# PSI from PaXoS:
# Fast, Malicious Private Set Intersection

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

## Abstract

We present a 2-party private set intersection (PSI) protocol which provides security against malicious participants, yet is almost as fast as the fastest known *semi-honest* PSI protocol of Kolesnikov et al. (CCS 2016).

Our protocol is based on a new approach for two-party PSI, which can be instantiated to provide security against either malicious or semi-honest adversaries. The protocol is unique in that the only difference between the semi-honest and malicious versions is an instantiation with different parameters for a linear error-correction code. It is also the first PSI protocol which is concretely efficient while having linear communication and security against malicious adversaries, while running in the OT-hybrid model (assuming a non-programmable random oracle).

State of the art semi-honest PSI protocols take advantage of cuckoo hashing, but it has proven a challenge to use cuckoo hashing for malicious security. Our protocol is the first to use cuckoo hashing for malicious-secure PSI. We do so via a new data structure, called a probe-and-XOR of strings (PaXoS), which may be of independent interest. This abstraction captures important properties of previous data structures, most notably garbled Bloom filters. While an encoding by a garbled Bloom filter is larger by a factor of $\Omega(\lambda)$ than the original data, we describe a significantly improved PaXoS based on cuckoo hashing that achieves constant rate while being no worse in other relevant efficiency measures.

## 1 Introduction

Private set intersection (PSI) allows two parties with respective input sets $X$ and $Y$ to compute the intersection of the two sets without revealing anything else about their inputs. PSI and its variants have numerous applications, such as for contact discovery, threat detection, advertising, etc. (see e.g., [IKN+17, PSZ18] and references within). Privately computing the size of the intersection (known as 'PSI cardinality') is also important for computing conditional probabilities, which are useful for computing different analytics of private distributed data.

While there are generic methods for secure multi-party computation of any function (MPC), finding a specific protocol for PSI is interesting in its own sake since generic MPC protocols are relatively inefficient for computing PSI: generic MPC operates on a circuit representation of the computed functionality, while the intersection functionality can be represented only by relatively large circuits (the naive circuit for computing the intersection of two sets of size $n$ is of size $O(n^2)$; a circuit based on sorting networks is of size $O(n \log n)$ [HEK12]; and new results reduce the circuit to size $O(n)$ by utilizing different hashing schemes, but seem to be hard to be adapted to the malicious setting [PSWW18, PSTY19].)

There has been tremendous progress in computing PSI in the semi-honest model, where the parties are assured to follow the protocol (see [PSTY19, PRTY19, KKRT16]). However, protocols for the malicious setting, where parties can behave arbitrarily, are much slower, with the protocol of Rosulek and Rindal [RR17b] being the best in terms of concrete efficiency. Protocols in both settings reduce the computation of PSI to computing many oblivious transfers (OT), which can be implemented extremely efficiently using oblivious transfer extension [IKNP03a, ALSZ13a]. The protocols also benefit from hashing the items of the input sets to many bins, and computing the intersection separately on each bin. In the semi-honest setting it was possible to use Cuckoo hashing, which is a very efficient hashing method that maps each item to one of *two* possible locations [PR04, KMW09]. However, it was unknown how to use Cuckoo hashing in the malicious setting: the problem was that a malicious party Alice can learn the location to which an input element of Bob is mapped. The choice of this location by Bob leaks information about the other inputs of Bob, including items which are not in the intersection.

## 1.1 Our Contributions

Our protocol is the first to use Cuckoo hashing for PSI in the malicious setting. This is done by introducing a new data structure, called a probe-and-XOR of strings (PaXoS). This is a randomized function, mapping $n$ binary strings to $m$ binary strings, where each of the $n$ original strings can be retrieved by XOR'ing a specific subset of the $m$ strings. PaXoS can be trivially implemented using a random $m \times n$ matrix, but then the encoding and decoding times are prohibitively high when $n$ is large. We show how to implement PaXoS using a Cuckoo graph (a graph representing the mapping in Cuckoo hashing), with efficient encoding and decoding algorithms. This is essentially equivalent to Cuckoo hashing where instead of storing an item in one of two locations, we set the values of these two locations such that their XOR is equal to the stored value. As a side-effect, this does away with the drawback of using Cuckoo hashing in malicious PSI. Namely, parties do not need to *choose one of two locations* in which an input item is stored, and thus there is no potential information leakage by Cuckoo hashing.

Our protocol uses a PaXoS data structure $D$ as a key-value store, mapping the inputs values (aka keys) of one of the parties to values which are encoded as linear combinations of the string in $D$. It then uses the OT extension protocol of Orrù *et al.* [OOS17] (OOS), to build a PSI protocol from this data structure. The OOS protocol is secure against malicious adversaries, and is parameterized by a linear error-correcting code. Our PSI construction is unique in that the only difference between the semi-honest and malicious instantiations is only in the parameters of this code.

The semi-honest instantiation improves over the state of the art KKRT protocol [KKRT16] by 25% in concrete communication cost, while having a comparable running time. More importantly, the malicious instantiation has only slightly higher overhead than the best *semi-honest* protocol, and significantly better performance than the state of the art for malicious security [RR17b] (about 8×

less communication, and 3× faster computation). Source code is available at

From a theory perspective, we introduce the first concretely efficient protocol in the the OT-hybrid model (assuming a non-programmable random oracle), which is secure in the malicious setting and has *linear* communication. The previous state-of-the-art [RR17b] has $O(n \log n)$ communication complexity.

## 1.2 Related Work

We focus on the discussion of the state-of-the-art of semi-honest PSI protocols. We note that the earliest PSI protocols, based on Diffie-Hellman assumptions, can be traced back to the 1980s [Sha80, Mea86a, HFH99], and refer the reader to [PSZ14] for an overview of the different PSI paradigms for PSI. Protocols [PRTY19] based on oblivious transfer extension have proven to be the fastest in practice.

A more popular public-key based approach to low-communication PSI is based on Diffie-Hellman key agreement, and presented in the 1980s [Mea86b, HFH99] in the random oracle model. The high-level idea is for the parties to compute the intersection of $\{(H(x_i)^k)^r \mid x \in X\}$ and $\{(H(y_i)^r)^k \mid y \in Y\}$ in the clear, where $r$ and $k$ are secrets known by receiver and sender, respectively. However, This protocol requires $O(n)$ exponentiations.

Current state-of-the-art semi-honest PSI protocols in the two-party setting are [KKRT16, PSZ18, PRTY19]. They all rely on oblivious transfer. Most work on concretely efficient PSI is in the random oracle model, and with security against semi-honest, rather than malicious, adversaries. Some notable exceptions are [FNP04, JL09, HL10] in the standard model, and [FNP04, RR17b, RR17a, CHLR18] with security against malicious adversaries.

We refer the reader to the Appendix D for a detailed and technical comparison of the many different protocol paradigms for PSI.

## 1.3 Organization

In Section 2 we present the preliminaries required in order to understand our techniques (linear codes, correlation-robustness, oblivious transfer and PSI). We then introduce the notion of Probe and Xor of Strings (PaXoS) in Section 3 and show an efficient construction of a PaXoS in Section 5. In section 4 we present and prove our PSI protocol, which is obtained from any PaXoS. We show our main construction of an efficient PaXoS in Section 3.2 (and an alternative, more compact one, in Appendix B ). We present a detailed, qualitative as well as experimental, comparison to previous work in Sections 6 and Section 7.

## 2 Preliminaries

We denote the computational and statistical security parameters by $\kappa$ and $\lambda$ respectively. We say that a function $\mu : \mathbb{N} \to \mathbb{N}$ is *negligible* if for every positive polynomial $p(\cdot)$ and all sufficiently large $\kappa$ it holds that $\mu(\kappa) < \frac{1}{p(\kappa)}$. For a bit string $x$ (or a vector) of length $m$, we refer to the $j$-th coordinate of $x$ by $x_j$. A matrix is denoted by a capital letter. For a matrix $X$, we refer to the $j$-th row of $X$ by $x_j$ and the $j$-th column of $X$ by $x^j$. For two bit strings $a, b$ with $|a| = |b|$, $a \wedge b$ (resp. $a \oplus b$) denotes the bitwise-AND (resp. bitwise-XOR) of $a$ and $b$.

**Error-correcting codes.**    A binary linear code $\mathcal{C}$ with length $n_\mathcal{C}$, dimension $k_\mathcal{C}$ and minimum distance $d_\mathcal{C}$ is denoted $[n_\mathcal{C}, k_\mathcal{C}, d_\mathcal{C}]$. So $\mathcal{C} : \mathbb{F}_2^{k_\mathcal{C}} \to \mathbb{F}_2^{n_\mathcal{C}}$ is a linear map such that for every nonzero $m \in \mathbb{F}_2^{k_\mathcal{C}}$, the Hamming weight of $\mathcal{C}(m)$ is at least $d_\mathcal{C}$.

**Code-correlation-robustness of random oracles.**    Our construction uses the fact that when $H$ is a random oracle, $C$ is a linear code with minimum distance $\kappa$, and $s$ is secret, terms of the form $H(a \oplus C(b) \wedge s)$ look random. This property was introduced in [KK13] as a generalization of correlation-robust hashing (from [IKNP03b]), and a variant is also used in the context of PSI in [KKRT16]. It is described in the following lemma.

**Lemma 1** ([KK13]). *Let $C$ be a linear error correcting code $[n, k, d]$ with $d \geq \kappa$. Let $H$ be a random oracle and let $s \leftarrow \{0,1\}^n$ be chosen uniformly at random. Then for all $a_1, \ldots, a_m \in \{0,1\}^n$ and nonzero $b_1, \ldots, b_m \in \{0,1\}^k$, the following values are indistinguishable from random:*

$$H(a_1 \oplus C(b_1) \wedge s), \ldots, H(a_m \oplus C(b_m) \wedge s),$$

*Proof Sketch.* If $C$ has minimum distance $\kappa$, then any nonzero codeword $C(b_i)$ has hamming weight at least $\kappa$, so the term $C(b_i) \wedge s$ involves at least $\kappa$ unknown bits of the secret $s$. Hence, each argument of the form $H(a_i \oplus C(b_i) \wedge s)$ has at least $\kappa$ bits of entropy, from the point of view of the distinguisher, so it is negligibly likely that the distinguisher will ever query $H$ at such a point.  □

**Oblivious transfer.**    Oblivious Transfer (OT) is a central cryptographic primitive in the area of secure computation. It was introduced by Rabin [Rab05, EGL85]. 1-out-of-2 OT is a two-party protocol between a sender, who inputs two messages $v_0, v_1$, and a receiver who inputs a choice bit $b$ and learns as output $v_b$ and nothing about $v_{1-b}$. The sender remains oblivious as what message was received by the receiver. The general case of 1-out-of-N OT on $\tau$-bit strings is defined as the functionality:

$$\mathcal{F}_{N\text{-OT}}^{\tau} [(v_0, \ldots, v_{N-1}), c] \to [\bot, v_c]$$

where $v_0, \ldots, v_{N-1} \in \{0,1\}^\tau$ are the sender's inputs and $c \in \{0, \ldots, N-1\}$ is the receiver's input. We denote by $\mathcal{F}_{N\text{-OT}}^{\tau,m}$ the functionality that runs $\mathcal{F}_{N\text{-OT}}$ for $m$ times on messages in $\{0,1\}^\tau$. An important variant is the *random* OT functionality, denoted $\mathcal{F}_{N\text{-ROT}}^{\tau,m}$ in which the sender provides no input, but receives from the functionality as output random messages $(v_0, \ldots, v_{N-1})$ (or a key which enables to compute these messages).

**The OOS oblivious transfer functionality.**    We will use a specific construction, by Orrù, Orsini and Scholl [OOS17] (hereafter referred to as OOS) that realizes $\mathcal{F}_{N\text{-ROT}}^{\tau,m}$, and supports an exponentially large $N$, e.g. $N = 2^\tau$. OOS is parameterized with a binary linear code $[n_\mathcal{C}, k_\mathcal{C}, d_\mathcal{C}]$ where $k_\mathcal{C} = \tau$ and $d_\mathcal{C} \geq \kappa$. OOS features a useful homomorphism property that we use in our PSI construction (see Section 4).

Specifically, we describe OOS as the functionality:

$$\mathcal{F}_{\text{OOS}} [s, (d_1, \ldots, d_m)] \to \left[ (q_1, \ldots, q_m), (r_1, \ldots, r_m) \right]$$

where $r_i = q_i \oplus s \wedge \mathcal{C}(d_i)$, $s, q_i \in \mathbb{F}_2^{n_\mathcal{C}}$ and $d_i \in \mathbb{F}_2^{k_\mathcal{C}}$ for every $i \in [m]$.

These outputs can be used for $m$ instances of 1-out-of-N OT as follows. The random OT values for the $i$th OT instance are $H(q_i \oplus s \wedge C(x))$, where $H$ is a random oracle and $x$ ranges over all $N$

possible $\tau$-bit strings. The sender can compute any of these values as desired, whereas the receiver can compute only $H(r_i) = H(q_i \oplus s \wedge \mathcal{C}(d_i))$, which is the OT value corresponding to choice index $d_i$. The fact that other OT values $H(q_i \oplus s \wedge \mathcal{C}(d'))$, for $d' \neq d_i$, are pseudorandom is due to Lemma 1. Specifically, we can write

$$q_i \oplus s \wedge \mathcal{C}(d') = q_i \oplus s \wedge \left[ \mathcal{C}(d_i) \oplus \mathcal{C}(d_i \oplus d') \right] = r_i \oplus s \wedge \mathcal{C}(d_i \oplus d')$$

and observe that $\mathcal{C}(d_i \oplus d')$ has Hamming weight at least $d_{\mathcal{C}} \geq \kappa$. Hence Lemma 1 applies. Note that the "raw outputs" of the OOS functionality are XOR-homomorphic in the following sense: for every $i, j \in [m]$,

$$r_i \oplus r_j = \left( q_i \oplus s \wedge \mathcal{C}(d_i) \right) \oplus \left( q_j \oplus s \wedge \mathcal{C}(d_j) \right) = q_i \oplus q_j \oplus s \wedge \mathcal{C}(d_i \oplus d_j)$$

In this expression we use the fact that $\mathcal{C}$ is a linear code.

