Labeled PSI from Fully Homomorphic Encryption with Malicious Security

Hao Chen[1], Zhicong Huang[2], Kim Laine[1], and Peter Rindal[3]

[1] Microsoft Research, Redmond, WA
[2] École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland
[3] Oregon State University, Corvallis, OR

Abstract. Private Set Intersection (PSI) allows two parties, the sender and the receiver, to compute the intersection of their private sets without revealing extra information to each other. We are interested in the *unbalanced* PSI setting, where (1) the receiver's set is significantly smaller than the sender's, and (2) the receiver (with the smaller set) has a low-power device. Also, in a *Labeled PSI* setting, the sender holds a label per each item in its set, and the receiver obtains the labels from the items in the intersection. We build upon the unbalanced PSI protocol of Chen, Laine, and Rindal (CCS 2017) in several ways: we add efficient support for arbitrary length items, we construct and implement an unbalanced Labeled PSI protocol with small communication complexity, and also strengthen the security model using Oblivious Pseudo-Random Function (OPRF) in a pre-processing phase. Our protocols outperform previous ones: for an intersection of 2^{20} and 512 size sets of arbitrary length items our protocol has a total online running time of just 1 second (single thread), and a total communication cost of 4 MB. For a larger example, an intersection of 2^{28} and 1024 size sets of arbitrary length items has an online running time of 12 seconds (multi-threaded), with less than 18 MB of total communication.

1 Introduction

1.1 Private Set Intersection

Private Set Intersection (PSI) is a secure computation protocol that allows two parties, the *sender* and the *receiver*, to compute the intersection of their private sets X and Y with pre-determined sizes, such that the receiver only learns $X \cap Y$ from the interaction and the sender learns nothing.

PSI has a long history, and motivating use-cases have ranged from two companies finding common customers, to private contact discovery [37], and to validate password leaks [1].

Unbalanced PSI Most of the work on PSI has been designed for the *balanced* case, where the two sets are roughly of equal size, and the two parties have similar computation and storage capabilities. These protocols typically perform only marginally better when one of the sets is much smaller than the other. In particular, their communication cost scales at least linearly with the size of the larger set. In certain applications, however, the receiver's set may be much smaller than the sender's. The receiver might be a mobile device with limited battery, computing power, and storage, whereas the sender could be a high-end computing device. Moreover, the bandwidth between the receiver and sender might be limited. This motivates the study of *unbalanced* PSI, where one set is much larger than the other. There have recently been several proposals optimizing for unbalanced PSI [12,44,47]. Among these works [12] achieves the smallest overall communication complexity, namely $O(|Y| \log |X|)$, where X denotes the sender's set and Y the receiver's set, and $|X| \gg |Y|$. However, their results were limited to 32-bit items due to the significant performance overhead of extending to longer items. In this work we improve their protocol in terms of functionality, performance, and the security model.

Labeled PSI In certain applications, the sender holds a *label* ℓ_i for each item x_i in its set, and we wish to allow the receiver to learn the labels corresponding to the items in the intersection. In other words, the receiver should learn $\{(x_i, \ell_i) : x_i \in Y\}$ as a result of the protocol execution. When the receiver's set Y consists of a single element, this is equivalent to the (single-server variant of) *Private Information Retrieval (PIR) by keywords* problem, considered first by [18]. For ease of exposition, we will stick to the concept of Labeled PSI in the rest of the paper, noting that it is equivalent to a batched single-server symmetric PIR by keywords.

Labeled PSI has some immediate practical applications to private web service queries. For example, querying stock prices, location specific information, travel booking information, or web domain name information can reveal information to the service providers allowing them to conduct highly targeted price discrimination [40], or to obtain sensitive personal or business information. Another example is a variant the private contact discovery problem [37], where a user wishes to retrieve a public key for every person in her contact list who has registered to an instant messaging service for peer-to-peer communication. In this regard, Labeled PSI can provide the necessary functionality while guaranteeing query privacy.

1.2 Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE) is a form of encryption that allows arbitrary (Boolean or arithmetic) circuits to be evaluated directly on encrypted data without requiring access to the decryption key [49,24,9,10,7,22,8,26,17,13]. The result of such an encrypted evaluation remains encrypted, and can be recovered by the party holding the decryption key. While FHE is still far from being a generic solution to computation over encrypted data, it can be possible to achieve good performance in specific scenarios, e.g. evaluating the AES circuit [25], computing edit distance on DNA sequences [16], and training logistic regression models [34].

One of the fundamental realizations on the path to improved performance was that in many cases so-called *leveled* FHE suffices, where the parameters of the encryption scheme are chosen to allow only circuits of a pre-determined multiplicative depth to be evaluated. Thus, for performance reasons, we only use leveled FHE in this work, and for simplicity we will sometimes drop the "leveled" prefix.

There are several FHE implementations that are publicly available. We chose to use the homomorphic encryption library SEAL[4], which implements the variant of [4] of the Brakerski/Fan-Vercauteren (BFV) scheme [22].

FHE parameters and the cost model The core parameters of the BFV scheme are three integers: n, q, and t.[5] We set the parameters to always achieve at least a 128-bit security level according to the recommendations provided in [11]. In order to compare different parameter choices, we need to provide some rough cost estimates for basic operations in SEAL. The size of each ciphertext is $2n \log q$ bits, and the size of the underlying plaintext is $n \log t$ bits. In terms of computation, a multiplication between two ciphertexts takes $O(n \log n (\log q)^2)$ bit operations, whereas a ciphertext-plaintext multiplication takes $O(n \log n \log q)$ bit operations (see e.g. [4]),

1.3 Our Contributions

At a high level, our contributions can be summarized as follows:

[4] Version 2.3.0-4 downloaded from http://sealcrypto.org

[5] Another important parameter is the error width σ for which we use the SEAL default value 3.19.

