamming correlation robustness. However, their definition has a flaw which makes it impossible to instantiate (there is a simple attack on any instantiation, even with a random oracle). When analyzing the security of our variant, we will provide a (more involved) corrected notion of correlation robustness, and formally prove that it is satisfied by a random oracle.

The above construction can be instantiated quite efficiently using the IKNP oblivious transfer extension [IKNP03]. It does not, however, directly imply a PSI protocol: since the keys K_i are distinct for each input, it can only help comparing items in the same position. To obtain a PSI protocol, KKRT uses hashing to map the datasets to bins, and reduce the intersection computation to a bin-by-bin comparison. With a naive hashing strategy, this would lead to a maximum of $\log n / \log \log n$ items per bin (for datasets of size n), which would still induce a significant overhead (since all pairs of items of Alice and Bob in the same bin must be compared).

To achieve better efficiency, the authors use *Cuckoo hashing with a stash*. In Cuckoo hashing, the items are mapped into $c \cdot n$ bins (where $c > 1$ is usually a small constant) using d hash functions $(h_1, \cdots, h_d)$, such that each item x is mapped to the bin number $h_i(x)$ for some $i \leq d$, and every bin contains at most a *single* item. The insertion follows a simple greedy procedure: x is inserted at the $h_1(x)$ location, and if an item y was already present, it is evicted and re-inserted at its next "authorized" location, $h_2(y)$, possibly evicting an item in turn. The process continues until insertion succeeds, or some threshold number of items have been evicted. If an item fails to be inserted, it is added to a special stack, called the *stash*.

Using Cuckoo hashing, Alice maps her dataset X to a length-cn vector $(x_1, \cdots, x_{cn})$ (possibly adding dummy items in bins that remained empty), and runs the BaRK-OPRF protocol with Bob, obtaining $(F_{\Delta,K_i}(x_i))_{i \leq cn}$. Then, Bob maps his own dataset Y to the $c \cdot n$ bins, this time using *all* hash functions $h_1, \cdots, h_d$ (that is, every $y \in Y$ is mapped to the bins $(h_1(y), \cdots, h_d(y))$), and computes $F_{\Delta,K_i}(y)$ for all items y in the i-th bin, for $i = 1$ to $c \cdot n$. In the end, Bob randomly shuffles and sends to Alice all these PRF evaluations. Alice computes the intersection with her own PRF evaluations, and learns $X \cap Y$.

KKRT also showed how to handle the items for which insertion failed (which are stored in the stash). However, the work of [PSZ18] heuristically analyzed the failure probability of generalized Cuckoo hashing with $k \geq 2$ hash functions, using large scale simulations and extrapolations. Based on their extrapolations, they determined that using as little as 3 hash functions and $N = 1.3 \cdot n$ bins suffices to guarantee a statistical failure probability below 2^{-40} (i.e., a stash size $s = 0$ with probability at least $1 - 2^{-40}$) for large enough set sizes (around $n = 2^{20}$).

Cost of KKRT. Using the heuristic parameters of [PSZ18] to get rid of the stash, the communication of KKRT consists in executing a BaRK-OPRF on $c \cdot n$ inputs, and sending $dn = 3n$ hashes. The BaRK-OPRF requires around $6\kappa n$ bits of communication, where κ is a computational security parameter (typically, $\kappa = 128$). The factor 6 overhead mainly comes from the fact that $\Delta \wedge \mathsf{Enc}(x)$ must retain $\approx \kappa$ bits of entropy, hence a large-ish value of Δ is required. Sending the $3n$ hashes costs $3 \cdot (\lambda + 2 \log n) \cdot n$ bits of communication, where λ is a statistical security parameter (typically, $\lambda = 40$ is the most common choice). Indeed, the hash outputs can be truncated to $\lambda + 2 \log n$ bits while maintaining the probability of collisions between the hashes of any pair $(x, y) \in X \times Y$ of distinct elements below $2^{-\lambda}$. For $n = 2^{20}$, this leads to the claimed $1008n$ bits of overall communication.