**Secure computation and 2-party PSI.**   Informally, security is defined in the real/ideal paradigm [Gol04, Chapter 7]. A protocol is secure if, for any attack against the protocol, there is an equivalent attack in an ideal world where the function is computed by a trusted third party. More formally, a functionality is a trusted third party who cannot be corrupted and who carries out a specific task in response to invocations (with arguments) from parties. This is considered as the ideal world. Parties interact with each other according to some prescribed protocol; in other words, the parties execute a protocol in the real world. Parties, who execute some protocol, may interact/invoke functionalities as well, in which case we consider this to be a hybrid world. A semi-honest adversary may corrupt parties and obtain their entire state and all subsequent received messages; a malicious adversary may additionally cause them deviate, arbitrarily, from their interaction with each other and with a functionality (i.e. modify/omit messages, etc.). In this work there are only two parties, sender and receiver, and the adversary may *statically* corrupt one of them (at the onset of the execution).

We denote by $\text{IDEAL}_{f,\mathcal{A}}(x,y)$ the joint execution of some task $f$ by an ideal world functionality, under inputs $x$ and $y$ of the receiver and sender, resp., in the presence of an adversary $\mathcal{A}$. In addition, we denote by $\text{REAL}_{\pi,\mathcal{A}}(x,y)$ the joint execution of some task $f$ by a protocol $\pi$ in the real world, under inputs $x$ and $y$ of the receiver and sender, resp., in the presence of an adversary $\mathcal{A}$.

**Definition 2.** *A protocol $\pi$ is said to securely compute $f$ (in the malicious model) if for every probabilistic polynomial time adversary $\mathcal{A}$ there exists a probabilistic polynomial time simulator $\mathcal{S}$ such that*

$$\{\text{IDEAL}_{f,\mathcal{S}}(x,y)\}_{x,y} \overset{c}{\equiv} \{\text{REAL}_{\pi,\mathcal{A}}(x,y)\}_{x,y}$$

We consider a 2-party PSI functionality, described in Figure 1, that **does not strictly enforce the size of a corrupt party's input set.** In other words, while ostensibly running the protocol on sets of size $n$, an adversary may learn as much as if he used a set of bounded size $n' > n$ in the ideal world (typically, $n' = c \cdot n$ for some constant $c$, This is the case in this work as well). This property is shared by several other 2-party malicious PSI protocols [RR17b, RR17a].

# 3 Probe-and-XOR of Strings (PaXoS)

## 3.1 Definitions

Our main tool is a mapping which has good linearity properties.

Parameters:

- Two parties: a sender and receiver.

- Set size $n$ for honest parties and $n'$ for corrupt parties.

Functionality:

1. Wait for input $Y = \{y_1, y_2, \ldots\}$ from the receiver. Abort if the receiver is corrupt and $|Y| > n'$.

2. Wait for input $X = \{x_1, x_2, \ldots\}$ from the sender. Abort if the sender is corrupt and $|X| > n'$.

3. Give output $X \cap Y$ to the receiver.

Figure 1: Ideal functionality for 2-party PSI.

**Definition 3.** *A $(n, m, 2^{-\lambda})$-**probe and XOR of strings** (PaXoS) is an oracle function $\boldsymbol{v}^H$ : $\{0,1\}^* \to \{0,1\}^m$ such that for any distinct $x_1, \ldots, x_n \in \{0,1\}^*$,*

$$\Pr[\boldsymbol{v}^H(x_1), \ldots, \boldsymbol{v}^H(x_n) \text{ are linearly independent}] \geq 1 - 1/2^\lambda$$

*where the probability is over choice of random function $H$, and linear independence is over the vector space $(\mathbb{Z}_2)^m$, i.e. for $x \in \{0,1\}^*$ we look at $\boldsymbol{v}^H(x)$ as a vector from $(\mathbb{Z}_2)^m$. We often let $H$ be implicit and eliminate it from the notation.*

In other words, this is a randomized function mapping $n$ binary strings to binary vectors of length $m$, satisfying the property that the output strings are independent except with probability $2^{-\lambda}$.

We would like the output/input rate, $m/n$, to be as close as possible to 1. A random mapping would satisfy the PaXoS definition and will have a good rate, but will be bad in terms of encoding/decoding efficiency properties that will be defined in Section 3.4.

A PaXoS has the implicit property that the mapping is independent of the inputs. Namely, the goal is not to find a function that works well for a specific set of inputs, but rather to find a function that works well with high probability for any input set. This is crucial in terms of privacy, since the function must not depend on any input, as this would leak information about the input.

## 3.2 PaXoS as Key-Value Mapping

A key-value store, or mapping, is a database which maps a set of keys to corresponding values.[1] A PaXoS leads to a method for **encoding a key-value mapping into a concise data structure**, as follows:

$\underline{\mathsf{Encode}((x_1, y_1), \ldots, (x_n, y_n))}$: Given $n$ items $(x_i, y_i)$ with $x_i \in \{0,1\}^*$ and $y_i \in \{0,1\}^\ell$, denote by $M$ the $n \times m$ matrix where the $i$th row is $\boldsymbol{v}(x_i)$. One can solve for a data structure (matrix) $D = (d_1, \ldots, d_m)^\top \in (\{0,1\}^\ell)^m$ such that $M \times D = (y_1, \ldots, y_n)^\top$. Namely, the following

---

[1]A hash table is a simple key-value mapping, but it encounters issues such as collisions. More importantly for our application, a hash table explicitly reveals whether an item is encoded in it and therefore has a privacy leakage.

linear system of equations (over the field of order $2^\ell$) is satisfied:

$$
\begin{bmatrix}
- \ \boldsymbol{v}(x_1) \ - \\
- \ \boldsymbol{v}(x_2) \ - \\
\vdots \\
- \ \boldsymbol{v}(x_n) \ -
\end{bmatrix}
\times
\begin{bmatrix}
d_1 \\
d_2 \\
\vdots \\
d_m
\end{bmatrix}
=
\begin{bmatrix}
y_1 \\
y_2 \\
\vdots \\
y_n
\end{bmatrix}
$$

When the $\boldsymbol{v}(x_i)$'s are linearly independent, a solution to this system of equations must exist. Therefore, when $\boldsymbol{v}(\cdot)$ is a PaXoS, the system has a solution except with probability $1/2^\lambda$.

$\underline{\mathsf{Decode}(D, x)}$: Given a data structure $D \in (\{0, 1\}^\ell)^m$ and a "key" $x \in \{0, 1\}^*$, we can retrieve its corresponding "value" via

$$
y = \langle \boldsymbol{v}(x), D \rangle \stackrel{\text{def}}{=} \bigoplus_{j : \boldsymbol{v}(x)_j = 1} d_j
$$

In other words, probing $D$ for a key $x$ amounts to computing the XOR of specific positions in $D$, where the choice of positions is defined by $\boldsymbol{v}(x)$ and depends only on $x$ (not $D$). It is easy to see that when $x$ is among the $x_i$ values that was used to create $D$ as above, then $y$ obtained this way is equal to the corresponding $y_i$. However, the PaXoS can be probed on any key $x$.

It is often more convenient to discuss PaXoS in terms of the corresponding Encode/Decode algorithms than the $\boldsymbol{v}$ mapping.

## 3.3 Homomorphic Properties

The Decode algorithm enjoys the following **homomorphic properties**. Let $D = (d_1, \ldots, d_m) \in (\{0, 1\}^\ell)^m$. Then:

- For any **linear** map $L : \{0, 1\}^\ell \to \{0, 1\}^{\ell'}$, extend the notation $L(D)$ to mean $(L(d_1), \ldots, L(d_m))$. Then we have
$$
\mathsf{Decode}(L(D), x) = L(\mathsf{Decode}(D, x)).
$$

- If $D$ and $D'$ have the same dimension, then define $D \oplus D' = (d_1 \oplus d_1', \ldots, d_m \oplus d_m')$. Then we have
$$
\mathsf{Decode}(D, x) \oplus \mathsf{Decode}(D', x) = \mathsf{Decode}(D \oplus D', x).
$$

- With $s \in \{0, 1\}^\ell$, define $D \wedge s = (d_1 \wedge s, d_2 \wedge s, \ldots, d_m \wedge s)$, where "$\wedge$" refers to bitwise-AND. Then we have
$$
\mathsf{Decode}(D \wedge s, x) = \mathsf{Decode}(D, x) \wedge s.
$$

## 3.4 Efficiency Measures

The following measures of efficiency are relevant in our work, and are crucial for the efficiency of the resulting PSI protocols:

- **Rate:** The Encode algorithm must encode $n$ values $(y_1, \ldots, y_n)$, which have total length $n\ell$ bits, into a data structure $D$ of total length $m\ell$ bits. The ratio $n/m$ defines the **rate** of the PaXoS scheme, with rate 1 being optimal and constant rate being desirable.

- **Encoding complexity:** What is the computational cost of the Encode algorithm, as a function of the number $n$ of key-value pairs? In general, solving a system of $n$ linear equations requires $O(n^3)$ computation using Gaussian elimination. However, the structure of the $\boldsymbol{v}(x)$ constraints may lead to a more efficient method for solving the system. We strive for an encoding procedure that is linear in $n$, for example $O(n\lambda)$ where $\lambda$ is the statistical security parameter.

- **Decoding complexity:** What is the computational cost of the Decode algorithm? The cost is proportional to the Hamming weight of the $\boldsymbol{v}(k)$ vectors — *i.e.*, the number of positions of $D$ that are XOR'ed to give the final result. We strive for decoding which is sublinear in $n$, for example $O(\lambda)$ or $O(\log n)$.[2]

## 3.5 Examples and Simple Constructions

Below are some existing concepts that fall within the abstraction of a PaXoS:

**Random Boolean Matrix.** A natural approach is to let $\boldsymbol{v}(x)$ simply be a random vector for each distinct $x$.[3] It is elementary to show that a random boolean matrix of dimension $n \times (n + \lambda)$ has full rank with probability at least $1 - 1/2^\lambda$. This leads to a $(n, m, 2^{-\lambda})$-PaXoS scheme with $m = n + \lambda$.

This scheme has excellent rate $n/(n + \lambda)$ (which is likely optimal), but poor efficiency of encoding/decoding. Encoding corresponds to solving a random linear system of equations, requiring $O(n^3)$ if done via Gaussian elimination. Decoding one item requires computing the XOR of $\sim n/2$ positions from the data structure.

**Garbled Bloom Filter.** A garbled Bloom filter works in the following way: Let $h_1, \ldots, h_\lambda$ be random functions with range $\{1, \ldots, m\}$. To query the data structure at a key $x$, compute the XOR of positions $h_1(x), \ldots, h_\lambda(x)$ in the data structure. In our terminology, $\boldsymbol{v}(x)$ is the vector that is 1 at position $i$ if and only if $\exists j : h_j(x) = i$.

Garbled Bloom Filters were introduced by Dong, Chen, Wen in [DCW13]. They showed that if the Bloom filter has size $m = \Theta(\lambda n)$ then the Encode algorithm succeeds with probability $1 - 1/2^\lambda$. The concrete error probability is identical to the false-positive probability of a standard Bloom filter.

Garbled Bloom filters are an instance of $(n, m, 2^{-\lambda})$-PaXoS with $m = \Theta(\lambda n)$ and therefore rate $\Theta(1/\lambda)$. Items can be inserted into the garbled Bloom filter in an online manner, leading to a total cost of $O(n\lambda)$ to encode $n$ items. Decoding requires taking the XOR of at most $\lambda$ positions per item.

**Garbled Cuckoo table.** We introduce in Section 5 a new PaXoS construction, garbled Cuckoo table, with a size which is almost optimal, and optimal encoding and decoding times.

It is also worth mentioning a variant of Bloomier filters that was introduced in [CC08], is similar to our garbled Cuckoo table construction, and yet is *insecure* for our purposes. The construction of [CC08] works for a specific input set $S$. It chooses random hash functions and generates a graph by mapping the items of $S$ to edges. The construction works well if the graph is acyclic. If the graph contains cycles then a new set of hash functions is chosen, until finding hash functions which map $S$

---

[2]When defining the cost of encoding and decoding we ignore the length ($m$) of the $y$-values.

[3]I.e., $\boldsymbol{v}^H(x) = H(x)$ where $H$ is a random oracle with $m$ output bits.

| scheme | size $m$ | encoding ($n$ items) | decoding (single item) |
|---|---|---|---|
| random matrix | $n + \lambda$ | $O(n^3)$ | $\Theta(n)$ |
| garbled Bloom filter | $O(\lambda n)$ | $O(\lambda n)$ | $\lambda$ |
| **garbled Cuckoo** | $(2 + \epsilon)n + d + \lambda$ | $O((\lambda + d)n)$ | $(\lambda + d + 2)/2$ in avg. |

Figure 2: A comparison between the different PaXoS schemes, where $n$ is the number of items, $\lambda$ is a statistical security parameter (e.g., $\lambda = 40$), $\epsilon$ is the a Cuckoo hash parameter (typically $\epsilon = 0.4$), and $d$ is an upper bound the number of cycles of a Cuckoo hash graph ($d = \log n$ except with negligible probability, and therefore for all reasonable input sizes $d < \lambda$).

to an acyclic graph. This construction is not a PaXoS since the choice of hash functions depends on the input and therefore leaks information about it. (Our garbled Cuckoo table construction, on the other hand, chooses the hash functions independently of the inputs, and works properly, except with negligible probability, even if the graph has cycles.)

## 4    PSI from PaXoS

In this section we describe a generic construction of PSI from PaXoS.

### 4.1    Overview

The fastest existing 2-party PSI protocols [KKRT16, RR17b] are all based on efficient OT extension and its variants. The leading OT extension protocol for malicious security is due to Orrù *et al.* [OOS17] (hereby called OOS), and it serves as the basis of our PSI protocol.

The OOS OT extension protocol implements the OOS functionality defined in Section 2, and provides many instances of 1-out-of-$N$ OT of random strings, where $N$ can even be exponentially large. Our PSI protocol involves the internals of the OOS protocol to some extent, so let us start by reviewing the relevant details. Suppose we are interested in 1-out-of-$N$ OT for $N = 2^t$. In OOS, the sender chooses a string $s$ and receives a string $q_i$ for each OT instance. In this OT instance, the sender can derive $N$ random values as follows:

$$H\big(q_i \oplus C(00\cdots 0) \wedge s\big); \quad H\big(q_i \oplus C(00\cdots 01) \wedge s\big); \quad \cdots \quad H\big(q_i \oplus C(11\cdots 1) \wedge s\big);$$

where $C$ is a linear error-correcting code with $t$ input/data bits, $H$ is a correlation-robust hash function, and "$\wedge$" denotes bitwise-AND (whenever we write $a \oplus b \wedge s$ we mean $a \oplus (b \wedge s)$).

