Faster Secure Comparisons with Offline Phase for Efficient Private Set Intersection

Florian Kerschbaum
University of Waterloo
Canada
florian.kerschbaum@uwaterloo.ca

Erik-Oliver Blass
Airbus
Germany
erik-oliver.blass@airbus.com

Rasoul Akhavan Mahdavi
University of Waterloo
Canada
rasoul.akhavan.mahdavi@uwaterloo.ca

***Abstract*—In a Private section intersection (PSI) protocol, Alice and Bob compute the intersection of their respective sets without disclosing any element not in the intersection. PSI protocols have been extensively studied in the literature and are deployed in industry. With state-of-the-art protocols achieving optimal asymptotic complexity, performance improvements are rare and can only improve complexity constants. In this paper, we present a new private, extremely efficient comparison protocol that leads to a PSI protocol with low constants. A useful property of our comparison protocol is that it can be divided into an online and an offline phase. All expensive cryptographic operations are performed during the offline phase, and the online phase performs only four fast field operations per comparison. This leads to an incredibly fast online phase, and our evaluation shows that it outperforms related work, including KKRT (CCS'16), VOLE-PSI (EuroCrypt'21), and OKVS (Crypto'21). We also evaluate standard approaches to implement the offline phase using different trust assumptions: cryptographic, hardware, and a third party ("dealer model").**

I. Introduction

Companies collect increasingly larger amounts of data about their customers' operation. Each company collects different data depending on their business, and the combination of these different data sets offers greater benefit than each set by itself. A standard example is Google collecting which user clicks on which online ad while Mastercard collects financial transactions performed by its clients using their cards. To allow Google to compute the number of successful transactions after a user clicked on an online ad, Google and Mastercard link their data based on a common user identifier, e.g., the user's phone number.

Abstractly, this is an instance of Private Set Intersection (PSI). In PSI, two parties, each have a set of (unique) elements and want to compute their intersection without revealing any element not in the intersection. PSI is indeed deployed by Google and Mastercard to analyze ad conversions [43, 83], but it has many more applications. Consequently, PSI has recently been extensively studied in the literature [1, 12–17, 19, 22, 26–28, 31–33, 39, 41, 43, 47, 52, 55–58, 60–63, 66–72, 74–76, 83]. The currently most efficient state-of-the-art PSI protocols are based on oblivious pseudo-random functions [15, 57, 76]. They require a constant number of public-key cryptography operations, linearly many symmetric key cryptography operations, and one round of interaction. This is asymptotically optimal, and any performance improvement can only stem from reduced constants which are, however, already very low. We note that PSI requires public-key operations, since two-party computation can be reduced to PSI (see our reduction in Section II-D), and two-party computation requires public-key operations. Hence, any (new) approach must deal with these unavoidable and expensive operations.

In this paper, we present a new (equality) comparison protocol that is simple, elegant, and very efficient. Our comparison protocol improves over the state-of-the-art in two aspects: first, it reduces (equality) comparison to *oblivious linear evaluation* (OLE), and, second, it enables the use of offline precomputed OLE tuples instead of computing the OLE online. This results in an alternative construction of PSI that off-loads all expensive cryptographic operations, public and symmetric key, to an initial offline phase. The offline phase precomputes correlated randomness and can be run in advance, independently of the inputs to the protocol. The online phase uses this randomness and the inputs to securely compute the output. Our online phase is highly efficient, comprising only four fast operations in a small field, i.e., one multiplication and three additions per comparison. Moreover, it takes only one round. The offline phase can be implemented using standard approaches based on only cryptographic assumptions, e.g., using lattice-based homomorphic encryption or Oblivious Transfer (OT), but also based on more efficient hardware or other trust assumptions, such as a trusted third party ("dealer model" [38]).

A. Why consider an offline phase?

Our offline phase enables a very fast online phase. In Figure 1 we display a comparison of computation time and communication cost to related work [15, 33, 57, 70, 76] for a data set size of 1 million elements *in the online phase online*. Note that VOLE-PSI [76] requires less communication for these small sets. On a larger data set with 16 million elements our online phase is between 2.4 and 3.5 times faster than Kolesnikov et al.'s work [57] and 1.2 and 14.6 times faster than Rindal and Schoppmann's work [76] while requiring less communication than either one of them. In general, our advantage increases as data set sizes grow, but Rindal and Schoppmann neither make their code available nor do they report evaluation results for larger data set sizes which makes

Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.23198
www.ndss-symposium.org

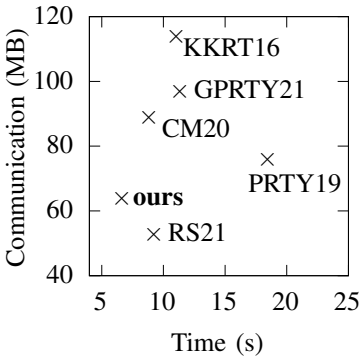

Fig. 1. Time (online) and communication of our protocol vs state of the art (KKRT16 [57], PRTY19 [70], CM20 [15], RS21 [76], and GPRTY21 [33]), sets of size $n = 2^{20}$, 100 MBit/s bandwidth.

comparison difficult.

Employing an offline phase is common in two-party computation, even if the total run-time (slightly) increases, and also in the context of PSI, it is not new. We emphasize that the commonly most efficient related work, VOLE-PSI by Rindal and Schoppmann [76], already requires an offline phase to derive vector OLE tuples and has been developed by industry, thus demonstrating the acceptability of such an offline phase. The type of offline phase we employ has further advantages, since it can be efficiently implemented using alternatives to cryptographic assumptions and similar code for the offline phase is re-usable among secure computation protocols based on Beaver multiplication triples [7]. This includes the prominent SPDZ family of protocols [24, 50, 51] for generic multi-party computation.

In summary, we believe that there are good reasons to consider our construction of a PSI protocol using an offline phase. There already exist many protocols for the offline phase, e.g., for secure two-party computation, and our work reuses those protocols in its construction or implements adjustments on top of them. This has the additional advantage that further research in improving these commonly used protocols will readily extend to our construction. Given additional security assumptions, such as the availability of trusted execution environments during the offline phase (only), our (entire) protocol reduces computation and communication costs compared to prior work.

B. Contributions

This paper contributes a new private set intersection protocol based on a new comparison protocol that can be divided into an *offline* and *online* phase. In particular, we provide

- a new, highly efficient *online* phase of our PSI/comparison protocol that uses no cryptographic operations, but uses precomputed, random *OLE tuples*.
- several adaptations of existing protocols for the *offline* phase, based on lattice-based homomorphic encryption, oblivious transfer, trusted execution environments, and a trusted third party.
- formal security proofs of our protocols.
- an extensive experimental evaluation of our protocol in comparison to related work [15, 33, 57, 70, 76].

Our PSI protocol's online phase is 1.2 (10 MBit/s network) to 3.5 (5 GBit/s network) times faster than the currently most efficient related work [15, 33, 57, 70, 76]. The maximum performance increase of our protocol in total time shrinks to 1.5 when used with an offline phase based on additional hardware trust assumptions not made in related work.

The next section contains preliminary definitions and building blocks used in our protocols. In Section III we describe the online phase of our comparison and PSI protocol. In Section IV we describe different protocols for the offline phase. We present our experimental results in Section V. We compare our work to related work in Section VI and present our conclusions in Section VII.

II. Preliminaries

A. Semi-Honest Security Model

We consider two-party PSI computation in the semi-honest, or passive, security model. In this model, parties Alice and Bob are assumed to follow the protocol as prescribed but keep a record of each message received, their random coins, and their input. From this information, called the view View_X of a party $X \in \{A, B\}$, they try to compute additional information beyond the output. Note that the output of a party X can be computed from its view View_X. Informally, a protocol is secure in the semi-honest security model, if a party cannot compute additional information. This can be proven by showing the existence of a simulator Sim_X that given the party's input x or y and the respective output χ or ψ produces a simulation of the view that is indistinguishable from the party's view View_X during the protocol.

Definition 1. *Protocol π is secure in the* semi-honest model*, if there exist simulators $\mathsf{Sim}_A(x, \chi)$ and $\mathsf{Sim}_B(y, \psi)$, such that*

$$\begin{aligned} \mathsf{View}_A^\pi(x, y) &= \mathsf{Sim}_A(x, \chi) \\ \mathsf{View}_B^\pi(x, y) &= \mathsf{Sim}_B(y, \psi) \end{aligned}$$

The semi-honest security model assumes that an adversary behaves passively, i.e., does not modify its messages. The malicious security model removes this assumption and considers active adversaries behaving arbitrarily. Any protocol secure in the semi-honest model can be compiled into one secure in the malicious model using the GMW compiler [36], but this compilation may lead to a very inefficient protocol. Hence, many previous works on PSI consider the semi-honest model. The malicious model for PSI has two limitations: First, since the intersection is revealed, a malicious party may simply substitute its input and learn (parts of) the other party's set which cannot be prevented in the malicious model. Second, in one-round protocols, such as ours, arbitrary behaviour is only possible in the input-carrying first message. Hence, additional information can only be computed from the message (output) received in response to this first message. Techniques such as certification of inputs [13, 22] can mitigate input substitution attacks. Our protocol can be augmented with input certification. However, we leave a detailed description to future work.

B. Oblivious Transfer

Oblivious transfer (OT) [73] is a protocol between two parties Alice and Bob. Its simplest variant is a 1-out-of-2 OT [29]. Alice has input $b \in \{0,1\}$. Bob has input y_0, y_1. An OT protocol is *correct* if Alice obtains y_b. An OT protocol is *secure* if Alice learns nothing about y_{1-b}, and Bob learns nothing about b. OT is a powerful primitive and can be used to implement secure two-party computations [36, 45, 54]. Hence, any OT protocol requires at least one public-key operation, since secure two-party computations require at least one public-key operation.

C. Oblivious Linear Evaluation

Oblivious Linear Evaluation (OLE) is a building block in many secure computation protocols [76] and a method to generate correlated randomness between two parties. From the slightly different OLE definitions in the literature [2, 6, 9, 76, 78], we use the following.

Oblivious Linear Evaluation is a secure two-party computation protocol between parties Alice and Bob, such that Bob samples $a, b \in \mathbb{F}$ for some field $\mathbb{F}$, and Alice samples $u \in \mathbb{F}$. After the protocol, Alice obtains $f(u)$ such that

$$f(u) = au + b. \tag{1}$$

The symmetric variant of OLE is referred to as product sharing [6], where Alice and Bob each know u and v, respectively, and they obtain additive shares of the product. In other words, after the protocol, Alice obtains a and Bob b, such that

$$uv = a + b. \tag{2}$$

As opposed to OLE, a and b are generated randomly. Product sharing can be used to construct OLE protocols [6].

In vector OLE (vOLE), which is a generalization of OLE, Bob knows two vectors $A, B \in \mathbb{F}^n$, and Alice knows $u \in \mathbb{F}$ and obtains $V \in \mathbb{F}^n$ where

$$V = Au + B. \tag{3}$$

Similarly in batch OLE (bOLE), Alice also samples a vector $U \in \mathbb{F}^n$ and obtains $V \in \mathbb{F}^n$ where

$$V = A \cdot U + B, \tag{4}$$

and the multiplication is element-wise. Note that, similarly to the notion of random OT, there exist also random alternatives of the above OLE protocols. So, parties receive random tuples (a, u) and (b, v) with $uv = a + b$.

In this work, we require a slightly different variant than (random) vOLE and bOLE for our PSI protocol. Parties will compute random batches of tuples, but the type of batches and choice of correlated randomness will be different. We will present details in Section III-A later.

