Oblivious Key-Value Stores and Amplification for Private Set Intersection

Gayathri Garimella[*] Benny Pinkas[†] Mike Rosulek[*] Ni Trieu[‡] Avishay Yanai[§]

June 26, 2021

Abstract

Many recent private set intersection (PSI) protocols encode input sets as polynomials. We consider the more general notion of an oblivious key-value store (OKVS), which is a data structure that compactly represents a desired mapping $k_i \mapsto v_i$. When the v_i values are random, the OKVS data structure hides the k_i values that were used to generate it. The simplest (and size-optimal) OKVS is a polynomial p that is chosen using interpolation such that $p(k_i) = v_i$.

We initiate the formal study of oblivious key-value stores, and show new constructions resulting in the fastest OKVS to date.

Similarly to cuckoo hashing, current analysis techniques are insufficient for finding *concrete* parameters to guarantee a small failure probability for our OKVS constructions. Moreover, it would cost too much to run experiments to validate a small upper bound on the failure probability. We therefore show novel techniques to amplify an OKVS construction which has a failure probability p, to an OKVS with a similar overhead and failure probability p^c. Setting p to be moderately small enables to validate it by running a relatively small number of $O(1/p)$ experiments. This validates a p^c failure probability for the amplified OKVS.

Finally, we describe how OKVS can significantly improve the state of the art of essentially all variants of PSI. This leads to the fastest two-party PSI protocols to date, for both the semi-honest and the malicious settings. Specifically, in networks with moderate bandwidth (e.g., 30 - 300 Mbps) our malicious two-party PSI protocol has 40% less communication and is 20-40% faster than the previous state of the art protocol, even though the latter only has heuristic confidence.

1 Introduction

Private set intersection (PSI) allows parties to learn the intersection of sets that they each hold, without revealing anything else about the individual sets. One common technique that has emerged in several PSI protocols (and protocols for closely related tasks) is to encode data into a polynomial. More precisely, a party interpolates a polynomial P so that $P(x_i) = y_i$, where the x_i's are their PSI input set and y_i are some values that are relevant in the protocol. The polynomial P compactly encodes a chosen mapping from x_i's to y_i's, but it has the additional benefit that it hides the x_i's, when the y_i's are random. This property is critical since the x_i's coincide with some party's private input set, which must be hidden.

[*]Oregon State University
[†]Bar-Ilan University
[‡]Arizona State University
[§]VMware Research

We present two major contributions. First, we abstract the properties of polynomials that are needed in these applications, and define "oblivious key-value stores" (OKVS) as objects satisfying these properties. We show how to construct a substantially more efficient OKVS that has linear size, similar to polynomials, and replaces the task of polynomial interpolation with an efficient linear time computation. Second, we observe that current analysis techniques are insufficient for setting concrete parameters to ensure a concrete upper bound (say, 2^{-40}) for the failure probability of our OKVS construction. (This is also true for many other randomized constructions, such as cuckoo hashing, used in PSI and in other cryptographic algorithms.) Furthermore, running experiments in order to validate this upper bound for a specific choice of parameters is extremely resource-intensive. Most previous work used heuristic techniques for setting the parameters for similar constructions. We overcome this issue by introducing new techniques for *amplifying* a randomized OKVS construction with a failure probability p, to an OKVS with a similar overhead and a failure probability p^c. Since p can be rather moderate, it is relatively easy to empirically validate that the failure probability of a specific choice of parameters is indeed bounded by p.

1.1 Polynomial Encodings for PSI

Cryptographic protocols which use polynomial encodings to hide input values date back to at least the work of Manulis, Pinkas, and Poettering [MPP10], in the context of "secret handshake" protocols (closely related to covert MPC and to PSI). Other examples that we are aware of include:[1]

- Cho, Dachman-Soled, and Jarecki [CDJ16] achieve 2-party PSI using a polynomial whose outputs (y_i values) are protocol messages from a suitable string-equality test protocol.
- Kolesnikov *et al.* [KMP+17] introduce a primitive called oblivious *programmable* PRF (OPPRF), which acts like an oblivious PRF with a twist. A sender selects (or learns) a PRF seed k and a receiver learns $PRF(k, a)$ for one or more values a of his/her choosing. But additionally, the sender gets to "program" the PRF on values of its choice as $PRF(k, x_i) = z_i$, where the special x_i points remain secret. This is achieved by combining a standard oblivious PRF $F(k, x_i)$ with a polynomial which encodes "output corrections" that the receiver applies in order to make the output match the sender's $x_i \mapsto z_i$ mappings.

  They use this OPPRF to construct a multi-party PSI protocol. Later, Pinkas *et al.* [PSTY19] also use an OPPRF to construct a protocol for computing arbitrary functions of the intersection (of two sets). Recently, OPPRFs were used by Chandran *et al.* for constructing circuit-PSI and multi-party PSI [CGS21, CDG+].
- Pinkas *et al.* [PRTY19] construct a low-communication PSI protocol using a polynomial whose outputs are values from the IKNP OT extension protocol [IKNP03].
- Kolesnikov *et al.* [KRTW19] construct a private set *union* protocol, using a variant of the OPPRF technique.

One downside to polynomials is that interpolating and evaluating them is not cheap. To interpolate a polynomial through n (unstructured) points, or to evaluate such a polynomial at n points, requires

[1] We note that there are also PSI constructions which use arithmetic manipulations of polynomials. These constructions encode input values as roots of polynomials [FNP04, KS05, GN19] or into separate monomials of a polynomial [GS19], and manipulate the polynomials in order to compute set operations. Our focus is on encodings, which is the more efficient versions of PSI, and do not require arithmetic manipulation of polynomials in order to compute the intersection.

$O(n \log^2 n)$ field operations, using the FFT algorithms of [MB72]. This cost becomes substantial for larger values of n, and raises the following natural question:

> *Is there a data structure that is better than a polynomial, for use in these PSI (and related) protocols?*

In addition to these applications of polynomials, Pinkas *et al.* [PRTY20] used a related technique to construct the fastest *malicious-secure* 2-party PSI protocol to date. They introduced a data structure called a PaXoS (probe and XOR of strings) which, similar to a polynomial, encodes a mapping from keys to values while hiding the keys. PaXoS took a significant step toward the abstraction of an OKVS, however, it is not sufficiently general. In particular, PaXoS is a specific, binary type of OKVS, whereas other types exist (like a linear OKVS, which is applicable in Oblivious Polynomial Evaluation [NP99]). The PaXoS data structure is the starting point for our constructions.

1.2 Correctness Amplification

One of the most challenging aspects of designing efficient PSI and OKVS constructions, is obtaining concrete bounds on extremal properties of randomized data structures. For example, exactly how many bins are required for cuckoo hashing with 3 hash functions, to ensure that the induced "cuckoo graph" avoids a certain structure with probability at least $1 - 2^{-40}$? This problem is crucial for PSI, since most PSI constructions are based on randomized data structures such as cuckoo hashing. Any failure in these constructions (e.g., too many collisions) leads to a violation of privacy. An implementation of PSI needs to be instantiated with specific parameters that will ensure a sufficiently small failure probability, but the available literature describing and analyzing the randomized constructions only describes asymptotic bounds, and it seems highly non-trivial to translate them to concrete numbers.

Prior PSI work which used such constructions, in particular variants of cuckoo hashing, either ran a small number of experiments in order to heuristically set the parameters, or, as in [PSZ18], invested significant efforts (*e.g.*, millions of core hours) to empirically measure the failure probability of these data structures. (This is needed since validating an upper bound of p on the failure probability requires running more than $1/p$ experiments.) But even after expending such efforts, it was not possible to validate the desired failure probabilities (*e.g.*, 2^{-40}), since they were too small. So ultimately in [PSZ18] and in other constructions which are based on the same set of experiments, the failure probabilities of the final constructions were only **extrapolated** from these empirical trials.

The lack of a concrete analysis for the failure probabilities of different randomized constructions, and the extreme cost of experimentally verifying small upper bounds on these probabilities, raise the following question:

> *Is is possible to start with a construction that has a moderately high failure probability, and which can therefore be validated through efficient experiments, and amplify it to obtain a construction which has a much smaller failure probability?*

For example, we can validate on a laptop an upper bound of 2^{-25} or 2^{-13}, whereas validating a 2^{-40} failure probability might require using a large cluster.

1.3 Our Results

In this work, we initiate the study of OKVS data structures and their properties.

- We introduce the abstraction of an **oblivious key-value store (OKVS)**. An OKVS consists of algorithms Encode and Decode. Encode takes a list of key-value pairs (k_i, v_i) as input and returns an abstract data structure S. Decode takes such a data structure and a key k as input, and gives some output. Decode can be called on any key, but if it is called on some k_i that was used to generate S, then the result is the corresponding v_i. The most basic property of an OKVS echoes the important property of polynomials; namely, S hides the k_i's, when the v_i's are random. We identify and formalize important properties that allow OKVS to be plugged into different protocols.

- We catalog existing OKVS constructions and introduce several new and improved ones.

- We describe **amplification techniques** that can be used to bootstrap strong OKVS out of weaker ones. Amplification only requires to validate a relatively high upper bound on the failure probability of the corresponding randomized construction, a task that can be accomplished through efficient experiments. As an example, we can construct an OKVS with *provable* error probability 2^{-40}, from an OKVS with error probability 2^{-25}. The latter probability is high enough that it can be empirically and efficiently verified with very high statistical confidence.

  Besides having more manageable error analysis, our new OKVS constructions improve considerably over the state of the art in terms of size and speed.

- We show that many existing PSI protocols can be written abstractly in terms of a generic OKVS. These PSI protocols are therefore automatically improved by instantiating with our improved OKVS constructions. As a flagship example, we demonstrate the improvement on the so-called "PaXoS-PSI" protocol of [PRTY20], which is the state of the art protocol with malicious security. Specifically, our protocol has 40% less communication and is 20% and 40% faster over medium and slow networks[2], respectively, for sets of a million items (over a fast network it is only 5% slower). In addition, on slow networks, our *malicious* protocol is even faster than the state of the art *semi-honest* protocol [PRTY19] (and is only about 10% and 20% slower than the best semi-honest protocols over fast [KKRT] and medium [CM20] networks, repectively).

  We also note that the covert MPC protocols of [MPP10, CDJ16] can be expressed using our OKVS constructions to exhibit a higher level of abstraction and to achieve a better runtime.

- Finally, we show two improvements to existing PSI protocols, beyond replacing their underlying OKVS with a better one.

  First, we observe that the leading state-of-the-art PaXoS PSI protocol of [PRTY20] can be generalized to be built from vector-OLE rather than 1-out-of-N OT extension. Since vector-OLE enjoys more algebraic structure, the generalized PSI protocol can take advantage of a more general class of OKVS, and also avoid one source of overhead in the construction.

  Second, we show that one of the multi-party PSI constructions of Kolesnikov et al. [KMP+17], which is the most efficient of the constructions presented in that paper but only has "augmented semi-honest security" rather than semi-honest security, actually enjoys *malicious security*. Hence, we obtain the most efficient malicious, multi-party PSI protocol to date.

[2]The slow network (33 Mib/s); medium network (260Mib/s); fast network (4.6 Gib/s)

2 Oblivious Key-Value Stores

2.1 Definitions

Definition 1. *A **key-value store** is parameterized by a set $\mathcal{K}$ of keys, a set $\mathcal{V}$ of values, and a set of functions H, and consists of two algorithms:*

- Encode_H *takes as input a set of (k_i, v_i) key-value pairs and outputs an object S (or, with statistically small probability, an error indicator $\bot$).*
- Decode_H *takes as input an object S, a key k, and outputs a value v.*

A KVS is **correct** *if, for all $A \subseteq \mathcal{K} \times \mathcal{V}$ with distinct keys:*

$$(k,v) \in A \textit{ and } \bot \neq S \leftarrow \mathsf{Encode}_H(A) \implies \mathsf{Decode}_H(S,k) = v$$

In the rest of the exposition we choose to omit the underlying parameter H as long as the text remains unambiguous.

In all the algorithms that we describe, the decision whether Encode outputs $\bot$ depends on the functions H and the keys k_i and is independent of the values v_i. If the data is encoded as a polynomial then Encode always succeeds.

To be clear, one may invoke $\mathsf{Decode}(S,k)$ on *any* key k, and indeed it is our goal that one cannot tell whether k was used to generate S or not. This is stated in the next definition.

Definition 2. *A KVS is an **oblivious KVS (OKVS)** if, for all distinct $\{k_1^0, \ldots, k_n^0\}$ and all distinct $\{k_1^1, \ldots, k_n^1\}$, if Encode does not output $\bot$ for $(k_1^0, \ldots, k_n^0)$ or $(k_1^1, \ldots, k_n^1)$, then the output of $\mathcal{R}(k_1^0, \ldots, k_n^0)$ is computationally indistinguishable to that of $\mathcal{R}(k_1^1, \ldots, k_n^1)$, where:*

```
$\mathcal{R}(k_1, \ldots, k_n)$:
    for $i \in [n]$: do $v_i \leftarrow \mathcal{V}$
    return $\mathsf{Encode}(\{(k_1, v_1), \ldots, (k_n, v_n)\})$
```

In other words, if the OKVS encodes random values (as it does in our applications), then for any two sets of keys K^0, K^1 it is infeasible to distinguish between an OKVS encoding of the keys of K^0 from an OKVS encoding of the keys of K^1. In fact, all our constructions satisfy the property that if the values encoded in the OKVS are random (as in the experiment R), then the two distributions are perfectly indistinguishable.

2.2 Linear OKVS

Some applications of an OKVS use it to encode data that is processed in some kind of homomorphic cryptographic primitive. In that case, it is convenient for $\mathsf{Decode}(\cdot, k)$ to be a **linear function** for all k.

Definition 3. *An OKVS is **linear** (over a field $\mathbb{F}$) if $\mathcal{V} = \mathbb{F}$ ("values" are elements of $\mathbb{F}$), the output of Encode is a vector S in $\mathbb{F}^m$, and the Decode function is defined as:*

$$\mathsf{Decode}(S,k) = \langle \mathsf{d}(k), S \rangle \overset{\text{def}}{=} \sum_{j=1}^{m} \mathsf{d}(k)_j S_j$$

for some function $\mathsf{d} : \mathcal{K} \to \mathbb{F}^m$. Hence $\mathsf{Decode}(\cdot, k)$ is a linear map from $\mathbb{F}^m$ to $\mathbb{F}$.

The mapping $\mathsf{d} : \mathcal{K} \to \mathbb{F}^m$ are typically defined by the hash function H.

For a linear OKVS, one can view the Encode function as generating a solution to the linear system of equations:

$$\begin{bmatrix} - \ \mathsf{d}(k_1) \ - \\ - \ \mathsf{d}(k_2) \ - \\ \vdots \\ - \ \mathsf{d}(k_n) \ - \end{bmatrix} S^\top = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

Hence, it is necessary that for all distinct $k_1, \ldots, k_n$, the set $\{\mathsf{d}(k_1), \ldots, \mathsf{d}(k_n)\}$ is linearly independent, with overwhelming probability. However, we also consider *how efficiently* Encode finds such a solution, since solving systems of linear equations is expensive in general . It is often convenient to characterize a linear OKVS by its d function alone.

Note that when Encode chooses *uniformly* from the set of solutions to the linear system, and the v_i values are uniform, the output S is uniformly distributed (and hence distributed independently of the k_i values). In other words, a linear OKVS satisfies the obliviousness property.

2.3 Binary OKVS

A **binary OKVS** over a field $\mathbb{F}$ is a special case of a linear OKVS, where the $\mathsf{d}(k)$ vectors are restricted to $\{0,1\}^m \subseteq \mathbb{F}^m$. Then $\mathsf{Decode}(S, k)$ is simply the sum of some positions in S.

We generally restrict our attention to $\mathbb{F} = GF(2^\ell) \cong \{0,1\}^\ell$, in which case the addition operation over $\mathbb{F}$ is XOR of strings. In [PRTY20], a binary OKVS is called a **probe and XOR of strings (PaXoS)** data structure.

In a binary OKVS we have (in addition to the usual properties of a linear OKVS) the property that:

$$\mathsf{Decode}\Big((S_1 \wedge x, \ldots, S_m \wedge x), k\Big) = \mathsf{Decode}\Big((S_1, \ldots, S_m), k\Big) \wedge x$$

where "$\wedge$" is bitwise-AND of strings, and $x \in \{0,1\}^\ell$. This additional property is used in one of the important applications of OKVS.