– We use an OPRF preprocessing phase to achieve improved performance and security over [12]. In particular, We achieve full simulation-based security against a malicious receiver, improving upon [12] which achieves security in the semi-honest model.
– We build upon [12] to support arbitrary length items by implementing a modified version of the generalized SIMD encoding algorithm of [27].
– We extend our protocol to achieve Labeled PSI with small communication complexity. We then apply Labeled PSI to provide improved security against a malicious sender.

First, we improve upon the protocol of [12] by leveraging a pre-processing phase, where the parties apply an Oblivious PRF (OPRF) to their input items. This change has two effects:

(i) The sender no longer needs to perform an expensive noise flooding operation on the result ciphertexts, as was necessary in [12]. This allows the implementation to utilize more efficient FHE parameters, which further improves our performance and adds flexibility to the parametrization.
(ii) The pre-processing phase allow us to argue that our protocol is secure against a malicious receiver. In particular, we show that after the pre-processing phase the simulator can extract the receiver's set and successfully simulate the rest of the protocol. We show that this pre-processing can either be performed using exponentiation [31,47] or oblivious transfer [42,36]. Crucially, the former allows the sender to perform the pre-processing only once and reuse it over many protocol executions, which significantly improves performance in an amortized setting, e.g. in private contact discovery.

Second, all examples in [12] were limited to performing comparisons of 32-bit items, whereas applications usually require longer items such as phone numbers, names, or public keys. The obvious extension of their protocol to fit larger items has suboptimal performance due to the significant performance overhead from using larger FHE parameters. We overcome this limitation by implementing a "lazy" version of the generalized SIMD encoding algorithm used in the HElib library [27], which allows encrypting fewer but longer items into a single ciphertext without requiring the FHE parameters to be increased.

Our third contribution is the design and implementation of a protocol for Labeled PSI. The challenge in this setting is how to allow the receiver to learn a label ℓ_i for each item $x_i \in X \cap Y$, while still keeping the communication sublinear in $|X|$. Although many existing PSI protocols [47,44] can perform this functionality with a simple modification (encrypt the labels under their respective OPRF value), all of them fall short on the requirement that the communication be sublinear in the larger set. We propose a method which is similar to the original [12] protocol, except that the sender evaluates an additional polynomial which interpolates the labels. When these labels are appropriately masked, the receiver will be able to recover the label for exactly the items in the intersection.

Fourth, we discuss how a variant of Labeled PSI can be leveraged to achieve a reasonable notion of security against a malicious sender. For example, the protocol of [12] suffers from an attack where the sender can force the receiver to output its full set Y. In our protocol the receiver will output $X \cap Y \cap \mathsf{leakage}(\widehat{Y})$, where $\mathsf{leakage}$ is specified by the malicious sender and is constrained in a reasonable way, and $\widehat{Y}$ denotes a hashing of the set Y (see Section 7.3). The high-level idea is to perform Labeled PSI where the label for an item z is $H(z)$ for some hash function H. We argue that the only efficient way for the sender to return the expected label for a receiver's item $y \in Y$ is to have the set X contain the item y. The exact assumption we make is that the sender can not homomorphically compute an encryption of $H(y)$ given an encryption of y and some predetermined list of encryptions of powers of y, when H is a sufficiently complex hash function. The validity of this assumption depends on the difficulty of using leveled FHE to evaluate a circuit of

depth higher than the pre-determined upper bound. Any efficient attack on this assumption would likely represent a very significant advancement in the state-of-the-art.

Fifth, we demonstrate how the output of the PSI computation can be secret-shared between the parties. This immediately allows for generic computation on the intersection using any general purpose MPC protocol. The idea behind this extension is for the sender to add an additional random value to all of the returned ciphertexts. When the receiver decrypts them the two parties will hold an additive sharing of the comparison results. From such a sharing additional computation can be performed on the intersection. For example, the cardinality of the intersection, or the sum of the labels can be computed.

1.4 Related Work

Unbalanced PSI As previously described, our protocol can be seen as a descendant of the of the FHE-based protocol of [12]. Due to the large overlap with our protocol, we delay a review of this work to Section 2 where a detailed description is provided.

Another closely related work is that of Resende *et al.* [47], which optimizes the communication overhead of PSI when the sender has a significantly larger set than the receiver. The central technique of their protocol is to apply an OPRF to the receiver's set to obtain a new set $Y' = \{\text{OPRF}_k(Y) : y \in Y\}$. Here the sender holds the key k and can locally apply the OPRF to its set to obtain $X' = \{\text{OPRF}_k(x) : x \in X\}$. To reduce communication, the sender compresses the set X' before forwarding it to the receiver. While this compression does reduce the communication, it remains linear in the size of the larger set, and can introduce false positives. That is, with high compression rates the receiver outputs an element in $Y \setminus X$ with non-negligible probability.

Another work following a similar framework as [47] is that of Kiss *et al.* [35]. The main difference between these protocols is in the choice of the OPRF and the compression technique. [47] uses a Diffie-Hellman based OPRF, while [35] uses garbled circuits to obliviously evaluate the AES function. This alteration significantly improves the computational work required by the sender to apply the OPRF to its set at the expense of increased communication and computation when the OPRF is applied to the receiver's set. The second difference is that [35] uses a more conservative compression technique and parameters, which do not introduce a significant false positive rate.

PSI with Computation Being able to compute functions on the intersection without revealing intermediate results, or even the intersection, is often a desirable property. For instance, [30] built a PSI-SUM protocol, which returns a weighted sum of all items in the intersection. This protocol is currently being used by Google to compute ad revenue conversion rates.

Pinkas *et al.* [44] presented a PSI protocol designed to perform arbitrary computation on the intersection. While their protocol is highly efficient, it has some limitations in the types of computations that can naturally be performed. Namely, a significant overhead is introduced if the function being computed is sensitive to the order of the inputs. Despite this limitation, several interesting applications were considered such as threshold-PSI, which returns true or false based on whether the intersection size reaches a certain threshold; [44,20] consider PSI cardinality, which returns the size of the intersection. [39] considers a private friend-finding scenario. Ciampi and Orlandi [19] also design a protocol that can compute an arbitrary function on the intersection. One limitation all of these works have is that the communication complexity is at least linear in both set sizes, whereas our approach remains sublinear in the sender's set size.