OLE was first formalized by Applebaum et al. [2] where they proposed that OLE is the arithmetic analog of oblivious transfer. One approach for OLE over an arithmetic field is to repeatedly invoke 1-out-of-2 bit-OT to retrieve the bits of the results one by one. This approach has a computational complexity proportional to the size of the binary circuit that evaluates the linear function [46]. Another approach is to use hardness assumptions in coding theory such as the pseudo-randomness of noisy random codewords in a linear code or the LPN problem [64].

Finally, protocols realizing OLE based on lattices and lattice-based encryption schemes are also common [6, 25]. All of these approaches (except LPN) are used in this work to implement our offline phase.

D. Reduction of Two-Party Computation to PSI

TABLE I. INTERSECTIONS $C \cap D$ FOR THE DIFFERENT INPUTS BY ALICE AND BOB.

Alice	Bob			
	$y_0 = 0$		$y_1 = 1$	
	$y_1 = 0$ $D = \{2,3\}$	$y_1 = 1$ $D = \{1,2\}$	$y_1 = 0$ $D = \{0,3\}$	$y_1 = 1$ $D = \{0,1\}$
$b = 0$ $C = \{0\}$	$\emptyset$	$\emptyset$	$\{0\}$	$\{0\}$
$b = 1$ $C = \{1\}$	$\emptyset$	$\{1\}$	$\emptyset$	$\{1\}$

To underpin the power of PSI, we now present a reduction of two-party computation to PSI. This reduction may be folklore, but we have not found it spelled out in the literature. We prove that the existence of one-sided PSI, where only Alice learns the intersection, implies the existence of OT by reducing OT to PSI. It also proves that public-key operations are necessary for one-sided PSI since they are necessary for oblivious transfer and two-party computation. The GMW protocol [36] over the binary field $\mathbb{Z}_2$ reduces (semi-honest) two-party and multi-party computation to OT. Reductions of maliciously secure two-party computation also exist [45, 54].

We compute a 1-out-of-2 OT for one-bit messages. Alice has $b \in \{0,1\}$. Bob has two messages $y_0, y_1 \in \{0,1\}$. Alice should obtain y_b, but not learn anything about y_{1-b}.

If $b = 0$, Alice chooses $C = \{0\}$ as her input set. If $b = 1$, Alice chooses $C = \{1\}$. Bob starts by creating an empty set D. If $y_0 = 0$, Bob adds $\{2\}$ to D. If $y_0 = 1$, Bob adds $\{0\}$ to D. If $y_1 = 0$, Bob adds $\{3\}$ to D. If $y_1 = 1$, Bob adds $\{1\}$ to D.

Alice and Bob perform a one-sided PSI protocol for sets C and D, such that Alice learns intersection $C \cap D$. Table I shows the resulting intersection. Observe that the size of input sets is constant ($|C| = 1$, $|D| = 2$). If the intersection $C \cap D$ is empty, then Alice outputs 0. If the intersection is either $\{0\}$ or $\{1\}$, Alice outputs 1.

Note that privately evaluating functions over the intersection is sufficient for this reduction, but not necessary, e.g., set intersection cardinality or set disjointness. Alice only obtains either of two possible outputs, either the empty set or her input set of size 1.

Furthermore, OT can be reduced to labelled PSI – an extended form of PSI. In labelled PSI Alice obtains a message (label) for her elements in the set. Bob simply sets the labels y_i for elements $i = \{0,1\}$. However, it does not seem obvious how to reduce labelled PSI to PSI which our reduction also implies.

III. Online Phase

For ease of exposition, we begin by describing the online phase of our comparison and PSI protocols.

Roadmap for this section. For now, assume that the offline phase has output so called *OLE tuples* which we define in Section III-A and show how to compute during the offline phase in Section IV. Given an OLE tuple, we will present how to compare two elements x held by Alice and y held by Bob (Section III-B). Finally, we will use hashing techniques to construct a full PSI protocol for sets $\mathbb{X}$ held by Alice and $\mathbb{Y}$ held by Bob, each of size n, in Section III-C. We assume that the sets have the same size for simplicity of exposition and the analysis of the algorithm. If they do not, the protocol's cost is dominated by the larger one as it is for any of the compared related work. The goal of the full PSI protocol is to reduce the number and cost of necessary computations. The online phase requires $O(n \log n / \log\log n)$ comparisons, but our constants are so low that, in concrete numbers, we improve over related work, most of which also require at least $O(n \log n / \log\log n)$ computational steps.

A. OLE Tuples

We operate over a prime field $\mathbb{F}_Q$. Let $\mathbb{F}_Q^* = \mathbb{F}_Q \setminus \{0\}$, and let r_B be a random number in $\mathbb{F}_Q^*$. Similarly, let r_A, s_A, s_B be three random numbers in $\mathbb{F}_Q$. The marginal distribution of each random number is uniform, but their joint distribution is correlated and satisfies

$$r_A r_B = s_A + s_B \tag{5}$$

If $s_A + s_B = 0$, then $r_A = 0$, but it always holds that $r_B \neq 0$. In our implementation of the offline phase s_A, s_B, and r_B are drawn independently, and r_A is computed as correlated randomness from their choices to satisfy Equation 5.

Such a random tuple (r_A, r_B, s_a, s_B) is called an *OLE tuple*. For our PSI online phase, we assume that the offline phase has generated a sufficient number of OLE tuples and distributed the (r_A, s_A) to Alice and the (r_B, s_B) to Bob.

Observe the similarity of Equation 5 to Equation 1. If we set $r_A = u$, $r_B = a$, $s_B = -b$, and $s_A = f(u)$, we obtain a re-ordering of Equation 5. As in OLE, r_A and s_A (u and $f(u)$) are known only to Alice, and r_B and s_B (a and b) are known only to Bob.

Note that, in Section III-C5, we will also introduce a new *set version* of OLE tuples that further optimizes our online communication cost. However, this optimization also comes at a cost: We then need to adapt existing protocols in the offline phase for our set OLE tuples.

B. Comparison Protocol

We describe the comparison of a single pair of input elements and then extend it to a full-fledged PSI protocol in the next section. For some input length $\sigma \in \mathbb{N}$, Alice has element $x \in \{0,1\}^\sigma$, and Bob has element $y \in \{0,1\}^\sigma$. As part of the PSI protocol, they want to determine whether $x = y$. Let $H : \{0,1\}^\sigma \mapsto \mathbb{F}_Q$ be a cryptographic hash function. In case the element's bit length is shorter than the hash's bit length ($\{0,1\}^\sigma \subset \mathbb{F}_Q$), we do not need a hash function.

The comparison consumes one OLE tuple (r_A, r_B, s_A, s_B), where (r_A, s_A) is know to Alice and (r_B, s_B) is known to Bob, and works as follows.

1) Alice starts by computing $c = s_A - H(x)$ and sending c to Bob.
2) Bob computes
$$d = (c + H(y) + s_B)/r_B \tag{6}$$
and sends d back to Alice.
3) Alice verifies whether $d \stackrel{?}{=} r_A$, and if they are equal, she outputs "match".

Note that we do not need to compute the multiplicative inverse of r_B. We can immediately choose it when generating the OLE tuples (see Section IV).

Theorem 1. *Our comparison protocol is* correct*, i.e., if and only if $x = y$, then Alice outputs "match", assuming no hash collisions.*

Proof: We can substitute c in Equation 6 and obtain:

$$d = (s_A - H(x) + H(y) + s_B)/r_B \tag{7}$$

We show perfect correctness, i.e., if $x = y$, then $d = r_A$ and if $x \neq y$, then $d \neq r_A$. We can reformulate Equation 7 as

$$d r_B = s_A - H(x) + H(y) + s_B$$

and subtract Equation 5

$$(d - r_A) r_B = H(y) - H(x)$$

Since $r_B \in \mathbb{F}_Q^* \neq 0$, it follows

$$d - r_A = 0 \Leftrightarrow H(y) - H(x) = 0$$

and then

$$d = r_A \Leftrightarrow H(x) = H(y)$$

■

Theorem 2. *Our comparison protocol is* secure*, i.e., there exists a simulator of Bob's and Alice's view.*

Proof: Bob's simulator: Since r_A and s_A are unknown to Bob, and he only learns Equation 5, s_A is uniformly distributed in $\mathbb{F}_Q$ for Bob. Consequently, c is uniformly distributed in $\mathbb{F}_Q$ for Bob and the simulator can output a uniformly chosen random number.

Alice's simulator: If $x = y$, then the simulator outputs $d = r_A$. We have shown perfect correctness in Theorem 1 and hence only need to deal with the case $d \neq r_A$ ($x \neq y$).

We show that since the set $\{r_B, s_B\}$ is unknown to Alice[1], d is uniformly distributed in $\mathbb{F}_Q \setminus \{r_A\}$ for Alice, if $x \neq y$.

Let

$$\mu = H(x) - H(y) \tag{8}$$

Since $x \neq y$ (and we assume no collisions), it holds that $\mu \in \mathbb{F}_Q^*$. Substituting Equation 8 into Equation 7 we get

$$d = (s_A + s_B + \mu)/r_B \tag{9}$$

[1]Note that if $r_A = 0$, Alice knows $s_B = -s_A$, but r_B is still unknown to Alice which suffices for the proof.

Solving Equation 5 for r_A, subtracting it from Equation 9, and solving for d we get

$$d = \mu / r_B + r_A \tag{10}$$

The simulator needs to output d, but is given r_A as input. μ is in $\mathbb{F}_Q^*$ and r_B is uniformly distributed in $\mathbb{F}_Q^*$. $\mathbb{F}_Q^*$ forms the multiplicative sub-group in $\mathbb{F}_Q$. Hence, μ/r_B is uniformly distributed in $\mathbb{F}_Q^*$. The simulator outputs the sum of a uniform random number v in $\mathbb{F}_Q^*$ and r_A. Since $\mu/r_B \neq 0$, it holds $d \neq r_A$. In summary, the simulator chooses a uniform random number in $\mathbb{F}_Q \setminus \{r_A\}$. ∎

OPPRFs. An oblivious *programmable* pseudo-random function (OPPRF) [57, 71] is an oblivious pseudo-random function (OPRF) that can be programmed for a number of input values. Interestingly, our proofs show that our comparison protocol can be interpreted as an OPPRF that can be programmed for a *single* input. However, our comparison protocol is much faster than OPPRFs for multiple inputs [57, 71] which explains its advantage over the state-of-the-art even when the comparison protocol is repeated multiple times.

C. Full PSI Protocol

In the full PSI protocol, Alice and Bob want to compare elements in the sets $\mathbb{X} = \{x_i\}$ held by Alice and $\mathbb{Y} = \{y_j\}$ held by Bob. In order not to compare each pair of elements (x_i, y_j), we use a well-known technique of hashing elements to bins and only comparing elements within each bin. Alice hashes her elements $x_i \in \mathbb{X}$ using cuckoo hashing with k hash functions and hashes into α bins. Bob uses regular hashing for his set $\mathbb{Y}$ with each of the k hash functions into the same hash table [32, 67, 69, 71].

1) Alice: Cuckoo Hashing: Cuckoo hashing [65] is a hashing technique that reduces the maximum number of elements per bin to 1. Cuckoo hashing uses $\alpha = O(n)$ bins and $k > 1$ hash functions h_i. The idea of cuckoo hashing is to iterate over the k hash functions for each element. First, an element x is inserted into bin $h_1(x)$. However, if this bin is already occupied by element y, as $h_1(x) = h_i(y)$, then x replaces y in that bin, and y will be inserted into bin $h_{i+1 \bmod k}(y)$ and so on. There is a chance of an infinite loop of replacements, so the algorithm stops after a logarithmic number of replacements and places the current remaining element on a small stash data structure. For our protocol, the size s of the stash needs to be fixed as well. Let λ be a statistical security parameter. We choose s, such that the probability of failure, i.e., exceeding the stash size, is less than $2^{-\lambda}$. In our experiments, we set $\lambda = 40$. We use the parameter sets of Pinkas et al. [69] which we evaluate for our protocol in Section V-A. In our optimal setting selected in Section V-A a stash is not even necessary.