B Dual PSI from mOPRF

Intuitively, we execute $\mathcal{F}_{\mathsf{oprf}}$ twice while Alice and Bob have the same role. The main idea behind our approach is as follows:

- Alice with the input set X invokes $\mathcal{F}_{\mathsf{oprf}}$ as a receiver to learn the set of PRF values of each element in X. Denote $F_A(x)$ for all $x \in X$. While Bob queries Y to $\mathcal{F}_{\mathsf{oprf}}$ to get $F_A(y)$.
- Alice and Bob follow exactly the previous steps with the roles reversed. While Bob with input set Y can only get the set of PRF values $F_B(y)$. Alice queries X to $\mathcal{F}_{\mathsf{oprf}}$ and obtains $F_B(x)$ for all $x \in X$.
- Alice computes the set $E = \{F(x) := F_A(x) \oplus F_B(x) \mid x \in X\}$ and sends it to Bob. Bob computes $F(y)$ for all $y \in Y$ then check whether it is in E or not. Formally, Bob outputs

$$\{y \in Y \mid F(y) \in E\}$$

Informally, there are two crucial properties to guarantee the correctness of this construction:

- Since Alice only knows $F_A(x)$ for $x \in X$ and for $x \notin X$ the PRF value $F_A(x)$ is pseudorandom in the view of Alice; Alice only can obtain the correct value of $F(x)$ for $x \in X$. Similarly, Bob can compute correctly only $F(y)$ for $y \in Y$.
- The outputs of PRF are pseudorandom, with a high probability

$$\forall x \in X, \forall y \in Y : F(x) = F(y) \Leftrightarrow x = y$$

For security, this scheme can be proven against a malicious adversary since it is possible to extract the input set of both sender and receiver based on $\mathcal{F}_{\mathsf{oprf}}$ ideal functionality. We leave the detailed construction in Figure 10 and the formal proof in the theorem 10. It requires the output domain of PRF is $v = \lambda + 2\log(Nd) \approx \lambda + 2\log n$ so that the probability of existing two distinct elements $x \in X$ and $y \in Y$ such that $F(x) = F(y)$ is negligible $2^{-\lambda}$ where $|X|, |Y| \leq Nd$.

PARAMETERS:

- Alice (sender) and Bob (receiver) have respectively input set $X = \{x_1, x_2, \ldots, x_n\} \in \mathbb{F}_p$ and $Y = \{y_1, y_2, \ldots, y_n\} \in \mathbb{F}_p$, all elements of bit-length ℓ.
- A random hash functions $h : \{0,1\}^* \to [N]$.
- A Permutation-based hashing $\mathsf{Per}_{h,X}$ maps a set X to table $\mathcal{B}_X$ consisting of N bins such that each bin has d slots where $Nd > |X|$, and $d = O(1)$. Denote $\mathsf{Per}(x) := (i, x')$ where x' is the stored value of x in bin i which defined by h and x then $\mathsf{Per}^{-1}(i, x') = x$.

PROTOCOL:

1. Preprocessing phase.
   - Alice and Bob use Per to map their own set to $\mathcal{B}_X, \mathcal{B}_Y$ respectively.
   - For each empty slot in each bin $\mathcal{B}_X[i]$ and $\mathcal{B}_Y[i]$, put here a dummy item of length $\ell - \log N$.
2. Alice sends $(\mathsf{receiver}, \mathsf{id}, \mathcal{B}_\mathsf{X})$ and Bob sends $(\mathsf{sender}, \mathsf{id})$ to $\mathcal{F}_{\mathsf{oprf}}$ then
   - Alice receives the $X' = \{F_A(x) \mid x \in X\}$ where $F_A(x) := PRF(\mathsf{Per}(x))$.
   - For each $y \in Y$, Bob queries $\mathsf{Per}(y)$ to $\mathcal{F}_{\mathsf{oprf}}$ and get $F_A(y)$.
3. Similarly, Alice sends $(\mathsf{sender}, \mathsf{id})$ and Bob sends $(\mathsf{receiver}, \mathsf{id}, \mathcal{B}_\mathsf{Y})$ to $\mathcal{F}_{\mathsf{oprf}}$ then
   - Bob receives the $Y' = \{F_B(y) \mid y \in Y\}$ where $F_B(y) := PRF(\mathsf{Per}(y))$.
   - For each $x \in X$, Alice queries $\mathsf{Per}(x)$ to $\mathcal{F}_{\mathsf{oprf}}$ and get $F_B(x)$.
4. Alice now sends to Bob the set
$$E = \{F_A(x) \oplus F_B(x) \mid x \in X\}$$
Now for each $y \in Y$, if $F_A(y) \oplus F_B(y) \in E$ then Bob outputs y as an element in the intersection.

Fig. 10. Our second malicious PSI protocol based on $\mathcal{F}_{\mathsf{oprf}}$ via dual execution

Theorem 10. *The PSI protocol on Figure 10 securely realizes the ideal functionality $\mathcal{F}_{\mathsf{psi}}$ (figure 8) over the field $\mathbb{F}_\mathsf{p}$ for set size n and malicious set size $n_X = n, n_Y = Nd$ with statistical security against malicious adversaries in $\mathcal{F}_{\mathsf{oprf}}$ hybrid model.*

Proof. **Alice is corrupted**. Sim interacts with Alice as below:

- When $\mathcal{A}$ plays the role of receiver in $\mathcal{F}_{\mathsf{oprf}}$, Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ to get $\mathcal{B}_X$. Sim samples a uniformly random sequence $Z = \{Z_{i,x'}\}_{i\leq N}$ as the output PRF values of the set $\mathcal{B}_X$, where $Z_{i,x'}$ corresponds to $x' \in \mathcal{B}_X[i]$.
- From $\mathcal{B}_X$, Sim extracts $X^* = \{\mathsf{Per}^{-1}(i,x') \mid \forall x' \in \mathcal{B}_X[i], i \in [1,N]\}$.
- When $\mathcal{A}$ plays the role of sender in $\mathcal{F}_{\mathsf{oprf}}$, Sim defines a set $\tilde{X}$ containing of all elements x such that $\mathsf{Per}(x)$ has been queried to $\mathcal{F}_{\mathsf{oprf}}$; Sim samples a uniformly random values as $\{F_B(x)\}_{x\in\tilde{X}}$ and give them to $\mathcal{A}$.
- On behalf of Bob, Sim receives the set E from $\mathcal{A}$, Sim defines the set

$$X = \{x \in X^* \cap \tilde{X} \mid \mathsf{Per}(x) = (i,x') \wedge F_B(x) \oplus Z_{i,x'} \in E\}$$

  Sim sends X to $\mathcal{F}_{\mathsf{psi}}$ functionality, receiving $I := X \cap Y$.

This simulation can not distinguish from the real protocol by the following hybrids:

- *Hybrid 0.* The same as real protocol, Bob is honest with the input set Y, and $\mathcal{F}_{\mathsf{oprf}}$ is implemented honestly.
- *Hybrid 1.* Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ then functionality then
  - When Alice is a receiver in $\mathcal{F}_{\mathsf{oprf}}$, Sim learns $\mathcal{B}_X$ which is the input set of Alice and then Sim samples a uniformly random sequence $Z = \{Z_{i,x'}\}_{i\leq N}$ as the output PRF values of the set $\mathcal{B}_X$, where $Z_{i,x'} \in \{0,1\}^v$ corresponds to $x' \in \mathcal{B}_X[i]$.
  - When Alice is a sender in $\mathcal{F}_{\mathsf{oprf}}$, Sim defines a set $\tilde{X}$ containing of all elements x such that $\mathsf{Per}(x)$ has been queried to $\mathcal{F}_{\mathsf{oprf}}$; Sim samples a uniformly random values as $\{F_B(x)\}_{x\in\tilde{X}}$ and give them to $\mathcal{A}$.