2.4 OKVS Overfitting

Often in malicious protocols, the simulator obtains an OKVS from a corrupt party and must "extract" the items that are encoded in that OKVS. Generally this is done by requiring an OKVS to include mappings $(k_i, v_i) \mapsto H(k_i)$ where H is a random oracle.[3] The simulator can observe the adversary's queries to H, and then later test which of those k sastisfy $\mathsf{Decode}(S, k) = H(k)$.

An OKVS whose parameters are chosen to encode n items can often hold even more than n items, especially when generated by an adversary. In the context of PSI, this leads to an adversary holding more items than advertised. It is therefore important to be able to bound the number of items that an adversary can "overfit" into an OKVS. In order to define this property we define a "game" which lets the adversary choose an arbitrary data structure S, of a size which can normally encode n (key,value) pairs. The adversary wins the game if it can find an S which encodes much more than n pairs of the form $(k_i, H(k_i))$. More formally, we use the following definition.

Definition 4. *The* (n, n')*-**OKVS overfitting game** is as follows. Let* $\mathsf{Encode}, \mathsf{Decode}$ *be an OKVS with parameters chosen to support n items, and let* $\mathcal{A}$ *be an arbitrary PPT adversary. Run*

[3] We abuse notation herein and use H to denote a random oracle rather than the underlying OKVS parameter, which remains implicit.

$S \leftarrow \mathcal{A}^H(1^\kappa)$. Define

$$X = \{k \mid \mathcal{A} \text{ queried } H \text{ at } k \text{ and } \mathsf{Decode}(S, k) = H(k)\}$$

If $|X| > n'$ then the adversary wins.

*We say the (n, n')-OKVS overfitting problem is **hard** for an OKVS construction if no PPT adversary wins this game except with negligible probability.*

The work in [PRTY20] gives an unconditional bound on the success probability in the overfitting game. They prove the bound for binary OKVS ("PaXoS", in their terminology), but the only property of OKVS they use is its correctness; hence it applies to any KVS:

Lemma 5 ([PRTY20])**.** *Let H be a random oracle with output length ℓ, and let* Encode, Decode *be an OKVS scheme supporting n key-value pairs, where the output of* Encode *is a bit string of length ℓ'. Then the probability that an adversary who makes q queries to H wins the (n, n')-OKVS overfitting game is $\leq \binom{q}{n'} 2^{\ell' - n'\ell}$.*

The nature of this bound is to argue that an OKVS that encodes n' items simply can't exist; for if it did exist, then it could be used to construct a compressed representation of the random oracle. One may further conjecture that an OKVS construction has a hard overfitting problem (for some relationship between n and n') against polynomial-time adversaries. For example, perhaps it may be hard to find a single polynomial of degree n that matches the random oracle on $n' = n + 100$ points, even in the case that such a polynomial exists.

Better cryptanalysis of these kinds of overfitting problems would lead to a tighter security analysis of our malicious-secure PSI protocols: the protocols would be proven to more strongly enforce the size of corrupt party's input sets.

2.5 Efficiency of OKVS

We can measure the efficiency of an OKVS based on the following measures: (1) The **rate** of an OKVS which encodes n elements from $\mathbb{F}$ is the ratio between the size of the OKVS and $n \cdot |\mathbb{F}|$, which is the minimal size required for this encoding. (2) The **encoding time** is the time which is required for encoding n items in the OKVS. (3) The **decoding time** is the time required for decoding (querying) a single element, while the **batch decoding time** is the time required for decoding n elements.

3 Existing OKVS constructions

In this section we list existing constructions that fit to the OKVS definition. These are summarized in Table 1.

Polynomials A simple and natural OKVS is a polynomial P satisfying $P(k_i) = v_i$. The coefficients of the polynomial are the OKVS data structure, and decoding amounts to evaluating the polynomial at a point k. This OKVS has optimal rate 1, and is **linear** since $P(k)$ is the inner product of $(1, k, k^2, \ldots)$ and the vector of coefficients. Encoding n items takes $O(n \log^2 n)$ field operations using the FFT interpolating algorithms of [MB72]. Batch decoding of n items likewise takes $O(n \log^2 n)$ operations, while decoding a single items takes $O(n)$ operations.

OKVS	type	rate	encoding cost	(batch) decoding cost
polynomial	linear	1	$O(n \log^2 n)$	$O(n \log^2 n)$
random matrix	linear	1	$O(n^3)$	$O(n^2)$
random matrix	binary	$1/(1+\lambda)$	$O(n^3)$	$O(n^2)$
garbled Bloom filter [DCW13a]	binary	$O(1/\lambda)$	$O(n\lambda)$	$O(n\lambda)$
PaXoS [PRTY20]	binary	$0.4 - o(1)$	$O(n\lambda)$	$O(n\lambda)$
Ours: 3H-GCT (§4.1)	binary linear	$0.81 - o(1)$	$O(n\lambda)$	$O(n\lambda)$

Figure 1: Different OKVS constructions and their properties, for error probability $2^{-\lambda}$. (The rate of the 3H-GCT construction can be improved to 0.91 by using the hypergraph construction of [Wal21b], but this improvement takes effect only for very large values of n.)

Dense matrix Another simple OKVS sets $\mathsf{d}(k)$ to be a random vector in $\mathbb{F}^m$ for each k. This means that the encoding matrix is a random matrix. It is well-known that a random matrix with n rows and $m \geq n$ columns has linearly *dependent* rows with probability at most

$$\sum_{j=1}^{n} \Pr[\text{row } j \in \text{span of first } j-1 \text{ rows} \mid \text{first } j-1 \text{ rows linearly ind.}] \tag{1}$$

$$= \sum_{i=0}^{n-1} \frac{|\mathbb{F}|^i}{|\mathbb{F}|^m} = \frac{1}{|\mathbb{F}|^m} \cdot \frac{|\mathbb{F}|^n - 1}{|\mathbb{F}| - 1} < |\mathbb{F}|^{n-m-1} \tag{2}$$

For an exponentially large field $\mathbb{F}$, we can have $m = n$ and hence achieve rate 1. If we desire a binary OKVS, then $\mathsf{d}(k)$ are $\{0,1\}$-vectors and we must have $m \geq n + \lambda - 1$ for error probability $2^{-\lambda}$.

While achieving a good rate, the encoding and decoding procedures are expensive. Encoding n items corresponds to solving a linear system of n random equations, which requires $O(n^3)$ operations using Gaussian elimination. Decoding each item costs $O(n)$. A random matrix OKVS has worse performance than a polynomial-based OKVS. The main reason for using a random matrix OKVS is if the underlying field $\mathbb{F}$ is smaller than n, for example, is a binary field, in which case it is impossible to define an n-degree polynomial over $\mathbb{F}$.

Garbled Bloom filter (GBF) In a garbled Bloom filter [DCW13a], n items are encoded into a vector of length $m = O(\lambda n)$, i.e. it has a rate of $O(1/\lambda)$. The scheme is parameterized by λ random functions $H = \{h_1, \ldots, h_\lambda\}$ with range $[m]$. We have $\mathsf{d}(k)$ zero everywhere except in the positions $h_1(k), \ldots, h_\lambda(k)$, where it is 1. Hence a garbled Bloom filter is a binary OKVS.

Encoding is done in an online manner, one item at a time. Encoding fails with probability $1/2^\lambda$, and the specific error probability is exactly the same as the false-positive probability for a standard Bloom filter with the same parameters (namely, using λ hash functions and a vector of size $m = 1.44\lambda n$ result in a failure probability of $1/2^\lambda$ [MU05]).

Encoding n items costs $O(n\lambda)$, and decoding each item likewise costs $O(\lambda)$, since only λ positions in $\mathsf{d}(k)$ are nonzero.

GBFs were used in multiple PSI papers, beginning in [DCW13b], and including the multi-party protocols of [IOP18, ZLL+19, BENOPC]. A major drawback of the usage of GBFs is the larger communication overhead of sending a GBF of length $O(\lambda n)$, instead of sending an object of size $O(n)$, and computing $O(\lambda n)$ oblivious transfers.

PaXoS [PRTY20] In a probe-and-xor of strings (PaXoS), n items are encoded into a vector S of length $m = (2+\varepsilon)n + \log(n) + \lambda$.

Let us describe a simplified version of PaXos for which S is of size $m = (2+\varepsilon)n$. This scheme is parameterized by 2 random hash functions $H = \{h_1, h_2\}$ with a range $[(2+\varepsilon)n]$. Decoding of a key x sums the vector entries at $h_1(x)$ and $h_2(x)$. Encoding is done by generating the "cuckoo graph" implied by the n keys and the functions h_1, h_2. In that graph, there are m vertices $u_1, \ldots, u_m$ such that each k_i implies an edge $(u_{h_1(k_i)}, u_{h_2(k_i)})$. The encoding then *peels* that graph, by recursively removing each edge $(u_{h_1(k_i)}, u_{h_2(k_i)})$ for which the degree of either $u_{h_1(k_i)}$ or $u_{h_2(k_i)}$ is 1, and pushing that k_i to a stack. That process ends when the graph is empty of edges. Then, the *unpeeling* process iteratively pops an item k_j from the stack and uses it to fill the vector's entries: If both $S[u_{h_1(k_j)}]$ and $S[u_{h_2(k_j)}]$ are unassigned yet, then they are assigned random values such that $S[u_{h_1(k_j)}] + S[u_{h_2(k_j)}] = v_j$. Otherwise, if only $S[u_{h_2(k_j)}]$ is unassigned (w.l.o.g) then assign $S[u_{h_2(k_j)}] = v_j - S[u_{h_1(k_j)}]$. This process succeeds as long as the peeling indeed removes all edges. However, there is a high probability for the peeling process to end with a non-empty graph where none of the vertices is of degree 1. The size of the remaining graph is known to be with at most $O(\log n)$ vertices. This is solved by extending the vector S with extra $O(\log n) + \lambda$ entries.

In a concrete instantiation of PaXoS [PRTY20] the authors set $\varepsilon = 0.4$, which becomes standard in Cuckoo hashing based constructions. However, that assignment is heuristic, and no failure probability was proven. Encoding is linear in the number of items and decoding takes $2 + \frac{c \cdot \log n + \lambda}{2}$ time, for some constant c ([PRTY20] used $c = 5$).

4 New OKVS Constructions

The main issue that the new OKVS constructions aim to improve over the existing polynomial-based or random matrix OKVS constructions, is improving the run time to be linear in the number of key-value pairs. This comes at the cost of slightly increasing the size of the OKVS.

4.1 OKVS based on a 3-Hash Garbled Cuckoo Table (3H-GCT)

The PaXoS construction of [PRTY20] uses cuckoo hashing with two hash functions. It is well-known that the efficiency of cuckoo hashing improves significantly when using three rather than two hash functions (see orientability analysis, with $\ell = 1$ and $k \in \{2, 3\}$ in [Wal21a, Table 1]). Hence, in this section we suggest generalizing the OKVS construction to three hash functions. (It is crucial that the construction uses not more than three hash functions. We describe in Footnote 5 that using more functions will result in better memory and network utilization, but will not support an efficient linear time peeling algorithm for finding the right assignment of values to memory locations. Therefore, with current techniques it seems that using three hash functions is optimal.)

Peeling. The construction follows a basic peeling based approach. The OKVS data structure S is a hypergraph $\mathcal{G}_{3,n,m}$, with m nodes and n hyperedges, each touching 3 nodes. The construction uses three hash functions h_1, h_2, h_3, and maps each key k to the hyperedge $(h_1(k), h_2(k), h_3(k))$.[4] The simplest OKVS construction is binary, and encodes a pair (k, v) into the graph to satisfy the property that $v = S(h_1(k)) \oplus S(h_2(k)) \oplus S(h_3(k))$. Namely, the value associated with a key k is

[4]The hyperedge is sampled uniformly at random from all subsets of 3 *different* nodes in the graph. We simplify the notation by referring to hash functions h_1, h_2, h_3, but these functions are invoked together under the constraint that the outputs of the three hash functions are distinct from each other.

encoded as the exclusive-or of the three nodes of the hyperedge to which it is mapped. The number of nodes m must be at least the number of values n, and our aim is to make it as close as possible to n.

This mapping is possible if the binary $n \times m$ matrix in which each row represents a key and has 1 entry corresponding to the three nodes to which the key is mapped, is of rank n, and can be therefore be found in time $O(n^3)$. However, our goal is to compute a mapping in time which is close to linear. This is done by a peeling based algorithm: Suppose that there is a key k with a corresponding hyperedge $(h_1(k), h_2(k), h_3(k))$, and that, say, $h_2(k)$ is a node to which no other key is mapped. Then we can set values to all other nodes in the graph, including nodes $h_1(k)$ and $h_3(k)$, and afterwards set the value of node $h_2(k)$ so that the equality $v = S(h_1(k)) \oplus S(h_2(k)) \oplus S(h_3(k))$ holds. To denote this property we can orient the hyperedge towards $h_2(k)$. This property also means that we can remove this hyperedge from the graph, solve the mapping for all other keys, and then set the value of node $h_2(k)$ so that the mapping of k is correct. This can of course be done for all hyperedges that touch nodes of degree 1. Moreover, removing these hyperedges might reduce the degrees of other nodes, and this enables removing additional hyperedges from the graph.

The peeling process that we described essentially works by repeatedly choosing a node of degree 0 or 1 and removing it (and the incident edge if present) from the hypergraph. The removed edge is oriented towards the node. If this process can be repeated until all nodes are removed then the graph is said to be "peelable". Otherwise, the process ends with a 2-core of the hypergraph (the largest sub-hypergraph where all nodes have a degree of at least 2). We first discuss the expected number of nodes that is required to ensure that the peeling process can remove all edges. We then discuss how to handle the case that the peeling process ends with a non-empty 2-core.

Peelability threshold. It is well known that for random 3-hypergraphs, peelability asymptotically succeeds with high probability when the number of nodes is at least $1.23n$. (See [Mol04, BPZ13] for an analysis, and [GL19] for implementation and measurements.) A recent result in [Wal21b] shows that choosing hyperedges based on a specific different distribution reduces the number of nodes to be as low as $1.1n$, but based on experiments in [Wal21b] and on our experiments these results seem to be applicable only to very large graphs of tens of millions of nodes.)[5] Of course, we also wish to ensure that the OKVS construction fails with only *negligible* probability, or with a sufficiently small *concrete* probability ($2^{-\lambda}$, for $\lambda = 40$). The known analysis methods do not provide concrete parameters for guaranteeing a $2^{-\lambda}$ failure probability. We will describe in Section 5 how to amplify OKVS constructions in order to verify experimentally that failures happen with sufficiently small probability.

Handling the 2-core in binary 3-hash OKVS. Let $\chi(G)$ be the number of hyperedges in the 2-core of a hypergraph G with n edges, and let $d(n)$ be an upper bound on $\chi(G)$ which holds with overwhelming probability ($d(n)$ will typically be very small). The peeling stops working when reaching the 2-core. We follow [PRTY20] in using a datastructure of the form $S = L||R$, where L

[5]For uniformly random d-regular hypergraphs (we use $d = 3$), increasing d improves the threshold of memory utilization that enables mapping values to hyperedges. Namely, increasing d enables to use a graph of fewer nodes in order to successfully orient the same number of hyperedges towards different nodes. Successfully orienting the nodes implies that it is possible to assign values to nodes to enable the recovery all values associated with hyperedges. However, this does not imply that mapping values to nodes can be efficiently found in linear time, such as by running by a peeling process. Unfortunately, increasing the degree d also makes it harder to succeed in peeling, and requires a substantially higher ratio between the number of nodes and the number of hyperedges in order for peeling to succeed (see first row of Table 1 in [Wal21b].) Our construction is based on peeling, and therefore our usage of hyperedges of size $d = 3$ is optimal.

consists of the nodes of the hypergraph, and R includes additional $d(n) + \lambda$ nodes, where $2^{-\lambda}$ is the allowed statistical failure probability. The hypergraph construction maps each key k to 3 nodes in L. Denote these nodes using a binary vector $l(k)$ of length L, which has 3 bits set to 1. In addition, we use another hash function to map k to a random binary string $r(k)$ of length $d(n) + \lambda$, where the bits which are set to 1 indicate a subset of the nodes in R. The value of a key k from the OKVS is retrieved as the exclusive-or of the values of the 3 nodes to which it is mapped in L and the values of the nodes to which it is mapped in R, namely it is $(l(k)||r(k)) \cdot S$. Therefore the encoding process must set the values in S to satisfy these requirements.