1.5 PIR by Keywords

Private Information Retrieval (PIR) allows a user to retrieve an entry in some database held by one or more servers. A variant called *PIR by keywords* was considered by Chor *et al.* in [18], where the user query consists of a keyword instead of the address of the entry in the database. Our Labeled PSI protocol can be viewed as a multi-query (single-server) PIR by keywords: we regard the receiver's set Y as a set of keywords, and the set of sender's labels as the database.

In [23] Freedman *et al.* introduced a protocol for single-server PIR by keywords based on additive homomorphic encryption. Our labeled PSI protocol in Section 5 uses the same interpolation polynomial as they did. Meanwhile, using leveled FHE, our communication per keyword is $O(\sigma \log |X| + \ell)$, whereas [23] uses additive homomorphic encryption, and their communication is linear in the size of the entire database, namely $|X| \cdot \ell$. Our Labeled PSI protocol also handles multiple keywords so it can be viewed as an improvement upon their single keyword search protocol. Labeled PSI is also considered in [31], where the authors constructed a solution based on Diffie-Hellman type assumptions, with communication linear in the larger set.

Angel *et al.* [3,2] used multi-query single-server PIR by keywords as the core component of an anonymous communication protocol. In order to reduce PIR by keywords to PIR, they pushed a Bloom filter representation of the index-to-keyword map to each client. They also optimized for multi-query by using power-of-two choices and cuckoo hashing techniques.

Olumofin and Goldberg [41] used information-theoretical PIR by keywords to protect the privacy of client queries to a public database. Their reduction from PIR by keywords to PIR relies on B+ trees, and perfect hash functions.

Private queries to databases Wang *et al.* [51] used functional secret sharing instead of PIR to achieve the same goal of protecting the query privacy to a public database. They model the query as a function to be applied to the database. The client sends secret-shares of the function to multiple servers, and later combines the responses to obtain the result. Query privacy is ensured as long as at least one server does not collude with the others.

Boneh *et al.* [6] used FHE to allow private conjunction queries. The protocol returns indexes of the matching records to the client, who can subsequently issue PIR queries in order to get the records themselves. The communication is proportional to the size of the entire database.

Cheon *et al.* [14,15] presented techniques that allow a user to perform a private search-and-compute over a homomorphically encrypted database under the user's own key. Their approach supports different kinds of queries, such as search-and-sum, search-and-max, and join queries, making use of binary circuits for equality checks and comparisons.

Khedr *et al.* [33] constructed a secure email spam filter and a secure multi-keyword search from homomorphic encryption. Exploiting the parallelism by working on a GPU platform, they achieve good performance. In contrast to the other keyword search protocols, theirs has binary output: true if a set of keywords exist in the encrypted file, and false otherwise.

1.6 Summary of Notations

- X is the sender's set; Y is the receiver's set. We assume $|X| \gg |Y|$.
- σ is the length of items in X and Y.
- ℓ is the length of labels in Labeled PSI.
- n is the ring dimension in our FHE scheme (a power of 2); q is the ciphertext modulus; t is the plaintext modulus [22,21].
- d is the degree of the extension field in the SIMD encoding.

– m is the cuckoo hash table size.
– α is the number of partitions we use to split the sender's set X in the PSI protocol (following [12]).
– $[i, j]$ denotes the set $\{i, i+1, ..., j\}$, and $[j]$ is shorthand for the case $i = 1$.

2 The CLR17 Protocol

We now review the protocol of [12] in detail. Following the architecture of [43], their protocol instructs the receiver to construct a cuckoo hash table of its set Y. Specifically, the receiver will use three hash functions h_1, h_2, h_3, and a vector $B_R[0], \ldots, B_R[m]$ of $O(|Y|)$ bins. For each $y \in Y$, the receiver will place y in bin $B_R[h_i(y)]$ for some i such that all bins contain at most one item. The sender will perform a different hashing strategy. For all $x \in X$ and all $i \in \{1, 2, 3\}$, the sender places x in bin $B_S[h_i(x)]$. Note that each bin on the sender's side will contain $O(|X|/m)$ items with high proabibility when $|X| \gg m$. It then holds that the intersection of $X \cap Y$ is equal to the union of all bin-wise intersections. That is,

$$X \cap Y = \bigcup_j B_R[j] \cap B_S[j] = \bigcup_j \{y_j\} \cap B_S[j]$$

where y_j is the sole item in bin $B_R[j]$ (or a special sentinel value in the case that $B_R[j]$ is empty). The protocol then specifies a method for computing $\{y\} \cap B_S[j]$ using FHE. The receiver first sends an encryption of y, denoted as $[\![y]\!]$, to the sender who locally computes

$$[\![z]\!] := r \prod_{x \in B_S[j]} ([\![y]\!] - x)$$

When $y \in B_S[j]$, observe that one of the terms in the product is zero and therefore $[\![z]\!]$ will be an encryption of zero. $[\![z]\!]$ is returned to the receiver, who concludes that $y \in X$ if $z = 0$. In the case that $y \notin B_S[j]$, the product will be the product of differences. The sender randomizes this product using a uniformly sampled element $r \in \mathbb{F}^*$ for some finite base field $\mathbb{F}$ used to encode the items. As a result, $z = 0$ if and only if $y \in X$. Otherwise z is uniformly distributed and independent of the set X.

Building on this general protocol, [12] proposed several optimizations which make computing this circuit computationally efficient. First, recall that the receiver has a vector of $m = O(|Y|)$ bins, each containing (at most) a single element $y_1, ..., y_m$. Each y_j must be intersected with $B_S[j]$. FHE naturally supports a technique which allows encrypting vectors and performing Single Instruction Multiple Data (SIMD) operations on the encrypted vectors ([50]). In this way many of the items y_j can be encrypted into a single ciphertext and processed concurrently, which results in a significant performance improvement.