Note that, in order to not leak information about her set, Alice pads empty bins with dummy elements, such that the total number of elements per bin is always 1.

2) Bob: Regular hashing: Bob uses regular hashing to map his input into a regular hash table of length α. Since Alice may use any of the k hash functions to bin element x, Bob has to use all k hash functions for his input y, guaranteeing that x and y will be in a common bin, if $x = y$. Inserting each element into (up to) k bins in Bob's table, increases the total number $\beta = O(k \log n / \log \log n)$ [37] of elements per bin. β is chosen such that the probability that Bob will exceed β in any hashtable bin is negligible in the statistical security parameter λ, i.e., smaller than 2^{λ}. Note that β does not depend on Bob's input. To not leak any information about Bob's set, he also has to pad all bins to β elements. Observe that β is crucial for performance, as Alice and Bob have to perform β comparisons for each of Alice's elements. Hence, reducing β significantly reduces the number of necessary comparisons (including those with dummy elements) and hence increases the overall performance of the protocol. We use the parameter sets of Pinkas et al. [67] for β which we evaluate for our protocol in Section V-A.

Algorithm 1 Our PSI protocol

Common Input: A set H of k hash functions
Input *Alice*: $\mathbb{X}$, OLE tuples $\{\{r_{A,i,j}\}, s_{A,i}\}$ $(i \in [\alpha], j \in [\beta])$

Input *Bob*: $\mathbb{Y}$, OLE tuples $\{\{r_{B,j,i}, s_{B,j,i}\}\}$
Output *Alice*: $\mathbb{X} \cap \mathbb{Y}$

Alice:
$\alpha \leftarrow (1+\epsilon) \cdot |\mathbb{X}|$
$T_A \leftarrow \{\mathsf{dummy}\}^{\alpha}$
for all $x_i \in \mathbb{X}$ **do**
  Permutation-based cuckoo hash x_i into T_A
end for
for all $t_i \in T_A$ **do**
  Send $i, c_i \leftarrow s_{A,i} - t_i$ to Bob
end for

Bob:
$T_B \leftarrow \{\{\mathsf{dummy}\}^{\beta}\}^{\alpha}$
for all $y_i \in \mathbb{Y}$ **do**
  for all $h_j \in H$ **do**
    Add permutation-based suffix of y_i to T_B using h_j
  end for
end for
for all $T_i \in T_B$ **do**
  for all $t_{i,j} \in T_i$ **do**
    Send $i, j, d_{i,j} \leftarrow (c_i + t_{i,j} + s_{B,j,i}) / r_{B,j,i}$ to Alice
  end for
end for

Alice:
for all $d_{i,j}$ **do**
  if $d_{i,j} = r_{A,j,i}$ **then**
    Output the x-value hashed into bin i
  end if
end for

3) Alice and Bob: Comparisons: Alice and Bob now need to run $\alpha \cdot \beta$ comparisons, i.e., β comparisons for each of the α bins. With a stash ($s > 0$), Alice and Bob also compare each element in the stash with each element of Bob, i.e., $s \cdot n$ additional comparisons. Each comparison uses the protocol from Section III-B. Note that all comparisons can be performed in parallel, and for each element in $\mathbb{X}$ or $\mathbb{Y}$, there can be at most one match.

While this already concludes the description of the main steps of the full PSI protocol, we also institute the following

crucial optimizations.

4) Permutation-Based Hashing: The comparison protocol operates over field $\mathbb{F}_Q$ and the up to $2n$ elements (from the union of $\mathbb{X}$ and $\mathbb{Y}$) need to be mapped to $\mathbb{F}_Q$, ideally without collision. Hence, the size of $\mathbb{F}_Q$ is determined by the domain of $\mathbb{X}$ and $\mathbb{Y}$. However, the size of $\mathbb{F}_Q$ is also linear in our communication complexity and any reduction again significantly reduces the overall communication cost. Thus, Alice uses permutation-based hashing [3], similar to the Phasing protocol [67], to further reduce the size of $\mathbb{F}_Q$.

Permutation-based hashing works as follows. Let x be an σ-bit string which consists of a prefix x_1 of σ_1 bits and a suffix x_2 of σ_2 bits, such that the concatenation of x_1 and x_2 is x. Let h be a hash function that maps prefixes to bins. Element x is inserted into bin $h(x_2) \oplus x_1$. This ensures that if two different elements $x \neq y$ have a common suffix, i.e., $x_2 = y_2$ and consequentially $h(x_2) = h(y_2)$, they will be mapped to different bins, since then it must be that $x_1 \neq x_2$. Hence, it suffices to compare x_2 and y_2 for each bin to determine equality of x and y. However, since we use cuckoo hashing, we need to ensure that the same hash function h_i is used. Otherwise, two different elements $x \neq y$ with the same suffix $x_2 = y_2$ could be mapped to the same bin $h_i(x_2) \oplus x_1 = h_j(y_2) \oplus y_1$, but have different prefixes $x_1 \neq y_1$ since $h_i \neq h_j$. We ensure this by using a field size of $\mathbb{F}_Q$ that is larger than $k2^{\sigma_2}$ and encoding suffixes into different ranges depending on the hash function, such that suffixes can only match if they use the same hash function. When using k hash functions, our technique increases the message length by $\log_2(k)$ bits. Permutation based hashing saves $\log_2(\alpha)$ bits. Hence, the net effect is $\log_2(\alpha) - \log_2(k)$ which is quite large for the parameters in Table II.

5) Communication Optimization for OLE Tuples: Using the above hashing techniques, Alice compares each of her (dummy and input) elements with β (dummy or input) elements from Bob. This allows for another optimization of Alice's (and thus the total) communication complexity.

Instead of using a new s_A for each comparison, Alice re-uses one s_A for all β comparisons of one x, but uses different r_A. Bob never re-uses a pair (s_B, r_B). So, we batch one s_A with β-many r_A, s_B, and r_B.

An *optimized* OLE tuple is tuple $(s_A, (s_{B,1}, r_{A,1}, r_{B,1}), \ldots, (s_{B,\beta}, r_{A,\beta}, r_{B,\beta}))$, where

$$\forall i \in \{1, \ldots, \beta\} : r_{A,i} r_{B,i} = s_A + s_{B,i}. \tag{11}$$

For the online phase of the PSI protocol, we assume that the offline phase has generated a sufficient number of optimized OLE tuples by choosing $s_A, s_{B,i} \in \mathbb{F}_Q, r_{B,i} \in \mathbb{F}_Q^*$ randomly and $r_{A,i} \in \mathbb{F}_Q$ as correlated randomness such that Equation 11 holds. For each tuple the $(s_A, r_{A,1}, \ldots, r_{A,\beta})$ are distributed to Alice and the $((s_{B,1}, r_{B,1}), \ldots, (s_{B,\beta}, r_{B,\beta}))$ to Bob.

In the remainder of the paper, we will only consider these optimized OLE tuples and show how to construct large amounts of such tuples in Section IV.

The computation of these sets is also more efficient than the computation of all distinct tuples. Overall, this reduces the number of messages sent by Alice in the online phase from $\alpha \cdot \beta$ to α.

We provide a proof of security for this optimization. Let x be an element held by Alice and Bob hold a set of elements, such that Alice compares her element against each element in Bob's set as described. We call this a set comparison protocol.

Theorem 3. *Our set comparison protocol is* secure*, i.e., there exists a simulator of Bob's and Alice's view.*

Proof:

Bob's simulator: Same as in Theorem 2.

Alice's simulator: The case $x = y_j$ is handled as above. We consider the simulation for each element $y_j \neq x$ in Bob's set. Let

$$\mu_j = H(x) - H(y_j)$$

Equation 10 is modified to

$$d_j = \mu_j / r_{B,j} + r_{A,j}$$

Since all $r_{B,j}$ are independently uniformly distributed in $\mathbb{F}_Q^*$, the same derivation as in the proof of Theorem 2 holds. The simulator chooses j uniform random numbers in $\mathbb{F}_Q \setminus \{r_A\}$ if $x \neq y_j$. ∎

Formal Presentation. We formalize our full PSI protocol, including the above optimizations and assuming that there is no stash, in Algorithm 1. It is a sequential composition of set comparison protocols, one for each bin in the hash table.

IV. Offline Phase (OLE Tuple Precomputation)

We now turn to the various options to realize the offline phase for securely computing optimized (set) OLE tuples from Section III-C5. These are (sometimes small) adaptations of existing state-of-the-art techniques that complete our comparison protocol into a full-fledged PSI protocol.

We consider purely cryptographic, hardware-supported, and trusted third-based approaches for the implementations of the offline phase. For fair comparison, we note that the related work with which we compare our performance [15, 33, 57, 70, 76] only uses purely cryptographic assumptions. However, other related works have considered our assumptions as well to speed up their (related) protocols. We start with an overview.

1) TTP: The first option we consider is that of a trusted third party (TTP) providing OLE tuples similar to Beaver triples as a service. The implementation of Beaver multiplication triples as a service by a TTP ("dealer model") to speed up their computation has been proposed multiple times in the literature, see, e.g., [38, 77, 81] for an overview. Also, some state-of-the-art PSI protocols assume trusted third parties. For example, the approach by Pinkas et al. [70] assumes the following trust model: each party outsources their data to a cloud provider and then runs the protocol. Sometimes they even use the same cloud provider. Clearly, this cloud provider is a real-world trusted third party, since it has access to both parties' data and could completely subvert the security of the protocol. However, Pinkas et al., in our opinion rightly, assume that the PSI protocol adds a layer of security, since the data is not revealed to the other party hosted by the cloud provider. Yet, given such a trusted cloud provider, it is easy to extend the model, such that the cloud provider also creates the output

of the offline phase (as our OLE tuples). This will significantly reduce the cost of our offline phase and hence our entire protocol significantly below the cost of Pinkas et al.'s protocol without changing the trust assumptions.

2) Trusted Execution Environments: Secure hardware-supported implementations of the offline phase are also possible. For example, Microsoft's Azure cloud offers trusted execution environments (TEE) such as Intel's SGX. Microsoft's Azure cloud's TEEs are also used to privately intersect the set of client's address book with the set of the Signal user base [20]. Using trusted hardware (and trusted software to run in the hardware), we will show that the cost of our offline phase is also very low. For fair comparison, we emphasize that this is an additional security assumption not made by the protocols with which we compare our performance [15, 33, 57, 70, 76]. However, the main advantage of our protocol stems from the shorter online time which does not make additional security assumptions. Hence, we still consider this a fair comparison and Signal's use of TEEs underpins the acceptability of this assumption. The advantage of only running the offline phase in the TEE instead of the entire PSI protocol is that the trusted software (which is difficult to produce and verify) can be reused across many secure computation protocols, such as SPDZ, and the memory requirement is small opposed to the large data set sizes for PSI that may need to be held in memory and randomly accessed in a hash table. We need a TEE at both parties, Alice and Bob, for maximum efficiency which is different from a single trusted third party.