After running the peeling process, we are left with $\chi(G) \leq d(n)$ hyperedges in a 2-core of G. We solve the system of linear equations $(l(k_i)||r(k_i)) \cdot S$ for all keys k_i whose corresponding hyperedges are in the 2-core.[6] Solving this system of equations sets values to the nodes in R, and to the nodes in L to which the edges in the 2-core are mapped. This can be done in $O((d + \lambda)^3)$ time. We can then run the peeling process in reverse: take the peeled hyperedges in reversed order and set values to the nodes in L to which they are oriented, to satisfy the decoding property for all other hyperedges in the graph. The entire algorithm is defined in Figure 2. The proof of Lemma 6 below is in Appendix A.1.

Lemma 6. *Let $d(n)$ be a parameter such that $\Pr[\mathcal{G}_{3,n,m}$ has 2-core $> d(n)] \leq \varepsilon_1$. Then the construction with $|R| = d(n) + \lambda$ is an OKVS with error $\varepsilon_1 + 2^{-\lambda}$.*

4.2 OKVS based on Simple Hashing and Dense Matrices

Another possible approach for constructing an OKVS is to randomly map the key-value pairs into many bins, and implement an independent OKVS per bin (using the polynomial-based or the random matrix approaches). The computation cost of these smaller OKVS instances is much smaller, and the space utilization only needs to take into account the maximum number of items that might be mapped into a bin.

Suppose we hash n pairs into m bins, where key-value pair (k, v) is placed into bin $h(k)$ based on a random function $h : \{0, 1\}^* \to [m]$. Encode each bin's set of key-value pairs into its own OKVS using any "inner OKVS" construction. The overall result is also an OKVS. More formally, if $(\mathsf{Encode}, \mathsf{Decode})$ is the inner OKVS, then given $(D_1, \ldots, D_m) \leftarrow \mathsf{Encode}(\{k_i, v_i\})$ the new OKVS is

$$\mathsf{Decode}^*\Big((D_1, \ldots, D_m), k\Big) \stackrel{\text{def}}{=} \mathsf{Decode}(D_{h(k)}, k)$$

The corresponding Encode^* is defined as explained above.

In choosing parameters for the inner OKVS, the naïve error analysis would proceed as follows. First compute a bound β such that all bins have at most β items except with the target ε probability. Choose parameters such that each bin's OKVS fails on β items with probability bounded by ε/m. Then by a union bound the entire encoding procedure fails with probability at most $m \cdot \varepsilon/m = \varepsilon$.

We can do better when the inner OKVS is a polynomial OKVS. If the field is small, we can use a random dense-matrix OKVS. For this OKVS the error probability within each bin drops

[6] An alternative approach is to use a graph without an R component, and try to solve the system of equations for the $l(k_i)$ nodes of the 2-core alone. However, experiments that we ran show that in many cases where the 2-core is small but not empty, the 2-core includes only two hyperedges. This means that these two hyperedges are mapped to exactly the same set of 3 nodes, and therefore the two associated linear equations are identical and cannot be solved.

We additionally note that PSI applications require using a Binary linear combination of the OKVS values. Other applications might allow using linear combinations with larger coefficients. In these cases there will likely be no need for adding the R nodes to the graph.

```
Encode($\{k_i, v_i\}$):
Parameters:

  • The algorithm is parameterized with the functions $H = \{h_1, h_2, h_3\}$, each has a range $[m]$.
  • In addition the algorithm uses the functions $l(\cdot)$ and $r(\cdot)$ where $l(x)$ outputs a bit-vector of length
    $m$ with zero at all entries except of entries $h_1(x), h_2(x)$ and $h_3(x)$. The function $r(x)$ outputs a
    random bit-vector of length $r$.

Algorithm:

  1. Initialize empty vectors $L \in \mathbb{F}^m$ and $R \in \mathbb{F}^r$.
  2. Initialize stack $P$.
  3. (Identify nodes which are touched by only a single hyperedge, and push them to $P$.) While there
     is a node $j \in [m]$ such that the set $\{k_i \notin P \mid j \in \{h_1(k_i), h_2(k_i), h_3(k_i)\}\}$ is a singleton: Let $k_i$ be
     the element of that singleton, and push $k_i$ onto $P$.
  4. Solve the system of equations $\langle l(k_i) \| r(k_i), L \| R \rangle = v_i$ for $k_i \notin P$, and assign the solutions to the
     corresponding locations in $S$.
  5. While $P$ not empty:
      (a) pop $k_i$ from $P$.
      (b) $L$ is undefined in at least one of the positions $h_1(k_i), h_2(k_i), h_3(k_i)$. Set the undefined
          position(s) so that $\langle l(k_i) \| r(k_i), L \| R \rangle = v_i$.
  6. Set any empty position in $L$ or $R$ with a random value from $\mathbb{F}$.
```

Figure 2: 3-Hash Garbled Cuckoo Table, fitting n key-value pairs (k_i, v_i) to a data structure $S \in \mathbb{F}^{m+r}$.

off gradually with the number of items (rather than having a sharp threshold). Suppose we have n items into m bins, and each bin is a dense-matrix OKVS with w slots (so that the entire data structure is mw in size). If exactly t items happen to be assigned to a particular bin, then that bin's OKVS fails with probability bounded by $|\mathbb{F}|^{w-t}$. Using the union bound, we bound the probability of the *overall* OKVS failing as:

$$m \cdot \Pr[\text{bin \#1 OKVS fails}] \leq m \sum_t \underbrace{\binom{n}{t} \left(\frac{1}{m}\right)^t \left(\frac{m-1}{m}\right)^{n-t}}_{\Pr[\text{bin \#1 holds exactly } t \text{ items}]} \min\left\{1, \frac{1}{|\mathbb{F}|^{w-t}}\right\}$$

It is straightforward to calculate this probability exactly, and it leads to better bounds on OKVS size.

Example. Consider the case of $|\mathbb{F}| = \{0, 1\}$, hashing $n = 1000$ items into $m = 100$ bins. How wide must each bin's dense-matrix OKVS be for an overall error probability of 2^{-40}? The naïve analysis proceeds as follows. With probability $1 - 2^{-40}$ all bins have at most 42 items. We must ensure $\Pr[\text{inner OKVS fails on 42 items}] < 2^{-47}$, so that the union bound over $m = 100$ bins bounds the overall failure probability by 2^{-40}. Hence, each bin must have $w = 42 + 47 = 89$ slots. In contrast, the more specialized analysis above shows that only $w = 61$ slots suffice per bin, for error probability 2^{-40} (a 31% improvement).

5 Amplifying OKVS Correctness

Premise: Empirically Measuring Failure Probabilities. The most efficient OKVS constructions are likely to be based on randomized constructions. Unfortunately, we lack techniques for finding tight concrete bounds of the relevant failure probabilities for constructions of this type, such as cuckoo hashing, and for choosing appropriate concrete parameters (*e.g.*, how many bins are needed to hash a concrete number of n items with k hash functions so that the 2-core of the cuckoo graph has size bounded by $2\log_2 n$ with probability $1-2^{-\lambda}$?[7][8]

The best we can currently hope for is to empirically measure failure probabilities. Since we seek data structures where the failure probabilities are extremely small (*e.g.*, 2^{-40}) empirical measurement is extremely costly. One would have to perform trillions of trials before expecting to see any failures at all. Alternatively, one must typically perform many trials with higher error probabilities, and extrapolate to the lower probabilities. This approach was used in, *e.g.*, [PSZ18, CLR17].

In this section we show methods for **amplifying** the probabilistic guarantees of an OKVS. For example, we show how to use an OKVS with failure probability ε to build an OKVS with failure probability $c \cdot \varepsilon^d$ (for explicit constants c, d). Think of ε as being moderately small, *e.g.*, $\varepsilon = 2^{-15}$, and therefore sufficiently large to enable running efficient empirical experiments to obtain 99.99% certainty about whether ε bounds the failure event. Using an OKVS with such an empirically-validated failure probability, we can construct a new OKVS with the desired failure probability (*e.g.*, 2^{-40}).

Since our amplification algorithms may instantiate two or more OKVS structures for the same set of keys and values, in this section we make the set of hash functions used in each instantiation explicit. That is, an OKVS scheme is a pair of algorithms $(\mathsf{Encode}_H, \mathsf{Decode}_H)$ as defined in Section 2.

In the following, we describe three amplification architectures for constructing a new OKVS scheme $(\mathsf{Encode}^*_H, \mathsf{Decode}^*_H)$ using an underlying OKVS scheme $(\mathsf{Encode}_H, \mathsf{Decode}_H)$. We assume that the OVKS is over a finite field and that randomly sampling a vector of appropriate length from that field samples a random OVKS. For the underlying scheme, we denote by $\mathsf{size}(n)$ the size of the resulting OKVS for encoding n items. (Recall that by the obliviousness property, it follows that the OKVS size depends only on the size of the key-value set and not on the keys themselves.) We note that the amplification constructions sometimes invoke Encode_H with a set of key-value pairs only to check whether encoding succeeds or fails, and do not necessarily use the outcome of that encoding. Recall that even though the input to Encode_H consists of key-value pairs, success or failure depend only on the keys.

5.1 Replication Architecture

The following construction is mainly described as a warmup towards more involved constructions, since it substantially increases the space requirements. The idea is to amplify the success probability by doubling the size and computation, by using two OKVS constructions and retrieving values as

[7]For cuckoo hashing, the relation between the number of items n, number of hash functions k, number of bins $m = (1+\beta)n$ for $\beta \in (0,1)$, stash size s, and the insertion failure probability ε, is proven in [KMW09]: for any $k \geq 2(1+\beta)\ln\frac{1}{\beta}$ and $s > 0$, mapping n items to $(1+\beta)n$ bins fails with probability $O(n^{1-c(s+1)})$ for a constant c and $n \to \infty$. However, the constants in the big "O" notation are unclear and therefore we do not know which concrete parameters are needed in order to instantiate such constructions.

[8]We stress that the failure events in Cuckoo hashing and in OKVS are slightly different. Specifically, an OKVS fails if the size of the 2-core is too large whereas CH can handle a large 2-core, as long as there are not too many intersecting cycles.

the sum of the retrieved values from both constructions. The encoding procedure checks if any of two random hash functions results in a successful OKVS for the given set of keys. The encoding fails only if both hash functions result in a failure. Its main disadvantage is the double space usage.

Formally:

- $\mathsf{Encode}^*_H(\{(k_i, v_i)\})$ views H as two sets of hash functions H_1 and H_2. It outputs two dictionaries S_1 and S_2 as follows:
  - Compute $S' \leftarrow \mathsf{Encode}_{H_1}(\{(k_i, v_i)\})$.
  - If $S' \neq \perp$: set $S_2 \leftarrow \mathbb{F}^{\mathsf{size}(n)}$ randomly, i.e. S_2 is a random OKVS independent of $\{(k_i, v_i)\}$. Then, define the set $\{(k_i, v'_i)\}$ where $v'_i = v_i - \mathsf{Decode}_{H_2}(S_2, k_i)$. Finally, compute $S_1 \leftarrow \mathsf{Encode}_{H_1}(\{(k_i, v'_i)\})$. We know that $S_1 \neq \perp$ (since $S' \neq \perp$ and S_1 uses the same set of keys as S') and therefore output $S = (S_1, S_2)$.
  - Otherwise ($S' = \perp$): set $S_1 \leftarrow \mathbb{F}^{\mathsf{size}(n)}$. Then, define the set $\{(k_i, v'_i)\}$ where $v'_i = v_i - \mathsf{Decode}_{H_1}(S_1, k_i)$ and compute $S_2 \leftarrow \mathsf{Encode}_{H_2}(\{(k_i, v'_i)\})$. If $S_2 \neq \perp$ then output $S = (S_1, S_2)$, otherwise, output $\perp$.
- $\mathsf{Decode}^*_H(S, x)$: Interpret $H = (H_1, H_2)$ and $S = (S_1, S_2)$. Output $y = \mathsf{Decode}_{H_1}(S_1, x) + \mathsf{Decode}_{H_2}(S_2, x)$.

Clearly, this construction only fails if both encodings fail. Therefore, if $(\mathsf{Encode}, \mathsf{Decode})$ fails with probability ε then $(\mathsf{Encode}^*, \mathsf{Decode}^*)$ fails with probability ε^2.

Generalization The above construction uses two 'replicas'. It could be generalized to $c > 2$ replicas, resulting in an OKVS of size $c \cdot \mathsf{size}(n)$, failure probability ε^c and overall encode/decode time that is c times greater than the underlying scheme. Denote an OKVS scheme with c replicas by $(\mathsf{Encode}^{*c}, \mathsf{Decode}^{*c})$. We use such a scheme in the generalized construction described below (Section 5.3).

The obvious undesirable property of this construction is that the size of the OKVS increases by a factor of c. (This is also true for the encoding and decoding times, but these performance parameters are typically less critical since they are small for hashing-based OKVS.) In the rest of this section we describe how to amplify the failure probability from ε to ε^c while keeping the size of the resulting OKVS not much larger than the underlying OKVS (certainly not larger by a factor of c).

5.2 Star Architecture

We next show how to reduce the error probability while keeping the OKVS size to be almost $\mathsf{size}(n)$. In our concrete instantiation (presented in Section 8) we are able to almost square the failure probability while increasing the OKVS size by less than 10% for $n = 2^{20}$ items.

At the high-level idea, imagine a star-shaped graph consisting of $q + 1$ nodes, one central node and q leaves. Each node, including the central node, is associated with an OKVS data structure and should be large enough to store about n/q items. Each item is retrieved from one leaf node and from the root node, and the returned value is the sum of the two retrieved values. More precisely, to probe for an item x, probe for x in the central OKVS and probe for x in the OKVS of leaf $\tilde{h}(x)$ (where $\tilde{h}$ is a random function), and add the results. The construction is robust to a hashing failure of a single node since we can set that node to have random values and can still set the values of all the other nodes to ensure that the correct sums are returned (this is true for either a leaf node or the root node). Therefore the system fails only if at least two nodes fail. Security holds since one node is set to be random, while the other nodes store random OKVS values.

Formally, the new OKVS scheme is defined in the following way: Let n' be an upper bound on the maximum load of a bin when mapping n balls into q bins, except with probability $2^{-\lambda}$. In the following description we treat the first OKVS (indexed by 0) as the center node, and the following q OKVS's, indexed 1 to q, as the leaf nodes.

- $\mathsf{Encode}^*_H(\{(k_i, v_i)\})$: Interpret $H = (\tilde{h}, H_0, \ldots, H_q)$.
  - Map the set $\{(k_i, v_i)\}$ to q subsets: $A_1, \ldots, A_q$ where $A_j = \{(k_i, v_i) \mid \tilde{h}(k_i) = j\}$.
  - For $j = 1, \ldots, q$ compute $S_j \leftarrow \mathsf{Encode}_{H_j}(A_j)$
  - **No failure.** ($\forall_{j\in[q]} : S_j \neq \perp$) In this case, set random values to the central node and adjust the values of other nodes accordingly.
    - Sample a random S_0 from $\mathbb{F}^{\mathsf{size}(n')}$.
    - For $j \in [q]$ compute the new set $A'_j = \{(k, v') \mid (k, v) \in A_j\}$ where $v' = v - \mathsf{Decode}_{H_0}(S_0, k)$; then, compute $S_j \leftarrow \mathsf{Encode}_{H_j}(A')$.
  - **One failure.** ($\exists_{j^*} : S_{j^*} = \perp \wedge \forall_{j\in[q]\setminus\{j^*\}} : S_j \neq \perp$) In this case, set the central node to ensure the correct decoding of the values mapped to the failed node, and adjust the values of other nodes accordingly.
    - Sample a random S_{j^*} from $\mathbb{F}^{\mathsf{size}(n')}$.
    - Compute a new set $A'_0 = \{(k, v') \mid (k, v) \in A_{j^*}\}$ where $v' = v - \mathsf{Decode}_{H_{j^*}}(S_{j^*}, k)$ and then $S_0 \leftarrow \mathsf{Encode}_{H_0}(A'_0)$. If $S_0 = \perp$ then output $S = \perp$ and halt.
    - For $j \in [q] \setminus \{j^*\}$ compute the new set $A'_j = \{(k, v') \mid (k, v) \in A_j\}$ where $v' = v - \mathsf{Decode}_{H_0}(S_0, k)$; then, compute $S_j \leftarrow \mathsf{Encode}_{H_j}(A')$.
  - **Two or more failures.** If $S_j = \perp$ for more than one OKVS j then output $S = \perp$ and halt.
  - Output $S_0, \ldots, S_q$.
- $\mathsf{Decode}^*_H(S, x)$: Interpret $H = (\tilde{h}, H_0, \ldots, H_q)$ and $S = (S_0, \ldots, S_q)$. Compute $j = \tilde{h}(x)$ and output $y = \mathsf{Decode}_{H_j}(S_j, x) + \mathsf{Decode}_{H_0}(S_0, x)$.