The receiver has a "choice string" $d_i \in \{0, 1\}^t$ for each instance, and as a result of the OOS protocol he receives

$$r_i = q_i \oplus C(d_i) \wedge s \tag{1}$$

Clearly $H(r_i)$ is one of the $N$ random values that the sender can compute for this OT instance. The security of the OOS protocol is that the $N - 1$ other values look pseudorandom to the receiver, given $r_i$, despite the fact that the same $s$ is used in all OT instances.

One important property of the OOS values is that they enjoy an XOR-homomorphic property:

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

Note that we use the fact that $C$ is a linear code.    The fact that these values have such a homomorphic property was already pointed out and used in the OOS protocol as a way to check

9

consistency for a corrupt receiver. Our main contribution is to point out how to leverage this homomorphic property for PSI as well.

Suppose the receiver uses the strings of a PaXoS $D = (d_1, \ldots, d_m)$ as its OOS inputs, and the parties further interpret their OOS outputs $Q = (q_1, \ldots, q_m)$ (for the sender) and $R = (r_1, \ldots, r_m)$ (for the receiver) as PaXoS data structures as well. Then we find that the identity $r_i = q_i \oplus C(d_i) \wedge s$ facilitates the homomorphic properties of PaXoS:

$$\begin{aligned} \mathsf{Decode}(R, x) &= \mathsf{Decode}(Q \oplus C(D) \wedge s, x) \\ &= \mathsf{Decode}(Q, x) \oplus C(\mathsf{Decode}(D, x)) \wedge s \end{aligned}$$

Suppose the receiver encodes the PaXoS $D$ so that $\mathsf{Decode}(D, x)$ is something "recognizable" (say, $x$ itself) for every item $x$ in his PSI input set. Then the expression above is something that both parties can compute: the receiver computes it as $\mathsf{Decode}(R, x)$, and the sender computes it as $\mathsf{Decode}(Q, x) \oplus C(x) \wedge s$.

Hence, we can obtain a PSI protocol by having the sender send $H(\mathsf{Decode}(Q, x) \oplus C(x) \wedge s)$ for each of her items $x$. The receiver compares these values to $H(\mathsf{Decode}(R, y))$ for each of his items $y$, to determine the intersection.

## 4.2 Protocol Details

Our full protocol follows the general outline described above, but with some minor technical changes to facilitate the security proof.

One change is that instead of generating a PaXoS $D$ where $\mathsf{Decode}(D, x) = x$, the receiver arranges for $\mathsf{Decode}(D, x) = H_1(x)$ (for $x$ in his input set) where $H_1$ is a random oracle. This modification allows the simulator to extract a malicious receiver's effective input set by observing $D$ (used as input to OOS) and the receiver's queries to $H_1$.

Also, instead of sending values of the form $H(\mathsf{Decode}(Q, x) \oplus C(H_1(x)) \wedge s)$, we have the sender send values of the form $H(x, \mathsf{Decode}(Q, x) \cdots)$. That is, the item $x$ is included in the clear as an additional argument to $H$ (named $H_2$ in our construction to avoid confusion with $H_1$). Additionally, $H$ ($H_2$) is a (non-programmable) random oracle. As above, this allows the simulator to extract a malicious sender's effective input by observing its random-oracle queries.

The protocol is described formally in Figure 3.

## 4.3 Security Analysis

Recall that we are using as our definition an ideal PSI functionality (Figure 1) that does not strictly enforce the size of a corrupt party's set. In other words, a corrupt party may provide more items ($n'$) than they claim ($n$). We prove security of our construction without making explicit reference to the relationship between $n'$ and $n$. That is, in the proofs below we show that a simulator is able to extract *some* set (of size polynomial in the security parameter) in the ideal interaction, but the proofs do not explicitly bound the size of these sets.

The protocol contains several parameters $\ell_1$ and $\ell_2$ which affect the value of $n'$ that can be proven. We discuss how to choose these parameters, and the resulting $n'$ that one obtains, in section 4.4.

**Theorem 4.** *The protocol of Figure 3 is a secure 2-party PSI protocol against malicious adversaries in the random oracle model.*

**Parameters:**

- Computational and statistical security parameters $\kappa$ and $\lambda$

- Sender with set $X \subseteq \{0,1\}^*$ of size $n$

- Receiver with set $Y \subseteq \{0,1\}^*$ of size $n$

- $(n, m, 2^{-\lambda})$-PaXoS scheme (Encode, Decode)

- Random oracles $H_1 : \{0,1\}^* \to \{0,1\}^{\ell_1}$ and $H_2 : \{0,1\}^* \to \{0,1\}^{\ell_2}$, where $\ell_2, \ell_1 \geq \lambda + 2 \log n$

- Linear error correcting code $C : [t, \ell_1, \kappa]$

**Protocol:**

1. The receiver generates a PaXoS $D = \mathsf{Encode}(\{(y, H_1(y)) \mid y \in Y\})$.

2. The parties run the OOS functionality (as defined in Section 2) where the receiver uses as input $D = (d_1, \ldots, d_m)$ and the sender uses a random string $s$ as input. As a result, the sender obtains output strings $Q = (q_1, \ldots, q_m)$ and the receiver obtains output strings $R = (r_1, \ldots, r_m)$ that follow Eq, (1). We interpret both $D, Q$ and $R$ as PaXoS data structures.

3. The sender computes and sends the set
$$M = \left\{ H_2\Big(x, \mathsf{Decode}(Q, x) \oplus C(H_1(x)) \wedge s\Big) \ \Big| \ x \in X \right\}$$
randomly permuted.

4. The receiver coutputs $\{y \in Y \mid H_2(y, \mathsf{Decode}(R, y)) \in M\}$.

Figure 3: Our PaXoS-PSI protocol

We prove the theorem in the following two lemmas:

**Lemma 5.** *The protocol of Figure 3 is secure against a malicious receiver in the random oracle model.*

*Proof.* The simulator for a corrupt receiver behaves as follows:

- It observes the receiver's input $D$ to OOS, and also observes all of the receiver's queries to random oracle $H_1$.

- The simulator computes $\tilde{Y} = \{y \mid y$ was queried to $H_1$ and $\mathsf{Decode}(D, y) = H_1(y)\}$ and sends this to the ideal functionality as the receiver's effective input.

- Upon receiving from the ideal functionality the intersection $Z = X \cap \tilde{Y}$, the simulator simulates the sender's message $M$ as $\{H_2(z, \mathsf{Decode}(R, z)) \mid z \in Z\}$ along with $|X \setminus Z|$ additional random values.

We prove the indistinguishability of this simulation in the following sequence of hybrids:

- Hybrid 1: Same as the real protocol interaction, but the simulator maintains a list $L$ of all queries that the adversary makes to random oracle $H_1$. When the adversary selects its OOS input $D$, the simulator checks all $y \in L$ and defines the set $\tilde{Y} = \{y \in L \mid \mathsf{Decode}(D, y) = H_1(y)\}$. This hybrid is indistinguishable from the real protocol interaction, since the only difference is in internal bookkeeping information that is not used.

- Hybrid 2: Same as Hybrid 1, except that immediately after defining $\tilde{Y}$, the simulator aborts if the honest sender holds an $x \in X$ where $\mathsf{Decode}(D, x) = H_1(x)$ but $x \notin \tilde{Y}$. It suffices to show that the probability of this artificial abort is negligible.

  - Case $x \in L$: then $H_1(x)$ was known at the time $\tilde{Y}$ was defined. Therefore it is by construction that $x \in \tilde{Y} \Leftrightarrow \mathsf{Decode}(D, x) = H_1(x)$. In other words, the abort does not happen in this case

  - Case $x \notin L$: then $H_1(x)$ is independent of $D$, and thus $\mathsf{Decode}(D, x) = H_1(x)$ with probability $1/2^{\ell_1}$ where $\ell_1$ is the output length of $H_1$.

  If $\ell_1 = \lambda + \log_2 n$ then by a union bound over at most $n$ possible sender's values $x \in X$, the abort probability is indeed bounded by $1/2^\lambda$.

- Hybrid 3: Same as Hybrid 2, except we can rewrite the computation that defines the sender's message $M$. Observe that

$$
\begin{aligned}
\mathsf{Decode}&(Q, x) \oplus C(H_1(x)) \wedge s \\
&= \mathsf{Decode}(R \oplus C(D) \wedge s, x) \oplus C(H_1(x)) \wedge s \\
&= \Big[\mathsf{Decode}(R, x) \oplus \mathsf{Decode}(C(D), x) \wedge s\Big] \oplus C(H_1(x)) \wedge s \\
&= \mathsf{Decode}(R, x) \oplus \Big[C(\mathsf{Decode}(D, x)) \oplus C(H_1(x))\Big] \wedge s \\
&= \mathsf{Decode}(R, x) \oplus C\Big(\mathsf{Decode}(D, x) \oplus H_1(x)\Big) \wedge s
\end{aligned}
$$

12

In particular, the term inside $C$ is zero if and only if $\mathsf{Decode}(D, x) = H_1(x)$. Furthermore, because of the artificial abort introduced in the previous hybrid, this happens for $x \in X$ if and only if $x \in X \cap \tilde{Y}$. Hence, we can rewrite the sender's message $M$ as:

$$M = \{H_2(x, \mathsf{Decode}(Q, x) \oplus C(H_1(x)) \wedge s) \mid x \in X\}$$
$$= \{H_2(x, \mathsf{Decode}(R, x)) \mid x \in X \cap \tilde{Y}\}$$
$$\cup \{H_2(x, \mathsf{Decode}(R, x) \oplus C(\delta_x) \wedge s) \mid x \in X \setminus \tilde{Y}\}$$

where the $\delta_x := \mathsf{Decode}(D, x) \oplus H_1(x)$ values are guaranteed to be nonzero. This hybrid is identical to the previous one, as we have only rewritten the same computation in an equivalent way.

- Hybrid 4: Same as Hybrid 3, except we replace every term of the form $H_2(x, \mathsf{Decode}(R, x) \oplus C(\delta_x) \wedge s)$ with random. The two hybrids are indistinguishable by Lemma 1 since $C(\delta_x)$ are nonzero codewords and hence have Hamming weight at least $\kappa$. Now note that the sender's message $M$ is generated as:

$$M = \{H_2(x, \mathsf{Decode}(R, x)) \mid x \in X \cap \tilde{Y}\} \cup \{m_1, \ldots, m_{|X \setminus \tilde{Y}|}\}$$

where each $m_i$ is uniformly chosen in $\{0, 1\}^{\ell_2}$.

- Hybrid 5: Same as Hybrid 4, except the simulator no longer artificially aborts in the manner introduced in Hybrid 2. The hybrids are indistinguishable for the same reasoning as before. Now the simulator does not use the items of $X \setminus \tilde{Y}$ at all. We conclude the proof by observing that this hybrid exactly describes the final ideal-world simulation: the simulator extracts $\tilde{Y}$, sends it to the ideal PSI functionality, receives $Z = X \cap \tilde{Y}$, and uses it to simulate the sender's message $M$.

$\square$

**Lemma 6.** *The protocol of Figure 3 is secure against a malicious sender in the random oracle model.*

*Proof.* The simulator for a corrupt sender behaves as follows:

- It observes the sender's input $s$ and output $Q$ from OOS, and also observes all of the sender's queries to random oracle $H_2$.

- When the sender produces protocol message $M$, the simulator computes

$$\tilde{X} = \{x \mid x \text{ was queried to } H_2 \text{ and } H_2(x, \mathsf{Decode}(Q, x) \oplus C(H_1(x)) \wedge s) \in M\}$$

and sends this to the ideal functionality as the sender's effective input.

We prove the indistinguishability of this simulation in the following sequence of hybrids:

- Hybrid 1: Same as the real protocol interaction, except that the simulator observes the sender's input $s$ and output $Q$ for OOS, and additionally observes all queries made to random oracle $H_2$. The simulator defines a set $L$ of all the values $x$ such that the adversary queried $H_2$ on the "correct" value $(x, \mathsf{Decode}(Q, x) \oplus C(H_1(x)) \wedge s)$. When the sender gives protocol message $M$, the simulator defines the set $\tilde{X} := \{x \in L \mid H_2(x, \mathsf{Decode}(Q, x) \oplus C(H_1(x)) \wedge s) \in M\}$. This hybrid is identical to the real protocol interaction, since the only change is to record bookkeeping information that is not used.

13

- Hybrid 2: Same as Hybrid 1, except the simulator aborts if the honest receiver holds $y \in Y \setminus \tilde{X}$ where $H_2(y, \mathsf{Decode}(Q, y) \oplus C(H_1(y)) \wedge s) \in M$. There are two cases for why such a $y$ may not be in $\tilde{X}$:

  - Case $y \in L$: then the value $H_2(y, \mathsf{Decode}(Q, y) \oplus C(H_1(y)) \wedge s)$ was defined at the time $\tilde{X}$ was computed, and $y$ was excluded because the correct value was not in $M$. The simulator will never abort in this case.
  - Case $y \notin L$: the adversary never queried $H_2$ at $H_2(y, \mathsf{Decode}(Q, y) \oplus C(H_1(y)) \wedge s)$ before sending $M$, so this output of $H_2$ is random and independent of $M$. The probability that this $H_2$-output appears in $M$ is thus $|M|/2^{\ell_2}$ where $\ell_2$ is the output length of $H_2$.

  Overall, the probability of such an artificial abort is bounded by $n|M|/2^{\ell_2} \leq n^2/2^{\ell_1} \leq 1/2^{\lambda}$ (since $\ell_1 < \ell_2$ and $\ell_1 \geq \lambda + 2\log n$). Hence the two hybrids are indistinguishable.