Despite this, computing $[\![z]\!] := r \prod_{x \in B_S[j]} ([\![y]\!] - x)$ directly was observed to be inefficient due to the performance penalty of using FHE to evaluate large degree polynomials on encrypted values. The multiplicative depth of directly computing z is $O(\log B)$ for $B \approx |X|/m$, and [12] reduced it to $O(\log \log B)$ using a windowing technique. Namely, observe that z can be viewed as a polynomial $P(y) = a_B y^B + ... + a_1 y + a_0$ where the a_i are determined by r and $B_S[j]$. The sender needs to compute encryptions of all the powers of y between 1 and B. Given an encryption of only y, this can be done in $O(\log B)$ depth using the square-and-multiply algorithm. However, the receiver can now assist in the computation by sending additional powers of y. For example, if the receiver sends encryptions of $y^{2^0}, y^{2^1}, y^{2^2}, ..., y^{2^{\log B}}$, the sender can use these terms to compute all necessary powers of y in multiplicative depth $O(\log \log B)$. They also partitioned the sender's bins into α subsets.

The sender can then process each of these subsets independently. This reduces multiplicative depth further to $O(\log\log\frac{B}{\alpha})$. The downside of this approach is that for each y several response ciphertexts $z_1, ..., z_\alpha$ must be sent back to the receiver, increasing the reply size by a factor of α. Finally, the authors used the modulus switching technique to reduce the return communication.

3 PSI with Long Items

The [12] protocol achieves good performance for 32-bit items and scales well to very large sets on the sender's side. However, it scales less well for longer items for the following reasons. Suppose the effective item length is σ bits. Then they need to set the plaintext modulus in the FHE scheme to $t \approx 2^\sigma$. Now let L denote the depth of the homomorphic evaluation at the sender's side. In [12] the depth L depends double-logarithmically on $|X|$. Hence for our purposes we assume L is a constant. Then, the BFV scheme requires $\log q \gtrsim L \log t$ for correctness, but using the complexity estimates in Section 1.2 we see that the communication cost of [12] grows linearly with σ; on the other hand, the computational cost grows *quadratically* with σ, which is undesirable.

Another side-effect of large σ comes from the security requirement: it drives up the FHE parameters t and q, and in order to keep the security level on par, the parameter n needs to increase as well. Now two cases can arise: if $|Y|$ is large compared to n then—since increasing n will increase the number of slots in each ciphertext—we end up using fewer ciphertexts, and the performance overhead is small; on the other hand, if $|Y|$ is of similar size as the previous n value then—after switching to new value of n—many slots could remain unused, which means the communication cost went up for no benefit.

Regardless of the initial length of the items, it is customary to apply a hash function of output length $\lambda + \log|X| + \log|Y|$, and perform the intersection protocol on these short hashes. Here λ denotes the statistical security parameter. In practice, we set $\lambda = 40$, thus we require t to be roughly 80 bits. However, using such a large value for t has a huge impact on the performance of the [12] protocol. In this work, we resolve this issue by using the general SIMD encoding method proposed in [50], which allows coefficient-wise operations on vectors of flexible length and width. More precisely, for a tunable parameter d, we can operate on vectors of length n/d, where each entry can take t^d different values (when $d = 1$, this is the SIMD method used in [12]).

More precisely, the plaintext space of BFV scheme equals $R_t = \mathbb{Z}_t[x]/(x^n + 1)$. Suppose t is a prime number, and $x^n + 1 \pmod t$ factors into a product of irreducible polynomials $\{F_j(x)\}$, each of degree d. Moreover, suppose d is the smallest positive integer such that $t^d \equiv 1 \pmod{2n}$. Then the SIMD encoding can be explained as the following two isomorphisms

$$R_t \xrightarrow{\phi} \prod_j \mathbb{Z}_t[x]/(F_j(x)) \xrightarrow{\psi} \prod_{i=1}^{n/d} \mathbb{F}_{t^d}$$

where $\mathbb{F}_{t^d}$ is a fixed finite field of t^d elements. Now SIMD encoding corresponds to $\phi^{-1} \circ \psi^{-1}$, and decoding is the isomorphism $\psi \circ \phi$.

3.1 Trade-Offs for SIMD with Wider Slots

Communication cost The concrete communication cost of our PSI protocol can be computed in the following way: the query consists of $m/k \cdot (\log(|X|/m))$ ciphertexts, where $k = n/d$ is the number of slots supported in a ciphertext, and $m = O(|Y|)$. Each ciphertext has size $2n \log q = 2nL \log t$ bits. Here $\log t \approx \sigma/d$, since every element in the finite field $\mathbb{F}_{t^d}$ can represent a $d \log t$-bit item. The reply from the sender consists of $m/k \cdot \alpha$ ciphertexts, each with size roughly $2n \log(tn)$ bits.[6]

[6] This is because we can perform the modulus switching trick to reduce the modulus to $q' \approx tn$.

We view $|Y|$ (and m), $|X|$, n, and L as constants. Then the query size is a constant multiple of $\log d + \log |X| - \log n$, so increasing d will increase the query size. However, in our setup we usually have $\log |X| \gg \log n$. Since $d < n$ always, the $\log |X|$ term dominates. The reply size is also slightly worse for larger d, but its effect is small.

Computational cost The sender's online computation consists of $\Theta(|X|/k)$ homomorphic multiplications, each taking $n \log n (\log q)^2$ bit operations. After hiding all the constants, we see that the computational cost is $\Theta(1/d)$, so increasing d has a direct positive effect on the online computing time.

Effect of choosing different n Using a larger value of d has another implicit benefit: it allows choosing a smaller t for the same item bit-length, hence one can choose a smaller q, which also opens the possibility of choosing a smaller n. We can use the above heuristics to analyze the effect of changing n. The computational cost is $|X|/d \log n \cdot L^2\sigma^2$ bit operations, and communication cost is $m(2d \log n + \sigma)\alpha + m \log(\frac{|X|d}{n}) \cdot 2L\sigma$, so both the computation and communication depend only logarithmically on the value of n. Therefore, the effect of n on performance is marginal when the other parameters are held fixed.