Failure probability The construction can tolerate a failure in any one of the $q + 1$ components (either a leaf or the center node). In other words, the new construction fails only when *two* of the $q + 1$ components fail. So if each of the underlying OKVS instances fails with probability ε, then the new construction fails with probability

$$\Pr[S = \perp] = \sum_{i=2}^{q+1} \binom{q+1}{i} \varepsilon^i (1 - \varepsilon)^{q+1-i} \tag{3}$$

$$= 1 - (1 - \varepsilon)^{q+1} - (q + 1)\varepsilon(1 - \varepsilon)^q \tag{4}$$

Looking at equation 3 and ignoring high order terms, we observe that if the failure probability of the underlying OKVS scheme is $\varepsilon = 2^{-\rho}$ then the failure probability of the star architecture is $\approx \binom{q+1}{2}\varepsilon^2 = 2^{\log\binom{q+1}{2} - 2\rho}$. Thus, in order for the star architecture to fail with probability $2^{-\lambda}$ we need $\log\binom{q+1}{2} - 2\rho = -\lambda$ and thus $\rho = \frac{\lambda + \log\binom{q+1}{2}}{2} \approx \frac{\lambda + 2\log(q) - \log 2}{2} \approx \lambda/2 + \log(q)$.

OKVS size and encoding/decoding time The size of the new OKVS is $(q+1) \times \mathsf{size}(n')$ where n' is the upper bound on the maximum load when mapping n balls to q bins, that is,

$$n' = \min_{\tilde{n}} : \Pr[\text{“there exists bin with} \geq \tilde{n} \text{ elements”}] \leq 2^{-\lambda} \tag{5}$$

where

$$\begin{aligned} \Pr[\text{“there exists bin with} \geq \tilde{n} \text{ elements”}] &\leq \sum_{i=1}^{q} \Pr[\text{“bin } i \text{ has } \geq \tilde{n} \text{ elements”}] \\ &= q \cdot \sum_{i=\tilde{n}}^{n} \binom{n}{i} \left(\frac{1}{q}\right)^{i} \left(1 - \frac{1}{q}\right)^{n-i} \end{aligned}$$

These equations enable to easily compute the maximal size $\tilde{n}$ of the bins. Note that since the number of bins q is typically very small compared to n, then $\tilde{n}$ is not much greater than the expected size of a bin which is n/q. Section 5.4 shows a concrete size analysis for a specific choice of parameters.

The new encoding requires at most $2q+1$ invocations of the underlying encoding algorithm. Decoding works exactly as in the replication architecture, with 2 calls to the underlying decoding algorithm.

5.3 Generalized Star Architecture

In this section we improve the amplification method to achieve a failure probability of $O(\varepsilon^d)$ for an arbitrary d. This enables to weaken the requirement from the underlying scheme, and only require that it fails with probability of at most $\varepsilon = O(2^{-\lambda/d})$ instead of $\varepsilon = O(2^{-\lambda/2})$. This is an important step if we wish to use an underlying OKVS scheme for which the failure probability is *empirically* proven, like our 3-hash garbled cuckoo table scheme presented in Section 4.1. The larger d is, the less experiments we have to conduct in order to empirically prove a failure probability of ε for the overall scheme.

The generalized idea is exactly the same as the star architecture, except that the center OKVS can tolerate up to $d-1$ failures of the OKVS instances in the leaves. The new OKVS is composed of two components: (1) q leaf nodes as before, each of size $\mathsf{size}(n')$, and (2) a center node of size $d \cdot \mathsf{size}(n')$ (whereas in the simple star architecture the center is of size only $\mathsf{size}(n')$). The center node uses the replicated scheme ($\mathsf{Encode}^{*d}, \mathsf{Decode}^{*d}$) described in Section 5.1. We require that both components fail with negligible probability in λ. Specifically, in order for the entire scheme to fail with probability $2^{-\lambda}$ each component has to fail with probability $2^{-(\lambda+1)}$.

The formal description of the new OKVS scheme is as follows:

- $\mathsf{Encode}^{*}_{H}(\{(k_i, v_i)\})$: Interpret $H = (\tilde{h}, \hat{H}, H_1, \ldots, H_q)$, then,
  - Map the set $\{(k_i, v_i)\}$ to q subsets: $A_1, \ldots, A_q$ where $A_j = \{(k_i, v_i) \mid \tilde{h}(k_i) = j\}$.
  - For $j = 1, \ldots, q$ compute $S_j \leftarrow \mathsf{Encode}_{H_j}(A_j)$ and record the set $F = \{j \mid S_j = \perp\}$ (the indices of leaf nodes for which encoding failed).
  - **Too many failures.** If $|F| \geq d$: output $S = \perp$ and halt.
  - **Otherwise.** If $|F| < d$:
    - For all $j \in F$ sample a random S_j from $\mathbb{F}^{\mathsf{size}(n')}$. (This procedure sets random values for all failed OKVS nodes.)

  * Define the set $\hat{A} = \bigcup_{j \in F} A_j$ of all items in the failed OKVS nodes. Compute a new set $A'_0 = \{(k, v')\}$ which contains for each $k \in \hat{A}$ the pair (k, v') where $v' = v - \mathsf{Decode}_{H_j}(S_j, k)$ where $j = \tilde{h}(k)$. (This ensures that the central node corrects the value assigend for the key in the node OKVS.)
    Set $\hat{S} \leftarrow \mathsf{Encode}_{\hat{H}}(A')$. If $\hat{S} = \perp$ then output $S = \perp$ and halt.
  * For $j \in [q] \backslash F$, define the set $A'_j = \{(k, v') \mid (k, v) \in A_j\}$ where $v' = v - \mathsf{Decode}_{\hat{H}}(\hat{S}, k)$ and compute $S_j \leftarrow \mathsf{Encode}_{H_j}(A'_j)$.
  * Output $S = (S_1, \dots, S_q, \hat{S})$.

- $\mathsf{Decode}^*_H(S, x)$: Interpret $H = (\tilde{h}, H_1, \dots, H_q, \hat{H})$ and $S = (S_1, \dots, S_q, \hat{S})$. Compute $j = \tilde{h}(x)$ and output $y = \mathsf{Decode}_{H_j}(S_j, x) + \mathsf{Decode}^{*d}_{\hat{H}}(\hat{S}, x)$.

In the description used above we denoted the central node's OKVS by $\hat{S}$ instead of S_0 as in the simple star architecture, to emphasize the fact that the central node is encoded using a stronger OKVS, namely a replicated OKVS scheme $(\mathsf{Encode}^{*d}, \mathsf{Decode}^{*d})$.

Failure probability The generalized star architecture fails if either the leaf nodes OKVS constructions or the central OKVS fail. Thus, we require that each component fails with probability $2^{-(\lambda+1)}$.

Let ε be the failure probability of the underlying OKVS scheme $(\mathsf{Encode}, \mathsf{Decode})$. The first component, with q leaf nodes, fails when $|F| \geq d$, which happens with probability $\sum_{i=d}^{q} \binom{q}{i} \varepsilon^i (1 - \varepsilon)^{q-i} = O(\varepsilon^d)$. The second component, which is a scheme with d replicas, fails with probability ε^d, corresponding to the event where all replicas fail.

OKVS size and encoding/decoding time The size of the new OKVS is $q \cdot \mathsf{size}(n') + \mathsf{size}^{*d}(n')$ where $\mathsf{size}(n')$ and $\mathsf{size}^{*d}(n')$ are the sizes of the resulting OKVS for the $(\mathsf{Encode}, \mathsf{Decode})$ and $(\mathsf{Encode}^{*d}, \mathsf{Decode}^{*d})$ schemes, respectively. The value n' is the upper bound on the maximum load when mapping n balls to q bins, as presented in Eq. (5).

The new encoding requires $2q$ invocations of Encode algorithm for the leaf nodes and a single invocation of Encode^{*d}. The new decoding requires one invocation of Decode and one invocation of Decode^{*d}.

5.4 A Concrete Instantiation

The underlying scheme $(\mathsf{Encode}_H, \mathsf{Decode}_H)$ is instantiated using the scheme of Section 4.1 where the resulting OKVS, when encoded using n' items, is $S = L \| R$ where $|L| = 1.3n$ and $|R| = \lambda + 0.5 \log n$ (i.e. $\mathsf{size}(n') = 1.3n' + \lambda + 0.5 \log n'$). In this scheme an encoding 'failure' happens when the 2-core which remains after peeling is of size larger than $0.5 \log n'$.

We conducted 2^{33} runs of such a scheme with $n' = 6600$, using different sets of hash functions in each run. There was only a single run in which the 2-core was greater than $0.5 \log n'$. By the Clopper-Pearson method [CP34], we get that for a random set of hash function H

$$\varepsilon = \Pr[\mathsf{Encode}_H(\{(k_i, v_i)\}) = \perp] = 2^{-29.355}$$

with confidence level of 0.9999.

We can use that result in order to construct a new scheme $(\mathsf{Encode}^*_H, \mathsf{Decode}^*_H)$ using the star architecture (Section 5.2, replication factor is $d = 1$, i.e., no replication):

- $n = 2^{16}$. We use $q = 10$ bins. Then, the maximum load according to Eq. (5) is $n' = 7117$, for which the above experiment applies[9]. Thus, the failure probability of the new scheme, according to equation (3), is $2^{-52.9}$.
- $n = 2^{20}$. We use $q = 160$ bins. Then, the maximum load according to Eq. (5) is $n' = 7163$. Thus, the failure probability of the new scheme, according to equation (3), is $2^{-45.05}$.

In both cases, the space usage is $(q+1) \cdot (1.3n/q + \lambda + 0.5 \log(n/q)) \approx 1.3n$.

6 Applications of OKVS

In this section we discuss how OKVS can be used as a drop-in replacement for polynomials in many protocols.

6.1 Sparse OT Extension

Pinkas *et al.* (SpOT-light [PRTY19]) proposed a semi-honest PSI protocol with very low communication, based on oblivious transfer techniques. Suppose the PSI input sets are of size n, and hold items from the universe $[N]$. There is a natural protocol for PSI that uses N OTs, where the receiver uses choice bit 1 in only n of them and choice bit 0 in the rest. This protocol will have cost proportional to N because communication is required for each OT, making it unsuitable for exponential N. The work in [PRTY19] introduces a technique called *sparse OT extension*, which reduces this cost.

Suppose the N OTs are generated with IKNP OT extension [IKNP03]. In IKNP, the receiver sends a large matrix with N rows. The parties perform the ith OT by referencing only the ith row of this matrix. Consider the mapping $i \mapsto [i\text{th row of IKNP matrix}]$. In the PSI protocol, the receiver only cares about n out of the N values of this mapping. So instead of sending the entire mapping (*i.e.*, the entire IKNP matrix), the receiver sends a polynomial P that satisfies $P(i) = [i\text{th row of matrix}]$, for the i-values of interest. Crucially, the communication has been reduced from N rows' worth of information to only n.

When the IKNP matrix is encoded in this way, the result is the $\mathsf{spot\text{-}low}$ PSI protocol of [PRTY19]. Any OKVS may replace the use of a polynomial in $\mathsf{spot\text{-}low}$.[10]

6.2 Oblivious Programmable PRF and its Applications

Kolesnikov *et al.* [KMP+17] introduced a primitive called **oblivious programmable PRF (OPPRF)**. In an OPPRF, the sender has a collection of n pairs of the form $x_i \mapsto y_i$, and the receiver has a list of x'_i values. The functionality chooses a pseudo-random function R, conditioned on $R(x_i) = y_i$ for all i. It gives (a description of) R to the sender and it gives $R(x'_i)$ to the receiver, for each i. In [KMP+17] a natural OPPRF protocol is described, based on polynomials. The parties invoke a (plain) oblivious PRF protocol, where the sender learns a PRF seed s and the receiver learns $PRF(s, x'_i)$ for each i. Then the sender interpolates a polynomial P containing "corrections" of the form $P(x_i) = PRF(s, x_i) \oplus y_i$, and sends it to the receiver. Now both parties define the function

[9] We assume that if $\Pr[\mathsf{Encode}_H(\{(k_i, v_i)\}) = \perp] = \varepsilon$ for encoding n' items then the same probability ε applies also to $n'' > n'$.

[10] [PRTY19] describe another protocol, $\mathsf{spot\text{-}fast}$, which also uses polynomials. Instead of using one polynomial of large degree n, $\mathsf{spot\text{-}fast}$ uses many polynomials of very small degree (and by this incurs a larger communication overhead). Due to the low degree, replacing these polynomials with an OKVS would have minimal effect.

$R(x) \overset{\text{def}}{=} PRF(s,x) \oplus P(x)$, which indeed agrees with the $x_i \mapsto y_i$ mappings of the receiver but is otherwise pseudo-random. In this application it is of course crucial that P hides the points which were used for interpolating it. Naturally, any OKVS can replace the polynomial in the OPPRF construction.[11]

Applications. [KMP+17] used an OPPRF to construct the first concretely efficient multi-party PSI. They described two protocols: The first protocol is fully secure against semi-honest adversaries. The second is more efficient but proven secure in a weaker *augmented semi-honest* model, where the corrupt parties are assumed to run the protocol honestly, but the simulator in the ideal world is allowed to change the inputs of corrupt parties. Intuitively, the protocol leaks no more to a semi-honest party than what can be learned by using *some input* (not necessarily the one they executed the protocol on) in the ideal model. We discuss this latter protocol in more detail in Section 7.2, where we show that, surprisingly, the protocol is secure against malicious adversaries despite not being secure in the semi-honest model.

OPPRF is also used in the PSI protocol in [PSTY19] for circuit PSI – computing arbitrary functions of the intersection rather than the intersection itself. In this protocol the overall effect of the OPPRF (and hence OKVS) is minor, accounting for only $\sim$1% of the total running time and communication. They also show an optimization, which can be viewed as its own OKVS constrution, where key-value pairs are hashed into bins, and a polynomial is interpolated for each bin. OPPRF is also used in the recent multi-party PSI protocols of Chandran *et al.* [CGS21, CDG+].

In a private set union protocol [KRTW19], a variant of OPPRF is used to perform a functionality of reverse private membership test. The functionality allows a party holding the set X to learn whether an input y of another party is in X, and nothing else. [KRTW19] also rely on simple hashing to improve the computation of the polynomial-based OKVS.

Finally, [RS21] proposes a new OPRF-based PSI protocol. Their construction combines a vector OLE with the PaXoS construction. We observe that it is possible to replace their use of PaXoS with any abstract OKVS, and with our new OKVS constructions in particular.

6.3 PaXoS PSI

The leading malicious 2-party PSI protocol is due to [PRTY20], and is known as PaXoS-PSI. The underlying data structure, a **probe and XOR of strings (PaXoS)**, is what we call a *binary OKVS* in this work. Their protocol and proofs are written in terms of an arbitrary PaXoS data structure, with definitions that are identical to the ones we require of a binary OKVS. Hence, the improved constructions of binary OKVS that we present in this work automatically give an improvement to the PaXoS-PSI protocol. We have implemented these improvements to PaXoS-PSI, and report on their concrete performance in Section 8.2.

In Section 7 we discuss more details of the PaXoS PSI protocol, and also introduce a new generalization that can take advantage of a non-binary OKVS.

[11]Besides encoding these "corrections" as a polynomial, [KMP+17] actually propose two other methods. One method is a garbled Bloom filter [DCW13a], which is indeed an OKVS (with expansion λ). Another method that they refer to as the "table" construction is not a true OKVS, as it only is oblivious when the mapping $k_i \mapsto v_i$ is such that all of the k_i (not just the v_i) are uniformly distributed except possibly one k_i which can be known to the distinguisher. As such, this "table" construction is suitable only when the receiver learns one output from the underling OPRF/OPPRF.

6.4 Covert Computation

Covert computation is an enhanced form of MPC (not to be confused with the definition of covert security) which ensures that participating parties cannot distinguish protocol execution from a random noise, until the protocol ends with a desired output. The constructions in [MPP10, CDJ16] enable two parties to run multiple such computations in linear time, while keeping the covertness property. The challenge is identifying the correspondence between the protocol invocation sets of both parties. This is solved using a primitive called Index-Hiding Message Encoding (IHME). The constructions in [MPP10, CDJ16] convert a protocol for single-input functionality into a secure protocol for multi-input functionality, by encoding as value $P(x)$ of a polynomial P the protocol message for input x. (Here, the polynomial P implements the IHME primitive.) The usage of a polynomial can be replaced by any OKVS, to result in improved performance.