3) Cryptographic Protocols: We will also describe two protocols for securely pre-computing OLE tuples using only cryptographic assumptions (lattice-based homomorphic encryption and OT). These are variations of previous work on product sharing. If a party is not willing to accept additional security assumptions during the offline phase (only), they would implement these offline phases and compare them as part of the whole protocol with related work. We emphasize that our performance gain during the online phase is unaffected by any performance loss during the combined online and offline phase. First, observe the relation between OLE tuples and Beaver's multiplication triples. In Beaver triples, equation $c = a \cdot b$ holds with $a = a_0 + a_1$, $b = b_0 + b_1$, $c = c_0 + c_1$, and one party holds (a_0, b_0, c_0), and the other holds (a_1, b_1, c_1). As $(c_0 + c_1) = (a_0 + a_1) \cdot (b_0 + b_1) = a_0b_0 + a_0b_1 + a_1b_0 + a_1b_1$, the only information parties have to interactively compute is a_0b_1 and a_1b_0. That is, the two parties compute two product sharings [46], where the factors stem from the parties, and the products are secret shared between the parties. The computation of product shares/OLE lies at the heart of offline phases to compute Beaver multiplication triples. Highly practical MPC systems typically use one of two approaches to realize product sharing: lattice-based, somewhat-homomorphic encryption (e.g., SPDZ [24] and Overdrive [51]) or Oblivious Transfer (e.g., MASCOT [50]). Both approaches were initially mentioned by Gilboa [35]. As the cryptographic protocols for product sharing and consequently Beaver triples in today's general MPC frameworks target (expensive) malicious security, they are significantly slower than lattice-based encryption and OT implementations that we will describe in the following.

All protocols we describe in detail below have computational complexity linear in the number of tuples. The protocol using TEEs features constant communication complexity, while the three others have linear communication complexity.

Discussion. Improvements in the underlying, adapted cryptographic protocols and techniques we build on can further enhance the performance of our offline phase. Recent advances in pseudo-random correlation generators (PRCG) [11] theoretically enable the generation of OLE tuples and even Beaver's multiplication triples [7] with very little communication effort. VOLE-PSI [76] uses a PRCG for vector-OLE (VOLE), and an implementation of vector-OLE is available [78]. However, vector-OLE, as used in VOLE-PSI [76], does not apply to the OLE tuples we require. Adapting existing implementations of vector-OLE to our needs leads to inferior performance than our implementations described in Section IV. Hence, we delay the development of improved PRCG techniques for the offline phase to future work.

We now present details of our four protocols for the offline phase.

A. Third Trusted Party (Dealer)

A naive implementation of the TTP would create the entire OLE tuples at the TTP and then distribute their respective pairs $(s_A, r_{A,1}, \ldots, r_{A,\beta})$ to Alice and $((r_{B,1}, s_{B,1}), \ldots, (r_{B,\beta}, s_{B,\beta}))$ to Bob. However, we found the following implementation using cryptographic assumptions to be more efficient even in current high-speed networks. The TTP chooses two random seeds R_A and R_B. TTP uses R_A as a seed in a pseudo-random number generator (PRG) to generate s_A and uses R_B to generate β (for each s_A) $s_{B,i}$s and $r_{B,i}$s. TTP then computes $r_{A,i} = (s_A + s_{B,i})/r_{B,i}$ and sends R_A and all $r_{A,i}$ to Alice and R_B to Bob. Alice and Bob use the same PRG to generate s_A, $s_{B,i}$, and $r_{B,i}$, respectively. This process repeats for all OLE tuples required.

B. Trusted Execution Environments

Some cloud providers, such as Microsoft's Azure, offer the use of Trusted Execution Environments (TEE) such as Intel's SGX.

A straightforward approach for PSI would be that Alice and Bob compute the entire PSI protocol in a TEE. However, this has a number of disadvantages. Programs to be executed in a TEE need to be hardened against side channel attacks and be free of any software vulnerability, such that the party controlling the host computer cannot infer the computation done in the TEE. This has significant costs for both parties – one developing the program and the other verifying the correct development (including compilation). Furthermore, in current TEEs, e.g., Intel's SGX, the encrypted memory for the enclave is small (96 MBytes). Such a small memory size requires memory paging even for the moderately-sized data sets we consider, and paging quickly becomes a performance bottleneck, particularly under random access in a hash table [4]. Splitting the PSI program into a secure part that runs in the TEE and unsecure part that has efficient memory access comes with significant implementation challenges. One would have to authenticate calls from the unsecure part to the secure part which may require additional security assumptions. Even without authentication of calls the performance of an insecure version that splits the PSI program can be disappointing.

Consider the following strawman construction. It computes messages authentication codes (MAC) using a key securely agreed between the enclaves at each party inside those enclaves and performs the intersection on the MACs in the unsecure parts. This construction is 29% to 109% slower in our cloud settings for data set sizes between 2^{20} and 2^{26} than our protocol with an offline phase in SGX even when the MACs are only sent from Bob to Alice. We attribute this disappointing performance to the computational cost of the MACs which is much higher than our simple field operations and generation of pseudo-random numbers.

Instead, we suggest to only generate OLE tuples in the TEE. Using the generation of OLE tuples as the program that is to be securely developed, verified, and deployed to the TEE has several advantages. First, a program for OLE tuple generation is very small and hence easier to harden. Second, it can be potentially be re-used among several applications using pre-computation of correlated randomness, such that its development costs amortize. Finally, OLE tuples can be sampled and communicated to untrusted program with constant memory requirements.

A naive implementation using TEEs would use one enclave and use that to generate all tuples and distribute them securely over an encrypted channel (e.g., TLS). However, this incurs significant communication costs, since all tuples of one party need to be sent from the TEE's host computer to that party. A more communication-efficient implementation uses two TEEs, one at Alice's site and one at Bob's. Both, Alice and Bob load the same program into their TEE which contains a public key of each party for authentication. Each party locally attests its deployment and verifies the remote deployment at the other party's TEE, such that the integrity of the programs at start is assured. The programs then jointly choose a random seed. Using that seed, they secretly generate the same OLE tuples using a pseudo-random number generator (PRG). Each party locally authenticates to its TEE and the TEE releases the pairs of the OLE tuples for that party. This implementation has only small, constant communication cost for the establishment of the joint seed.

C. Lattice-Based Homomorphic Encryption (LBE)

We can precompute the OLE tuples using LBE. Alice chooses m keys over plaintext fields $\mathbb{F}_{q_i}, 0 < i \leq m$. If our prime Q (as used in the comparison protocol) is a possible plaintext size for the LBE scheme, we set $m = 1$ and $q_1 = Q$. If prime Q is larger than the possible plaintext sizes for a chosen security parameter, then we set $m > 1$ and for a statistical security parameter λ we choose

$$Q' = \prod_{i=1}^{m} q_i > Q^2 2^\lambda$$

Alice chooses a uniform $s_A \in \mathbb{F}_Q$. Let $E_i()$ denote the homomorphic encryption under the i-th key (with plaintext modulus q_i). Alice encrypts as

$$c_i = E_i(s_A \bmod q_i)$$

for $1 \leq i \leq m$ and sends all c_i to Bob. Bob chooses uniform $s_B \in \mathbb{F}_Q$ and $r_B \in \mathbb{F}_Q^*$. If $m > 1$, Bob uniformly chooses u in $\mathbb{F}_{2^\lambda}$, else Bob sets $u = 0$. Then he computes

$$d_i = (c_i + E_i(s_B))((r_B^{-1} \bmod Q) \bmod q_i) + E_i(uQ \bmod q_i)$$

and sends all d_i to Alice. Alice decrypts d_i and if $m > 1$, Alice uses the Chinese Remainder Theorem to recover $v = uQ + (s_A + s_B)/r_B \in \mathbb{F}_{Q'}$. Alice sets $r_A = v \bmod Q$.

We can re-use the optimization of the comparison protocol (see Section III-C) and Alice sends one s_A, but Bob chooses r_B, s_B and computes r_A for each pair of comparison with the same element x. This reduces the number of ciphertexts that Alice needs to send. Furthermore, ciphertext operations can be easily batched over vectors of plaintexts, a common SIMD technique in LBE.

Homomorphically encrypted ciphertexts conceal Alice's inputs from Bob, but not vice-versa. Specifically, the noise level in the ciphertexts after the protocol might reveal information about Bob's inputs to Alice. To prevent this, there exists techniques to provide *circuit privacy*. In short, circuit privacy allows the function that Bob evaluates ($f(x) = ax + b$ in this case) to remain hidden from Alice. *Noise flooding* is one technique to make the noise level of the ciphertext after the protocol statistically indistinguishable from the noise level a freshly encrypted ciphertext. Before communicating the results back to Alice, Bob uses such techniques to make the noise level of the ciphertexts statistically indistinguishable from fresh ciphertexts.

Proposition 1. *The above OLE tuple precomputation protocol using LBE is* correct*, i.e., $r_A r_B = s_A + s_B$.*

This trivially follows from the construction of the protocol.

Theorem 4. *The above OLE tuple precomputation protocol using LBE is* secure*, i.e., there exists a simulator of Bob's and Alice's view.*

Proof: Bob's simulator: Bob receives m IND-CPA secure homomorphic ciphertexts.

Alice's simulator: In case $m = 1$, Alice's messages can be simulated by the message r_A, i.e., part of the output of the protocol. In case $m > 1$, Alice's messages can be simulated by a choosing random number $u' \in \mathbb{F}_{2^\lambda}$ and the message $r_A + u'Q$. The statistical indistinguishability of these views has been proven by Damgård and Thorbek [23]. Due to circuit privacy, the noise level of the ciphertexts are statistically indistinguishable from fresh ciphertexts. ■

D. Oblivious Transfer

We use a second, alternative cryptographic approach to prepare OLE tuples that is based on Gilboa's product sharing [35]. This is technique is implicitly used as a sub-routine in various general MPC frameworks, see, for example, Keller [49] for an overview.

a) Intuition: To generate an OLE tuple, Alice chooses r_A randomly from $\mathbb{F}_Q$, and Bob randomly chooses r_B from $\mathbb{F}_Q^*$. Let $\ell = \log Q$ and $r_B[i]$ be the i-th bit of r_B. The main idea is to rewrite $r_A \cdot r_B$ using the multiplication formula

$$r_A \cdot r_B = \sum_{i=1}^{\ell} r_B[i] \cdot r_A \cdot 2^{i-1}.$$

Alice and Bob run ℓ instances of 1-out-of-2 OT. In each instance, receiver Bob uses $r_B[i]$ as choice bit, and sender Alice inputs 0 and $r_A \cdot 2^{i-1}$. By summing up his OT output, Bob gets $r_A \cdot r_B$. To restrict Bob to only get a share of $r_A \cdot r_B$, Alice offsets each input to the OT by a random ρ_i. Alice's share s_A of $r_A \cdot r_B$ is then the sum of the ρ_i, and Bob's share s_B is the sum of the OT output.

b) Details: More formally, to generate an OLE tuple (r_A, r_B, s_A, s_B), Alice and Bob run the following protocol.

1) Alice randomly chooses r_A from $\mathbb{F}_Q$ and ℓ random values ρ_i from $\mathbb{F}_Q$. Bob randomly chooses r_B from $\mathbb{F}_Q^*$.
2) Alice and Bob run ℓ instances of 1-out-of-2 OT. In round i, sender Alice inputs $-\rho_i$ and $r_A \cdot 2^{i-1} - \rho_i$. Receiver Bob inputs choice bit $r_B[i]$ and receives output o_i.
3) Alice sets $s_A = \sum_{i=1}^{\ell} \rho_i$, and Bob sets $s_B = \sum_{i=1}^{\ell} o_i$.

The correctness and security of this protocol has been shown by Gilboa [35, Section 4].