  This hybrid is indistinguishable from the previous hybrid since the the outputs of PRF are pseudorandom.
- *Hybrid 2.* Sim computes a set

$$X^* = \{\mathsf{Per}^{-1}(i,x') \mid \forall x' \in \mathcal{B}_X[i], i \in [1,N]\}$$

  Note that, $|X^*| \leq Nd$.
- *Hybrid 3.* On behalf of Bob, Sim gets the set E sending from Alice. Sim defines the set

$$X = \{x \in X^* \cap \tilde{X} \mid \mathsf{Per}(x) = (i,x') \wedge F_B(x) \oplus Z_{i,x'} \in E\}$$

  This hybrid will abort if there exist some $x_1, x_2 \in X$ such that

$$F_B(x_1) \oplus Z_{i,x'_1} = F_B(x_2) \oplus Z_{i,x'_2}$$

  Observe that $\{Z_{i,x'}\}_{i\leq N}$ is first fixed for elements in X^* and then the function F_B is sampled. Therefore, the probability of aborting is bounded to $(Nd)^2/2^v$ which is negligible $O(2^{-\lambda})$ when $v = \lambda + 2\log n$. This deduces that $|X| \leq n$.
- *Hybrid 4.* Sim inputs X to $\mathcal{F}_{\mathsf{psi}}$ functionality, receiving $X \cap Y$ and then outputs it as the output of honest Bob.

Bob is corrupted. Sim interacts with Bob as below:

- Similarly, Sim plays the role of $\mathcal{F}_{\mathsf{oprf}}$ to get $\tilde{Y}, \mathcal{B}_Y$ then Sim samples a uniformly random sequence $T = \{T_{i,y'}\}_{i\leq N}$ as the output PRF values of the set $\mathcal{B}_Y$, where $T_{i,y'} \in \{0,1\}^v$ corresponds to $y' \in \mathcal{B}_Y[i]$.
- From $\mathcal{B}_Y$, Sim extracts $Y^* = \{\mathsf{Per}^{-1}(i,y') \mid \forall y' \in \mathcal{B}_Y[i], i \in [1,N]\}$.
- Sim emulates $\mathcal{F}_{\mathsf{psi}}$ functionality with input set $Y := Y^* \cap \tilde{Y}$, receiving $I := X \cap Y$.
- Sim sends to Bob the set containing of
  - $F_A(y) \oplus T_{i,y'}$ for $y \in I$.
  - $n - |I|$ uniformly random values in $\{0,1\}^v \setminus T$.

This simulation is indistinguishable from the real protocol by the following hybrids:

- *Hybrid 0.* The same as real protocol, Alice is honest with the input set X, and $\mathcal{F}_{\mathsf{oprf}}$ is implemented honestly.

– *Hybrid 1.* Sim emulates $\mathcal{F}_{\mathsf{oprf}}$ functionality then
  - When Bob is a sender in $\mathcal{F}_{\mathsf{oprf}}$, Sim extracts the set $\tilde{Y}$ such that for $y \in Y$, $\mathsf{Per}(y)$ has been queried to $\mathcal{F}_{\mathsf{oprf}}$; Sim give to Bob a uniformly random sequences defined as $\{F_A(y)\}_{y\in\tilde{Y}}$.
  - When Bob is a receiver in $\mathcal{F}_{\mathsf{oprf}}$, Sim learns $\mathcal{B}_Y$ which is the input set of Bob and then Sim samples a uniformly random sequence $T = \{T_{i,y'}\}_{i\leq N}$ as the output PRF values of the set $\mathcal{B}_Y$, where $T_{i,y'} \in \{0,1\}^v$ corresponds to $y' \in \mathcal{B}_Y[i]$.
– *Hybrid 2.* From $\mathcal{B}_Y$, Sim computes a set

$$Y^* = \{\mathsf{Per}^{-1}(i, y') \mid \forall y' \in \mathcal{B}_Y[i], i \in [1, N]\}$$