7 Other PSI Improvements

We present several improvements to leading PSI schemes which use OKVS.

7.1 Generalizing PaXoS-PSI to Linear OKVS

The PaXoS-PSI protocol [PRTY20] uses any *binary* OKVS data structure. We now present a generalization that can support any *linear* (not necessarily binary) OKVS. First, we review the protocol to understand its restriction to binary OKVS: The PaXoS-PSI protocol starts with the parties invoking the malicious OT-extension protocol of Orrú, Orsini & Scholl [OOS17]. The receiver chooses a vector of strings $D = (d_1, \ldots, d_m)$, and learns an output vector $R = (r_1, \ldots, r_m)$. The sender chooses a random string s and learns output $Q = (q_1, \ldots, q_m)$. The important correlation among these values is:

$$r_i = q_i \oplus C(d_i) \wedge s \tag{6}$$

where C is a binary, linear error correcting code with minimum distance κ, and $\wedge$ denotes bitwise-AND.

If we view D, R, and Q as OKVS data structures, we will see that equation (6) is compatible with the homomorphic properties of a binary OKVS (see Section 2.3). Hence:

$$\mathsf{Decode}(R, k) = \mathsf{Decode}(Q, k) \oplus C\big(\mathsf{Decode}(D, k)\big) \wedge s$$

Now, suppose the receiver has chosen their input D (an OKVS) so that $\mathsf{Decode}(D, y) = H(y)$, for each y in their PSI input set, where H is a random oracle. Suppose that for each x in their set, the sender computes

$$m_x = H'\Big(\mathsf{Decode}(Q, x) \oplus C(\mathsf{Decode}(D, x)) \wedge s\Big),$$

where H' is a random oracle. If that x is in the intersection, then the receiver can also compute/recognize m_x, since it is equal to $H'(\mathsf{Decode}(R, x))$. If x is not in the intersection, then $\mathsf{Decode}(D, x) = H(x) \oplus \delta$ for some nonzero string δ. Then through some simple substitutions, we get $m_x = H'(\mathsf{Decode}(R, k) \oplus C(\delta) \wedge s)$.

When H' is a correlation-robust hash function, values of the form $H'(a_i \oplus b_i \wedge s)$ are indistinguishable from random, when each b_i has hamming weight at least κ (as is guaranteed by the code) and s is uniform. In other words, when the sender has an item x and computes m_x, this value looks random to the receiver.

Binary OKVS and the generalization. Revisiting equation (6), we see that the relation $r_i = q_i \oplus C(d_i) \wedge s$ is homomorphic with respect to xor:

$$r_i \oplus r_j = (q_i \oplus q_j) \oplus C(d_i \oplus d_j) \wedge s.$$

This is what makes these correlated values compatible with a binary OKVS. However, if we view all strings as elements of a binary field, we see that more general linear combinations of r_i's do not work because the $\wedge$ operation is *bit-wise*, i.e. it is not compatible with the field operation.

The fact that $\wedge$ is not a field operation is also the reason for the error-correcting code C in the expression $r_i = q_i \oplus C(d_i) \wedge s$. For any nonzero d_i, we use the fact that $C(d_i) \wedge s$ is an expression with at least κ bits of uncertainty (*i.e.*, we are bitmasking at least κ bits of s).

Now suppose that the parties had values that were not correlated according to equation (6), but instead used a field operation $\cdot$ in place of $\wedge$:

$$r_i = q_i \oplus d_i \cdot s \tag{7}$$

Then we could view D, R, and Q each as OKVS data structures, and if they were linear OKVS we would have:

$$\mathsf{Decode}(R, k) = \mathsf{Decode}(Q, k) \oplus \mathsf{Decode}(D, k) \cdot s.$$

Additionally, for any a_i, b_i pairs with nonzero b_i, a value of the form $H(a_i \oplus b_i \cdot s)$ would look random to the receiver.

Indeed, replacing the correlation of equation (6) with that of (7) and using any *linear* (not necessarily binary) OKVS will lead to a secure PSI protocol whose proof follows closely to PaXoS-PSI. Additionally, since an error-correcting code is not needed, communication is reduced relative to PaXoS-PSI. A protocol that generates correlations that follow equation (7) is called a **vector oblivious linear evaluation (vOLE)** protocol [BCGI18, BCG+19, SGRR19]. Our protocol would require a malicious-secure vOLE protocol, but to date no such vOLE has been implemented. We leave it to future work to determine whether a vOLE-based approach will be competitive with the original PaXoS (OT-extension) approach.

Theorem 7. *If* (Encode, Decode) *is a linear OKVS, and other parameters ℓ_1, ℓ_2 are as in [PRTY20], then the protocol in Figure 3 securely realizes 2-party PSI against malicious adversaries.*

7.2 Malicious Multi-Party PSI

Multi-party Private Set Intersection($\mathcal{F}_{\text{m-psi}}$) allows a set of parties, each with a private set of items (P_i owns a set X_i), to learn the intersection of their sets $X_0 \cap X_1 \cap \cdots \cap X_n$ and nothing beyond that. The work of Kolesnikov *et al.* in [KMP+17] presents generic transformations from any 2-party oblivious PRF to a multi-party PSI protocol. One of these transformations is secure in the semi-honest model, and a more efficient transformation is secure in the weaker "augmented semi-honest" model, in which the ideal-world simulator is allowed to change the inputs of the corrupt parties. Here we observe that this more efficient protocol can actually be made secure in the **malicious model** with only a minor modification (post-processing of the OPRF outputs with a random oracle).

Malicious-secure but not Semi-honest secure? Here, we briefly address this apparent paradoxical situation of a protocol being malicious-secure but not semi-honest secure. For a semi-honest secure protocol the simulator cannot change the inputs of the corrupt parties; that is, it should be able to explain any well-defined input provided by the environment on behalf of the corrupt parties. We

Parameters:

- Computational and statistical security parameters κ and λ
- Sender with set $X \subseteq \{0,1\}^*$ of size n
- Receiver with set $Y \subseteq \{0,1\}^*$ of size n
- Linear OKVS scheme $(\mathsf{Encode}, \mathsf{Decode})$ mapping n items to m slots
- Random oracles $H_1 : \{0,1\}^* \to \{0,1\}^{\ell_1}$ and $H_2 : \{0,1\}^* \to \{0,1\}^{\ell_2}$

Protocol:

1. The parties invoke the vOLE functionality where the sender's input is random string $s \leftarrow \{0,1\}^{\ell_1}$ and the receiver's input is:
$$D = (d_1, \ldots, d_m) = \mathsf{Encode}(\{(y, H_1(y)) \mid y \in Y\}).$$
As a result, the sender obtains output $Q = (q_1, \ldots, q_m)$ and the receiver obtains output $R = (r_1, \ldots, r_m)$ satisfying $q_i = r_i \oplus d_i \cdot s$, with $\cdot$ denoting the field operation in $GF(2^{\ell_1})$.
2. The sender computes and sends a random permutation of the set
$$M = \Big\{ H_2\Big(x, \mathsf{Decode}(Q, x) \oplus H_1(x) \cdot s\Big) \;\Big|\; x \in X \Big\}.$$
3. The receiver coutputs $\{y \in Y \mid H_2(y, \mathsf{Decode}(R, y)) \in M\}$.

Figure 3: Our generalized PaXoS-PSI protocol, adapted from [PRTY20]

can interpret the "augmented semi-honest" secure protocol as "the protocol is semi-honest secure apart from the issue of simulators changing inputs". In contrast, simulators changing a corrupt party's inputs is no issue while proving malicious-security. It just so happens, that without the issue of "simulators changing inputs" the protocol in [KMP+17] is malicious-secure.

We discuss the protocol in detail in Appendix B, as well as its cost analysis, proof of security and possible extensions. We also discuss there the interesting interaction between semi-honest and malicious security.

To the best of our knowledge, [ZLL+19, BENOPC] are the only other works that study concretely efficient malicious multi-party PSI. Their constructions rely heavily on BF/GBF, which is the most communication-expensive construction amongst the three PSI constructions presented in [KMP+17]. While our protocol achieves almost the same cost as that of the most efficient construction in [KMP+17], with only a minor (inexpensive) modification, the protocols of [ZLL+19] and [BENOPC] are about 10× and 2× slower than [KMP+17]. We present a more detailed qualitative comparison with the recent work of [BENOPC] in Appendix B.

8 Concrete Performance

We now benchmark different OKVS constructions and our PSI schemes. We also present a comparison based on implementations of state-of-the-art semi-honest and malicious PSI protocols. We used the implementation of semi-honest protocols (KKRT [KKRT], SpOT-low and SpOT-fast [PRTY19], CM [CM20]) and malicious protocols (RR [RR17b], PaXos [PRTY20]) from the open source-code provided by the authors, and perform a series of benchmarks on the range of set size $n = \{2^{12}, 2^{16}, 2^{20}\}$. All cuckoo hash functions are public parameters of the protocols, and can be simply implemented as one party chooses the hash functions and broadcasts them to other parties.

We assume there is an authenticated secure channel between each pair of participants (e.g., with TLS). We evaluated the PSI protocols over three different network settings (so-called fast, medium, slow networks). The LAN setting (i.e, fast network) has two machines in the same region (N.Virginia) with bandwidth 4.6 Gib/s; The WAN1 (i.e, medium network) has one machine in Ohio and the other in Oregon with bandwidth 260 Mib/s; and the WAN2 (i.e, slow network) has one machine in Sao Paolo and the other in Sydney with bandwidth 33 Mib/s. While our protocol can be parallelized at the level of bins, all experiments, however, are performed with a single thread (with an additional thread used for communication). In all tables and figures of this section,"SH" and "M" stand for semi-honest and malicious, respectively. We describe detailed *microbenchmarking* results for OKVS in Appendix A.2.

8.1 Parameters for OKVS and PSI

Some OKVS schemes ('bins+polynomials', 'bins+dense matrix' and 'star architecture' in Table 2; 'SpOT-fast' and 'star arch.' in Table 1) rely on a simple hashing which maps n pairs into m bins. The number of items assigned of any bin leaks a distribution about input set. Therefore, all bins must be padded to some maximum possible size. Using a standard ball-and-bin analysis based on the input size and number of bins, one can deduce an upper bound bin size m such that no bin contains more than m items with high probability $1 - 2^{-\lambda}$. When n balls are mapped at random to m bins, the probability that the most occupied bin has μ or more balls is $m\binom{n}{\mu}\frac{1}{m^{\mu}}$ [RS98, PSZ14]. We provide our choices of μ for which the probability of a bin overflow is most $1 - 2^{-\lambda}$, as well as other relevant parameters for the OKVS schemes and PSI protocols in Figure 4.

n		2^{12}	2^{16}	2^{20}
Simple hashing	#bins (m)	10	100	2000
	bin size (μ)	555	854	714
GBF	# hash functions	40		
	table size	$60n$		
2hf Cuckoo expansion		$2.4n$		
3hf Cuckoo expansion		$1.3n$		
codeword length (SH)		448	473	495
codeword length (M)		627	616	605
ℓ_2 (SH) (see [PRTY20])		64	72	80
ℓ_2 (M) (see [PRTY20])		256		
λ		40		

Figure 4: Parameters for OKVS and PSI.

A garbled Bloom filter (GBF) [DCW13a] fails if a false-positive even occurs. Using λ hash functions and a vector of size $1.44\lambda n$ results in a failure probability of $1/2^{\lambda}$ [MU05]. Therefore, we use λ hash functions and an OKVS table size of $60n$. We use $m = 2.4n$ and $m = 1.3n$ bins as the acceptable heuristic for the PaXoS and 3H-GCT OKVS constructions, respectively, and the PSI protocols that use them. We use the concrete parameters for the star architecture based OKVS that are described in Section 5.4.

8.2 Improving PSI Protocols

A detailed benchmark and comparison of different PSI protocols is given in Table 1. Note that the SpOT-low [PRTY20] and RR [RR17b] protocols run out of memory for set size $n = 2^{20}$, and are not included in the comparison for this case.

Communication improvement. The overall communication of our 3H-GCT and star-arch. based malicious PSI is 1.61× and 1.43×, respectively, less than the previous state of the art, PaXoS. This is greatly due to the fact that our protocols invoke $1.3n$ and $1.41n$ OTs, respectively, compared to $2.4n$ in PaXoS.

Computation improvement. Over fast networks (4.6Gbits/sec) and $n = 2^{20}$, our protocol is only 1.05×–1.1× slower than the fastest PSI protocols (KKRT and PaXoS), where the running time is

Protocol	**Sett.**	**comm (MB)**			**4.6 Gbits/sec**			**260 Mbits/sec**			**33 Mbits/sec**		
		2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}	2^{12}	2^{16}	2^{20}
KKRT [KKRT]		0.48	7.73	128.49	201	**368**	**4512**	665	2390	12568	4352	10220	146067
SpOT-low [PRTY19]		**0.25**	**3.9**	**63.18**	495	10035	220525	894	11154	—	3406	20337.7	—
SpOT-fast [PRTY19]		0.3	4.61	76.46	**173**	1795	24676	678	7455	26050	4364	17923	38737
PaXoS-2hf (2-core) [PRTY20]	SH	0.59	9.9	169.67	217	410	4680	443	1395	11935	1974	8448	60159
CM* [CM20]		0.36	5.34	87.6	149	518	7251	807	2816	**7966**	4395	10303	85476
Ours: 3H-GCT (§4.1)		0.34	5.63	96.71	216	416	5831	**300**	1890	10604	**1264**	**7248**	38349
Ours: Star arch. (§5.4)		0.39	6.09	104.04	227	483	4938	355	**1343**	9504	1373	9491	**34870**
RR (EC-ROM variant) [RR17b]		4.54	75.52	1260.82	122	951	16240	3505	9127	45962	19220	24867	271442
RR (SM variant, $\sigma = 64$) [RR17b]		48.66	815.43	—	534	7694	—	4506	33236	—	35959	187801	—
PaXoS (2-core) [PRTY20]	M	0.92	14.23	223.89	221	**418**	**4779**	392	2119	12042	2531	8152	60771
Ours: 3H-GCT (§4.1+§6.3)		**0.57**	**8.68**	**136.66**	**219**	420	5855	**300**	2929	10417	**1365**	**6981**	37695
Ours: Star arch. (§5.4+§6.3)		0.64	9.27	145.42	227	496	4987	308	**1350**	**9631**	1375	7654	**36871**

Table 1: Communication in MB and run time in milliseconds. All protocols run with inputs of length $\sigma = 128$ except RR (SM) that supports 64 bits at most. The upper part of the table refers to semi-honest (SH) protocols whereas the lower part refers to malicious (M) protocols. Missing entries refer to experiments that failed due to lack of memory or took too much time. Reported results are by running over AWS c5d.2xlarge.
Note that we found an issue with the implementation of [KKRT, PRTY19, CM20, RR17b], which use network connection library [Rin]. Specifically, over a real network their protocols take more time than over a simulated network with similar bandwidth and latency. The difference is noticeable in CM [CM20].

dominated by computation. Over slower networks our protocols are almost always the fastest in the semi-honest setting and always fastest in the malicious setting. For example, over a 33 Mbits/sec network, our malicious star architecture-based construction is almost 2× faster than PaXoS.

Acknowledgements. We would like to thank Dan Boneh and Laliv Tauber, as well as the anonymous referees, for their valuable comments on earlier drafts of this paper. The first and third authors are partially supported by a Facebook research award. The second author is supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office, and by a grant from the Alter family. The fourth author is partially supported by NSF awards #2031799, #2115075.

References

[BCG$^+$19] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, and Peter Scholl. Efficient pseudorandom correlation generators: Silent OT extension and more. In *CRYPTO (3)*, volume 11694 of *LNCS*, pages 489–518. Springer, 2019.

[BCGI18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, and Yuval Ishai. Compressing vector OLE. In *ACM Conference on Computer and Communications Security*, pages 896–912. ACM, 2018.

[BENOPC] Aner Ben-Efraim, Olga Nissenbaum, Eran Omri, and Anat Paskin-Cherniavsky. PSImple: practical multiparty maliciously-secure private set intersection. ePrint, 2021/122.

[BPZ13] Fabiano C. Botelho, Rasmus Pagh, and Nivio Ziviani. Practical perfect hashing in nearly optimal space. *Inf. Syst.*, 38(1):108–131, 2013.

[CDG$^+$] Nishanth Chandran, Nishka Dasgupta, Divya Gupta, Sai Lakshmi Bhavana Obbattu, Sruthi Sekar, and Akash Shah. Efficient linear multiparty PSI and extensions to circuit/quorum psi. ePrint2021/172.