Recall that the optimized communication during the online phase (Section III-C5) requires only one $s_{A,i}$ for β-many $(s_{B,i,j}, r_{A,i,j}, r_{B,i,j})$. That is, we need to compute the tuple $(s_{A,i}, \{r_{A,i,j}, r_{B,i,j}, s_{B,i,j}\}_{j=1\ldots\beta})$ such that $s_{A,i} + s_{B,i,j} = r_{A,i,j} r_{B,i,j}$ for $j \leq \beta$. To enable OT-based offline preparation of communication optimized OLE tuples, we consequently must institute one additional change. In the above protocol, instead of choosing ℓ random ρ values and setting $s_{A,i} = \sum_{\iota=1}^{\ell} \rho_\iota$ in Step 3, Alice randomly chooses one $s_{A,i}$ and $\beta\ell$-many random $\rho_{j,\iota}$ such that $s_{A,i} = \sum_{\iota=1}^{\ell} \rho_{j,\iota}$ for each $j \leq \beta$. While computing the j^{th} product sharing, during round ι, Alice uses $-\rho_{j,\iota}$ and $r_{A,i,j} \cdot 2^{\iota} - \rho_{j,\iota}$ as her input to the OT.

We omit proofs of correctness and security, since they are simple corollaries of the theorems by Gilboa [35] and to abide by page length restrictions.

V. Evaluation

We have implemented our protocol's online and offline phases in C++. The source code is available for download at https://github.com/BlazingFastPSI/NDSS23. Using this implementation, we have evaluated our protocol's communication cost (at the application layer in the network stack) and its running time. We compare our implementation to the recent works of Chase and Miao [15], Kolesnikov et al. [57], and Pinkas et al. [70] ("SpOT-low") using their publicly available implementations. We also compare to the recent VOLE-PSI protocol by Rindal and Schoppmann [76] and OKVS by Garimella et al. [33] using their measurements in their paper, since no implementation is publicly available. These two protocols are especially interesting, as they have low communication requirements (in bits per element). Our protocol is a set intersection protocol, revealing the set intersection, but we also compare to state-of-the-art circuit PSI (or PSI analytics) protocols [14, 71] to demonstrate the difference. VOLE-PSI requires an offline phase such as our protocol, but we concentrate on online timings and communication in this section. Note that VOLE-PSI by Rindal and Schoppmann [76] can also be extended to PSI circuit PSI.

We stress that there exists a large body of related work on practical PSI, e.g., OT-based PSI [67, 69], but our experiments only focus on the very recent works mentioned above as they typically outperform other approaches.

All of the related work with which we directly compare uses purely cryptographic assumptions. If a party is not willing to accept any additional security assumption during the offline phase only, the relevant comparison is with our offline phase implemented based on lattice-based encryption. However, since we focus on the online time of our protocol which makes no additional security assumption, we also present the times for offline phases using additional security assumptions, such as TEEs or a trusted third party.

In all of our experiments, we use an element bit length of $\sigma = 32$. This bit length is used in the evaluation of many related works [66–69, 71]. It is sufficient for several applications, including matching on phone numbers ("contact discovery") or IPv4 addresses, without using hashing and hence without collisions. Often, longer domains can be mapped to 32 bits without collisions using a public dictionary, since the largest data set sizes we consider are only 2^{26} (which already exceeds most previous work).

We consider the four implementations for our offline phase as described in Section IV:

- We implement the TTP (Section IV-A), where the TTP derives all $(r_{A,i,j}, r_{B,i,j}, s_{A,i}, s_{B,i,j})$, sends R_A and all r_{A_j} to Alice, and sends R_B to Bob. Alice then computes the $(s_{A,i})$ and Bob the $(r_{B,i,j}, s_{B,i,j})$.
- We use Intel's SGX (Version 1) to implement the offline phase in a trusted execution environment (Section IV-B).
- We use the Fan–Vercauteren cryptosystem [30] for the offline phase using LBE (Section IV-C). Our implementations employs Microsoft's SEAL library [79]. There, we set the polynomial modulus degree and coefficient modulus bit length to be 8192 and 218, respectively. We also perform modulus switching to a 40-bit coefficient modulus. We do not implement the noise flooding technique for our experiments since the SEAL library does not support the necessary operations at the moment. However, based on the analysis of Castro et al. [25], which is summarized in Lemma 4.2 of their paper, our parameter set provides circuit privacy for (v)OLE with more than 40 bits of statistical indistinguishability. The overhead of adding noise flooding to the protocol is negligible compared to the rest of the protocol (at most the cost of one homomorphic addition).
- We implement the OT-based offline phase (Section IV-D), building on IKNP OT extensions [44] from EMP toolkit [82].

A. Theoretical communication cost

We differentiate between online and offline communication cost. First, we theoretically analyze the online communication cost and estimate the optimal parameters for the number of hash functions, hash table length, and the stash size in our

TABLE II. PARAMETER ANALYSIS FOR OUR SYSTEM WITH STATISTICAL SECURITY PARAMETER $\lambda = 40$. n: NUMBER OF ELEMENTS, $\log Q$: NUMBER OF BITS TO COMPARE, s : SIZE OF STASH, $\frac{\text{BIT}}{\text{ELEMENT}}$: ONLINE COMMUNICATION COST PER ELEMENT (IN BIT, SEE TEXT)

	$k=2,\ \alpha=2.4n$				$k=3,\ \alpha=1.27n$ (no stash $s=0$)			$k=4,\ \alpha=1.09n$ (no stash $s=0$)		
n	β	$\log Q$	s	$\frac{\text{Bit}}{\text{element}}$	β	$\log Q$	$\frac{\text{Bit}}{\text{element}}$	β	$\log Q$	$\frac{\text{Bit}}{\text{element}}$
2^{20}	19	13	3	663	28	14	**516**	33	15	556
2^{22}	20	11	3	588	28	12	**442**	34	13	496
2^{24}	20	9	2	472	29	10	**381**	35	11	432
2^{26}	21	7	2	384	29	8	**305**	35	9	353

TABLE III. THEORETICAL COMMUNICATION COST IN BIT PER ELEMENT OF OUR SCHEME VS. RELATED WORK, $\lambda = 40, \kappa = 128$. GPRTY21 [33] AND CGS22 [14] SPECIFY ONLY ASYMPTOTIC COMMUNICATION COMPLEXITY, SO WE EXTRAPOLATE FROM THEIR BENCHMARKS.

	Ours ($k=3, \alpha=1.27n$)					PSTY19	KKRT16	PRTY19	CM20	RS21 [76]		GPRTY21	CGS22 (PSM_2)
n	**online**	total SGX	total TTP	total LBE	total OT	[71]	[57]	[70]	[15]	**online**	total	[33]	[14]
2^{20}	516	516	1014	2922	78179	20079	972	514	688	**394**	**419**	(774)	(8856)
2^{22}	442	442	869	2956	65303	20079	980	516	691	**393**	**400**	(943)	(8870)
2^{24}	**381**	**381**	749	2893	54889	20079	988	518	694	396	398	(1622)	(8926)
2^{26}	**305**	**305**	600	2928	42733	20079	997	520	697	400	400	(4338)	(9150)

TABLE IV. OFFLINE BENCHMARKS. CPU TIME: COMPUTATION TIME ONLY, TOTAL OFFLINE TIME: END-TO-END TOTAL TIME (INCLUDING COMMUNICATION TIME, WAITING FOR OTHER PARTY ETC.) UNTIL PARTIES HAVE ALL OLE TUPLES. DNF: DID NOT FINISH IN 15MIN.

n		CPU Time (s)	Total Offline Time (s) 5 GBit/s	1 GBit/s	100 MBit/s	10 MBit/s	Intra-Cont.	Inter-Cont.	Communication (MB)
2^{20}	SGX	0.2	0.3	0.3	0.3	0.3	0.4	0.4	$1.5 \cdot 10^{-5}$
	TTP	0.2	0.3	0.6	5.2	52.2	3.2	3.6	62.23
	LBE	10.7	14.7	18.1	55.6	430.3	46.5	48.0	482
	OT	13.7	44.7	205.7	DNF	DNF	DNF	DNF	23896
2^{22}	SGX	1.1	1.5	1.5	1.5	1.5	2.4	2.4	$1.5 \cdot 10^{-5}$
	TTP	0.6	1.0	2.4	18.8	182.4	11.5	12.1	213
	LBE	42.4	58.6	71.6	221.6	DNF	184.9	191.3	1915
	OT	50.2	150.4	703.0	DNF	DNF	DNF	DNF	81930
2^{24}	SGX	7.8	7.9	7.9	7.9	7.9	12.4	12.4	$1.5 \cdot 10^{-5}$
	TTP	2.5	3.8	8.8	65.9	636.4	40.6	43.0	737.6
	LBE	175.6	240.3	289.8	DNF	DNF	760.6	787.4	7951
	OT	178.9	519.9	DNF	DNF	DNF	DNF	DNF	282855
2^{26}	SGX	39.1	39.1	39.1	39.1	39.1	61.8	61.8	$1.5 \cdot 10^{-5}$
	TTP	11.5	15.5	31.4	214.7	DNF	136.0	143.9	2357
	LBE	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
	OT	576.8	DNF	DNF	DNF	DNF	DNF	DNF	905135

TABLE V. ONLINE BENCHMARKS. TIME IN SECONDS, COMMUNICATION IN MBYTE. DNF: DID NOT FINISH IN 15 MIN OR CRASHED. "—": SOURCE CODE NOT AVAILABLE. CPU: TIME FOR COMPUTATION ONLY, ONLINE TOTAL: END-TO-END TOTAL TIME (INCLUDING COMMUNICATION TIME, WAITING FOR OTHER PARTY ETC.) UNTIL INTERSECTION HAS BEEN COMPUTED. VALUES IN "()" ARE TAKEN FROM ORIGINAL PAPERS DUE TO LACK OF SOURCE CODE ([33, 76]). GPRTY21 [33] BENCHMARK WITH 4.6 GBIT/S, 260 MBIT/S, 33 MBIT/S BANDWIDTH.