We conclude that under the same setup, using wider slots (i.e. a larger value of d) to encode items results in larger communication and smaller on-line computation.

3.2 Lazy SIMD Encoding Algorithm

[50] suggested that an FFT algorithm can be utilized for fast SIMD encoding and decoding. The FFT algorithm is very efficient when d is small, but for larger values of d there exist more efficient algorithms. For example, the HElib library [27] performs the encoding in two steps. Let $\mathbb{F}_j = \mathbb{Z}_t[x]/(F_j)$. They first map $(\mathbb{F}_{t^d})^k$ to $\mathbb{F}_1 \times \cdots \times \mathbb{F}_k$ through k field isomorphisms. Then a tree-based CRT algorithm is used to invert the first map ϕ: given $f_j(x) \in \mathbb{Z}_t[x]/(F_j)$, return $f \in \mathbb{Z}_t[x]/(F)$, such that $f_j = f \mod F_j$.

We make the observation that the second map ψ can sometimes be omitted in the encoding. Indeed, it is necessary if one wishes to homomorphically permute items in the $\mathbb{F}_{t^d}$ slots. Since our current application does not require such permutations, we could skip this step and solely use ϕ and its inverse for decoding and encoding. This saves computation time as well as storage, since evaluating ψ requires the information of the isomorphisms $\mathbb{F}_j \to \mathbb{F}_{t^d}$.

On the other hand, the FFT algorithm can perform $\psi \circ \phi$ in one step, and there are fast algorithms that work with the $x^n + 1$ modulus. However, a natural way to utilize FFT in this scenario seems to require working with data of length n, using the $2n$-th roots of unity in $\mathbb{F}_{t^d}$. The complexity of such an algorithm is $O(n \log n)$ operations in $\mathbb{F}_{t^d}$.

To determine the optimal strategy, we performed a comparison by implementing both the lazy encoding algorithm and the FFT algorithm using FLINT [28] and present the results in Figure 1. From the results we can see that the lazy encoding algorithm has a speed advantage which grows with the extension degree d.

4 OPRF Pre-processing

We now demonstrate how a pre-processing phase can be performed to facilitate a more efficient online phase. The core idea is to first update the values being intersected using an oblivious PRF, where only the sender knows the key. This has the effect that several costly countermeasures employed by [12] to protect the senders set can be eliminated.

$\log n$	t	d	FFT time (ms)	Lazy SIMD time (ms)
11	0x2E01	8	3.3	1.5
12	0x13FF	8	12	4
13	0x3401	16	32.6	9.2
14	0x2A01	64	512	23.5

Table 1. Comparison of SIMD encoding algorithms.

4.1 The CLR17 Approach

The [12] protocol performs noise flooding to prove the security for the sender. The need for this stems from the fact that noise growth in homomorphic operations depends not only on the ciphertexts being operated on, but also on the underlying plaintexts. Thus, their security proof cannot work if the result ciphertexts are not re-randomized at the end, and if the underlying noise distribution is not hidden by flooding the noise by an appropriate number of bits.

There are at least two different problems with this approach. First, it requires the sender to estimate a heuristic upper bound on the size of noise, and ensure that there is enough noise room left to perform an appropriate amount of noise flooding. This makes it impossible to run their protocol with small FHE parameters, even for very small sets. Also, their protocol is fragile against malicious attacks. For example, the receiver can insert more noise into its ciphertexts, causing the sender to noise-flood by fewer bits than it thinks. Now, by examining the noise distribution after the PSI computation, the receiver can potentially obtain extra information about the sender's set.

4.2 Our Solution

We take a different approach to solving this problem, allowing us to get rid of noise flooding altogether. Namely, we use an OPRF to hash the items on both sides before engaging in the PSI protocol. This ensures that the sender's items $X \setminus Y$ are pseudo-random in the receiver's view, preventing the receiver from learning anything about the original items, even if it learns the hashed values in full.

Abstractly, we have the sender sample a key k and instruct it to locally compute $X' = \{F_k(x) \mid x \in X\}$. The receiver then interactively applies the OPRF to its set, obtaining $Y' = \{F_k(y) \mid y \in Y\}$. From a security perspective it is now safe to send X' to the receiver, who can infer the intersection from $X' \cap Y'$. However, this approach incurs a very high communication overhead, since X' can easily be over a gigabyte. Several recent works [44,47] try to tackle this problem by encoding X' in a compressed format, e.g. a Bloom filter or a cuckoo filter. However, the communication remains linear in the set $|X|$, and the compression can introduce false positives such as in the case of [47].

Our approach sidesteps this issue by applying an FHE-based PSI protocol to the sets X' and Y'. Overall, the communication complexity of our protocol is $O(|Y| \log |X|)$, as opposed to $O(|Y|+|X|)$ in the case of e.g. [44,47]. As previously described, we do not need to worry about noise flooding unlike [12], since the OPRF already provides sufficient protection. This allows our protocol to use FHE parameters that are highly optimized, improving our performance and communication overhead.

More broadly, applying the OPRF to the sets also eliminates the need to perform two other procedures which protect the sender's set. First, recall that the sender performs simple hashing where its $|X|$ items are mapped to $O(|Y|)$ bins using three hash functions. In the original [12] protocol all of these bins must then be padded with dummy items to an upper bound. This prevents some partial information from being leaked to the receiver, e.g. m items hash to bin i, which implies

that $|\{x \in X \mid h(x) = i\}| = m$. However, in the case that the OPRF is applied, the number of items in any given bin is a function of X', and therefore can be made public.