[CDJ16] Chongwon Cho, Dana Dachman-Soled, and Stanislaw Jarecki. Efficient concurrent covert computation of string equality and set intersection. In Kazue Sako, editor, *CT-RSA 2016*, volume 9610 of *LNCS*, pages 164–179. Springer, Heidelberg, February / March 2016.

[CGS21] Nishanth Chandran, Divya Gupta, and Akash Shah. Circuit-PSI with linear complexity via relaxed batch OPPRF. Cryptology ePrint Archive, Report 2021/034, 2021.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1243–1255. ACM Press, October / November 2017.

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

[CP34] C. J. Clopper and Egon Sharpe Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, 26(4):pp. 404–413, 1934.

[DCW13a] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 789–800. ACM Press, November 2013.

[DCW13b] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *ACM CCS 2013*, pages 789–800, 2013.

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19, 2004.

[GL19] Thomas Mueller Graf and Daniel Lemire. Xor filters: Faster and smaller than bloom and cuckoo filters. *CoRR*, abs/1912.08258, 2019.

[GN19] Satrajit Ghosh and Tobias Nilges. An algebraic approach to maliciously secure private set intersection. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 154–185. Springer, Heidelberg, May 2019.

[GS19] Satrajit Ghosh and Mark Simkin. The communication complexity of threshold private set intersection. In *CRYPTO (2)*, volume 11693 of *LNCS*, pages 3–29, 2019.

[HL10] Carmit Hazay and Yehuda Lindell. A note on the relation between the definitions of security for semi-honest and malicious adversaries. Cryptology ePrint Archive, Report 2010/551, 2010. http://eprint.iacr.org/2010/551.

[HV17] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Scalable multi-party private set-intersection. In *PKC 2017, Part I*, volume 10174 of *LNCS*, pages 175–203, 2017.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 145–161. Springer, Heidelberg, August 2003.

[IOP18] Roi Inbar, Eran Omri, and Benny Pinkas. Efficient scalable multiparty private set-intersection via garbled bloom filters. In *SCN*, pages 235–252, 2018.

[Kil00] Joe Kilian. More general completeness theorems for secure two-party computation. In *32nd ACM STOC*, pages 316–324. ACM Press, May 2000.

[KKRT] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *ACM CCS 2016*, pages 818–829.

[KMP$^+$17] Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *ACM CCS 2017*, pages 1257–1272. ACM Press, October / November 2017.

[KMW09] Adam Kirsch, Michael Mitzenmacher, and Udi Wieder. More robust hashing: Cuckoo hashing with a stash. *SIAM J. Comput.*, 39(4):1543–1561, 2009.

[KRTW19] Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. Scalable private set union from symmetric-key techniques. In *ASIACRYPT 2019, Part II*, volume 11922 of *LNCS*, pages 636–666. Springer, Heidelberg, 2019.

[KS05] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *CRYPTO 2005*, volume 3621 of *LNCS*, pages 241–257. Springer, Heidelberg, August 2005.

[MB72] R. Moenck and Allan Borodin. Fast modular transforms via division. In *Switching and Automata Theory*, pages 90–96, 1972.

[Mol04] Michael Molloy. The pure literal rule threshold and cores in random hypergraphs. In *SODA*, pages 672–681. SIAM, 2004.

[MPP10] Mark Manulis, Benny Pinkas, and Bertram Poettering. Privacy-preserving group discovery with linear complexity. In *ACNS 10*, volume 6123 of *LNCS*, pages 420–437, 2010.

[MU05] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005.

[NP99] Moni Naor and Benny Pinkas. Oblivious transfer and polynomial evaluation. In *31st ACM STOC*, pages 245–254. ACM Press, May 1999.

[OOS17] Michele Orrù, Emmanuela Orsini, and Peter Scholl. Actively secure 1-out-of-N OT extension with application to private set intersection. In Helena Handschuh, editor, *CT-RSA 2017*, volume 10159 of *LNCS*, pages 381–396. Springer, Heidelberg, February 2017.

[PRTY19] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. SpOT-light: Lightweight private set intersection from sparse OT extension. In *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 401–431. Springer, Heidelberg, August 2019.

[PRTY20] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXoS: Fast, malicious private set intersection. In *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 739–767. Springer, Heidelberg, May 2020.

[PSTY19] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 122–153. Springer, Heidelberg, May 2019.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 797–812. USENIX Association, August 2014.

[PSZ18] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.*, 21(2):7:1–7:35, 2018.

[Rin] Peter Rindal. cryptotools. `https://github.com/ladnir/cryptoTools`.

[RR17a] Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In *EUROCRYPT 2017, Part I*, volume 10210, pages 235–259, 2017.

[RR17b] Peter Rindal and Mike Rosulek. Malicious-secure private set intersection via dual execution. In *ACM CCS 2017*, pages 1229–1242. ACM Press, October / November 2017.

[RS98] Martin Raab and Angelika Steger. “balls into bins” - a simple and tight analysis. In *Workshop on Randomization and Approximation Techniques in Computer Science*, RANDOM ’98, page 159–170. Springer-Verlag, 1998.

[RS21] Peter Rindal and Phillipp Schoppmann. VOLE-PSI: fast OPRF and circuit-psi from vector-ole. *IACR Cryptol. ePrint Arch.*, 2021:266, 2021.

[SGRR19] Phillipp Schoppmann, Adrià Gascón, Leonie Reichert, and Mariana Raykova. Distributed vector-ole: Improved constructions and implementation. In *ACM Conference on Computer and Communications Security*, pages 1055–1072. ACM, 2019.

[Wal21a] Stefan Walzer. Peeling close to the orientability threshold - spatial coupling in hashing-based data structures. In Dániel Marx, editor, *SODA*, pages 2194–2211. SIAM, 2021.

[Wal21b] Stefan Walzer. Peeling close to the orientability threshold – spatial coupling in hashing-based data structures. *SODA*, pages 2194–2211, 2021.

[ZLL$^+$19] En Zhang, Feng-Hao Liu, Qiqi Lai, Ganggang Jin, and Yu Li. Efficient multi-party private set intersection against malicious adversaries. In *ACM SIGSAC Conference on Cloud Computing Security Workshop*, CCSW’19, page 93–104, 2019.

A OKVS

A.1 Sketched Proof of Lemma 6

We solve the system of linear equations $(l(k_i)||r(k_i)) \cdot S$ for all keys k whose corresponding hyperedges are in the 2-core. There at most $d(n)$ such equations, and the R-parts of these equations include random binary combinations of $d(n) + \lambda$ variables. Based on eq. (2) the probability that the system is unsolvable, is at most $1/2^{\lambda}$. After assigning values to S in the positions corresponding to this solution, the unpeeling process can set values so the remaining positions in S to satisfy all remaining key-value pairs.

A.2 Benchmarking OKVS

We implement and evaluate the existing and new OKVS schemes described in Sections 3-4. For polynomial interpolation and batch evaluation we use the algorithm of Moenck and Borodin [MB72], which has an overhead of $O(n \log(n)^2)$ field operations for both tasks. Since interpolation always succeeds, this scheme's error probability is 0. To speed up the computation of the polynomial-based OKVS, we use a simple hashing to bins. Then, we interpolate and evaluate for each bin separately. The 'bins+polynomial' scheme takes less computation cost than a single big polynomial, but requires more bandwidth since we pad all bins up to μ items. The failure probability of this scheme equals the probability of having a bin to which more than μ items are mapped, which is fully analyzed.

As expected, Table 2 shows that all hashing based OKVS's (GBF, PaXoS, 3H-GCT and star architecture) are much faster than the polynomial-based ones, since their running time is linear in n. The fastest among those are PaXoS, 3H-GCT and star architecture, whose run times are comparable. We note that in the context of PSI the smaller size of the 3H-GCT and star constructions leads to a better run time of the PSI protocol. Also, although the star construction is a little slower it is the only one with an empirically proven error probability.

We now describe detailed *microbenchmarking* results for OKVS in Table 3. We measured the time (in milliseconds) and percentage of each sub operation in the three hash-based OKVS's.

- **Star architecture**'s encoding has five main sub operations (we highlight in **bold** the tasks which take the bulk of the run time): (1) generate the cuckoo graph of all bins and perform the peeling algorithm to check whether or not the encoding fails; (2) encode the center bin according to whether one leaf node's encoding failed or not; (3) **adjust** the new values of all leaves according to the encoded center node; (4) **encode** the leaf nodes using the new values; and (5) merge all the tables of all bins to a single consistent vector, which is the final OKVS. The decoding has three main sub operations: (1) preprocessing - in which a lookup tables are prepared, in order to reduce the XOR time of the additional $\lambda + 0.5 \log n$ entries (see Section 5.4 for the description of the concrete instantiation); (2) **getting the indices** of the entries of all keys, this consists of many invocations of the scheme hash functions; and (3) performing the **XOR operation** to obtain the decoded results.
- **3H-GCT and PaXoS.** Encoding has 3 main sub operations: (1) generating the cuckoo-graph; (2) peeling, in order to get the 2-core sub-graph; and (3) **unpeeling** – which fills the entries with values. The decoding has 3 main sub-operations as well: (1) getting indices, as above, consists of many invocations of the scheme's hash functions; (2) Xor main performs the XOR on the main graph; and (3) **Xor extra** performs the XOR on the additional $\lambda + 0.5 \log n$ entries.

Construction	**Error Analysis**	$n = 2^{12}$			$n = 2^{16}$			$n = 2^{20}$		
		Size	**Time (ms)**		**Size**	**Time (ms)**		**Size**	**Time (ms)**	
		(MB)	**Enc.**	**Dec.**	**(MB)**	**Enc.**	**Dec.**	**(MB)**	**Enc.**	**Dec.**
Polynomial	no error	0.07	196	168	1.05	4876	4371	16.78	111920	102834
Bins + polynomial	analytic	0.09	187	336	1.37	2938	2413	22.85	48099	39096
GBF [DCW13a]	analytic	3.93	21	16	62.91	367	275	1006.63	6412	4884
PaXoS [PRTY20]	heuristic	0.15	5	2	2.40	120	42	38.40	3114	853
Dense matrix (§3)	analytic	0.07	–	–	1.05	–	–	16.78	–	–
Bins + dense matrix (§4.2)	analytic	0.09	52310	130	1.36	184220	290	23.01	109320	210
3H-GCT (§4.1)	heuristic	0.08	6	3	1.30	154	49	20.80	4996	1034
Star architecture (§5.4)	empiric	0.12	9	4	1.56	180	71	22.83	2915	1625

Table 2: Performance of various OKVS constructions, for n items from $GF(2^{128})$, with error $1/2^{-40}$. Reported results are by running over AWS c5d.4xlarge.

		Star architecture			**3H-GCT**		**PaXoS**	
	Sub operation	$n = 2^{20}$	$n = 2^{16}$	**Sub operation**	$n = 2^{20}$	$n = 2^{16}$	$n = 2^{20}$	$n = 2^{16}$
Encode	*Check bins*	453 (14.2%)	25 (14.3%)	Generate graph	1198 (26.2%)	29 (21.1%)	520 (18.3%)	14 (13.5%)
	Encode center	106 (3.7%)	5 (2.8%)	*Peeling*	1962 (43%)	34 (24.8%)	753 (26.5%)	14 (13.5%)
	Adjust values	884 (31.0%)	57 (32.7%)	*Solve system*	0	0	2 (0%)	0
	Encode leaves	1046 (36.7%)	63 (36.2%)	*Unpeeling*	1408 (30.8%)	74 (54%)	1563 (55%)	75 (72.8%)
	Merge bins	405 (14.2%)	24 (13.8%)					
	Total	2846	174		4568	137	2838	103
Decode	*Preprocess*	36 (5.5%)	2 (8.1%)	*Get indices*	93 (7.7%)	5.7 (10%)	68.7 (6.9%)	4.3 (8.6%)
	Get indices	220 (33.8%)	12.5 (51%)	*Xor main*	1032 (85.7%)	44 (80.7%)	845 (85.3%)	41 (82%)
	Xor all	393 (60.5%)	10 (40%)	*Xor extra*	79 (6.5%)	4.8 (8.8%)	76.8 (7.7%)	4.7 (9.4%)
	Total	649	24.5		1204	54.5	990.5	50

Table 3: Micro-benchmarks of the main OKVS constructions. Time in ms.

B Multi-Party PSI

B.1 Overview and Intuition

Multi-party Private Set Intersection($\mathcal{F}_{\text{m-psi}}$) allows a set of parties, each with a private set of items (P_i owns a set X_i), to learn the intersection of their sets $X_0 \cap X_1 \cap \cdots \cap X_n$ and nothing beyond that. We observe that, one of the multi-party PSI protocols proposed by Kolesnikov *et al.* in [KMP+17] that is "augmented semi-honest" secure (and semi-honest secure with significant additional cost) is malicious secure. We describe the protocol, proof of security against malicious adversaries and discuss the cost analysis. We explicitly address this apparent contradiction between a protocol achieving malicious but not semi-honest security in Section B.6 and other extensions in Section B.7.

B.2 Protocol

We highlight here the main ideas of the protocol. The protocol is a generic transformation from any (malicious-secure in this case) 2-party **oblivious PRF** (OPRF) functionality. An OPRF allows a sender to learn (or specify) a PRF key, while the receiver learns the output of the PRF on a limited number of chosen inputs. The sender learns nothing about the receiver's choice of inputs.

Following the precedent set by [HV17], the protocol is in the star topology communication model, where most communication is with a central party P_0 which is also the only one that learns the output. P_0 runs many instances of the malicious secure $\mathcal{F}_{\text{oprf}}$ (which is almost equivalent to the 2-party PSI) functionality, once with every other party.

The main challenge is to ensure that P_0 doesn't learn any of the intermediate intersection values from the two-party $\mathcal{F}_{\text{oprf}}$ outputs, unless *all parties* hold the corresponding item. In order to achieve this, the protocol is augmented by a zero-sharing setup, where the $n + 1$ parties non-interactively generate an *unlimited* number of additive shares of the value zero. That is, for every h (from an exponentially large domain), each party P_i can generate its individual share $sh_i(h)$ such that $\sum_{i=0}^{n} sh_i(h) = 0$.

Parties use these zero-shares to mask their OPRF outputs in the following way. Each party P_i generates an OKVS (in [KMP+17], only polynomial and GBF are considered) $\mathcal{Q}_i$ where $\mathsf{Decode}(\mathcal{Q}_i, h)$ is the OPRF output on h, masked by the zero-share $sh_i(h)$, for every $h \in X_i$. Only if *all* parties hold an item do the shares cancel out, revealing the presence of *all* OPRF outputs.

The protocol is described in detail in Figure 5. We generalize the treatment of [KMP+17], writing the protocol in terms of an arbitrary OKVS rather than explicitly a polynomial. Our only deviation from the protocol of [KMP+17] is to wrap the OPRF output in a random oracle before using it to generate the OKVS. As we will see, the random oracle layer allows the simulator to extract a corrupt party's input.

B.3 Costs

Computational cost. Every party computes a single instance of OKVS on a set of m points. Additionally, the central party performs decode of m points on OKVS.

Communication cost. Zero-sharing involves the exchange of keys (of length κ) between every pair of parties, costing $\mathcal{O}(n^2\kappa)$. Every party participates in a 2-party $\mathcal{F}_{\text{oprf}}$ protocol with the central and in step 2(c) sends a OKVS of m points to the central party. Thus, we can express our communication complexity as $\mathcal{O}(n^2\kappa + n \cdot |\mathsf{OPRF}| + n \cdot m|\log \mathbb{F}|)$, where κ is the security parameter, OPRF is the 2-party OPRF protocol and $\mathbb{F}$ is the field of inputs.

Round complexity. Zero-sharing can be run in parallel with the call to the $\mathcal{F}_{\text{oprf}}$ since the shares are used only after the the parties have finished querying (in step 2). Additionally, if the chosen instantiation of the $\mathcal{F}_{\text{oprf}}$ sends the key to the sender (here the non-central parties) before the receiver learns all his outputs, the non-central parties can send their OKVS (step 2(c)) before the $\mathcal{F}_{\text{oprf}}$-interaction ends. Thus, we achieve a round complexity that's almost equal to that of the $\mathcal{F}_{\text{oprf}}$ instantiation, namely, $|\mathsf{OPRF}| + 1$ rounds.

B.4 Security Proof

In this section we outline the security proof of the protocol in Figure 5.

We assume without loss of any generality that set $\mathcal{C} = \{P_1^*, P_2^*, \ldots, P_k^*\}$ indicates the set of corrupt *non-central* parties. Our proof considers two cases, depending on whether the central party P_0 is corrupt.