		KKRT16 [57] LBE		PRTY19 [70] LBE		GPRTY21 [33] LBE		CM20 [15] LBE		RS21 [76] Online LBE		RS21 [76] Online+Offline LBE		**Ours Online**			Ours Online+Offline Time Security Assumption				
Sec. n	Assumption Bandwidth	Time	Comm	Time	Comm	Time	Comm	Time	Comm	Time	Comm	Time	Comm	CPU	**Time**	**Comm**	SGX	TTP	LBE	OT	Comm
2^{20}	5 GBit/s	1.3	114	16.8	76	(5.8)	(97)	5.1	89	(4.4)	**(53)**	(5.4)	(54)	0.3	**0.4**	64	0.7	0.7	15.1	45.1	
	1 GBit/s	1.4		17.6		—		5.1		—		—			**0.8**		1.1	1.5	18.9	206.5	SGX: 64
	100 MBit/s	10.3		17.7		(10.6)		8.1		(8.5)		(9.9)			**5.9**		6.1	11.1	64.4	DNF	TTP: 126
	10 MBit/s	99.0		66.7		(38.3)		75.9		(48.7)		(54.4)			**55.9**		56.1	108.0	486.1	DNF	LBE: 546
	Intra-Cont.	4.1		26.0		—		8.8		—		—		0.5	**2.7**		3.1	5.9	49.2	DNF	OT: 23960
	Inter-Cont.	5.8		26.2		—		9.2		—		—			**3.4**		3.8	7.0	51.4	DNF	
2^{22}	5 GBit/s	5.1	471	66.1	314	—		25.2	358	(23.9)	**(209)**	(25.6)	(210)	1.1	**1.5**	221	3.1	2.5	60.1	151.9	
	1 GBit/s	5.7		66.4				25.6		—		—			**3.0**		4.5	5.4	74.6	706.0	SGX: 221
	100 MBit/s	42.2		73.3				31.7		(40.7)		(43.0)			**20.2**		21.7	38.9	241.7	DNF	TTP: 434
	10 MBit/s	408.3		275.2				309.4		(199.0)		(204.7)			**191.8**		193.3	374.2	DNF	DNF	LBE: 2136
	Intra-Cont	14.2		103.2				41.9		—		—		2.1	**7.6**		10.0	19.1	192.5	DNF	OT: 82151
	Inter-Cont.	19.8		103.0				DNF		—		—			**9.6**		12.0	21.7	200.9	DNF	
2^{24}	5 GBit/s	21.4	1894	293.6	1271	—		106.3	1442	(90.74)	(850)	(92.8)	(851)	5.0	**6.2**	**762**	14.0	9.9	246.5	526.1	
	1 GBit/s	22.6		294.1				108.2		—					**11.4**		19.3	20.3	301.2	DNF	SGX: 762
	100 MBit/s	169.5		321.8				128.0		(156.4)					**70.8**		78.6	136.6	DNF	DNF	TTP: 1500
	10 MBit/s	DNF		DNF				DNF		(814.2)					**662.7**		670.5	DNF	DNF	DNF	LBE: 8713
	Intra-Cont.	53.1		460.5				169.8		—		—		9.6	**27.1**		39.5	67.7	787.7	DNF	OT: 283617
	Inter-Cont.	126.7		459.2				DNF		—		—			**31.3**		43.7	74.3	818.7	DNF	
2^{26}	5 GBit/s	86.3	7800	DNF		—		543.0	5769	—				15.8	**19.2**	**2438**	58.4	34.8	DNF	DNF	
	1 GBit/s	97.1						549.9							**35.8**		74.9	67.2	DNF	DNF	SGX: 2438
	100 MBit/s	697.3						670.3							**225.8**		264.9	440.5	DNF	DNF	TTP: 4795
	10 MBit/s	DNF						DNF							DNF		DNF				LBE: DNF
	Intra-Cont.	229.7						DNF						24.9	**96.2**		158.0	232.2	DNF	DNF	OT: 907573
	Inter-Cont.	553.8						DNF							**99.4**		161.2	243.3	DNF	DNF	

scheme. We set the statistical security parameter $\lambda = 40$ to minimize the probability of failure that cuckoo hashing exceeds stash size s or regular hashing exceeds the expected maximum number β of elements per bin. Equation (3) in Section 7.1

of Pinkas et al. [67] provides an upper bound for β, i.e., the expected maximum number of elements per bin, given n and α. Similarly, Section 3.2.2 in Pinkas et al. [69] provides estimations for the stash size based on extensive experimentation and extrapolation where necessary. We compute the communication cost per element, i.e., the number of Bit per element, as

$$\frac{\text{Bit}}{\text{element}} = \frac{\alpha \cdot (\beta + 1) + s \cdot (n + 1)}{n} \log Q. \tag{12}$$

Table II summarizes our results. We conclude that for set sizes between 2^{20} and 2^{26} using $k = 3$ has the lowest online communication cost and also does not require a stash.

Next, we theoretically analyze the offline and the total communication cost in bits per element and compare it to related work. We consider the four possible implementations of the offline phase mentioned before: trusted execution environments, a trusted third party, lattice-based homomorphic encryption, and OT.

- The communication cost of using an SGX offline phase is 256 bits to jointly choose a common seed. We did not include the time and communication for attestation, since it can be amortized over multiple runs.
- The communication cost of the offline phase using a trusted third party is 128 bits to Bob (R_B) and $128 + \alpha\beta \log Q$ bits to Alice (R_A and $r_{A,i}$s).
- Let N and q be the polynomial modulus degree and coefficient modulus in SEAL. Also, denote the coefficient modulus after modulus switching by q_0. Each ciphertext sent from Alice to Bob has a size of $N \log_2 q$ bits and Alice can pack N different s_A values into one ciphertext. In sum, Alice sends $N \log_2 q \cdot \lceil \frac{\alpha}{N} \rceil$ bits of data to Bob. For each ciphertext received, Bob sends β ciphertexts back to Alice. Ciphertexts that are sent from Bob to Alice have a size of $2N \log_2 q_0$. The total communication cost using lattice-based homomorphic encryption is $N \cdot (2\beta \log_2 q_0 + \log_2 q) \lceil \frac{\alpha}{N} \rceil$ bits.
- For the OT-based offline phase, we need $\alpha \cdot \beta$ product sharings. Each product sharing is implemented by $\log Q$ instances of 1-out-of-2 OT extensions of length $\log Q$ bit strings. This results in a total of $m = \alpha\beta \log Q$ OT extensions with a total of $2m \log Q = \alpha\beta \cdot 2 \log^2 Q$ Bit of communication.
  To bootstrap m instances of IKNP OT extensions [44], we need, first, to perform κ (security parameter) base OTs of length κ Bit, e.g., using the base OTs by Chou and Orlandi [18, Figure 1]. When implemented over an elliptic curve, these base OTs require a total of $\kappa \cdot (2|P| + 2\kappa)$ Bit of communication, where $|P|$ denotes the bit length of a curve point. Then, we apply a transformation scheme, e.g., the one by Asharov et al. [5], where the OT receiver sends κ strings of length m Bit to the receiver. So, this preparation phase requires in total $\kappa(2|P| + 2\kappa) + \kappa m$ Bit of communication.
  In conclusion, the total communication cost for the OT offline phase (preparation plus extensions) computes to $2m \log Q + \kappa(2|P| + 2\kappa) + \kappa m = \alpha\beta \log Q(2 \log Q + \kappa) + 2(\kappa^2 + \kappa|P|)$ Bit. Note that a curve with G points offers $\kappa \approx \log \sqrt{\frac{G \cdot \pi}{4}}$ Bit security [8], so for our security parameter $\kappa = 128$ in Table III we use a 256 Bit curve and have $|P| = 257$ Bit (using point compression).

Note that an alternative to standard IKNP OT extensions could be recent Silent OT [10, 21]. Silent OT could reduce communication costs at the expense of higher computational costs which might be interesting for low bandwidth networks such as the 10 MBit/s configuration we consider in Section V-B. We leave a further analysis to determine the best OT extensions for different network configurations to future work.

Using the above for the offline phases and Equation 12 for the online phase, we can compute the total communication cost in Bit per element of our PSI protocol. Table III summarizes our results. For related work [15, 57, 70, 76], we use the equations from Table 1 (column 3) of Rindal and Schoppmann [76]. As Garimella et al. [33] and Chandran et al. [14] specify only asymptotic communication complexity and also do not provide source code, we extrapolate their communication costs from their benchmarks.

We conclude that our online communication cost and our total communication cost using Intel SGX is the lowest among related work for larger sets starting from $n = 2^{24}$.

B. Benchmarks

We evaluate the runtime in two different environments and compare it to recent related work [14, 15, 33, 57, 70, 71, 76]. The two environments we consider are, first, a controlled environment where we can precisely adjust network bandwidth using Wondershaper [42]. This controlled environment is a workstation with a 3 GHz Intel Xeon W-1290 CPU and 64 GByte RAM. The second environment comprises different data centers in the Microsoft Azure cloud. There, we emulate an intracontinental scenario ("Intra-Cont.") by benchmarking PSI protocols between two data centers on the US East coast and the US West coast. We also emulate an intercontinental scenario ("Inter-Cont.") by benchmarking protocols between two data centers on the US East coast and Western Europe. Each data center is running our implementation in a standard Azure H8 instance, i.e., on an Intel 3.2 GHz (3.6 GHz turbo frequency) Xeon E5-2667 CPU with 56 GByte RAM.

Our implementation and all benchmarks use statistical security parameter $\lambda = 40$ and computational security parameter $\kappa = 128$.

a) Offline: We first report on our evaluation of the offline phase in the controlled environment. Note that although our protocol is highly parallelizable, all experiments both in the offline and online phases are run single threaded for fair comparison with related work. Our results are summarized in Table IV.

As expected, the TTP and SGX offline phases significantly outperform the OT- and LBE-based ones. For very large values of $n = 2^{26}$ or low bandwidth networks (10 MBit/s), they do not complete within our set time limit of 15 minutes, so we do not report measurement results. For $n = 2^{26}$, we can measure

TABLE VI. ONLINE END-TO-END TIME COMPARED TO CIRCUIT-PSI SCHEMES. DNF: DID NOT FINISH IN 15 MIN OR CRASHED. VALUE IN "()" TAKEN FROM ORIGINAL PAPER [71]

		PSTY19 [71]		CGS22 (PSM_2) [14]		**Ours**	
n	Bandwidth	Time	Comm	Time	Comm	**Time**	**Comm**
2^{20}	5 GBit/s	37.5	(2540)	17.0	1127	**0.4**	**64**
	1 GBit/s	52.5		23.1		**0.8**	
	100 MBit/s	255.6		107.2		**5.9**	
	10 MBit/s	DNF		DNF		**55.9**	
2^{22} – 2^{26}		DNF		DNF		see Table V	

CPU time and communication requirements for the OT-based offline phase, simply by releasing the network throttle. This is not possible for the LBE-based offline phase, as even computation does not complete in 15 min. As we have theoretically analyzed in Section IV, the dual TEE configuration in the SGX offline phase has a small constant communication cost for setting up the seed whereas all our other offline phases have linear communication cost. Hence, the running time of the SGX offline phases is largely independent of the network environment different from our other offline phases.

We conclude that the offline phase based on SGX is by far the most efficient in communication cost and total latency. However, we note that it introduces an addition security assumption compared to the related work with which we directly compare. Its communication cost is much smaller than other implementations and the time to communicate the tuples, even in plain, exceeds the computation cost in a protected environment such as a trusted enclave. The offline phase implemented with a TTP, such as a cloud service provider, could be scheduled for times of low utilization, thus reducing its cost. It turns out that, due to smaller communication requirements, the offline phase using LBE is more efficient than the OT-based ones. Moreover, it makes the least trust assumptions of all offline phases. Hence, LBE is the method of choice when relying only on cryptographic assumptions as related work [15, 33, 57, 70, 76] does.

b) Online: Next, we report on the running time and exchanged data of our online phase in comparison to related work [15, 33, 57, 70, 76]. We compare to the publicly available implementations where available and measure running times and the amount of data exchanged within the same environments. All protocols are executed single-threaded as is common practice for fair comparison. Only for VOLE-PSI [76] and OKVS [33], we resort to their published numbers in the most comparable setting, since there is no publicly available implementation. The faster of the two, VOLE-PSI [76] was evaluated on an Amazon EC2 M5.2xlarge VMs, Intel Xeon Platinum 8175M 2.5 GHz (3.5 GHz turbo frequency), 32 GiB of RAM. We chose Micrsoft Azure, since it provides access to TEEs, and consider our configuration the most similar to the Amazon one among the ones that were available. For the SpOT-Light PSI protocol by Pinkas et al. [70], we use the computation-optimized version, since the communication-optimized version had very high running times and performed worse in our tested environments.

The benchmarks are summarized in Table V. We measure our implementation and implementations of related work and present total, end-to-end running time and communication exchanged. For our scheme, we additionally show CPU-only time (including hashing, all comparisons, and Alice's computation of the intersection) to allow a better understanding of our scheme's performance.