Secondly, the polynomials that the sender evaluates using the receiver's set need not be randomized. Recall that in [12] the sender evaluates homomorphically a polynomial of the form $F(y) = r\prod_{x\in X}(y-x)$, where $r \leftarrow \mathbb{F}^*$ is sampled uniformly at random each time the protocol is executed. This additional randomization was required to ensure that the receiver does not learn the product of differences between y and X. It also has a significant impact on performance, as it increases the multiplicative depth by one. However, after the OPRF is applied, this polynomial is formed with respect to X'—not X—and therefore revealing the product of differences is no longer a security risk, since X' can securely be made public.

We consider two types of OPRFs which have different trade-offs. The first is a Diffie-Hellman based OPRF described by [31,47], which allows the sender to reuse OPRF values with many independent receivers, allowing for the cost of applying the OPRF to the sender's set to be amortized. Alternatively, an oblivious transfer based OPRF [42,36] case be used, which is computationally more efficient, but can not be reused across several receivers.

4.3 DH-OPRF Pre-Processing

The Diffie-Hellman based OPRF protocol of [31] computes the function $F_\alpha(x) = H'(H(x)^\alpha)$, where H is a hash function modeled as a random oracle. This style of OPRF has been used several times in the context of PSI, e.g. in [38,23,31,47]. In more detail, let G be a cyclic group with order q, where the One-More-Gap-Diffie-Hellman (OMGDH) problem is hard. H is a random oracle hash function with range $\mathbb{Z}_q^*$. The sender has a key $\alpha \in \mathbb{Z}_q^*$ and the receiver has an input $x \in \{0,1\}^*$. The receiver first samples $\beta \leftarrow \mathbb{Z}_q^*$ and sends $H(x)^\beta$ to the sender, who responds with $(H(x)^\beta)^\alpha$. The receiver can then output $H'(H(x)^\alpha) = H'(((H(x)^\beta)^\alpha)^{1/\beta})$. The outer hash function H' is used to map the group element to a sufficiently long bit string, and helps facilitate extraction in the malicious setting.

In particular, by observing the queries made to $H(x_i)$, the simulator can collect a list of pairs $\{(x_i, H(x_i))\}$ which are known to the receiver. From this set the simulator can compute the set $A = \{(x_i, H(x_i)^\alpha)\}$. For some subset of the $H(x_i)$, the receiver sends $\{H(x_i)^{\beta_i}\}$ to the simulator, who sends back $\{H(x_i)^{\beta_i\alpha}\}$. For the receiver to learn the OPRF value for x_i, it must send $H(x_i)^\alpha$ to the random oracle H'. At this time the simulator extracts x_i if $(x_i, H(x_i)^\alpha) \in A$. Although this OPRF does not facilitate extracting all x_i at the time the first message is sent, extraction is performed before the receiver learns the OPRF value, which will be sufficient for our purposes.

In the context of our PSI protocol, this OPRF has the property that the sender can use the same key with multiple receivers. This allows the sender, who has a large and often relatively static set, to pre-process its set only once. This is particularly valuable since our protocol also allows for efficient insertions and deletions of data from the pre-processed set.

4.4 OT-OPRF Pre-Processing

An alternative approach is to use recent advances in Oblivious Transfer (OT) extension protocols [36,42], which enable a functionality very similar to a traditional OPRF. The relevant difference is that the receiver can only learn one OPRF output per key. This restriction mandates that the OPRF be used in a different way. In particular, we follow the PSZ paradigm [45,43,46,36], where the OPRF is applied to the items after cuckoo hashing. First the parties perform cuckoo hashing, which ensures that the receiver has at most one item per hash table bin. The parties then run an OT-based OPRF protocol, where the sender assigns a unique key to each bin. The receiver updates

the values in the cuckoo table with the OPRF outputs, while the sender similarly updates the values in its simple hash table. Note that the sender can learn an arbitrary number of OPRF outputs per key, which allows it to update all the values in each bin.

Another restriction with this approach is that the sender must pad its simple hash table with dummy items to ensure that the receiver does not infer any partial information. That is, since the OPRF is applied after hashing, the number of items in any given bin is a function of X instead of the hashed set X'. It is therefore critical that the bins be padded to their maximum possible size, as was done in [12].

As with the Diffie-Hellman based OPRF, the receiver's input can be extracted. The exact details how how these protocols extract is quite involved, and we defer to [42] for a detailed explanation.

5 Labeled PSI

We present two related approaches for enabling Labeled PSI. The first is compatible with the [12] protocol, while the latter is optimized to take advantage of the OPRF pre-processing phase. This section will be presented in terms of the receiver with a singleton set $\{y\}$ and obtaining the label ℓ_i if and only if the sender has a pair (x_i, ℓ_i) in its set for which $y = x_i$. The approaches naturally extend to the general setting by using cuckoo hashing on the receiver's side.

5.1 Compatible With CLR17

We employ an idea of interpolation polynomials, also used in [23], to build our labeled PSI protocol. Recall that in the [12] protocol the server homomorphically evaluates the polynomial $F(y) = r^* \prod_{x \in X}(x - y)$, where r^* is a random nonzero element in some finite field $\mathbb{F}$, and the coefficients of the polynomial $F(y)$ are elementary symmetric polynomials in the $x \in X$. It has the property that if $y \in X$, then $F(y) = 0$; otherwise $F(y)$ is a random element in $\mathbb{F}^*$. In the Labeled PSI case, the sender's input is a list of pairs $\{(x_i, \ell_i)\}_{i=1}^{D}$, where for simplicity we assume x_i and ℓ_i are elements of $\mathbb{F}$. Our goal is to construct a polynomial G, such that for any $y \in \mathbb{F}$

$$G(y) = \begin{cases} \ell_i & \text{if } y = x_i; \\ \text{random element in } \mathbb{F} & \text{otherwise.} \end{cases}$$

Note that there exists a polynomial $H(x)$ of degree less than $D = |Y|$, such that $H(y_i) = \ell_i$ for all $1 \leq i \leq D$. Then, we select $r \in \mathbb{F}$ randomly, and let

$$G(y) = H(y) + rF(y).$$

It is easy to verify that G has the desired property: if $y = x_i$, then $G(y) = H(x_i) = \ell_i$; if $y \notin X$, then since $F(y) \neq 0$ and r is random, we know that $G(y)$ is a random element in $\mathbb{F}$. At a high level, our protocol has the receiver encrypt and send each of its items y using the FHE scheme; the sender evaluates the polynomials G and F homomorphically, and sends the results back. The receiver then decrypts and obtains $(F(y), G(y))$, which is either equal to $(0, p_i)$ if $y = x_i$, or uniformly random in $\mathbb{F}^* \times \mathbb{F}$.