- Hybrid 3: Same as Hybrid 2, except we change the way the honest receiver's output is computed. In Hybrid 2, the honest receiver computes output as in the protocol specification:

$$\{y \in Y \mid H_2(y, \mathsf{Decode}(R, y)) \in M\}$$

  In this hybrid we make the honest receiver compute its output as, simply, $\tilde{X} \cap Y$. These two expressions are in fact equivalent, from the definition of $\tilde{X}$, the artificial abort introduced in the previous expression, and the equivalence of $\mathsf{Decode}(R, y)$ and $\mathsf{Decode}(Q, y) \oplus C(H_1(y)) \wedge s$ discussed in the previous proof.

- Hybrid 4: Same as Hybrid 3, except we remove the artificial abort condition that was introduced in Hybrid 2. The hybrids are indistinguishable for the same reason as before. Note that in this hybrid, the simulator does not use the honest receiver's input $Y$ except to compute the receiver's final output. We conclude the proof by observing that this hybrid exactly describes the ideal world simulation: The simulator observes $s, Q$ and the sender's oracle queries to determine a set $\tilde{X}$. It sends $\tilde{X}$ to the ideal functionality and $\tilde{X} \cap Y$ is delivered to the receiver.

$\square$

## 4.4 Choosing Parameters

The protocol contains several parameters:

- A linear binary code $C : \{0, 1\}^{\ell_1} \to \{0, 1\}^t$.
- Random oracle output lengths $\ell_1, \ell_2$.

As shown in the security proof, the following facts must be true in order for security to hold:

- $C$ must have minimum distance at least $\kappa$ (the computational security parameter).
- $\ell_1, \ell_2 \geq \lambda + 2\log n$, where $\lambda$ is the statistical security parameter.

However, the parameters $\ell_1, \ell_2$ also have an effect on the size of the corrupt party's set, as extracted by the simulator. In particular, increasing these values causes the protocol to more tightly enforce the size ($n'$) of the corrupt party's input set.

| $n$ | $\ell_1 = \ell_2 = \lambda + 2\log n$ | codeword lengh $t$ | choice of code |
|---|---|---|---|
| $2^{12}$ | 64 | 448 | $RS[28, 13, 16]_{32}$ composed with $(16, 5, 8)_2$ |
| $2^{16}$ | 72 | 473 | $RS[42, 12, 32]_{64}$ composed with $(11, 6, 4)_2$ |
| $2^{20}$ | 80 | 495 | $RS[45, 14, 32]_{64}$ composed with $(11, 6, 4)_2$ |
| $2^{24}$ | 88 | 506 | $RS[46, 15, 32]_{64}$ composed with $(11, 6, 4)_2$ |

Figure 4: Parameters for semi-honest instantiation of PaXoS-PSI, with $\kappa = 128$ and $\lambda = 40$.

We note that the communication cost of the protocol is roughly $\ell_2$ bits per item from the sender and roughly $t$ bits per item from the receiver (sent as part of the OOS protocol, where $t$ is the length of the code used in the OOS protocol).

**Semi-honest security.** To instantiate our protocol for semi-honest security, it is enough to set $\ell_1 = \ell_2 = \lambda + 2\log n$, the minimum possible value for security. The issue of extracting a corrupt party's input, which involves further increasing $\ell_1, \ell_2$, is not relevant in the semi-honest case.

It therefore suffices to identify linear (binary) codes with suitable minimum distance, for the different values of $\ell_1$ that result. We identify good choices in Figure 4, all of which are the result of concatenating a Reed-Solomon code with a small (optimal) binary code.

**Malicious sender's set size.** Consider a malicious sender and recall how the simulator extracts an effective input for that sender. The sender gives protocol message $M$ and the simulator extracts via

$$\tilde{X} := \{x \in L \mid H_2(x, \text{Decode}(Q, x) \oplus C(H_1(x)) \wedge s) \in M\}$$

where $L$ is the set of $x$ values such that the adversary has queried $H_2(x, \cdot)$. The protocol limits the protocol message $M$ to have $n$ items, but still $\tilde{X}$ may have many more than $n$ items if the adversary manages to find collisions in $H_2$. If we set $\ell_2$ (the output length of $H_2$) to be $2\kappa$, then collisions are negligibly likely and indeed $|\tilde{X}| \leq n$ except with negligible probability.

While it is possible to set $\ell_2 < 2\kappa$, doing so has less impact on the protocol than the other parameters ($\ell_1$ and hence $t$). One can reduce $\ell_2$ only very slightly before the adversary can find a very large amount (*e.g.*, superlinear in $n$) of collisions. For these reasons, **we recommend setting $\ell_2 = 2\kappa$ in our malicious instantiation.**

**Malicious receiver's set size.** Consider a malicious receiver and recall how the simulator extracts an effective input for that receiver. The simulator observes the receiver's input $D$ (a PaXoS) to OOS and also observes all queries made to the random oracle $H_1$. Then the simulator extracts via:

$$\tilde{Y} := \{y \in L \mid \text{Decode}(D, y) = H_1(y)\}$$

where $L$ is the set of queries made to $H_1$. The question becomes: as a function of $|D|$ and $\ell_1$ (the output length of $H_1$), what is an upper bound on the number of items in $\tilde{Y}$?

In Appendix A we prove the following, using an information-theoretic compression argument:

**Claim 7.** *Suppose an adversary makes $q$ queries to random oracle $H_1$ with output length $\ell_1$ and then generates a PaXoS $D$ of size $m$ (hence $m\ell_1$ bits) total. Fix a value $n'$ and let $\mathcal{E}$ denote the event that $\text{Decode}(D, y) = H_1(y)$ for at least $n'$ values $y$ that were queried to $H_1$. Then*

$$\Pr[\mathcal{E}] \leq \binom{q}{n'}/2^{(n'-m)\ell_1}.$$

| $m$ | $n'$ | $\ell_1$ | codeword len $t$ | choice of code |
|---|---|---|---|---|
| $2^{12}$ | $2m$ | 233 | 776 | $RS[97,34,64]_{128}$ composed with $(8,7,2)_2$ |
| $2^{12}$ | $3m$ | 174 | 660 | $RS[60,29,32]_{64}$ composed with $(11,6,4)_2$ |
| $2^{12}$ | $4m$ | 154 | 627 | $RS[57,26,32]_{64}$ composed with $(11,6,4)_2$ |
| $2^{12}$ | $5m$ | 144 | 605 | $RS[55,24,32]_{64}$ composed with $(11,6,4)_2$ |
| $2^{16}$ | $2m$ | 225 | 768 | $RS[64,33,32]_{128}$ composed with $(12,7,4)_2$ |
| $2^{16}$ | $3m$ | 168 | 649 | $RS[59,28,32]_{64}$ composed with $(11,6,4)_2$ |
| $2^{16}$ | $4m$ | 149 | 616 | $RS[56,25,32]_{64}$ composed with $(11,6,4)_2$ |
| $2^{16}$ | $5m$ | 139 | 605 | $RS[55,24,32]_{64}$ composed with $(11,6,4)_2$ |
| $2^{20}$ | $2m$ | 217 | 744 | $RS[62,31,32]_{128}$ composed with $(12,7,4)_2$ |
| $2^{20}$ | $3m$ | 162 | 638 | $RS[58,27,32]_{64}$ composed with $(11,6,4)_2$ |
| $2^{20}$ | $4m$ | 144 | 605 | $RS[55,24,32]_{64}$ composed with $(11,6,4)_2$ |
| $2^{20}$ | $5m$ | 134 | 594 | $RS[54,23,32]_{64}$ composed with $(11,6,4)_2$ |
| $2^{24}$ | $2m$ | 209 | 732 | $RS[61,30,32]_{128}$ composed with $(12,7,4)_2$ |
| $2^{24}$ | $3m$ | 156 | 627 | $RS[57,26,32]_{64}$ composed with $(11,6,4)_2$ |
| $2^{24}$ | $4m$ | 138 | 594 | $RS[54,23,32]_{64}$ composed with $(11,6,4)_2$ |
| $2^{24}$ | $5m$ | 129 | 583 | $RS[53,22,32]_{64}$ composed with $(11,6,4)_2$ |

Figure 5: Parameters for malicous PaXoS-PSI with $\kappa = 128$ and Pr[simulator extracts $> n'$ items from malicious receiver] $< 1/2^{40}$, where adversary makes $2^{128}$ queries to $H_1$.

The idea behind the proof is that if a PaXoS $D$ happens to encode many $H_1(y)$ values, then $D$ could be used to compress $H_1$. However, this is unlikely due to $H_1$ being a random object and therefore incompressible.

For reference, we have computed some concrete parameter settings so that $\Pr[\mathcal{E}] < 2^{-40}$ (the probability that the simulator extracts more than $n'$ items). The values are given in Figure 5. We consider an adversary making $q = 2^{128}$ queries to $H_1$, which is rather conservative (in terms of security). In practice significantly smaller parameters may be possible.[4] Note that if the PaXoS has size $m$, then a compression argument such as the one we use only starts to apply when $n' > m$. Hence all of our bounds are expressed as $n' = cm$ where $c > 1$ is a small constant.

Recall that $\ell_1$ is the input length to the linear code $C$, so increasing it has the effect of increasing $t$ (the codeword length) as well. We include good choices of codes (achieving minimum distance $\kappa = 128$) in the figure as well.

## 5  Garbled Cuckoo Table

We introduce a new approach for PaXoS that enjoys the best of all worlds: it has the same asymptotic encoding and decoding costs as a garbled Bloom filter, but with constant rate (e.g., $\sim 1/(2 + \epsilon)$) rather than a $O(1/\lambda)$ rate. Furthermore, it has a linear time construction, just like the modified Bloomier filter of [CC08], but with the advantage of having the hash function(s) independent of the keys/values.

---

[4]For example, considering an adversary who makes $q = 2^{80}$ queries to $H_1$ leads to $\ell_1$ in the range of 70 to 90, and codeword length $t$ in the range of 460 to 510.

## 5.1 Overview

Our construction uses ideas from both garbled Bloom filters as well as Cuckoo hashing. Recall that in Cuckoo hashing, it is typical to have only 2 hash functions $h_1, h_2$, where an item $x$ is associated with positions $h_1(x)$ and $h_2(x)$ in the data structure.

So as a starting point, consider a garbled Bloom filter with just 2 hash functions rather than $\lambda$. Such a data structure corresponds to the decoding function $\mathsf{Decode}(D, x) = d_{h_1(x)} \oplus d_{h_2(x)}$.[5] (Using the PaXoS key-value mapping terminology of Section 3.2, the vector $\boldsymbol{v}(x)$ has only two non-zero entries, in locations $h_1(x)$ and $h_2(x)$.) Given $n$ key-value pairs $(x_i, y_i)$, how can we generate a data structure $D = (d_1, \ldots, d_m)$ that encodes them in this way?

An important object in analyzing our construction is the **cuckoo graph**. The vertices in the cuckoo graph are numbered 1 through $m$, and correspond to the positions in the data structure $D$. The (undirected) edges of the graph correspond to items that are meant to be inserted. An item $x$ corresponds to the edge $\{h_1(x), h_2(x)\}$. (The graph may contain self-loops and repeated edges.) We refer to such graphs with $m$ vertices and $n$ edges as $(n, m)$-**cuckoo graphs** and note that the distribution over such graphs is independent of $X$. We write $G_{h_1,h_2,X}$ to refer to the specific $(n, m)$-cuckoo graph corresponding to a particular set of hash functions and keys $X$. All properties of our PaXoS can be understood in terms of properties of random $(n, m)$-cuckoo graphs.

In the simplest case, suppose that $G_{h_1,h_2,X}$ happens to be a **tree**. Our goal is to encode the items $X$ into the data structure $D$. Each node $g$ in the graph corresponds to a row $d_g$ of $D$. Then we can do this encoding in linear time as follows: We choose an arbitrary root vertex $r$ of the tree and set $d_r$ of the data structure arbitrarily. We then traverse the tree, say, in DFS or BFS order. Each time we visit a vertex $j$ for the first time, we set its corresponding value $d_j$ in the data structure, to agree with the edge we just traversed. This is done as follows.

Recall that each edge $ij$ corresponds to a key-value pair $(x, y)$ in the sense that $\{i, j\} = \{h_1(x), h_2(x)\}$ and our goal is to arrange that $d_i \oplus d_j = y$. As we cross an edge from $i$ to $j$ in the traversal, we have the invariant that position $d_i$ in the data structure has been already fixed but $d_j$ is still undefined. Hence, we can always set $d_j := d_i \oplus y$.

**Handling Cycles.** When $m = O(n)$, corresponding to a PaXoS of constant rate, the corresponding Cuckoo graph is unlikely to be acyclic [HMWC93]. In this case the encoding procedure that we just outlined does not work, since when the graph traversal closes a circuit it encounters a vertex whose value has already been defined and cannot be set to satisfy the constraint imposed by the current edge.

We can handle acyclic Cuckoo graphs by adding $d + \lambda$ additional entries to the data structure $D$. We first describe an analysis where $d$ is an upper bound on the size $\chi$ of the 2-core of the graph, and then an analysis where $d$ is an upper bound on the cyclomatic number $\sigma$ of the graph. (These bounds are $O((\log n)^{1+\omega(1)})$ and $\log n$, respectively.) We recall below the definitions of both these values, and note that $\sigma < \chi$ always.

The **2-core** of a graph is the maximum subgraph where each node has degree at least 2 (namely, the subgraph containing all cycles, as well as all paths connecting cycles). We use $\chi$ to denote the number of edges in the 2-core. The **cyclomatic number** of a graph is the minimum number of edges to remove to leave an acyclic graph. Equivalently, it is the number of non-tree edges (back edges) in a DFS traversal of the graph. We use $\sigma$ to denote the cyclomatic number. The cyclomatic

---

[5]For now, we ignore the case where $h_1(x) = h_2(x)$.

number is equal to the minimal number of independent cycles in the graph, and is therefore smaller than or equal to the number of cycles. It is also always strictly less than the size of the 2-core.

**The construction.** $D$ will be structured as $D = L\|R$, where $|L| = m$ (the number of vertices in the Cuckoo graph) and $|R| = d + \lambda$. Each decoding/constraint vector $\boldsymbol{v}(x)$ then has the form $\boldsymbol{v}(x) = \boldsymbol{l}(x)\|\boldsymbol{r}(x)$, where $\boldsymbol{l}(x)$ determines the positions of $L$ to be XOR'ed and $\boldsymbol{r}(x)$ determines the positions of $R$ to be XOR'ed. We will let $L$ correspond to the simple Cuckoo hashing idea above, so each $\boldsymbol{l}(x)$ vector is zeroes everywhere except for two 1s. We will let $\boldsymbol{r}(x)$ be determined uniformly at random for each $x$ (similar to the random matrix construction of a PaXoS).