On larger data sets, starting at $n = 2^{24} = 16$ million elements, our online phase is 2.4 to 3.5 times faster than the protocol by Kolesnikov et al. [57] and 1.2 to 14.6 times faster than the one by Rindal and Schoppmann [76], the two currently most competitive related works. At the same time, our protocol requires less communication than either one of them. Kolesnikov et al.'s protocol [57] performs best in high-performance network environments, which may be the future given that CPU performance currently increases slower than network performance, and it is already outperforming VOLE-PSI [76] in the same-continent cloud setting (compared to the fastest time measured for VOLE-PSI). Schoppmann and Rindal's protocol VOLE-PSI [76] has the lowest communication cost for smaller sets with $n = 2^{20}$ and $n = 2^{22}$ and performs best in low-performance network environments. VOLE-PSI has lower communication complexity ($O(n)$) compared to our protocol ($O(n \log n / \log\log n)$) but higher computation complexity ($O(n)\,\mathrm{polylog}\, n$) compared to our protocol's complexity of $O(n \log n / \log\log n)$. However, in the common network settings as experienced in our real-world cloud experiments, we expect that the advantage in total latency of the online phase of our protocol compared to VOLE-PSI increases as data set sizes continue to grow, since the protocols in these environments are computation-bound and not communication-bound.

Although circuit-PSI provides a richer functionality than our PSI protocol, we compare communication cost and running times with the two most recent, cirucit-PSI protocols [14, 71] in Table VI. Not surprisingly, our protocol is much faster, and implementations of circuit-PSI protocols do not even scale beyond set sizes of 2^{20}.

c) Total time: When using lattice-based cryptography for our offline phase, Kolesnikov et al.'s protocol [57] is the fastest in high-performance network environments and VOLE-PSI [76] is the fastest in low-performance network environments. Only when allowing additional security assumptions during the offline, such as TEEs, our protocol becomes the fastest in total running time of the combined offline and online phase. Note that this does not affect our performance advantage during the online phase which is the focus of this work.

VI. RELATED WORK

Freedman et al. [31] coined the term private set intersection (PSI) and since then progress has been fast and steady. There exist many variants of PSI protocols. We can distinguish between PSI protocols which reveal the intersection itself and PSI analytics (or circuit PSI) protocols which allow to compute functions over the intersection, such as the (differentially private) set intersection cardinality. We can also distinguish

between protocols secure in the semi-honest model or the malicious model. However, we argue in Section II-A that this distinction is of minor practical importance in PSI (not necessarily PSI analytics) protocols.

A. Industrial Work and Applications

Google has deployed PSI for computing the value of ad conversion with Mastercard [43, 83]. Their protocol uses commutative elliptic curve encryption [61, 80], one of the first techniques to compute PSI. The communication cost of this protocol is asymptotically optimal, very low in practice, and has only been recently matched by other protocols [71, 76]. The computation cost of this protocol is rather high, but Yung mentions its flexibility as one of its advantages when deployed [83]. Their protocol is a PSI protocol revealing the intersection and they use extensions to accommodate certain functions, such as the differentially private set intersection cardinality [43]. These extensions are also applicable to our and other PSI protocols. Their protocol is secure in the semi-honest, but not the malicious model. These properties match our protocol, and our protocol is also conceptually simple as can been seen in Algorithm 1.

There exist other industrial implementations of PSI protocols such as by Facebook [12], Microsoft [16, 17], VISA [76] and VMWare [33]. The VOLE-PSI protocol supported by Google and VISA [76] also has an offline phase like our protocol. Miao et al. [62] have developed a maliciously secure, circuit PSI protocol for restricted functions, such as sum and cardinality.

A practical proposal is also to use three servers similar to our trusted third party setup [48]. In this setup, the parties share a symmetric key and send keyed hashes of the elements to a third party that compares them but does not have access to the key. Compared to our construction – even when using a trusted third party – this setup has two disadvantages. First, the third party is involved during the online phase whereas in our construction the third party is online involved in the offline phase. Second, the online phase still requires cryptographic operations albeit only fast symmetric key ones whereas our online phase does not require any cryptographic operations and hence is significantly faster.

B. PSI protocols

PSI protocols compute comparison between the elements of the two sets. The first approach by Meadows [61] and Shamir [80] is to compute the comparisons over pseudo-random functions (PRF) of the inputs. Their construction still requires to compute one public-key operation per element. The number of public-key operations can be reduced by using PCRGs or PRFs based on OT extensions [44]. Using OT extensions, it is possible to compute a small number of public-key operations and only use symmetric key operations for each element. This significantly speeds up the entire protocol. Several such constructions of oblivious PRFs (OPRF) exist [27, 57, 70, 72, 75, 76]. These constructions can often be made secure in the malicious model with little overhead [72, 75, 76]. The current state-of-the-art protocols for PSI with the best communication and computation cost [57, 76] are based on this construction.

We use a different construction than these protocols. Instead of computing a PRF per element which has communication complexity $O(n)$, we perform a secure computation per comparison (which has higher communication complexity). However, we are able to provide an incredibly fast implementation of comparisons (assuming an offline phase) using only four field operations and no cryptographic operations. This improves the practical latency over medium-fast to fast networks as our experiments in Section V-B show. Our protocol for comparisons is inspired by oblivious linear function evaluation (OLE) and Beaver multiplication triples [7]. Ghosh and Nilges [34] already present a PSI protocol based on OLE, but use a complicated construction based on polynomials that cannot achieve our efficiency even when adapted to the semi-honest model.

When computing comparisons using a dedicated protocol instead of a plain comparison algorithm over PRFs, a challenge is to reduce the number of comparisons necessary, since each requires interaction. Naively, there are $O(n^2)$ comparisons. In a series of works, Pinkas et al. [66, 67, 69, 71] refined a strategy for Alice and Bob to hash their elements to common bins, such that only $O(n \log n / \log \log n)$ comparisons are necessary between pairs of common bins. We follow this strategy as described in Section III-C, but replace the comparison protocol with our own which makes this construction more efficient than OT extension-based OPRF ones. Note that Pinkas et al. also provide a different strategy for dual cuckoo hashing which leads to a circuit PSI protocols [68].

PSI analytics or circuit PSI protocols support a larger set of functionalities and hence applications, but are usually slower than PSI protocols. A first PSI protocol based on the generic secure two-party computation protocol by Yao (garbled circuits) was presented by Huang et al. [41]. The most efficient current constructions base on OPRF protocols, but use a technique called oblivious, programmable PRF (OPPRF) [58]. The idea of an OPPRF is that the value of the PRF at given inputs can be fixed by the key holder. This allows the comparison result to be secret shared and available for subsequent secure computations over the intersection. Garimella et al. [33] generalize OPPRF to oblivious key-value stores (OKVS). The most recent and most efficient circuit PSI protocol (under common wired network conditions) is VOLE-PSI [76] which follows this construction using OPPRFs. VOLE-PSI also can be used as a PSI protocol (as any circuit PSI protocol) to which we compare favourably in Section V.

C. Specialized PSI protocols

There exist several variations of PSI and circuit PSI that cater for specific applications scenarios. Circuit PSI performs a secure computation on the secret shared outputs of the PSI protocol. However, the PSI protocols may be a part of a larger secure computation that does not start with a PSI protocol and consequently the inputs must also be secret shared. Mohassel et al. [63] present a variation of PSI for this setting.

It has been proposed to perform contact discovery in mobile phone messaging apps, such as Signal, using PSI. In this case, one set, the cell phone users' one, is much smaller than the other, the messaging provider's one, i.e., the set sizes are imbalanced. Modified OPRF protocols have been proposed

for this setting [55, 74]. However, they require to store the PRFs of the larger set in the downloaded software with a key common to all users and consequently hinge on the security of this key. Recent protocols without this restriction, which turn out to be also faster, use fully homomorphic encryption (FHE) [16, 17, 19]. FHE-based constructions can also be used for circuit PSI [47].

PSI with a semi-trusted third party has received attention under several different names. It is beneficial when the communication between the two parties is restricted. Kerschbaum introduced the concept in 2012 as outsourced PSI [52]. Dong et al. [26] named the third party an arbiter in 2013 and Abadi et al. [1] named it delegated PSI in 2015. Some functions on the intersection, such as intersection cardinality, can be computed in this setting [28]. Our offline phase using a trusted third party could be considered this setting, but our communication complexity between Alice and Bob remains large ($o(n)$).

PSI can also be computed among multiple, more than two, parties. The protocol with the currently best complexity is by Hazay and Venkitasubramaniam [39] and the protocol with the best practical performance due to the extensive use of symmetric encryption is by Kolesnikov et al. [58]. Protocols with variants for the definition of intersection, such as the intersection of a subset with a size above a threshold, exist [56, 60].

PSI requires all elements to be unique within one set, i.e., there are no duplicates in each set. This is different from database joins where elements in a relational table can be duplicated. Database joins require different matching procedures [59, 63]. PSI also performs only exact matches of the elements. Private intersection with approximate matches is often referred to as private record linkage (PRL). The challenge in PRL is to find the $O(n)$ matches among the $O(n^2)$ pairs where the hashing strategy used by PSI protocols does not apply. Instead one can use locality-sensitive hashing [40] and then differentially privately pad the bins or locality-preserving hashing and use sliding windows [53].

VII. Conclusions

PSI protocols have been extensively studied in the literature, and progress in practical latency is only possible by improving constants. In this paper, we present the first construction of a PSI protocol that is practically efficient and uses no cryptographic operations during the online phase. Instead, all cryptographic operations are moved to an offline phase. For the online phase, we present a new comparison protocol which consumes only one OLE tuple per comparison. The online phase of our protocol outperforms the currently best related work [15, 33, 57, 70, 76] by factors between 1.2 and 3.5 depending on the network performance. There exist many different implementations for the offline phase which can be shared with other privacy-preserving protocols using precomputed, correlated, random secret shares. Among others, we present a very efficient construction using TEEs, such as Intel's SGX. The performance of our PSI protocol with a TEE-based offline phase is currently unmatched.

Acknowledgements

We gratefully acknowledge the support of NSERC for grants RGPIN-05849, IRC-537591, and the Royal Bank of Canada for funding this research.

References

[1] A. Abadi, S. Terzis, and C. Dong. O-psi: delegated private set intersection on outsourced datasets. In *30th IFIP International Information Security and Privacy Conference (SEC)*, 2015.

[2] B. Applebaum, I. Damgård, Y. Ishai, M. Nielsen, and L. Zichron. Secure arithmetic computation with constant computational overhead. In *37th International Cryptology Conference (CRYPTO)*, 2017.

[3] Y. Arbitman, M. Naor, and G. Segev. Backyard cuckoo hashing: constant worst-case operations with a succinct representation. In *51st IEEE Symposium on Foundations of Computer Science (FOCS)*, 2010.

[4] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. Stillwell, D. Goltzsche, D. M. Eyers, R. Kapitza, P. R. Pietzuch, and C. Fetzer. SCONE: secure linux containers with intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation OSDI*, 2016.

[5] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *19th ACM Conference on Computer and Communications Security (CCS)*, 2013.

[6] C. Baum, D. Escudero, A. Pedrouzo-Ulloa, P. Scholl, and J. R. Troncoso-Pastoriza. Efficient protocols for oblivious linear function evaluation from ring-lwe. In *12th International Conference on Security and Cryptography for Networks (SCN)*, 2020.

[7] D. Beaver. Efficient multiparty protocols using circuit randomization. In *11th International Cryptology Conference (CRYPTO)*, 1991.

[8] D. J. Bernstein and T. Lange. SafeCurves: choosing safe curves for elliptic-curve cryptography, 2021. https://safecurves.cr.yp.to.

[9] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai. Compressing vector OLE. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *24th ACM Conference on Computer and Communications Security (CCS)*, 2018.

[10] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, P. Rindal, and P. Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In *25th ACM Conference on Computer and Communications Security (CCS)*, 2019.

[11] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl. Efficient pseudorandom correlation generators from ring-lpn. In *40th International Cryptology Conference (CRYPTO)*, 2020.

[12] P. Buddhavarapu, A. Knox, P. Mohassel, S. Sengupta, E. Taubeneck, and V. Vlaskin. Private matching for compute. *IACR Cryptology ePrint Archive*, 2020(599), 2020.