Note that in the above discussion, we implicitly assumed that labels are of the same size as the items. In case the labels are longer than the items, we can break down each label in chunks, and have the server repeat its computation several times. Finally, the receiver can decrypt the parts of the label and re-assemble. The security proof is not affected, because in the case of a non-match, all the decrypted parts will be random strings.

Communication complexity We utilize the optimizations in [12], with the modification that the sender homomorphically evaluates several polynomials instead of one. Hence, the communication complexity for our Labeled PSI is equal to $O(|Y| \log |X| \sigma + |Y| \ell)$, and the online computation complexity is $O(|X| \ell)$, where σ and ℓ denote the lengths of items and labels, respectively. Note that by hashing the items beforehand, we can assume $\sigma = \lambda + \log |X| + \log |Y|$, where λ is the statistical security parameter.

Computational complexity This Labeled PSI protocol introduces two additional computational tasks on top of the PSI protocol of [12]. In the offline phase, the sender needs to interpolate a polynomial of degree $B' \approx \frac{|X|}{m\alpha}$. The Newton interpolation algorithm has complexity $O(B'^2)$, and is fast for small values of B'. The algorithm needs to be executed $m\alpha$ times, so the total complexity is $O(\frac{|X|^2}{m\alpha}) \cdot \ell/\sigma$. In the online phase, the sender needs to evaluate the interpolated polynomials homomorphically, which has a cost of $O(\frac{|X||Y|}{m^2} \cdot \ell/\sigma)$ FHE operations. Compared to the complexity of the PSI protocol in [12], the sender's computation of labeled PSI grows by a factor of ℓ/σ, i.e. by the ratio of the label length and the item length.

5.2 OPRF Optimized Labeled PSI

If the parties perform the OPRF pre-processing phase, this procedure can be significantly improved. The core idea is to first encrypt all of the labels using the associated OPRF values as the key. All of these encrypted labels can then be sent to the receiver, who uses the OPRF values for the items in its set to decrypt the associated labels. We stress that this approach requires no security guarantees from the homomorphic encryption scheme to ensure that information is not leaked to the receiver about labels for items not in the intersection.

To avoid linear communication when sending these encrypted labels, we have the sender evaluate a polynomial which interpolates the encrypted labels, effectively compressing the amount of data that needs to be communicated. In more detail, the sender first computes $(x'_i, x''_i) = OPRF_k(x_i)$ for all x_i in its set X. Here, x'_i will be used as the OPRF value for computing the intersection as before, while the second part x''_i will be used to one-time-pad[7] encrypt the label as $\ell'_i = \ell_i + x''_i$. The sender then computes a polynomial G with minimal degree such that $G(x'_i) = \ell'_i$.

One of the main advantages of this approach is that the degree of the online computation is reduced due to G not requiring additional randomization. Recall from above that the result of evaluating the symmetric polynomial F has to be randomized by multiplying with a nonzero r. This increases the degree of the computation by one, which can require larger FHE parameters.

Looking forward, our approach for improved security against a malicious sender requires the use of Labeled PSI, but interestingly does not require the evaluated symmetric polynomial $F(y)$ to be sent back to the receiver. As such, by not requiring F in the computation of the labels, we gain an additional performance improvement in the malicious setting by not computing or evaluating F.

5.3 Full Protocol

We present our full protocol for labeled PSI in Figure 2.

[7] Note that x''_i can be extended to an arbitrary size ℓ_i using a PRG.

Input: Receiver inputs set $Y \subset \{0,1\}^*$ of size N_{Y}; sender inputs set $X \subset \{0,1\}^*$ of size N_{X}. $N_{\text{X}}, N_{\text{Y}}$ are public. κ and λ denote the computational and statistical security parameters, respectively.
Output: The receiver outputs $Y \cap X$; the sender outputs $\perp$.

1. **[Sender's OPRF]** The sender samples a key k for the [31] OPRF $F : \{0,1\}^* \to \{0,1\}^\kappa$. The sender updates its set to be $X' = \{H(F_k(x)) : x \in X\}$. Here H is a random oracle hash function with a range of $\sigma = \log_2(N_{\text{X}} N_{\text{Y}}) + \lambda$ bits which is sampled using a coin flipping protocol.
2. **[Hashing]** The parameter m is agreed upon such that cuckoo hashing N_{Y} balls into m bins succeed with probability $\geq 1 - 2^{-\lambda}$. Three random hash function $h_1, h_2, h_3 : \{0,1\}^\sigma \to [m]$ are agreed upon using coin flipping. The sender inserts all $x \in X'$ into the sets $\mathcal{B}[h_1(x)], \mathcal{B}[h_2(x)], \mathcal{B}[h_3(x)]$.
3. **[Choose FHE parameters]** The parties agree on parameters (n, q, t, d) for an IND-CPA secure FHE scheme. They choose t, d to be large enough so that $d \log_2 t \geq \sigma$.
4. **[Choose circuit depth parameters]** The parties agree on the split parameter $B < O(|Y|/m)$ and windowing parameter $w \in \{2^1, 2^2, \ldots, 2^{\log_2 B}\}$ as to minimize the overall cost.
5. **[Pre-Process X]**
   (a) **[Splitting]** For each set $\mathcal{B}[i]$, the sender splits it into α subsets of size at most B, denoted as $\mathcal{B}[i,1], \ldots, \mathcal{B}[i,\alpha]$.
   (b) **[Computing Coefficients]**
      i. **[Symmetric Polynomial]** For each set $\mathcal{B}[i,j]$, the sender computes the symmetric polynomial $S_{i,j}$ over $\mathbb{F}_{t^d}$ such that $S_{i,j}(x) = 0$ for $x \in \mathcal{B}[i,j]$.
      ii. **[Label Polynomial]** If the sender has labels associated with its set, then for each set $\mathcal{B}[i,j]$, the sender interpolates the polynomial $P_{i,j}$ over $\mathbb{F}_{t^d}$ such that $P_{i,j}(x) = \ell$ for $x \in \mathcal{B}[i,j]$ where ℓ is the label associated with x.
   (c) **[Batching]**
   View the polynomials $S_{i,j}$ as a matrix where i indexes the row. For each set of n/d rows (non-overlapping and contiguous), consider them as belonging to a single *batch*. For b-th batch and each j, take the k-th coefficient of the n/d polynomials, and batch them into one FHE plaintext polynomial $\overline{S}_{b,j,k}$.
   For Labeled PSI, perform the same batching on the label polynomials $P_{i,j}$ to form batched FHE plaintext polynomials $\overline{P}_{b,j}$.