To encode $n$ key-value pairs into the data structure in this way, first consider the system of linear equations induced by the constraints $\langle \boldsymbol{v}(x_i), D \rangle = y_i$, **restricted to only the $\chi$ items (edges) in the 2-core.** (Once we set values that encode the items in the 2-core, we will be able to encode the other items using graph traversal as in an acyclic graph.) These constraints refer to a vertex $u$ of $G$ only if that vertex is in the 2-core. We get $\chi$ equations over $m + d + \lambda$ variables, where the coefficients of the last $d + \lambda$ variables (the $\boldsymbol{r}(x)$ part) are random. If we set $d$ to be an upper bound on $\chi$ then we get that the system has a solution with probability $1 - 2^{-\lambda}$.

So, using a general-purpose linear solver we can find values for $R$ and for the subset of $L$ corresponding to the vertices in the 2-core, that satisfies these constraints. This can be done in $O((d + \lambda)^3)$ time. For vertices $u$ outside of the 2-core, the value of $d_u$ in the data structure remains undefined. But after removing the 2-core, the rest of the graph is such that these values in the data structure can be fixed according to a tree traversal process:

Every edge *not* in the 2-core can be oriented away from all cycles (if an edge leads to a cycle in both directions, then that edge would have been part of the 2-core). We traverse those edges following the direction of their orientation. Let edge $i \to j$ correspond to a key-value pair $(x, y)$. Let $d_j$ denote the position in $D$ (in its "$L$ region") corresponding to vertex $j$. By our invariant, $d_j$ is not yet fixed when we traverse $i \to j$. Yet it is the only undefined value relevant to the constraint $\langle \boldsymbol{v}(x), D \rangle = y$, so we can satisfy the constraint by solving for $d_j$. Hence with a linear pass over all remaining items, we finish constructing the data structure $D$.

The total cost of encoding is therefore $O((d + \lambda)^3 + n\lambda)$. We explained above that we can set $d$ to be an upper bound on the size of the 2-core.[6] As we shall see (in Section 5.2), it is possible to set $d$ to be the cyclomatic number of the Cuckoo graph, which is logarithmic in $n$. Therefore the dominating part of the expression is $n\lambda$.

## 5.2 Details

The garbled-cuckoo construction is presented formally in Figure 6.

**Analysis & Costs** We show in Lemma 12 in Appendix C that the number $\sigma$ of cycles is smaller than $\log n + O(1)$ except with negligible probability. Therefore we can set $d = (1 + \varepsilon) \log n$.[7] This bound also applies to the cyclomatic number (which is always smaller than or equal to the number of cycles). Theorem 8 shows that it is sufficient to set $d$ to be equal to this upper bound on the cyclomatic number.

---

[6] Such an upper bound for the case of Cuckoo hashing can be derived from [PP08, Lemma 3.4], but that analysis assumes that the graph has $8n$ edges, and shows that an upper bound of size $d$ fails with probability $n/2^{-\Omega(d)}$. Therefore we must set $d = (\log n)^{1+\epsilon}$ to get a negligible failure probability.

[7] The parameter $\varepsilon$ used here is independent of the parameter $\epsilon$ used in Cuckoo hashing.

Parameters:

- upper bound $d$ on the cyclomatic number of the Cuckoo graph

- error parameter $\lambda$

- random functions $h_1, h_2 : \{0,1\}^* \to \{1, \ldots, m\}$

- random function $\boldsymbol{r} : \{0,1\}^* \to \{0,1\}^{d+\lambda}$

$\underline{\mathsf{Decode}(D, x)}$:

1. Parse $D$ as $D = L \| R$ where $|L| = m$ and $|R| = d + \lambda$

2. Set $\boldsymbol{l}(x) \in \{0,1\}^m$ to be all zeroes except 1s at positions $h_1(x)$ and $h_2(x)$

3. Return $\langle \boldsymbol{l}(x), L \rangle \oplus \langle \boldsymbol{r}(x), R \rangle$

$\underline{\mathsf{Encode}((x_1, y_1), \ldots, (x_n, y_n))}$:

1. Construct the Cuckoo graph $G_{h_1, h_2, X}$ for $X = \{x_1, \ldots, x_n\}$ and let $\tilde{V}, \tilde{E}$ be the vertices and edges of its 2-core. If the number of cycles is greater than $d$ then abort.

2. Initialize variables $L = (l_1, \ldots, l_m)$ and $R = (r_1, \ldots, r_{d+\lambda})$.

3. Solve (*e.g.*, with Gaussian elimination) for variables $\{l_u \mid u \in \tilde{V}\} \cup R$ that satisfy:

$$\langle \boldsymbol{l}(x_i) \| \boldsymbol{r}(x_i), L \| R \rangle = y_i, \quad \forall x_i \in \tilde{E}$$

where $\boldsymbol{l}(\cdot), \boldsymbol{r}(\cdot)$ are as above.

4. For each connected component which is a tree, pick an arbitrary vertex $v$ as the root of the tree. Set the variable $l_v$ to a random value.

5. For each item/edge $x_i \notin \tilde{E}$, in order of a DFS traversal directed away from the 2-core (in connected components which include a cycle), or directed away from the root (in connected components which do not include a cycle)

   (a) Let $\{u, v\} = \{h_1(x_i), h_2(x_i)\}$ so that $l_u$ is already defined and $l_v$ is not.

   (b) Set $l_v := l_u \oplus \langle \boldsymbol{r}(x_i), R \rangle \oplus y_i$

6. Output $D = L \| R$

Figure 6: Garbled Cuckoo PaXoS

Recall that each item is mapped to a row $\boldsymbol{l}(x_i)\|\boldsymbol{r}(x_i)$ which contains an $\boldsymbol{l}(x_i)$ part with two 1 entries, and a random binary vector $\boldsymbol{r}(x_i)$ of length $\lambda + d$. We set $\lambda = 40$, and therefore for all practical input sizes we get that $d < \lambda$. We conclude that the number of 1 entries in the row vector is $O(\lambda)$.

The encoding processes each of $n$ edges once during the traversal. The computation involves XORing the locations pointed to by 1 entries in the row. The overhead of encoding all rows is $O(n\lambda)$. The decoding of a single item involves XORing the rows pointed by the two rows to which it is mapped, and is $O(\lambda)$.

**Theorem 8.** *When setting $d = (1 + \varepsilon) \log n$, the garbled cuckoo* PaXoS *of Figure 6 with parameter $\lambda$ is a $(n, m, \epsilon + 2^{-\lambda})$-*PaXoS *where*

$$\epsilon = \Pr[\text{the cyclomatic number of a random } (n, m)\text{-cuckoo graph} > \log n + O(1)]$$

*Proof.* As discussed above, we use here an upper bound $d$ for the cyclomatic number of the graph. Setting the bound to $d = (1 + \varepsilon) \log n$ works excepts with a negligible failure probability.

The proof bounds the probability that the Encode algorithm fails to satisfy the linear constraints $\langle \boldsymbol{v}(x_i), D \rangle = y_i$ for every $i$. For items $x_i$ that do not correspond to edges in the 2-core, Step 4 of Encode satisfies the appropriate linear constraint, by construction. For items in the 2-core, their linear constraints are fixed all at once in Step 3 of Encode. Hence, the construction only fails if Step 3 fails. Step 3 solves for the following system of equations:

$$\langle \boldsymbol{l}(x_i)\|\boldsymbol{r}(x_i), L\|R \rangle = y_i, \quad \forall x_i \in \tilde{E}$$

We interpret $\{\boldsymbol{l}(x_i)\|\boldsymbol{r}(x_i)\}_{x_i \in \tilde{E}}$ as a matrix $M_L|M_R$ where the first $m$ columns (i.e., $M_L$) are $\{\boldsymbol{l}(x_i)\}_{x_i \in \tilde{E}}$ and the remaining $d + \lambda$ columns (i.e., $M_R$) are $\{\boldsymbol{r}(x_i)\}_{x_i \in \tilde{E}}$. We therefore ask whether the rows of the matrix $M_L|M_R$ are linearly independent.

There are up to $d$ cycles in the graph, denoted as $C_1, \ldots, C_d$. Let us focus on the matrix $M_L$, and more specifically on the rows corresponding an arbitrary cycle $C_i$ (each of these rows has two 1 entries, at the locations of the vertices touching the corresponding edge). It is easy to see that there is a single linear combination $D_i$ of these rows which is 0 (the XOR of all these rows). Any linear combination of $D_1, \ldots, D_d$ is 0, and these are the only linear combinations of rows which are equal to 0. Therefore there are at most $2^d$ such combinations and the kernel of $M_L$ is of dimension at most $d$.

Our goal is to find the probability of the existence of a zero linear combination of the rows of $M_L|M_R$, rather than the rows of $M_L$ alone. Since in $M_R$ each row contains $d + \lambda$ random bits, this probability is at most $2^{-\lambda}$. $\qquad\square$

$\square$

## 5.3 Comparison

Our construction shares many features with garbled Bloom filters (GBF), and indeed is somewhat inspired by them. Both our construction and GBF involve probing about the same number of positions per item ($\frac{\lambda + \chi + 2}{2}$ in average vs. $O(\lambda)$), however we are able to obtain constant rate while GBFs have rate $O(1/\lambda)$. We point out that GBFs inherit from standard Bloom filters their support for fully online insertion; that is, their analysis proves that items can be added to a GBF in any order. Our approach builds the data structure in a very particular order (according to a global tree

or tree-like structure of a graph). This qualitative difference seems important for achieving constant rate.

We also use much of the analysis techniques and terminology from cuckoo hashing (especially cuckoo hashing with a stash). However, one important difference with typical cuckoo hashing is that our construction can handle multiple cycles in a connected component of the cuckoo graph. Indeed, usual cuckoo hashing (without a stash) succeeds if each connected component of the graph has at most one cycle. The items in a cycle can be handled by arbitrarily assigning an orientation to the cycle, and assigning each edge (item) to its forward endpoint (position in the table). In our case, if some items form a cycle, their corresponding constraint vectors become linearly dependent and we cannot solve the system of linear equations. In general, our approach has a larger class of subgraphs which present a "barrier" to the process (where graphs with only 1 cycle are a barrier for us but not for standard cuckoo hashing), making the analyses slightly different.

## 5.4 An Alternative Construction

In Appendix B we describe a modified construction which is based on a DFS traversal of the graph, and has a similar overhead to the construction described in this section.

# 6 A Theoretical Comparison

In table 1 we show the *theoretical* communication complexity of our protocol compared with the Diffie-Hellman based PSI, the KKRT protocol [KKRT16] and the SpoT protocol [PRTY19] in the semi-honest setting, and the Rindal-Rosulek [RR17b] and Ghosh-Nilges [GN19] protocols in the malicious setting. Find a description of these protocols in Appendix D. This comparison measures how much communication the protocols require on an idealized network where we do not care about protocol metadata, realistic encodings, byte alignment, etc. In practice, data is split up into multiples of bytes (or CPU words), and different data is encoded with headers, etc. — empirical measurements of such real-world costs are given later in Section 7.

| Protocol | Communication | $n = n_1 = n_2$ | | |
|---|---|---|---|---|
| | | $2^{16}$ | $2^{20}$ | $2^{24}$ |
| **Semi Honest** | | | | |
| DH-PSI | $\phi n_1 z + (\phi + \lambda + \log(n_1 n_2))n_2$ | $584n$ | $592n$ | $600n$ |
| KKRT [KKRT16] | $(3 + s)(\lambda + \log(n_1 n_2))n_1 + 1.2\ell n_2$ | $1042n$ | $1018n$ | $978n$ |
| SpOT-low-comm [PRTY19] | $1.02(\lambda + \log_2(n_2) + 2)n_1 + \ell n_2$ | $488n$ | $500n$ | $512n$ |
| SpOT-fast [PRTY19] | $2(\lambda + \log(n_1 n_2))n_1 + \ell(1 + 1/\lambda)n_2$ | $583n$ | $609n$ | $634n$ |
| ours | $(\lambda + \log_2(n_1 n_2))n_1 + \ell(2.4n_2 + \lambda + \chi)$ | $\sim 1207n$ | $\sim 1268n$ | $\sim 1302n$ |
| **Malicious** | | | | |
| RR (EC-ROM) [RR17b] | $3\kappa n + n(2\kappa + \kappa \log n + \log^2 n)$ | $10112n$ | $10576n$ | $11024n$ |
| RR (SM) [RR17b] | $3\kappa n + n(2\kappa + \sigma\kappa \log n + \log^2 n)$ | $(200\text{k})n$ | $(220\text{k})n$ | $> (240\text{k})n$ |
| GN [GN19] | at least $8(n + 1)(\kappa + 2\sigma)$ | $> 3072n$ | $> 3072n$ | $> 3072n$ |
| ours | $(\lambda + \log_2(n_1 n_2))n_1 + \ell(2.4n_2 + 2\lambda + \chi) + \lambda(2.4n_2 + 2\ell)$ | $\sim 1623n$ | $\sim 1621n$ | $\sim 1602n$ |

Table 1: Theoretical communication costs of PSI protocols (in bits), calculated using computational security $\kappa = 128$ and statistical security $\lambda = 40$. Ignores cost of base OTs (in our protocol,KKRT, Sp) which are independent of input size. $n_1$ and $n_2$ are the input sizes of the sender and receiver respectively. $\phi$ is the size of elliptic curve group elements (256 is used here). $\ell$ is width of OT extension matrix (depends on $n_1$ and protocol. $\chi$ is the upper bound on the number of cycles in a cuckoo graph. $\sigma$ is the length of items ($\sigma = 64$ in the concrete numbers). "SH" and "M" denotes semi-honest and malicious setting. In RR protocols, EC-ROM and SM respectively denote their encode-commit model and the standard model dual execution variant.

**PaXoS PSI has linear communication complexity.** Let us clarify our claim of linear communication. Consider the *insecure* intersection protocol where Alice sends $H(x)$ for every $x$ in her set. $H$ could have output length equal to security parameter, giving $O(n \cdot \kappa)$ communication. But with semi-honest parties $H$ can also have output length as small as $\lambda + 2 \log(n)$ to ensure correctness with probability $1 - 1/2^\lambda$. When viewed this way, it looks like the protocol has complexity $O(n \log n)$! However, if $1/2^\lambda$ is supposed to be negligible then certainly $\log n \ll \lambda$, so one could still write $O(n \cdot \lambda)$.