  Sim emulates $\mathcal{F}_{\mathsf{psi}}$ functionality with input set $Y = \tilde{Y} \cap Y^*$, receiving $I := X \cap Y$.
– *Hybrid 3.* Sim sends to Bob the set E containing of
  - $\mathrm{T} = \{F_A(y) \oplus T_{i,y'} \mid \forall y \in I\}$.
  - $n - |I|$ uniformly random values in $\{0,1\}^v \setminus T$.

  This hybrid is indistinguishable from the real protocol since in the view of Bob E consists of the XOR of pseudorandom values and a arbitrary values. This concludes the proof.

C The OT-Based PSI of PSZ14

For completeness, we recall in this Appendix a PSI protocol of [PSZ14, Section 5], which is particularly relevant to our approach. In essence, this protocol combines an OT-based secure equality test with a standard hash-based approach. While this protocol had initially much higher communication than its later competitor [KKRT16], recent advances in pseudorandom correlation generators (which we exploit in this paper) have enabled the construction of *silent OT extension* protocols [BCG+19a, CRR21] which can be used to make this protocol considerably more competitive. For the sake of comparing this protocol to our new proposals, we work out its parameters with silent OT extension here.

C.1 Description of the Protocol

OT-based equality test. The core component of the protocol is a private equality test (PEQT) which allows two parties with respective inputs x, y of length ℓ to learn whether $x = y$, and nothing more. The protocol proceeds as follows:

– The sender with input x picks ℓ pairs of random strings $(s_0^i, s_1^i)_{i\leq\ell}$ and computes $x' \leftarrow \bigoplus_i s_{x_i}^i$. He sends x' to the receiver
– The sender and the receiver execute ℓ parallel instances of string-OT, where the selection bits of the receiver with input y are the bits $y_1, \cdots, y_\ell$. The receiver obtains $(s_{y_i}^i)_{i\leq\ell}$ and computes $y' \leftarrow \bigoplus_{i=1}^{\ell} s_{y_i}^i$.
– If $x' = y'$, the receiver decides that $x = y$; otherwise, she decides that $x \neq y$.

Using silent OT extension, executing ℓ random OT hash vanishing communication (which approaches 0 when amortized over multiple PEQT). Then, to derandomize the random OTs into OTs with selection bits $y_1, \cdots, y_\ell$, the receiver must send exactly ℓ bits. Eventually, sending x' costs λ bits (assuming the OTs are on λ-bit strings). Hence, ignoring the vanishing cost, each PEQT will have amortized communication $\ell + \lambda$.

Private set inclusion. The above PEQT can be extended to a private set inclusion test: assume that the sender has now an input set $X = \{x^1, \cdots, x^n\}$. Then the sender and the receiver simply execute the above protocol, but the pairs of random strings $(s_0^i, s_1^i)_{i\leq\ell}$ are now $n \cdot \lambda$-bit long each. Given a bit b and $i \leq \ell$, us denote each of the n λ-bit long substrings of s_b^i by $(s_b^{i,1}, \cdots, s^{i,n})$. The sender sets $x'_j \leftarrow \bigoplus_i s_{x_i^j}^{i,j}$ for $j = 1$ to n, and the receiver sets $y' \leftarrow \bigoplus_i s_{y_i}^i$. Eventually, the sender sends all of the x'_j to the receiver in a random order. The receiver outputs $y \in X$ if one of the x'_j is a substring of y'. Observe that this is essentially performing n PEQT in parallel, while hiding which of the elements of X was equal to y. The total (amortized) communication is now $\ell + n \cdot \lambda$.