We start by noticing that $\sum_{i=0}^{n} sh_i(h) = 0$ for any value h. This implies that the corrupt parties can learn the sum of the shares of the honest parties for any value.

$$\sum_{i \in \mathcal{C}} sh_i(h) = -\sum_{i \notin \mathcal{C}} sh_i(h), \tag{8}$$

We start our proof with the following lemma:

Lemma 8. *Given a set of parties that run the zero-sharing setup (step 1 in Figure 5) such that a pair of parties P_i, P_j are honest and the adversary's view is independent of P_i's share $sh_i(x)$, then P_j's share $sh_j(x)$ is indistinguishable from uniform to the adversary.*

Proof. The parties P_i, P_j share a key k_{ij} that is unknown to the adversary as per the zero-sharing setup. As such, the value $\mathsf{PRF}'(k_{ij}, x)$ is indistinguisahble from uniform. The only place this value is used in the protocol is as a term in both $sh_i(x)$ and $sh_j(x)$ (referring to the definition of $sh_i(x)$ in Step 1 of Figure 5). But since $sh_i(x)$ is independent of the adversary's view, the *only* place $\mathsf{PRF}'(k_{ij}, x)$ is used to influence the adversary's view is as a term in $sh_j(x)$. Hence, this pseudorandom value causes the entirety of $sh_j(x)$ to be indistinguishable from random, in the adversary's view. □

Theorem 9. *The protocol in Figure 5 realizes UC-secure PSI in the $\mathcal{F}_{oprf}$-hybrid model, when the field $\mathbb{F}$ is exponentially large.*

Proof. The central party is either corrupt or honest and we write a simulator for these two cases. We always assume that some subset of the non-central parties are corrupt.

(i) We begin with the scenario when the **central party** P_0 **is corrupt**. The goal of the simulator is to extract the effective input of the central party and simulate the interaction between the honest non-central parties and P_0 using the output of the $\mathcal{F}_{\text{m-psi}}$ functionality. Consider all the technical details in the simulation below:

- *Hybrid 0* We begin with the real interaction where all honest parties follow the protocol on their given inputs. Each honest P_i computes his OKVS $\mathcal{Q}_i$ (sent in step 2(c)) by encoding the points $h \in X_i$ as follows:

$$\mathcal{Q}_i = \mathsf{Encode}\Big(\Big\{\big(h, \mathcal{H}(\mathsf{PRF}_i(h), h) + sh_i(h)\big) \;\Big|\; h \in X_i\Big\}\Big) \tag{9}$$

- *Hybrid 1* An honest P_0 is supposed to use the same input X_0 in each instance of $\mathcal{F}_{\text{oprf}}$, but a corrupt P_0 may not. In this hybrid we write X_0^i to denote P_0's input to the $\mathcal{F}_{\text{oprf}}$ used with party P_i. We refer to $\mathcal{A}_i = X_i \cap X_0^i$ as the ith "active set".

  In this hybrid we modify each honest party P_i to generate its OKVS to satisfy the following different m constraints (instead of as Equation 9)

$$\mathcal{Q}_i = \begin{cases} \mathsf{Encode}\big(h, \mathcal{H}(\mathsf{PRF}_i(h), h) + sh_i(h)\big) & \text{if } h \in \boxed{\mathcal{A}_i} \\ \mathsf{Encode}\big(h, s_i^h\big) & \text{if } h \in \boxed{X_i \setminus \mathcal{A}_i} \text{ where } s_i^h \xleftarrow{\$} \mathbb{F} \end{cases} \tag{10}$$

  To see why the hybrids are indistinguishable, we use the fact that $\mathsf{PRF}_i(h)$ is indistinguishable from random for $h \notin X_0^i$ (*i.e.*, the adversary did not query PRF_i at this point). Hence, the probability that the adversary queries $\mathcal{H}$ on input containing $\mathsf{PRF}_i(h)$ is negligible, and so the output of $\mathcal{H}$ at this point is indistinguishable from random.

- *Hybrid 2* In this hybrid we further modify the way the honest parties create their OKVS. Here, P_i creates its OKVS to satisfy the following different set of m constraints:

$$\mathcal{Q}_i = \begin{cases} \mathsf{Encode}\big(h, \mathcal{H}(\mathsf{PRF}_i(h), h) + sh_i(h)\big) & \text{if } h \in \boxed{\bigcap_{P_i \notin \mathcal{C}} \mathcal{A}_i} \\ \mathsf{Encode}\big(h, s_i^h\big) & \text{if } h \in \boxed{X_i \setminus \bigcap_{P_i \notin \mathcal{C}} \mathcal{A}_i} \\ & \quad \text{where } s_i^h \leftarrow \mathbb{F} \end{cases} \tag{11}$$

First, we notice that if there is only one honest party then this hybrid is indentical to the previous one. The subsequent reasoning assumes that we have at least two honest parties.

It's easy to see that if $h \in \bigcap_{P_i \notin \mathcal{C}} \mathcal{A}_i$, then this hybrid is identical to hybrid 1. P_i's behavior only differs for a value $h \in \mathcal{A}_i$ where $h \notin \mathcal{A}_j$ for some other honest party P_j. When $h \notin \mathcal{A}_j$, we see that in Hybrid 1, P_j does not use the value $sh_j(h)$ anywhere. By Lemma 8, the corrupt central party's view is independent of the share $sh_i(h)$. Thus, even when the adversary knows the evaluation $\mathsf{PRF}_i(h)$ (since $h \in \mathcal{A}_k$), the view of $\mathsf{Decode}(\mathcal{Q}_i, h) = \mathcal{H}(\mathsf{PRF}_i(h), h) + sh_i(h)$ is indistinguishable from a independently random field element because $sh_i(h)$ is uniformly random. Thus, this hybrid is indistinguishable from the previous one.

- *Final simulation:* The previous hybrid can be carried out by a simulator in the ideal interaction, as follows:
  - The simulator computes the collection X_0^i from the adversary's inputs to instances of $\mathcal{F}_{\text{oprf}}$.
  - The simulator sends $\tilde{X}_0 := \bigcap_{i \notin \mathcal{C}} X_0^i$ to $\mathcal{F}_{\text{m-psi}}$, *as the input of all corrupt parties*, and receives $\tilde{X}_0 \cap (\bigcap_{i \notin \mathcal{C}} X_i)$ as the output. Note that $\tilde{X}_0 \cap (\bigcap_{i \notin \mathcal{C}} X_i) = \bigcap_{i \notin \mathcal{C}} \mathcal{A}_i$.
  - The simulator simulates the OKVS $\mathcal{Q}_i$ from honest party P_i as in Equation 11. Generating the OKVS in this way requires only the output of $\mathcal{F}_{\text{m-psi}}$.

(ii) Now, we consider the case when **central party P_0 is honest**. The goal of the simulator is to extract the inputs of the corrupt non-central parties and simulate their interaction with P_0 in a way that is consistent with the output he computes.

Now, we present all the hybrids to simulate the central party's interaction with every corrupt non-central party P_i.

- *Hybrid 0:* We start with the real interaction with all honest parties running the protocol honestly on their inputs. Honest P_0 computes the protocol output as:

$$\left\{ h \in X_0 \;\middle|\; \sum_{i \in \mathcal{C}} \mathsf{Decode}(\mathcal{Q}_i, h) + \sum_{i \notin \mathcal{C}} \mathsf{Decode}(\mathcal{Q}_i, h) = 0 \right\} \tag{12}$$

- *Hybrid 1:* In this hybrid, we modify how P_0 computes the output as:

$$\left\{ h \in \boxed{\bigcap_{i \notin \mathcal{C}} X_i} \;\middle|\; \sum_{i \in \mathcal{C}} \mathsf{Decode}(\mathcal{Q}_i, h) + \sum_{i \notin \mathcal{C}} \mathsf{Decode}(\mathcal{Q}_i, h) = 0 \right\} \tag{13}$$

  Note that this computation differs only for values $h \in X_0$ where $h \notin X_i$ for some honest P_i (in particular, there must be two honest parties for this hybrid to be any different). Such an h will never satisfy Equation 13 (since $h \notin X_i$), and it suffices to show that h satisfies Equation 12 only with negligible probability.

  For such an h, since $h \notin X_i$, $\mathcal{Q}_i$ is generated independently of $sh_i(h)$. But $\mathsf{Decode}(\mathcal{Q}_0, h)$ contains a term $sh_0(h)$, which renders the entire expression $\sum_i \mathsf{Decode}(\mathcal{Q}_i, h)$ uniformly random (by Lemma 8). Hence, the probability that h satisfies the condition in Equation 12 is $1/|\mathbb{F}|$ which is negligible.

Note that if $h \in \bigcap_{i \notin \mathcal{C}} X_i$, then all honest parties have intentionally chosen $\mathsf{Decode}(\mathcal{Q}_i, h)$ on their OKVS, according to the protocol. Using this fact along with Equation 8 we can rearrange Equation 13 as:

$$\Big\{ h \in \bigcap_{i \notin \mathcal{C}} X_i \;\Big|\; \sum_{i \in \mathcal{C}} \mathsf{Decode}(\mathcal{Q}_i, h) = \sum_{i \in \mathcal{C}} sh_i(h) + \sum_{i \in \mathcal{C}} \mathcal{H}(\mathsf{PRF}_i(h), h) \Big\} \tag{14}$$

- *Hybrid 2:* This hybrid differs from the previous one, in that we again modify how P_0 computes the final protocol output. During the interaction we monitor all random oracle queries made by the adversary. When the adversary sends the final $\mathcal{Q}_i$, we define the set

  $$O = \{x \mid \text{ adversary made a query of the form } \mathcal{H}(\mathsf{PRF}_i(x), x)\}$$

  Note that the PRF_i functions are determined before the OKVS tables are sent in Step 2c. Then P_0 computes the output with a modified computation as follows:

  $$\Big\{ h \in \boxed{O \cap} \bigcap_{i \notin \mathcal{C}} X_i \;\Big|\; \sum_{i \in \mathcal{C}} \mathsf{Decode}(\mathcal{Q}_i, h) = \sum_{i \in \mathcal{C}} sh_i(h) + \sum_{i \in \mathcal{C}} \mathcal{H}(\mathsf{PRF}_i(h), h) \Big\} \tag{15}$$

  It suffices to show that if $h \notin O$, then it is only with negligible probability that h would have satisfied the condition in Equation 14. Indeed, if $h \notin O$ then at the time the adversary sends $\mathcal{Q}_i$, the value of $\mathcal{H}(\mathsf{PRF}_i(x), x)$ is distributed independently of the adversary's view (it hasn't queried $\mathcal{H}$ at this input). Since the condition in Equation 14 is a linear equality involving this term, the probability that it is satisfied is $1/|\mathbb{F}|$ which is negligible.

- *Final simulation:* We can observe that Equation 15 can be written as:

  $$\Big\{ h \in O \;\Big|\; \sum_{i \in \mathcal{C}} \mathsf{Decode}(\mathcal{Q}_i, h) = \sum_{i \in \mathcal{C}} sh_i(h) + \sum_{i \in \mathcal{C}} \mathcal{H}(\mathsf{PRF}_i(h), h) \Big\} \cap \bigcap_{i \notin \mathcal{C}} X_i$$

  Therefore the interaction can be carried out in the ideal world as follows:

  – The simulator plays the role of $\mathcal{F}_{\text{oprf}}$ honestly and also observes the adversary's oracle queries.

  – When the adversary sends $\mathcal{Q}_i$ in Step 2c, the simulator defines the set O (of oracle queries) as above, and computes

  $$\tilde{X} := \Big\{ h \in O \;\Big|\; \sum_{i \in \mathcal{C}} \mathsf{Decode}(\mathcal{Q}_i, h) = \sum_{i \in \mathcal{C}} sh_i(h) + \sum_{i \in \mathcal{C}} \mathcal{H}(\mathsf{PRF}_i(h), h) \Big\}$$

  It sends $\tilde{X}$ to ideal $\mathcal{F}_{\text{m-psi}}$ as the input for *every* corrupt party.

  – $\mathcal{F}_{\text{m-psi}}$ delivers output $\tilde{X} \cap \bigcap_{i \notin \mathcal{C}} X_i$ to P_0.

One subtlety is how big the set $\tilde{X}$ is, which the simulator sends to the ideal functionality. We can think of the adversary as choosing $\sum_{i \in \mathcal{C}} \mathsf{Decode}(\mathcal{Q}_i, h)$, which stores m key-value pairs, and the simulator extracts by checking which pairs of this aggregate OKVS satisfy a certain condition. In principle there may be more than m pairs that satisfy the condition that defines $\tilde{X}$. (This issue does not affect the case of a corrupt P_0 since in that case the simulator extracts by finding all *pairs* in the OKVS.) It is for this reason that our *ideal functionality allows corrupt parties to provide larger input sets than honest parties* (see Figure 5).[12]

We can bound the size m' of the set $\tilde{X}$ as follows. Suppose the adversary makes q queries to $\mathcal{H}$ and outputs $\mathcal{Q}_i$ such that the resulting $\tilde{X}$ has m' items. We could use this adversary to "compress" the random oracle $\mathcal{H}$ as follows (the following analysis is information-theoretic, as it operates on the exponentially large object $\mathcal{H}$):

- Run the adversary on oracle $\mathcal{H}$ until it outputs $\mathcal{Q}_i$ for $i \in \mathcal{C}$.
- Compute the set of values $\tilde{X}$ as above.
- For every $h \in \tilde{X}$, consider all oracle queries of the form $\tilde{H}(\mathsf{PRF}_i(h), h)$. Mark whichever of these queries was made *last* by the adversary as **"overdetermined."**
- Output the following:
  - The list of overdetermined oracle queries made by the adversary, given by which order they are made by the adversary (*i.e.*, "the 3rd, 5th, 19th oracle queries are overdetermined"). This list consists of $\log \binom{q}{m'}$ bits.
  - For every non-overdetermined oracle query made by the adversary, in the order that they are made, give the output of $\mathcal{H}$ on that query. This list consists of $(q - m') \log |\mathbb{F}|$ bits.
  - For all queries *not* made by the adversary, in lexicographic order, give the output of $\mathcal{H}$. This list consists of $(N - q') \log |\mathbb{F}|$ bits, where N is the size of the input domain for $\mathcal{H}$.
  - Output $\sum_{i \in \mathcal{C}} \mathcal{Q}_i$.

Note that all of $\mathcal{H}$ can be reconstructed from this information: Simply run the adversary, answering its oracle queries from the information provided. If the query is not overdetermined, then the correct response is given in the list explicitly. If the query is overdetermined, then it can be solved for in the equation $\sum_i \mathsf{Decode}(\mathcal{Q}_i, h) = \sum_{i \in \mathcal{C}} sh_i(h) + \sum_{i \in \mathcal{C}} \mathcal{H}(\mathsf{PRF}_i(h), h)$, since by construction, at the time the $\mathcal{H}$-output is needed, all other $\mathcal{H}$-outputs in this expression are already known. Note that since h is included in the oracle query (whose output we are trying to reconstruct), so we know the identity of h in this expression as well. All other outputs of $\mathcal{H}$ are given explicitly in the information provided above.

Overall, we have provided information consisting of $\log \binom{q}{m'} (N - m' + m) \log |\mathbb{F}|$, from which the random oracle can be reconstructed. The size of the random oracle is $N \log |\mathbb{F}|$. Hence, we must have

$$\begin{aligned} \log \binom{q}{m'} (N - m' + m) \log |\mathbb{F}| &\geq N \log |\mathbb{F}| \\ \implies \log \binom{q}{m'} &\geq (m' - m) \log |\mathbb{F}| \end{aligned}$$

Take $|\mathbb{F}| = 2^{2\kappa}$ and $q = 2^{\kappa}$. Since $\binom{q}{m'} \geq (q/m')^{m'}$, it suffices to have:

$$m' \log(q/m') = \kappa m' - m' \log m' \geq 2\kappa(m' - m)$$

Ignoring the insignificant term $m' \log m'$, we see that it suffices to have $m' = 2m$, because it makes both sides equal to $2\kappa m$.

In other words, the simulator will extract a set for the corrupt parties, with size bounded by $2m$. □

[12]This property is common to many malicious PSI protocols, for example those of [RR17b, RR17a].

We leave it as an interesting open problem to understand whether it is possible to restrict corrupt parties to input sets of size m within this PSI protocol framework.

B.5 Comparison

In this section, we present a detailed comparison of our work to existing protocols on various parameters. The works of [HV17, KMP$^+$17] describe multi-party PSI protocols that are semi-honest secure, and [GN19, HV17, BENOPC] and the protocol described in this paper are robust against malicious adversaries.