If we let $L$ be a length that depends on the security parameters and $\log n$ (which is inherent to all intersection protocols, secure or not), then insecure PSI and PaXoS-PSI have complexity $O(L \cdot n)$, while previous OT-based malicious PSI [RR17b] has complexity $O(L \cdot n \log n)$ or even $O(L \cdot n\kappa)$ [RR17a]. For comparison, semi-honest KKRT [KKRT16] protocol has complexity $\omega(L \cdot n)$ (from the stash growing as $\omega(1)$) and semi-honest PRTY [PRTY19] has complexity $O(L \cdot n)$.

In [PRTY19] and in this work, $L$ can depend on the security parameter alone, leading to a $O(n \cdot \kappa)$ communication, which we would characterize as linear in $n$. But when choosing concrete parameters (just like in the insecure protocol) $L$ can be made smaller by involving a $O(\log n)$ term. Again, this is endemic to all intersection protocols.

# 7 Implementation & Evaluation

## 7.1 Implementation Details

We now present a comparison based on implementations of all protocols. We used the implementation of KKRT [KKRT16], RR [RR17b], HD-PSI, spot-low, spot-fast [PRTY19] from the open source-code[8] provided by the authors.

We evaluate the DH-PSI protocol, instantiated with two different elliptic curves: Curve25519 [Ber06] and Koblitz-283. Curve25519 elements are 256 bits while K-283 elements are 283 bits. Using the Miracl library, K-283 operations are faster than Curve25519, giving us a tradeoff of running time vs. communication for DH-PSI.

All OT-based PSI protocols [KKRT16, RR17b, PRTY19] (including our protocols) require the same underlying primitives: a Hamming correlation-robust function $H$, a pseudorandom function $F$, and base OTs for OT extension. We instantiated these primitives exactly as in previous protocols (e.g, KKRT, RR): both $H$ and $F$ instantiated using AES, and base OTs instantiated using Naor-Pinkas [NP01]. We use the implementation of base OTs from the libOTe library[9]. All protocols use a computational security parameter of $\kappa = 128$ and a statistical security parameter $\lambda = 40$.

For our own protocols, we implemented two variants of our PaXoS. We implemented the DFS traversal of the cuckoo graph (Appendix B ) using the boost library. We used additional libraries linbox, gmp, ntl, givaro iml, blas for solving systems for linear equations and generating the required concatenated linear codes needed for the 2-core based variant of Section 5. We use $2n$ bins in our DFS based PaXoS, and $2.4n$ bin in our 2-core based variant.

---

## 7.2 Experimental setup

We performed a series of benchmarks on the Amazon web services (AWS) EC2 cloud computing service. We used the M5.large machine class with 2.5GHz Intel Xeon and 8 GB RAM.6

We tested the protocols over three different network settings: LAN – two machines in the same region (N.Virginia) with bandwidth 4.97 GiB/s; WAN1 – one machine in N.Virginia and the other in Oregon with bandwidth 155 MiB/s; and WAN2 – one machine in N.Virginia and the other in Sydney with bandwidth 55 MiB/s. All experiments are performed with a single thread (with an additional thread used for communication).

## 7.3 Experimental Results

A detailed benchmark for set sizes $n = \{2^{12}, 2^{16}, 2^{20}\}$ is given in Table 2.

**Semi-honest PSI Comparison.** Our best protocol in terms of communication is PaXoS-DFS. The communication of this protocol is less than 10% larger than that of KKRT [KKRT16], and slightly more than twice the communication of SpOT-low.

Our best protocol in terms of run time is PaXoS 2-core. In the LAN setting for $2^{20}$ inputs, it runs only 18% slower than KKRT. In the two WAN settings it is about 80% slower.

**Malicious PSI Comparison.** The communication of both implementations of our protocol is better than that of RR. For $2^{20}$ items, PaXoS-DFS uses almost 8 times less communication, and PaXoS 2-core uses 6.5 less communication.

In terms of run time, PaXoS 2-core is faster than RR by a factor of about 2.5 on a LAN, and factors of 3.7-4 in the two WAN settings. The larger improvement in the WAN settings is probably due to the larger effect that the improvement in the communication has over a WAN.

**Semi-honest vs. Malicious.** In both our implementations of PaXoS the malicious implementation uses only about 25% more communication than the semi-honest implementation. In the LAN setting, our malicious protocols run about 4% slower than our corresponding semi-honest protocols.

## References

[ACT11]    Giuseppe Ateniese, Emiliano De Cristofaro, and Gene Tsudik. (if) size matters: Size-hiding private set intersection. In *PKC*, pages 156–173, 2011.

[ALSZ13a]  Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM CCS*, pages 535–548, 2013.

[ALSZ13b]  Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 535–548. ACM Press, November 2013.

[Ber06]    Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In *PKC*, pages 207–228, 2006.

| Protocol | comm (MB) | | | LAN | | | WAN1 | | | WAN2 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $2^{12}$ | $2^{16}$ | $2^{20}$ | $2^{12}$ | $2^{16}$ | $2^{20}$ | $2^{12}$ | $2^{16}$ | $2^{20}$ | $2^{12}$ | $2^{16}$ | $2^{20}$ |
| **Semi Honest** | | | | | | | | | | | | |
| DH-PSI (K-283) | 0.32 | 5.2 | 84.0 | 4597 | 73511 | | 6529 | 75839 | | 36979 | 80775 | |
| DH-PSI (25519) | 0.29 | 4.7 | 76.1 | 8797 | 140507 | | 12558 | 142922 | | 17641 | 150703 | |
| KKRT [KKRT16] | 0.53 | 8.06 | 127 | 177 | 339 | 4551 | 586 | 1361 | 9809 | 932 | 1983 | 22887 |
| SpOT-low [PRTY19] | **0.25** | **3.9** | **63.1** | 898 | 10173 | | 3693 | 18068 | | 5425 | 22152 | |
| SpOT-fast [PRTY19] | 0.3 | 4.71 | 76.4 | 460 | 1964 | 24442 | 6464 | 11602 | 31944 | 9565 | 11088 | 36187 |
| PaXoS 2-core (Sec. 5) | 0.65 | 10.19 | 163.63 | 16 | 235 | 5378 | 641 | 1664 | 17628 | 1592 | 4140 | 40524 |
| PaXoS DFS (App. B) | 0.55 | 8.59 | 138.03 | 94 | 838 | 17140 | 790 | 1968 | 33707 | 1805 | 19265 | 50218 |
| **Malicious** | | | | | | | | | | | | |
| RR (EC-ROM) [RR17b] | 4.8 | 79 | 1322 | 144 | 828 | 13996 | 1723 | 5061 | 69003 | 4320 | 12979 | 161632 |
| RR (SM, $\sigma = 64$) [RR17b] | 92 | 1317 | 22183 | 596 | 7330 | | 6190 | 67310 | | 14867 | 161868 | |
| PaXoS 2-core (Sec. 5) | 0.81 | 12.59 | 202.04 | **120** | **257** | **5598** | **644** | **1800** | **18621** | **1770** | **4546** | 40810 |
| PaXoS DFS (App. B) | **0.69** | **10.59** | **170.04** | 198 | 913 | 17903 | 881 | 2351 | 29078 | 2329 | 8660 | **31737** |

Table 2: Communication in MB and run time in milliseconds for related works over $n = \{2^{12}, 2^{16}, 2^{20}\}$ items and over three network settings as described in the text. DH-PSI has two versions, with two different curves: K-283 and 25519. EC-ROM is the encode-commit version in [RR17b] and $\sigma$ is the input length of the parties. All protocols run with $\sigma = 128$ except RR (SM) that can run with 64 at most bit items. 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 experiment that failed due to lack of memory or they took too much time.

[Can17]  Clment Canonne. A short note on poisson tail bounds. web page, 2017. http://www.cs.columbia.edu/~ccanonne/files/misc/2017-poissonconcentration.pdf.

[CC08]  Denis Xavier Charles and Kumar Chellapilla. Bloomier filters: A second look. In *Algorithms - ESA*, pages 259–270, 2008.

[CHLR18]  Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1223–1237, 2018.

[CKT10]  Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *ASIACRYPT*, pages 213–231, 2010.

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

[CT12]  Emiliano De Cristofaro and Gene Tsudik. Experimenting with fast private set intersection. In *TRUST 2012*, pages 55–73, 2012.

[DCW13]  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.

[DMRY09]  Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud, editors, *ACNS 09*, volume 5536 of *LNCS*, pages 125–142. Springer, Heidelberg, June 2009.

[EGL85]  Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28:637–647, 1985.

[FHNP16]  Michael J. Freedman, Carmit Hazay, Kobbi Nissim, and Benny Pinkas. Efficient set intersection with simulation-based security. *Journal of Cryptology*, 29(1):115–155, January 2016.

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

[FNO18]      Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Private set intersection with linear communication from general assumptions. ePrint Archive, Report 2018/238, 2018.

[FNP04]      Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19. Springer, Heidelberg, May 2004.

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

[GNN17]      Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. Maliciously secure oblivious linear function evaluation with constant overhead. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 629–659. Springer, Heidelberg, December 2017.

[Gol04]      Oded Goldreich. *Foundations of Cryptography, Volume 2: Basic Applications*. Cambridge University Press, 2004.

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

[HFH99]      Bernardo A. Huberman, Matthew K. Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *EC*, pages 78–86, 1999.

[HL10]       Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. *J. Cryptology*, 23(3):422–456, 2010.

[HMWC93]     George Havas, Bohdan S. Majewski, Nicholas C. Wormald, and Zbigniew J. Czech. Graphs, hypergraphs and hashing. In *Graph-Theoretic Concepts in Computer Science, 19th International Workshop, WG '93, Utrecht, The Netherlands, June 16-18, 1993, Proceedings*, pages 153–165, 1993.

[HN12]       Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. *Journal of Cryptology*, 25(3):383–433, July 2012.

[IKN+17]     Mihaela Ion, Ben Kreuter, Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, David Shanahan, and Moti Yung. Private intersection-sum protocol with applications to attributing aggregate ad conversions. *ePrint Archive 2017/738*, 2017.

[IKNP03a]    Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, pages 145–161, 2003.

[IKNP03b]    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.

[IPS09]      Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 294–314. Springer, Heidelberg, March 2009.

[JL09]       Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, pages 577–594, 2009.

[KK13]       Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 54–70. Springer, Heidelberg, August 2013.

[KKRT16]   Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched OPRF with applications to PSI. In *ACM CCS*, 2016.

[KLS+17]    Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. Private set intersection for unequal set sizes with mobile applications. *PoPETs*, 2017(4):177–197, 2017.

[KMRS14]  Seny Kamara, Payman Mohassel, Mariana Raykova, and Seyed Saeed Sadeghian. Scaling private set intersection to billion-element sets. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *FC 2014*, volume 8437 of *LNCS*, pages 195–215. Springer, Heidelberg, March 2014.

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

[Lam16]     Mikkel Lambæk. Breaking and fixing private set intersection protocols. Master's thesis, Aarhus University, 2016.

[Mea86a]   C. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *IEEE S&P*, 1986.

[Mea86b]   Catherine A. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *Proceedings of the 1986 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 7-9, 1986*, pages 134–137, 1986.

[NP01]       Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In S. Rao Kosaraju, editor, *12th SODA*, pages 448–457. ACM-SIAM, January 2001.

[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.

[PP08]       Anna Pagh and Rasmus Pagh. Uniform hashing in constant time and optimal space. *SIAM J. Comput.*, 38(1):85–96, 2008.

[PR04]       Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.

[PRTY19]   Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse OT extension. In *CRYPTO*, pages 401–431, 2019.

[PSS17]     Arpita Patra, Pratik Sarkar, and Ajith Suresh. Fast actively secure OT extension for short secrets. In *NDSS*, 2017.

[PSSZ15]   Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX*, pages 515–530, 2015.

[PSTY19]    Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient circuit-based PSI with linear communication. In *EUROCRYPT*, pages 122–153, 2019.

[PSWW18]    Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In *Advances in Cryptology - EUROCRYPT 2018, Part III*, pages 125–157, 2018.

[PSZ14]    Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In *USENIX 2014*, pages 797–812, 2014.

[PSZ18]    Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. *ACM Trans. Priv. Secur.*, 21, 2018.

[RA18]    Amanda C. Davi Resende and Diego F. Aranha. Unbalanced approximate private set intersection. *Financial Crypto*, 2018.

[Rab05]    Michael O. Rabin. How to exchange secrets with oblivious transfer. *ePrint Archive 2005/187*, 2005, 2005.

[RR17a]    Peter Rindal and Mike Rosulek. Improved private set intersection against malicious adversaries. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 235–259. Springer, Heidelberg, May 2017.

[RR17b]    Peter Rindal and Mike Rosulek. Malicious-secure private set intersection via dual execution. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 17*, pages 1229–1242. ACM Press, October / November 2017.

[Sha80]    Adi Shamir. On the power of commutativity in cryptography. In *Automata, Languages and Programming*, 1980.

[Tak88]    Lajos Takcs. On the limit distribution of the number of cycles in a random graph. *Journal of Applied Probability*, 25:359–376, 1988.

# A    Bound on Malicious Receiver's Set Size

In this section we prove a lower bound on the number of items an adversary can pack into a PaXoS. The proof uses an incompressibility argument, which is based on the following structure:

**Proposition 9.** *Suppose $E : A \to B$ and $D : B \to A$ are deterministic functions. Then,*

$$\Pr_{a \leftarrow A}[D(E(a)) = a] \leq |B|/|A|.$$

In other words, it is not possible to compress $A$ into a smaller $B$ without losing information.

*Proof.* The simple reason is that $D$ only has $|B|$ possible outputs. So $D(E(a)) = a$ can only be correct if $a$ is one of the possible outputs of $D$.     □

We now use this approach to prove the lower bound on items in a PaXoS.