[13] J. Camenisch and G. M. Zaverucha. Private intersection of certified sets. In *13th International Conference Financial Cryptography and Data Security (FC)*, 2009.

[14] N. Chandran, D. Gupta, and A. Shah. Circuit-psi with linear complexity via relaxed batch OPPRF. *Proceedings of Privacy Enhancing Technologies*, 2022(1), 2022.

[15] M. Chase and P. Miao. Private set intersection in the internet setting from lightweight oblivious PRF. In *40th International Cryptology Conference (CRYPTO)*, 2020.

[16] H. Chen, K. Laine, and P. Rindal. Fast private set intersection from homomorphic encryption. In *23rd ACM Conference on Computer and Communications Security (CCS)*, 2017.

[17] H. Chen, Z. Huang, K. Laine, and P. Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In *24th ACM Conference on Computer and Communications Security (CCS)*, 2018.

[18] T. Chou and C. Orlandi. The Simplest Protocol for Oblivious Transfer. In *4th International Conference on Cryptology and Information Security in Latin America (LATINCRYPT)*, 2015.

[19] K. Cong, R. C. Moreno, M. B. da Gama, W. Dai, I. Iliashenko, K. Laine, and M. Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. In *27th ACM Conference on Computer and Communications Security (CCS)*, 2021.

[20] M. Corp. Scaling secure enclave environments with Signal and Azure confidential computing. https://customers.microsoft.com/en-us/story/1374646124015821 54-signal-nonprofit-azure-security, 2021.

[21] G. Couteau, P. Rindal, and S. Raghuraman. Silver: Silent VOLE and oblivious transfer from hardness of decoding structured LDPC codes. In *41st International Cryptology Conference (CRYPTO)*, 2021.

[22] E. D. Cristofaro, J. Kim, and G. Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *16th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2010.

[23] I. Damgård and R. Thorbek. Efficient conversion of secret-shared values between different fields. *IACR Cryptology ePrint Archive*, 2008(221), 2008.

[24] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *32nd Annual Cryptology Conference (CRYPTO)*, 2012.

[25] L. de Castro, C. Juvekar, and V. Vaikuntanathan. Fast Vector Oblivious Linear Evaluation from Ring Learning with Errors. In *9th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC)*, 2021.

[26] C. Dong, L. Chen, J. Camenisch, and G. Russello. Fair private set intersection with a semi-trusted arbiter. In *27th IFIP Conference Data and Applications Security and Privacy (DBSEC)*, 2013.

[27] C. Dong, L. Chen, and Z. Wen. When private set intersection meets big data: an efficient and scalable protocol. In *19th ACM Conference on Computer and Communications Security (CCS)*, 2013.

[28] T. Duong, D. H. Phan, and N. Trieu. Catalic: delegated PSI cardinality with applications to contact tracing. In *26th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2020.

[29] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28, 1985.

[30] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012(144), 2012.

[31] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2004.

[32] M. J. Freedman, C. Hazay, K. Nissim, and B. Pinkas. Efficient set intersection with simulation-based security. *Journal of Cryptology*, 29(1), 2016.

[33] G. Garimella, B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. Oblivious key-value stores and amplification for private set intersection. In *41st International Cryptology Conference (CRYPTO)*, 2021.

[34] S. Ghosh and T. Nilges. An algebraic approach to maliciously secure private set intersection. In *38th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2019.

[35] N. Gilboa. Two party RSA key generation. In *19th International Cryptology Conference (CRYPTO)*, 1999.

[36] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *19th ACM Symposium on Theory of Computing (STOC)*, 1987.

[37] G. H. Gonnet. Expected length of the longest probe sequence in hash code searching. *Journal of the ACM*, 28(2), 1981.

[38] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic. Sok: General purpose compilers for secure multi-party computation. In *40th IEEE Symposium on Security and Privacy (S&P)*, 2019.

[39] C. Hazay and M. Venkitasubramaniam. Scalable multi-party private set-intersection. In *20th International Conference on Practice and Theory in Public-Key Cryptography (PKC)*, 2017.

[40] X. He, A. Machanavajjhala, C. J. Flynn, and D. Srivastava. Composing differential privacy and secure computation: a case study on scaling private record linkage. In *23rd ACM Conference on Computer and Communications Security (CCS)*, 2017.

[41] Y. Huang, D. Evans, and J. Katz. Private set intersection: are garbled circuits better than custom protocols? In *19th Network and Distributed System Security Symposium (NDSS)*, 2012.

[42] B. Hubert, J. Geul, and S. Sehier. Wondershaper, a command-line utility for limiting an adapter's bandwidth, 2021. https://github.com/magnific0/wondershaper.

[43] M. Ion, B. Kreuter, A. E. Nergiz, S. Patel, S. Saxena, K. Seth, M. Raykova, D. Shanahan, and M. Yung. On deploying secure computing: private intersection-sum-with-cardinality. In *5th IEEE European Symposium on Security and Privacy (EuroS&P)*, 2020.

[44] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *23rd Annual International Cryptology Conference (CRYPTO)*, 2003.

[45] Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer — efficiently. In *28th International Cryptology Conference (CRYPTO)*, 2008.

[46] Y. Ishai, M. Prabhakaran, and A. Sahai. Secure arithmetic computation with no honest majority. In *6th Theory of*

Cryptography Conference (TCC), 2009.

[47] B. Kacsmar, B. Khurram, N. Lukas, A. Norton, M. Shafieinejad, Z. Shang, Y. Baseri, M. Sepehri, S. Oya, and F. Kerschbaum. Differentially private two-party set operations. In *5th IEEE European Symposium on Security and Privacy (EuroS&P)*, 2020.

[48] S. Kamara, P. Mohassel, M. Raykova, and S. S. Sadeghian. Scaling private set intersection to billion-element sets. In *18th International Conference on Financial Cryptography and Data Security (FC)*, 2014.

[49] M. Keller. MP-SPDZ: A versatile framework for multi-party computation. In *26th ACM Conference on Computer and Communications Security (CCS)*, 2020. https://github.com/data61/MP-SPDZ.

[50] M. Keller, E. Orsini, and P. Scholl. MASCOT: faster malicious arithmetic secure computation with oblivious transfer. In *22nd ACM Conference on Computer and Communications Security (CCS)*, 2016.

[51] M. Keller, V. Pastro, and D. Rotaru. Overdrive: Making SPDZ great again. In *37th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2018.

[52] F. Kerschbaum. Collusion-resistant outsourcing of private set intersection. In *27th ACM Symposium on Applied Computing (SAC)*, 2012.

[53] B. Khurram and F. Kerschbaum. SFour: a protocol for cryptographically secure record linkage at scale. In *36th IEEE International Conference on Data Engineering (ICDE)*, 2020.

[54] J. Kilian. Founding cryptography on oblivious transfer. In *20th ACM Symposium on Theory of Computing (STOC)*, 1988.

[55] Á. Kiss, J. Liu, T. Schneider, N. Asokan, and B. Pinkas. Private set intersection for unequal set sizes with mobile applications. *Proceedings of Privacy Enhancing Technologies*, 2017(4), 2017.

[56] L. Kissner and D. Song. Private and threshold set-intersection. Technical Report, Carnegie-Mellon University, 2004.

[57] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu. Efficient batched oblivious PRF with applications to private set intersection. In *22nd ACM Conference on Computer and Communications Security (CCS)*, 2016.

[58] V. Kolesnikov, N. Matania, B. Pinkas, M. Rosulek, and N. Trieu. Practical multi-party private set intersection from symmetric-key techniques. In *23rd ACM Conference on Computer and Communications Security (CCS)*, 2017.

[59] S. Krastnikov, F. Kerschbaum, and D. Stebila. Efficient oblivious database joins. *Proceedings of the VLDB Endowment*, 13(11), 2020.

[60] R. A. Mahdavi, T. Humphries, B. Kacsmar, S. Krastnikov, N. Lukas, J. A. Premkumar, M. Shafieinejad, S. Oya, F. Kerschbaum, and E. Blass. Practical over-threshold multi-party private set intersection. In *36th Annual Computer Security Applications Conference (ACSAC)*, 2020.

[61] C. A. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *7th IEEE Symposium on Security and Privacy (S&P)*, 1986.

[62] P. Miao, S. Patel, M. Raykova, K. Seth, and M. Yung. Two-sided malicious security for private intersection-sum with cardinality. In *40th International Cryptology Conference (CRYPTO)*, 2020.

[63] P. Mohassel, P. Rindal, and M. Rosulek. Fast database joins and PSI for secret shared data. In *26th ACM Conference on Computer and Communications Security (CCS)*, 2020.

[64] M. Naor and B. Pinkas. Oblivious polynomial evaluation. *SIAM Journal of Computing*, 35:1254–1281, 2006.

[65] R. Pagh and F. F. Rodler. Cuckoo hashing. *Journal of Algorithms*, 51(2), 2004.

[66] B. Pinkas, T. Schneider, and M. Zohner. Faster private set intersection based on OT extension. In *23rd USENIX Conference on Security Symposium (USENIX Security)*, 2014.

[67] B. Pinkas, T. Schneider, G. Segev, and M. Zohner. Phasing: private set intersection using permutation-based hashing. In J. Jung and T. Holz, editors, *24th USENIX Security Symposium (USENIX Security)*, 2015.

[68] B. Pinkas, T. Schneider, C. Weinert, and U. Wieder. Efficient circuit-based PSI via cuckoo hashing. In *37th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2018.

[69] B. Pinkas, T. Schneider, and M. Zohner. Scalable private set intersection based on OT extension. *ACM Transactions on Privacy and Security*, 21(2), 2018.

[70] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. SpOT-Light: Lightweight private set intersection from sparse OT extension. In *39th International Cryptology Conference (CRYPTO)*, 2019.

[71] B. Pinkas, T. Schneider, O. Tkachenko, and A. Yanai. Efficient circuit-based PSI with linear communication. In *38th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2019.

[72] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai. PSI from PaXoS: fast, malicious private set intersection. In *39th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2020.

[73] M. O. Rabin. How to exchange secrets with oblivious transfer. Technical Report TR-81, Aiken Computation Lab, Harvard University, 1981.

[74] A. C. D. Resende and D. F. Aranha. Faster unbalanced private set intersection. In *22nd International Conference on Financial Cryptography and Data Security (FC)*, 2018.

[75] P. Rindal and M. Rosulek. Improved private set intersection against malicious adversaries. In *36th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2017.

[76] P. Rindal and P. Schoppmann. VOLE-PSI: fast OPRF and circuit-PSI from vector-OLE. In *40th International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2021.

[77] P. Scholl, N. P. Smart, and T. Wood. When it's all just too much: Outsourcing mpc-preprocessing. *IACR Cryptology ePrint Archive*, 2017(262), 2017.

[78] P. Schoppmann, A. Gascón, L. Reichert, and M. Raykova. Distributed vector-ole: Improved constructions and implementation. In *25th ACM Conference on Computer and Communications Security (CCS)*, 2019.

[79] SEAL. Microsoft SEAL (release 3.6). https://github.com/Microsoft/SEAL, Nov. 2020. Microsoft Research, Redmond, WA.

[80] A. Shamir. On the power of commutativity in cryptography. In *7th International Colloquium on Automata, Languages and Programming (ICALP)*, 1980.

[81] N. P. Smart and T. Tanguy. Taas: Commodity MPC via triples-as-a-service. In *ACM Cloud Computing Security Workshop, (CCSW)*, 2019.

[82] X. Wang, A. J. Malozemoff, and J. Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016.

[83] M. Yung. From mental poker to core business: why and how to deploy secure computation protocols? In *21st ACM Conference on Computer and Communications Security (CCS)*, 2015.