Hash-based PSI from PEQT. Equipped with the above protocol, the PSI proceeds as follows: the sender and the receiver map their items into bins, using for example Cuckoo hashing on the receiver side, and simple hashing on the sender side (as in KKRT16).

For each bin, and for each item in the bin on the receiver side, the receiver will execute a private set inclusion with the sender, as above, with the following tweak: the length of the random strings is increased from $\lambda + 2\log n$, where n is now the number of items in the sets of each party. The sender does not send the x'_j shuffled for each bin: rather, the sender accumulates all the x'_j across all bins, shuffles then randomly, and sends the shuffled values all at once (this allows to mask the load of each bin on the sender side).

C.2 Communication Costs

The total communication of this protocol is given by

$$\ell \cdot N + (\lambda + 2 \cdot \log N) \cdot k \cdot d + o(\ell \cdot N),$$

where $o(\ell \cdot N)$ is the communication cost of setting up the silent OT extension (the protocol requires $(\ell - \log N + 2) \cdot N$ oblivious transfers in total), k is the number of hash functions and d is the maximum load of a bin on the receiver side. For large values of n, the $o(\ell \cdot N)$ term vanishes; however, for small values of n, its impact on the communication is significantly higher than the impact of the $o(n)$ term in our protocol (since we need much shorter VOLEs).

Note that here, an important difference with our approach is that d influences the sender-to-receiver communication: unlike with our method, therefore, it is impractical to use values of d above 1. As for the KKRT protocol, if hashing with a stash is used, a dedicated protocol must be used to handle the stash, and the most efficient solution is typically to choose parameters guaranteeing that there will be no stash with high probability. In consequence, in our estimations, we choose $k = 3$, $d = 1$, and set the number of bins to $N = 1.3$, which by the analysis of [PSZ18] suffices to guarantee a 2^{-40} failure probability. We also apply the technique of [TLP+17] to reduce the $\lambda + 2\log n$ term to $\lambda + \log n$, and the Phasing method to reduce ℓ to $\ell - \log N + 2$ (the two bits are required to indicate the index of the hash function, since we use three hash functions). This way, the total communication becomes

$$(1.3(\ell + 2 - \log N) + 3 \cdot (\lambda + \log N)) \cdot n + o(\ell \cdot n).$$

We compare this communication cost to our new protocols on Table 3. As the table shows, PSZ14 provides a competitive alternative to RS21, but is still outperformed by our new protocols on essentially all parameters, by up to 40%.

Table 3. Comparison of the communication cost of our new PSI protocol with the PSZ14 protocol using silent OT extension (with all relevant optimizations applied to PSZ14).

	$n = 2^{14}$	$n = 2^{16}$	$n = 2^{20}$	$n = 2^{24}$
RS21 [RS21] enhanced	$257n$	$207n$	$197n$	$196n$
PSZ14 [PSZ14] + sOT, $\ell = 64$	$275n$	$245n$	$240n$	$246n$
PSZ14 [PSZ14] + sOT, $\ell = 48$	$253n$	$224n$	$219n$	$225n$
PSZ14 [PSZ14] + sOT, $\ell = 32$	$231n$	$203n$	$199n$	$205n$
Ours ($\ell = 64$, GCH)	$246n$	$220n$	$210n$	$209n$
Ours ($\ell = 48$, GCH)	$215n$	$189n$	$179n$	$178n$
Ours ($\ell = 32$, GCH)	$184n$	$158n$	$148n$	$147n$
Ours ($\ell = 64$, 2CH)	$214n$	$190n$	$183n$	$185n$
Ours ($\ell = 48$, 2CH)	$193n$	$169n$	$162n$	$164n$
Ours ($\ell = 32$, 2CH)	$171n$	$148n$	$141n$	$142n$
Ours ($\ell = 64$, SH, $N = n/10$)	$332n$	$302n$	$284n$	$276n$
Ours ($\ell = 48$, SH, $N = n/10$)	$261n$	$230n$	$209n$	$198n$
Ours ($\ell = 32$, SH, $N = n/10$)	$191n$	$158n$	$133n$	$120n$

C.3 Computational Costs

It remains to compare the (enhanced) protocol of PSZ14 to ours computation-wise.