We start by observing that the protocol of Hazay *et al.* in [HV17] relies heavily on public-key encryption and calls on expensive tools like threshold homomorphic encryption and zero-knowledge proofs. Further, they use a broadcast channel in 3 out the 8 rounds in their malicious secure protocol. In comparison, all other mentioned protocols can be instantiated using fast symmetric-key based operations and do not assume a broadcast channel.

To the best of our knowledge, all multi-party PSI protocols use the star-topology communication model to avert the cost associated with a complete network of communication. Therefore, a relevant parameter of efficiency is the amount of *non-star* communication cost of the protocols. The work of Ghosh *et al.* [GN19] and the protocol in our paper are comparable, with a single round of *input-independent* setup where each pair of parties exchanges κ bits of information. In contrast, the work of [HV17] in the malicious setting, involves 3 rounds of communication between all pairs of parties. One of those rounds, is a setup key-generation phase for threshold homomorphic encryption scheme. The other two rounds are during the online phase, and involve a coin toss and n broadcasts, one by each of the parties.

We wish to highlight that the multi-party PSI protocols of Kolesnikov *et al.* [KMP$^+$17] and in this paper are the most "general" transformation from *any* 2-party protocol that can be expressed as an OPRF functionality (where this is true for a vast majority of protocols). Another distinguishing feature is that our protocol restricts all parties to participate with input sets of the same size. The works of [GN19, HV17] do not suffer from this drawback.

The concurrent work of [BENOPC] uses garbled Bloom filters to construct a concretely efficient malicious-secure protocol. In their work, P_0 needs to do an exhaustive search (that scales with the number of parties) to learn the intersection. That is, P_0 expects to see a codeword y_x to learn that x belongs to the intersection, other parties send unordered sets of their codewords. P_0 needs to check all combinations of codewords to see if any of them XOR to match y_x. In contrast, our OKVS-based solution does not suffer from this drawback, the central party always knows where to look to check if x is in the intersection, and its computation does not scale with the number of parties. Further, the asymptotic cost of our 2 party OPRF is much cheaper than the GBF based instantiation of [BENOPC] that requires many random OTs (as many as the size of the garbled Bloom filter) augmented with a costly cut-and-choose check to make the construction malicious-secure. Lastly, we compare the sizes of the GBF and OKVS sent to the central party. For sets of size m, we send an OKVS with field elements, of total size $O(m(\lambda + \log m))$ while [BENOPC] send strings of size $O(m\kappa)$ (security parameter), while it almost always holds that $\lambda + \log m < \kappa$. To summarize, the [BENOPC] construction relies heavily on BF/GBF, which is the most communication-expensive construction amongst three PSI constructions presented in [KMP$^+$17]. While our protocol achieves almost the same cost as that of the most efficient construction (the augmented semi-honest protocol) in [KMP$^+$17] with only a minor (inexpensive) modification.

Protocol	**Security**	**Tools, Assump.**	**Communication**	**Computation**	**Rounds**
KMPRT17 [KMP+17]	aug SH	OPPRF	$\mathcal{O}(n^2\kappa + nm(\kappa + \lambda + \log m))$	$\mathcal{O}(n\kappa)$	$\lvert\mathsf{OPRF}\rvert + 1$
	SH	OPPRF	$\mathcal{O}(n^2\kappa + nmt(\kappa + \lambda + \log m))$	$\mathcal{O}(nt\kappa)$	$\lvert\mathsf{OPRF}\rvert + 1$
HV17 [HV17]	SH	THE	$\mathcal{O}(nm\kappa)$	$\mathcal{O}((nm + m^2)\log \mathbb{F})$	4
	M	THE, RO/CRS	$\mathcal{O}((n^2 + nm\log m)\kappa)$	$\mathcal{O}((nm + m^2)\log \mathbb{F})$	8
GN19 [GN19]	M	OLE	$\mathcal{O}((n^2 + nm)\kappa)$	$\mathcal{O}(nm\log^2 m)$	4
ENO21 [BENOPC]	M	GBF, OPPRF	$\mathcal{O}((nm\kappa^2 + mn\kappa\log(m\kappa))$	$\mathcal{O}(mn\kappa)$	6
Ours	M	vOLE, RO	$\mathcal{O}(n^2\kappa + nm(\kappa + \lambda + \log m)) + o(nm)$	$\Omega(nm)$	4

Table 4: We compare protocols assuming n parties, each owning private sets of equal size m with λ as the statistical parameter and κ as the computational security parameters. Some protocols require us to indicate the number of colluding parties as t. All protocols are designed in the star-topology communication model and only one abritrarily chosen central party learns the output. The protocols are robust against $n-1$ corruptions. All computational cost is written in terms of field multiplications. "aug SH", "SH", and "M" refer to augmented semi-honest, semi-honest and malicious, respectively.

B.6 Comparison with KMPRT17 (How Can it be Secure Against Malicious but not Semi-Honest Adversaries?)

As we have mentioned, our construction is "isomorphic" to a construction of Kolesnikov *et al.* (KMPRT) [KMP+17]. We briefly review their construction now. The construction of KMPRT achieves an intermediate functionality that they call "oblivious **programmable** PRF" (OPPRF). This is similar to an 2-party OPRF, where the sender can additionally "program" the random function with a small set of desired points. Formally, the sender provides points $(x_1, y_1), (x_2, y_2), \ldots$, and the receiver provides points $z_1, z_2, \ldots$. The receiver receives $F(z_1), F(z_2), \ldots$ and the sender receives (a description of) F, where F is a pseudorandom function constrained so that $F(x_i) = y_i$ for all i. Importantly, the receiver doesn't learn whether any given point was one of the "special" ones x_i.

KMPRT construct an OPPRF from an OPRF as follows (among other constructions): The parties run an OPRF for PRF F, where the sender learns key k. Then the sender generates and sends a polynomial P such that $P(x_i) = y_i \oplus F(k, x_i)$ for all pairs (x_i, y_i) on which he wishes to "program" the OPRF. Defining the new function $F'(k, x) = F(k, x) \oplus P(x)$ we can see that indeed $F'(k, x_i) = y_i$ for all programming-pairs (x_i, y_i).

Furthermore, KMPRT use an OPPRF in their multi-party PSI protocol by programming it so that the y_i values are shares of zero (using the same non-interactive zero-sharing approach that we also use). Hence, their approach boils down to sending a polynomial interpolated through points $P(x) = OPRF(k, x) + sh_i(x)$, just as we have presented here. The only notable difference is to apply the transformation to a malicious-secure OPRF protocol.

Semi-honest vs Malicious. How can it be that this construction is apparently not secure enough for semi-honest adversaries but we prove it to be secure against malicious adversaries?

Hazay and Lindell [HL10] explicitly address this seemingly paradoxical scenario, and we present a synopsis here: Here, we start by considering a simpler example. Suppose Alice has a bit $a \in \{0, 1\}$ and Bob has a bit $b \in \{0, 1\}$. Bob should receive the bit $a \wedge b$, while Alice should receive nothing. The protocol where Alice simply sends a to Bob is actually a secure protocol against malicious adversaries. Indeed, if Bob is malicious then (in the ideal world) he can always use input $b = 1$ and learn a anyway, without Alice noticing. However, this trivial protocol is not secure for semi-honest adversaries because in the case where semi-honest Bob has input $b = 0$ the protocol reveals too much. In fact, it is highly nontrivial to construct a semi-honest protocol for this functionality, as it is known to be equivalent to oblivious transfer [Kil00].

The problem is that the simulator for a *malicious* Bob always sends input $b = 1$ to the ideal functionality (so it can learn a from the output $a \wedge b$ and simulate Alice's protocol message). However, in the semi-honest model all parties (even corrupt parties) have a well-defined input (provided by the environment) and the semi-honest simulator is obligated to send that input to the ideal functionality. If semi-honest Bob has input $b = 0$ and his semi-honest simulator sends $b = 0$ to the ideal functionality, it clearly does not receive enough information to simulate the message a from Alice, making the protocol "insecure."

In short, semi-honest security considers a strict subset of possible real-protocol adversaries (so in that sense is less demanding), but also is limited to a strict subset of possible ideal-world simulators (so in that sense is more restrictive).

Our multi-party PSI protocol is similar to the simple $a \wedge b$ example. Kolesnikov *et al.* showed that it does achieve "augmented semi-honest" security which is essentially semi-honest security where the simulator is now allowed to send a different input to the ideal functionality. Intuitively, we can interpret their result as "the protocol is semi-honest secure apart from the issue of simulators changing inputs." It just so happens that the protocol furthermore is malicious-secure, and simulators "changing" a corrupt party's input is completely irrelevant in the malicious model. Thus we achieve the standard notion of malicious security.

B.7 Extensions

Sets of different sizes. Our protocol is written for the case where all parties have sets of the same size n. It is natural to extend our protocol so that each party P_i has input of size n_i and sends an OKVS consisted of $n_i - 1$ key-value pairs. Perhaps surprisingly, there is a subtle security issue associated with this generalization. Specifically, the protocol cannot guarantee the fact that the intersection will have size at most $\min_i\{n_i\}$.

More concretely, consider a 3-party scenario where P_1 and P_2 are corrupt and the central party P_0 is honest. For simplicity, we consider the polynomial OKVS. P_1 advertises 10 items and therefore sends a polynomial $\mathcal{Q}_1$ of degree 9, while P_2 advertises 1000 items and sends a polynomial $\mathcal{Q}_2$ of degree 999. Note that $\mathcal{Q}_1$ and $\mathcal{Q}_2$ do not have an *individual* effect on P_0's ultimate output — only their sum $\mathcal{Q}_1 + \mathcal{Q}_2$ has an effect.[13] This observation indeed leads to a concrete "attack" whereby P_1 and P_2 arrange so that $\mathcal{Q}_1 + \mathcal{Q}_2$ (which, recall, has degree 999) has 1000 pairs h that satisfy:

$$\mathsf{Decode}(\mathcal{Q}_1, h) + \mathsf{Decode}(\mathcal{Q}_2, h) = \mathcal{H}(\mathsf{PRF}_1(h), h) + \mathcal{H}(\mathsf{PRF}_2(h), h) + sh_1(h) + sh_2(h)$$

When this is true, the intersection can easily contain more than 10 items (depending on which items honest P_0 has).

On the positive side, this is the only issue with the generalized protocol. It does achieve security with respect to a slightly weaker ideal PSI functionality. In particular, if a party is corrupt then it is allowed to provide a set of size $n_{\mathsf{max}} = \max_i\{n_i\}$ where i ranges over the indices of corrupt parties. In other words, corrupt parties may have more items than they "advertise" by the size of their OKVS, but still learn no more than from an intersection on (perhaps slightly larger) sets. This relaxation of PSI ideal functionality is not unprecedented, as several other malicious PSI protocols (*e.g.*, [RR17b, RR17a]) cannot strictly enforce the cardinality of the corrupt parties' input sets to what is "advertised" by the structure/size of the protocol messages.

We leave it as an interesting open question to extend our multi-party protocol to strictly enforce the size of each corrupt party's input set.

[13] Note that when the simulator extracts inputs for P_1, P_2, it only inspects the aggregate OKVS $\mathcal{Q}_1 + \mathcal{Q}_2$ and sends the same input set to the ideal functionality on behalf of both parties!

Output to more than one party. Our protocol and ideal functionality provide output only to the central party P_0. We can provide output to all parties with the following modifications to the protocol:

- All parties, even P_0 publicly commit to their OKVS $\mathcal{Q}_i$. (Recall that P_0 computes an OKVS in the original protocol but only uses it locally.)
- After all parties have committed, all parties reveal their OKVS.
- All parties compute output as $\{h \in X_i \mid \sum_i \mathsf{Decode}(\mathcal{Q}_i, h) = 0\}$.

In this way, all parties will receive output. However, there are some important subtleties to consider.

Suppose P_0 is corrupt and commits to a garbage OKVS. Other parties will compute empty output, while P_0 knows the "correct" $\mathcal{Q}_0$ (that it should have committed/sent) and can compute the "true" output. More generally, P_0 can commit/send an "incorrect" $\mathcal{Q}_0$ that encodes through the correct values only on a subset $X_0^* \subsetneq X_0$ of the true items. Other parties will output $X_0^* \cap (\bigcap_i X_i)$ while P_0 will be able to learn $X_0 \cap (\bigcap_i X_i)$. It can be shown that this is the only kind of attack possible – a corrupt party P_0 can only cause the honest parties to learn a strict subset of the "correct" output, but cannot cause them to accept an item outside of X_0 in their output.

More formally, this protocol can be shown to achieve a variant multi-party PSI ideal functionality that works as follows. Corrupt parties provide both an "honest" input set X_i^h as well as a "corrupt" input set X_i^c, with the restriction that $X_i^h \subseteq X_i^c$. The corrupt parties learn the intersection of all sets, computed using their "corrupt" input sets, but the honest parties are given the intersection computed using the "honest" input sets.

This kind of behavior regarding outputs is endemic to many other malicious PSI protocols. It is a significant challenge for the receiver to convey the output to other parties, even in the 2-party case! It is for this exact reason that existing malicious 2-party PSI protocols like [RR17a, RR17b] provide output to only one party — they too could be made to provide output to both parties but the receiver (who gets output first) could cause the sender to learn a strict subset of the "true" output. The multi-party PSI protocol of [GN19] has this property as well.

We point out one interesting difference between our approach here and the one of [GN19]. In that work, they prove that the protocol realizes an ideal functionality that works as follows: The corrupt parties provide input, then receive output, and then the corrupt parties choose which subset of the true output is delivered to the honest parties. In other words, the adversary can decide what outputs to deliver to the honest parties based on the output it learns. In our protocol, the adversary is committed, *before seeing the output*, to which subset of the true output it will deliver to the honest parties.

PARAMETERS:

- $n+1$ parties $P_0, \ldots, P_n$, where P_0 is designated to receive output.
- An exponentially large field $\mathbb{F}$, and random oracle $\mathcal{H} : \{0,1\}^* \to \mathbb{F}$.
- A malicious-secure $\mathcal{F}_{\text{oprf}}$ ideal functionality for underlying PRF PRF.
- A PRF PRF' (possibly the same as above) with output in $\mathbb{F}$.

INPUTS:

- Each party P_i owns an input set X_i, such that $X_i \subseteq \mathbb{F}$ and $|X_i| = m$.

PROTOCOL:

1. Zero-sharing setup: Assume an ordering on the parties $\{P_0, P_1, \ldots P_n\}$. Every P_i samples a key for PRF' uniformly at random $k_{ij} \xleftarrow{\$} \mathbb{F}$ and sends it to every P_j, where $i > j$. We use the following notation for non-interactive zero-shares throughout the protocol:

$$sh_i(h) = \sum_{j<i} \mathsf{PRF}'_{k_{ij}}(h) - \sum_{j>i} \mathsf{PRF}'_{k_{ji}}(h)$$

   Note that party P_i can compute $sh_i(h)$ for any h, and also note that $\sum_i sh_i(h) = 0$ for any h.

2. P_0 interacts with every other party P_i as follows:

   (a) The two parties invoke an instance of $\mathcal{F}_{\text{oprf}}$ where P_i acts as sender and learns a key, and P_0 acts as receiver with input X_0. We write PRF_i to denote the underlying PRF specialized to the key used in this instance. P_0 learns the set $\{\mathsf{PRF}_i(h) \mid h \in X_0\}$.

   (b) P_i computes OKVS $\mathcal{Q}_i$ via:

$$\mathcal{Q}_i = \mathsf{Encode}\Big(\Big\{\big(h, \mathcal{H}(\mathsf{PRF}_i(h), h) + sh_i(h)\big) \;\Big|\; h \in X_i\Big\}\Big)$$

   (c) P_i sends $\mathcal{Q}_i$ to P_0.

3. P_0 computes OKVS $\mathcal{Q}_0$ via:

$$\mathcal{Q}_0 = \mathsf{Encode}\Big(\Big\{\big(h, sh_0(h) - \sum_{i=1}^{n} \mathcal{H}(\mathsf{PRF}_i(h), h)\big) \;\Big|\; h \in X_0\Big\}\Big)$$

4. After P_0 receives OKVS from the other n parties he determines the output $\{h \in X_0 \mid \sum_{i=0}^{n} \mathsf{Decode}(\mathcal{Q}_i, h) = 0\}$

Figure 5: Malicious-secure Multi-party PSI protocol Π_{mpsi} in the $\mathcal{F}_{\text{oprf}}$-hybrid model