**Lemma 10.** *Suppose an adversary makes $q$ queries to random oracle $H_1$ with output length $\ell_1$ and then generates a PaXoS $D$ of size $m$ (hence $m\ell_1$ bits) total. Fix a value $n'$ and let $\mathcal{E}$ denote the event that $\mathsf{Decode}(D, y) = H_1(y)$ for at least $n'$ values $y$ that were queried to $H_1$. Then*

$$\Pr[\mathcal{E}] \leq \binom{q}{n'}/2^{(n'-m)\ell_1}.$$

*Proof.* Given such an adversary, we construct a "compression" scheme $E, D$ that compresses the random oracle $H_1$ whenever the event $\mathcal{E}$ happens.

Without loss of generality, $H_1$ is restricted to inputs of some finite length $v$, and there are $(2^{\ell_1})^{2^v}$ possible values for $H_1$. The algorithm $E$ works as follows, given $H_1$ as input.

1. Run the adversary with oracle $H_1$.

2. The adversary outputs a PaXoS $D$. If event $\mathcal{E}$ does not occur ($D$ correctly encodes less than $n'$ values), then output $\perp$ and abort. Otherwise continue.

3. The PaXoS $D$ has size $m$; include $D$ as part of the output (there are $2^{\ell_1 m}$ possible values for $D$).

4. Since $\mathcal{E}$ happened, there are $n'$ queries that were made (out of the $q$ total) which are correctly encoded in $D$. Include in the output the indices of these $n'$ oracle queries. (there are $\binom{q}{n'}$ possible values for these indices)

5. For the remaining $q - n'$ queries $x$ made by the adversary, include $H_1(x)$ in the output, in the order they were queried. (there are $(2^{\ell_1})^{q-n'}$ possible values for these oracle outputs)

6. For all other $2^v - q$ values $x$ in $H_1$'s domain that were *not* queried by the adversary, include $H_1(x)$ in the output, in lexicographic order of $x$. (there are $(2^{\ell_1})^{2^v - q}$ possible values for these oracle outputs)

The corresponding "decompression" algorithm $D$ works in a natural way to reconstruct all of $H_1$ given the output of $E$.

1. Run the adversary, and when it makes the $i$th oracle query $x$:

    (a) If $i$ is among the $n'$ "distinguished" indices, then retrieve $H_1(x)$ as $\mathsf{Decode}(D, x)$

    (b) Otherwise, retrieve $H_1(x)$ as the next unused value from the list generated in Step 5 of $E$.

2. After the adversary terminates, fill in the missing parts of $H_1$ from the list generated in Step 6 of $E$.

Hence, $\Pr[\mathcal{E}] \leq \Pr[D(E(a)) = a]$. Now, $E : A \to B$ where (as we argued above) $|A| = (2^{\ell_1})^{2^v}$, and $|B| = 2^{\ell_1 m} \cdot \binom{q}{n'} \cdot (2^{\ell_1})^{q-n'} \cdot (2^{\ell_1})^{2^v - q}$.

From Proposition 9, we therefore have:

$$\Pr[\mathcal{E}] \leq \Pr[D(E(a)) = a]$$

$$\leq \frac{2^{\ell_1 m} \cdot \binom{q}{n'} \cdot (2^{\ell_1})^{q-n'} \cdot (2^{\ell_1})^{2^v - q}}{(2^{\ell_1})^{2^v}}$$

$$= \binom{q}{n'} \frac{(2^{\ell_1})^{m + (q - n') + (2^v - q)}}{(2^{\ell_1})^{2^v}}$$

$$= \binom{q}{n'} (2^{\ell_1})^{m - n'}$$

This completes the proof. □

# B  An Alternative Garbled Cuckoo PaXoS

In this section we describe another PaXoS construction based on cuckoo hashing. This new construction has size $m + d + \lambda$ where $d$ is an upper bound on, $\sigma$, the *cyclomatic number* of the cuckoo graph. (In the following text, we interchange between $d$ and $\sigma$.)

Our construction is indeed inspired by the idea of a DFS traversal of the cuckoo graph. When encountering a new tree edge in a DFS traversal, we traverse an edge towards a vertex that has not been visited previously. Such a vertex corresponds to a position in the PaXoS that we can then fix in order to satisfy the linear constraint for that edge/item. The only concern, therefore, is how to deal with inevitable non-tree edges in the traversal.

We do so by using the additional $\sigma + \lambda$ positions of the PaXoS. Each DFS back-edge corresponds to a cycle in the graph. Say the cycle includes items $x_1, \ldots, x_k$. Our idea is to insert a constraint that expresses the condition $\mathsf{Decode}(D, x_1) \oplus \cdots \oplus \mathsf{Decode}(D, x_k) = y_1 \oplus \cdots y_k$. The set of all such constraints (one per each cycle, so at most $\sigma$ in total) can be expressed using only the $\sigma + \lambda$ "extra" positions of the PaXoS. One way to interpret this constraint is: "if all but one of the $x_i$'s on this cycle give correct output $y_i$, then the remaining $x_i$ must also give correct output." And since all but one edge in the cycle is a DFS tree edge, we can indeed satisfy the condition and ensure even the DFS back-edge gives correct output.

The details of the construction are given in Figure 7.

**Theorem 11.** *The PaXoS of Figure 7 with parameters $\sigma$ and $\lambda$ is a $(n, m, \epsilon + \lambda)$-PaXoS where*

$$\epsilon = \Pr[\text{cyclomatic number of a random } (n, m)\text{-cuckoo graph is more than } \sigma]$$

*Proof.* We focus on the correctness of the construction, as the error probability of the construction follows closely the analysis of the other PaXoS scheme.

When an edge/item $x_i$ is not a back edge then the construction indeed generates a $D$ that satisfies the constraint $\mathsf{Decode}(D, x_i) = y_i$. This is by construction of step 5b. Note that step 5b itself always succeeds since the defining property of a DFS tree edge is that one of its endpoints is being visited for the first time (vertex $v$ in this case). Hence, the variable $l_v$ is always free to be fixed in step 5b.

It suffices to show why the construction correctly achieves $\mathsf{Decode}(D, x_{i^*}) = y_{i^*}$ when $x_{i^*}$ is a back edge in the DFS traversal, even though these edges are ignored in step 5a.

Parameters:

- upper bound $\sigma$ on cyclomatic number

- error parameter $\lambda$

- random functions $h_1, h_2 : \{0,1\}^* \to \{1, \ldots, m\}$

- random function $\boldsymbol{r} : \{0,1\}^* \to \{0,1\}^{\sigma+\lambda}$

$\underline{\mathsf{Decode}(D, x):}$

1. Parse $D$ as $D = L\|R$ where $|L| = m$ and $|R| = \sigma + \lambda$

2. Set $\boldsymbol{l}(x) \in \{0,1\}^m$ as follows: if $h_1(x) \neq h_2(x)$ then set $\boldsymbol{l}(x)$ to be all zeroes except 1s at positions $h_1(x)$ and $h_2(x)$. If $h_1(x) = h_2(x)$ then set $\boldsymbol{l}(x) = 0^m$.

3. Return $\langle \boldsymbol{l}(x), L \rangle \oplus \langle \boldsymbol{r}(x), R \rangle$

$\underline{\mathsf{Encode}((x_1, y_1), \ldots, (x_n, y_n)):}$

1. Construct the Cuckoo graph $G_{h_1, h_2, X}$ for $X = \{x_1, \ldots, x_n\}$.

2. Initialize variables $L = (l_1, \ldots, l_m)$ and $R = (r_1, \ldots, r_{\chi+\lambda})$ and an initially empty set of linear constraints $S$.

3. Perform a DFS on $G_{h_1, h_2, X}$. For each non-tree edge (back edge), of which there are at most $\sigma$:

   (a) The back edge closes a cycle of edges/items. Let $C$ denote the indices of those items, so that the cycle corresponds to items $\{x_i \mid i \in C\}$.

   (b) Add to $S$ the following constraint (in formal variable $R$):

$$\left\langle \left( \bigoplus_{i \in C} \boldsymbol{r}(x_i) \right), R \right\rangle = \bigoplus_{i \in C} y_i$$

4. Solve (*e.g.*, with Gaussian elimination) for variables $R$ satisfying the constraints of system $S$.

5. Perform another DFS on $G_{h_1, h_2, X}$ with identical ordering as before. Set variable $l_v$ arbitrarily, where $v$ is the root vertex of the traversal. Then, for each edge/item $x_i$ in the DFS traversal:

   (a) If $x_i$ is a back-edge, then do nothing.

   (b) Otherwise, let $\{u, v\} = \{h_1(x_i), h_2(x_i)\}$ where $l_u$ has already been fixed and $l_v$ has not. Set $l_v := l_u \oplus \langle \boldsymbol{r}(x_i), R \rangle \oplus y_i$.

6. Output $D = L\|R$

Figure 7: DFS-Based Garbled Cuckoo PaXoS

Let $C$ be the cycle corresponding to $x_{i^*}$. Being a cycle, we have that

$$\bigoplus_{i \in C} \boldsymbol{l}(x_i) = 0$$

(This includes the case where the cycle is a self-loop, since in that case we define $\boldsymbol{l}(x_i) = 0$.) Since $x_{i^*}$ is the only back-edge among $C$, all other edges in $C$ are correctly encoded in $D$; that is:

$$(\forall i \in C \setminus \{i^*\}) \quad \mathsf{Decode}(D, x_i) \overset{\text{def}}{=} \langle \boldsymbol{l}(x_i), L \rangle \oplus \langle \boldsymbol{r}(x_i), R \rangle = y_i$$

We also have from Steps 3b-4 of the construction that:

$$\left\langle \bigoplus_{i \in C} \boldsymbol{r}(x_i), R \right\rangle = \bigoplus_{i \in C} y_i$$

Putting everything together, we get:

$$
\begin{aligned}
\mathsf{Decode}(D, x_{i^*}) &\overset{\text{def}}{=} \langle \boldsymbol{l}(x_{i^*}), L \rangle \oplus \langle \boldsymbol{r}(x_{i^*}), R \rangle \\
&= \langle \boldsymbol{l}(x_{i^*}), L \rangle \oplus \langle \boldsymbol{r}(x_{i^*}), R \rangle \oplus \bigoplus_{i \in C \setminus \{i^*\}} \underbrace{\left( \langle \boldsymbol{l}(x_i), L \rangle \oplus \langle \boldsymbol{r}(x_i), R \rangle \oplus y_i \right)}_{\text{zero!}} \\
&= \left[ \bigoplus_{i \in C} \langle \boldsymbol{l}(x_i), L \rangle \right] \oplus \left[ \bigoplus_{i \in C} \langle \boldsymbol{r}(x_i), R \rangle \right] \oplus \left[ \bigoplus_{i \in C \setminus \{i^*\}} y_i \right] \\
&= \left\langle \bigoplus_{i \in C} \boldsymbol{l}(x_i), L \right\rangle \oplus \left\langle \bigoplus_{i \in C} \boldsymbol{r}(x_i), R \right\rangle \oplus \left[ \bigoplus_{i \in C \setminus \{i^*\}} y_i \right] \\
&= 0 \oplus \left[ \bigoplus_{i \in C} y_i \right] \oplus \left[ \bigoplus_{i \in C \setminus \{i^*\}} y_i \right] \\
&= y_{i^*}
\end{aligned}
$$

And hence even the non-tree edge/item $x_{i^*}$ is encoded into $D$ correctly.

The encoding process fails only if the cyclomatic number of the graph exceeds $\sigma$, or the set of constraints $S$ has no solution in step 4. Observe that the constraints in $S$ are all independently random, since each constraint includes a random term $\boldsymbol{r}(x_i)$ from a back-edge that is independent of all other $\boldsymbol{r}(\cdot)$ values included so far. Overall $S$ consists of at most $\sigma$ *random* constraints of length $\sigma + \lambda$, and so has a solution except with probability $1/2^\lambda$. $\qquad\square$

## C  Bounding the Number of Cycles in a Cuckoo Graph

**Lemma 12.** *The number of cycles in a Cuckoo graph with $(2 + \epsilon)n$ nodes is at most $\log n$, except with negligible probability.*

**Proof:** It was shown in [Tak88] that the number of cycles in a Cuckoo graph has a Poisson distribution. More accurately, it was shown that in a graph where each edge is connected with iid probability $p$, the total number of cycles has a Poisson distribution with expectation $-0.5 \log(1 - pm) - \frac{pm}{2} - \frac{(pm)^2}{4}$.

In Cuckoo hashing of $n$ items, the number of nodes is $m = (2+\epsilon)n$. Therefore $p = n/m^2 = \frac{1}{(2+\epsilon)^2 n}$ and $pm = \frac{1}{(2+\epsilon)}$. The expectation of the distribution is

$$E = -0.5 \log(1 - \frac{1}{(2+\epsilon)}) - \frac{1}{2(2+\epsilon)} - \frac{1}{4(2+\epsilon)^2} = O(1)$$

It is also known (see, e.g., [Can17]) that for a Poisson variable $X$ with expectation $E$,

$$\Pr(X > E + \Delta) \leq e^{-\frac{\Delta^2}{2E} h(\Delta/E)}$$

where $h(u) = 2 \frac{(1+u)\ln(1+u)-u}{u^2}$.

Since $E = O(1)$, setting $\Delta = O(E)$ will only result in a constant failure probability. However, setting $\Delta = \log n$, we get that the failure probability is negligible:

$$
\begin{aligned}
Pr(X > \log n + \Omega(1)) &\leq e^{-\frac{\log^2 n}{2E} h(\frac{\log n}{E})} = e^{-\Omega(\log^2 n)\Theta(\frac{\log\log n}{\log n})} \\
&= e^{-\Omega(\log n \log\log n)} = O(n^{-\log\log n})
\end{aligned}
$$

# D   Comparison

In the following we compare our results to relevant related work, focusing on qualitative differences between the protocols. Even though our protocol is maliciously secure, we find that its performance is competitive even with state-of-the-art protocols with only semi-honest security. Therefore, we chose to present a comparison to concretely efficient protocols in both settings.

**Protocols based on Diffie-Hellman key exchange.**   These protocols are based on ideas presented in [Mea86a, HFH99]. For a hash function $H$ modeled as a random oracle and underlying acyclic group $\mathbb{G}$ of prime order, for which the DDH problem is hard, the sender and receiver sample $\alpha$ and $\beta$, respectively, from $[|\mathbb{G}|]$. The sender computes $\tilde{Y} = \{H(y)^\alpha\}_{y \in Y}$ and sends $\tilde{Y}$ to the receiver. Similarly, the receiver computes $\tilde{X} = \{H(x)^\beta\}_{x \in X}$ and sends $\tilde{X}$ to the sender. The sender, then, computes $\hat{Y} = \{H((\tilde{x})^\alpha)\}_{\tilde{x} \in \tilde{X}}$ and sends $\hat{Y}$ to the receiver. The receiver computes $\hat{X} = \{H((\tilde{y})^\beta)\}_{\tilde{y} \in \tilde{Y}}$ and outputs $\{x \mid \hat{x} \in \hat{X} \cap \hat{Y}\}$.