Receiver computation. In our protocol, receiver computation boils down to

- Setting up the VOLE correlation
- Hashing his input set into bins
- Interpolating N degree-d polynomials
- Computing $d \cdot N$ hashes

In contrast, in PSZ14, the receiver computation boils down to

- Setting up the $(\ell - \log N + 2) \cdot N$ random OTs
- Hashing his input set into bins
- Computing N hashes

Let us first compare the correlation setups. Using [CRR21], computing 10 million random OTs on one core of a standard laptop takes 300ms. We note that silent OT extension requires in particular setting up a subfield VOLE first, and then hashing the outputs with a correlation-robust hash function to obtain OTs (*a la* [IKNP03]). The cost of this final hashing step was estimating to take roughly a third of the total computation time in [CRR21], hence setting up a length-10 millions subflied VOLE should be roughly 1.5 times faster.

For $n = 2^{20}$ and $\ell = 48$ (to take a middle ground for the parameters), our protocol requires a length-dN VOLE, which (using $N = 0.65n$, $d = 3$) should therefore take significantly less than 50 milliseconds on a laptop similar to the one used in [CRR21]. In contrast, the PSZ14 protocol requires $(\ell - \log N + 2) \cdot N \approx 40 \cdot n$ random OTs (using $N = 1.3n$), which should take about 1200 milliseconds on the same laptop (more than 20 times more than the VOLE cost in our protocol).

The cost of computing the N hashes using a hash function built from fixed-key AES takes a few dozen milliseconds on the laptop of [CRR21], and is dominated by the cost of setting up the VOLE / silent OT correlations (since the latter involve computing significantly more hashes with fixed-key AES anyway). The cost of hashing into the bins is identical in both our protocol and PSZ14, and should again be largely dominated by the cost of setting up the VOLE/OT: it requires inserting n items into N bins using Phasing [PSSZ15], which boils down to computing around n hashes, which can be implemented with a very fast hash function (since it needs not be a cryptographically strong hash function). Eventually, using generalized Cuckoo hashing, our protocol requires interpolating $0.65 \cdot n$ degree-3 polynomials, which boils down to computing $5 \cdot 0.65n = 3.25n$ multiplications over the field $\mathbb{F}_{2^{\ell - \log N + 1}} = \mathbb{F}_{2^{30}}$. While we do not have benchmarks for this cost, it should again be largely dominated by the cost of performing n fixed-key AES evaluations.

While the above are approximate estimations and real runtimes would significantly differ (due to, e.g., cache misses), it should be clear that the dominant cost is that of setting up the VOLE/OT, and that this cost is an order of magnitude higher for the protocol of PSZ14. Therefore, we estimate that our protocol should require significantly less computation than that of PSZ14 (our back-of-the-envelope calculation suggests an order of magnitude, though of course it can vary significantly).

Sender computation. The sender costs are identical to the receiver costs in both protocols, up to one difference: the sender must compute $k \cdot dN = 3.9n$ hashes in our protocol, and $k \cdot N = 3.9n$ in PSZ14 (using the simple hashing scheme), where k is the number of hash functions (2 in our protocol, 3 in PSZ14). In addition, the sender in our protocol must also interpolate $k \cdot N = 1.3n$ degree-d polynomial. Hence, the added cost (compared to the receiver computation) is $1.9n$ hashes and $3.25n$ multiplications in our protocol, and $2.6n$ hashes in PSZ14. Since the cost of computing $0.7n$ hashes is likely larger than that of computing $3.25n$ multiplications over $\mathbb{F}_{2^{30}}$, the computational gap should be similar to or larger than the receiver case.