This protocol sends a group element of size $\phi$ for each $x \in X$ and $y \in Y$, in addition, it sends a hash value for every $\tilde{x} \in \tilde{X}$, with the hash output being of length $\lambda + 2 \log n$. The total communication is $n(2\phi + \lambda + 2 \log n)$.

The protocol computes $O(n)$ exponentiations (or elliptic curve multiplications), which are each computed using $\log |\mathbb{G}|$ multiplications in $\mathbb{G}$) ($O(n \log |\mathbb{G}|)$ overall). Note that for all relevant input sizes ($n$) it follows that $\log |\mathbb{G}| > \log n$.

The experiments that we report below demonstrate that our protocol is substantially faster than DH-PSI for all realistic set sizes and on all network configurations (we report run times and bandwidth for both Koblitz K-283 and Curve25519 elliptic curves).

PaXoS-based PSI has security against both semi-honest and malicious adversaries. In contrast, DH-PSI is not easily adapted to malicious security, even against just one party.[10] In order to harden

---

[10]The main challenge is that a simulator would have to extract effective inputs $\{x_1, \ldots, x_n\}$ from a corrupt party, seeing only $\{H(x_1)^\alpha, \ldots, H(x_n)^\alpha\}$.

DH-PSI against malicious parties, the leading protocol of De Cristofaro et al. [CKT10] requires *both parties* to run zero-knowledge proofs involving all of their input items, which incurs significant overhead to the semi-honest protocol.

**Protocols based on an RSA accumulator.** The protocol of [ACT11] has a very low communication overhead of roughly $\lambda + \log^2 n$ bits per item, which may even be optimal (even for an insecure protocol). Furthermore, it also hides the receiver's set size. On the other hand, it computes $O(n \log n)$ RSA exponentiations, and as such is slower than DH-PSI by at least an order of magnitude (due to the $\log n$ factor, and to RSA exponentiations being slower than elliptic curve multiplications). Our protocol is substantially faster than both of these protocols. This protocol is only secure against a semi-honest adversary and requires a random oracle, whereas ours is in the standard model (when assuming semi-honest adversary).

**OT-based protocols.** The fastest PSI protocol with security against a semi-honest adversary is KKRT [KKRT16] with $\sim 4$ seconds for $\approx 1$ million inputs (over LAN).[11] Similarly to our protocol, KKRT requires the receiver to map its items to a Cuckoo table of size $m = 1.2n$, however, it does so using a Cuckoo hashing with 3 hash functions, which ends up having at most one item in each table entry (bin), so each item $x$ is mapped to location $h_1(x)$, $h_2(x)$ or $h_3(x)$-th in the table. Items that could not find an empty table entry are mapped to a special bin called 'stash', which has a bounded size $s = \omega(1)$. Denote the $i$-th item in the Cuckoo table by $R[1], \ldots, R[m]$ and the items in the stash by $R[m+1], \ldots, R[|s|]$. On the other side, the sender maps its items to a table of size $m$, but using a simple hashing, resulting with up to $O(\log n)$ items in the most occupied bin (with high probability). Then, for each $i \in [m + s]$ the parties run an oblivious PRF (OPRF) [FIPR05], by which the sender obtains a secret key $k_i$ and the receiver obtains $F_{k_i}(R[i])$. The sender then, for each $y \in Y$, uses keys $k_{i_1}, k_{i_2}, k_{i_3}$ where $i_1 = h_1(y), i_1 = h_3(y), i_1 = h_3(y)$ and adds $\{F_{k_{i_1}}(y), F_{k_{i_2}}(y), F_{k_{i_3}}(y)\}$ to an initially empty set $\tilde{Y}$. In addition, for each $j \in [s]$ and $y \in Y$, it adds $F_{k_{m+j}}(y)$ to $\tilde{Y}$. Finally, the sender sends $\tilde{Y}$ to the receiver, who computes the intersection $\{x_i \in X | F_{k_i}(x) \in \tilde{Y}\}$. Note that KKRT immediately has a 20% overhead, since the parties run $m = 1.2n$ OPRF instances. Our protocol is worse on that aspect as it has $\sim 140\%$ overhead, because we have $m = 2.4n$. On the other hand, the sender in our protocol sends only $n$ masks to the receiver, whereas the sender in KKRT sends $(3 + s)n$ masks, which narrows the communication gap of the two protocols. Although our protocol is maliciously secure, it requires only $\sim 25\%$ more communication and is only $1.75\times$ slower than KKRT. Our protocol achieves such competitive results, even compared to the state-of-the-art semi-honest protocol, by overcoming the main difficulty of adapting Cuckoo hashing to the malicious setting. We stress that our protocol has linear communication complexity of $O(n\kappa)$ whereas KKRT has a super-linear complexity of $\omega(n\kappa)$. This is due to handling the stash handling, see a detailed explanation in [FNO18, PRTY19].

Recently, Falk, Noble and Ostrovsky [FNO18] presented a semi-honest PSI that achieves linear communication complexity relying on standard assumption (i.e. in the OT-hybrid model, assuming the existence of correlation robust hash and one-way functions) and in the standard model (i.e. without a random oracle). This is in contrast to previous protocols that achieve linear communication but rely on stronger assumptions (like [CKT10, CT12] that are based on the one-more RSA assumption and a random oracle); and to previous OT-based protocols that achieve only super-linear communication complexity due to the stash handling (like [KKRT16]). In the

---

[11]Independently, Lambæk [Lam16] and Patra et al. [PSS17] showed how to enhance the protocols of [PSZ14, KKRT16] with a security against a *malicious receiver* (but not a malicious sender) with almost no additional overhead.

protocol of [FNO18], just like previous OT-based protocols, Bob maps his $n$ items to $O(n)$ bins using a Cuckoo hashing, hence, it has at most one item in each bin. Bob also maintains a stash with $s = \omega(1)$ items, as described above. Alice maps her items to $O(n)$ bins using simple hashing, hence, she has at most $O(\log n / \log \log n)$ items in each bin with high probability. Then, Bob can obtain the intersection between items in its 'regular' bins and Alice's set using the OPRF of [KKRT16] with communication complexity $O(n \cdot \kappa)$ (where $\kappa$ is the computational security parameter). It remains to compare the items in Bob's stash to all Alice's items; since the stash is of size $\omega(1)$ this comparison would naively require $\omega(n \cdot \kappa)$ communication overall (as it indeed takes in [KKRT16]). However, the observation in [FNO18] is that this comparison can be performed using a separate PSI protocol that is specialized for *unbalanced* set sizes in which Alice has much more items than Bob; such a protocol can achieve communication complexity that depends only on the larger set size, therefore, the overall communication complexity of [FNO18] is $O(n \cdot \kappa)$ rather than $\omega(n \cdot \kappa)$. This work was not implemented and therefore it does not appear in our empiric comparison.

Another work, called SpOT-light [PRTY19], solves the same problem described above, achieving the same asymptotic results as [FNO18], under the same assumptions, but demonstrating concrete efficiency. In particular, it presents a surprisingly low communication complexity that beats even DH-PSI. This is done using a technique called Sparse-OT, by which the receiver sends the correction matrix to the sender in the form of a polynomial. Due to this fact, the computational complexity of [PRTY19] is dominated by an expensive polynomial interpolation and evaluation that incurs $O(n \log^2 n)$ overhead and renders that protocol less competitive in run-time. With regards to communication, our protocol has $\sim 2 - 2.5\times$ more communication than SpOT-light. Our protocol, even though providing stronger security guarantees (malicious vs. semi-honest), is $\sim 5\times$ faster than the fast version in [PRTY19] and $\sim 54\times$ than their slow version.

Ghosh and Nilges [GN19] proposed a maliciously secure PSI protocol based on oblivious linear function evaluation (OLE). This protocol requires $2n$ passive OLE invocations, 3 polynomial interpolations (one of degree $n$, and two of degree $2n$), and 4 polynomial evaluations on $2n+1$ points. In terms of communication, the required passive OLE instances [IPS09, GNN17] entails $8(n+1)$ elements sent from the receiver to the sender to create a noisy encoding, and the cost of doing $4n$-out-of-$8(n+1)$ OT which incurs an overhead of at least $8(n+1)$ on the number of Correlated OT [ALSZ13b]. Hence, this OLE-based PSI protocol requires at least $8(n+1)(\kappa + 2\sigma)$ bits communication, where $\sigma$ is bit-length of item. We note that our protocol does not depend on the bit length of the inputs. For example, when $\sigma = 128$, our protocols show a factor of at least $2.5\times$ improvement in terms of communication.

**Maliciously secure OT-based PSI.** The previous fastest maliciously secure PSI protocol, by Rindal and Rosulek [RR17b], is also based on OT. Like many previous results, it works by first having the parties compute a simple hash of their items into bins (e.g., item $x$ is placed in bin $h(x)$), and a smaller PSI is performed in each bin. The idea is to take an efficient OT-based protocol that is secure against a malicious sender (or a malicious receiver) and use it 'twice', such that in the second time the roles (of sender and receiver) are reversed. Hence, the protocol executes a 'dual execution'. The starting point of the protocol has a quadratic complexity in which the two parties Alice and Bob proceed as follows: denote by $[x_i]_i^B$ an encoding protocol in which Alice has an item $x_i$ and learns an 'encoding of $x$', $[x_i]_i^B$ whereas Bob obtains the ability to compute $[\cdot]_i^B$ on any value $v$, where $[\cdot]_i^B$ is a random oracle. This encoding functionality can be realized by the OOS protocol. So first Alice obtains $[x_i]_i^B$ for every $i \in [n]$ and Bob obtains $[\cdot]_i^B$. Then, they switch roles so Bob obtains $[y_j]_j^A$ and Alice obtains $[\cdot]_j^A$ for every $j \in [n]$. Finally, Bob sends to

Alice $\mathcal{B} = \left\{ \{[y_j]_j^A \oplus [y_j]_i^B\}_{i \in [n]} \right\}_{j \in [n]}$. Likewise, Alice computes $\mathcal{A} = \left\{ \{[x_i]_i^B \oplus [x_i]_j^A\}_{j \in [n]} \right\}_{i \in [n]}$ and outputs $\mathcal{A} \cap \mathcal{B}$. Note that for any value $z \in X \cap Y$, say $z = x_i = y_j$, the value $[z]_j^A \oplus [z]_i^B$ belongs to both $\mathcal{A}$ and $\mathcal{B}$ and will be found in the intersection. On the other hand, for $y_j \in Y \setminus X$, Bob sends $\{[y_j]_j^A \oplus [y_j]_i^B\}_{i \in [n]}$ to Alice (among other values), but all of those values look random to Alice, because all of $\{[y_j]_i^B\}$ look random to her as she has never queried $y_i$ and these values serve as a one-time-pad.

To overcome the quadratic (communicational / computational) complexities, [RR17b] suggests that the parties first hash their items to $O(n/\log n)$ bins, so each bin holds at most $O(\log n)$ items with high probability. Then, run the quadratic protocol on each bin, which has overall complexities $O(\kappa n / \log n \cdot \log^2 n) = O(\kappa n \log n)$.

Although that protocol heavily relies on fast symmetric-key primitives, the asymptotic complexity kicks in and renders it less efficient than our protocol $O(\kappa n)$ protocol. Concretely, our protocol has $\sim 10\times$ less communication and is $\sim 3.5\times$ faster.

**Bloom filter based protocols.** Dong, Chen & Wen [DCW13] describe an approach for PSI based on representing the parties sets as Bloom filters. This approach also relies heavily on OT extension, and is reasonably efficient.[12] Another work [RR17a] presented a protocol based on Bloom filter with security against malicious adversaries.

**Other paradigms.** Freedman, Nissim and Pinkas [FNP04] proposed an approach for PSI based on oblivious polynomial evaluation. This technique was later extended to the malicious setting [DMRY09, FHNP16, HN12]. The idea behind these protocols is to construct a polynomial $Q$ whose roots are Alice's items. The coefficients of $Q$ are homomorphically encrypted and sent to Bob. For each of Bob's items $y$, he homomorphically evaluates $\hat{y} = r \cdot Q(y) + y$ for a random $r$. When Alice decrypts the result, she will see $\hat{y} = y$ for all $y$ in the intersection. These protocols require expensive public-key operations for each item and are not competitive with our protocol.

Huang, Evans & Katz [HEK12] use a general-purpose MPC to perform PSI. Later improvements, also fall under the name 'circuit-based-PSI' were suggested in [PSSZ15, PSTY19]. At the time of [HEK12], such general-purpose PSI protocols in the semi-honest setting were actually faster than other special-purpose ones. Since then, the results in OT-based PSI have made special-purpose PSI protocols significantly faster.

Pinkas et al. [PSZ14, PSSZ15] performed a comprehensive comparison of semi-honest PSI techniques and found the OT-extension paradigm to strictly dominate others in terms of performance. They found that the best Bloom-filter approach is 2x worse in runtime, 4x worse in communication; best generic-MPC-based approach is 100x worse in runtime and 10x worse in communication. For this reason, we do not include these protocol paradigms in further comparisons.

Kamara et al. [KMRS14] presented techniques for both semi-honest and malicious secure PSI in a server-aided model. In this model the two parties who hold data enlist the help of an untrusted third party who carries out the majority of the computation. Their protocols are extremely fast (roughly as fast as the plaintext computation) and scale to billions of input items. In our work, we focus on on the more traditional (and demanding) setting where the two parties do not enlist a third party.

---

[12]The authors of [DCW13] describe a protocol that was claimed to have security against malicious adversaries, but that was later found to be insecure [Lam16, RR17a].

Several recent PSI protocols are optimized specifically for the case of highly asymmetric set sizes [CLR17, PSSZ15, KLS$^+$17, RA18]. In this work we focus on the setting in which the parties has roughly the same set size, nevertheless, we note that our protocol can be adapted to the unbalanced setting as well.