Fig. 1. Full Labeled PSI protocol (sender's offline pre-processing).

6 PSI with Computation

A recent trend in PSI protocols has been to enable additional computation on the intersection [44,19,20,30]. However, in our setting where the set sizes are extremely unbalanced these protocols impose (at least) a linear communication overhead. We show that our protocol can be extended to output the intersection along with the associated labels in a form suitable for further computation. With these techniques, both parties can input a set of key-value pairs $\{(x_i, \ell_i)\}_i, \{(y_j, t_j)\}_j$ where the values $\{(\ell_i, t_j)\}$ with matching keys $x_i = y_j$ are the input to the function

$$\mathsf{f}(\{(\ell_i, t_j)\})$$

The communication complexity of this computation is $O(|Y| \log |X| + |\mathsf{f}|)$ where $|\mathsf{f}|$ is the communication complexity of computing f on $|Y|$ inputs. This deviates from other techniques such as [44,19], which require that the communication complexity be at least $O(|Y| + |X| + |\mathsf{f}|)$. In the case that $|X| \gg |Y|$, this represents a significant saving. Moreover, in our protocol the receiver's work is proportional to $O(|Y| \log |X|)$, which can be of practical importance when the receiver is a lightweight device, e.g. a cellphone.

The core idea is to return an additive secret share of the result, which can be forwarded as input to a secondary MPC protocol. For simplicity, first let us consider the case without labels and where our PSI protocol is performed using a single split. In the standard PSI protocol having one

7. [**Encrypt** Y]
   (a) [**Receiver's OPRF**] The receiver performs the interactive OPRF protocol of [31] using its set Y in a random order as private input. The sender uses the key k as its private input. The receiver learns $F_k(y)$ for $y \in Y$ and set $Y' = \{H(F_k(y)) : y \in Y\}$.
   (b) [**Cuckoo Hashing**] The receiver performs cuckoo hashing on the set Y' into a table $\mathcal{C}$ with m bins using h_1, h_2, h_3 has the hash functions.
   (c) [**Batching**] The receiver interprets $\mathcal{C}$ as a vector of length m with elements in $\mathbb{F}_{t^d}$. For the bth set of n/d (non-overlapping and contiguous) in $\mathcal{C}$, the receiver batches them into a FHE plaintext polynomial $\overline{Y}_b$.
   (d) [**Windowing**] For each $\overline{Y}_b$, the receiver computes the component-wise $i \cdot w^j$-th powers $\overline{Y}_b^{i \cdot w^j}$, for $1 \leq i \leq w-1$ and $0 \leq j \leq \lfloor \log_w(B) \rfloor$.
   (e) [**Encrypt**] The receiver uses $\mathsf{FHE.Encrypt}$ to encrypt each power $\overline{Y}^{i \cdot 2^j}$ and forwards the ciphertexts $c_{i,j}$ to the sender.
8. [**Intersect**] For the bth batch,
   (a) [**Homomorphically compute encryptions of all powers**] The sender receives the collection of ciphertexts $\{c_{i,j}\}$ and homomorphically computes a vector $\mathbf{c} = (c_0, \ldots, c_B)$, such that c_k is a homomorphic ciphertext encrypting $\overline{Y}_b^k$.
   (b) [**Homomorphically evaluate the dot product**] For each $\overline{S}_{b,1}, ..., \overline{S}_{b,\alpha}$, the sender homomorphically evaluates

$$z_{b,j} = \sum_{k=0}^{B} c_k \cdot \overline{S}_{b,j,k}$$

   and optionally performs modulus switching on the ciphertexts $z_{b,j}$ to reduce their sizes. All $z_{b,j}$ are sent back to the receiver. If Labeled PSI is desired, repeat the same operation for $\overline{P}$ and denote the returned ciphertexts $q_{b,j}$.
9. [**Decrypt and get result**] For the b-th batch, the receiver decrypts the ciphertexts $z_{b,1}, ..., z_{b,\alpha}$ to obtain $r_{b,1}, ..., r_{b,\alpha}$, which are interpreted as vectors of n/d elements in $\mathbb{F}_{t^d}$.
   Let $r_1^*, ..., r_\alpha^*$ be vectors of m elements in $\mathbb{F}_{t^d}$ obtained by concatenating $r_j^* = r_{1,j}^* || ... || r_{md/n,j}^*$. For all $y' \in Y'$, output the corresponding $y \in Y$ if

$$\exists j : r_j^*[i] = 0\,,$$

   where i is the index of the bin that y' occupies in $\mathcal{C}$.
   If Labeled PSI is desired, perform the same decryption and concatenation process on the $q_{b,j}$ ciphertexts to obtain the m element vectors $\ell_1^*, ..., \ell_\alpha^*$. For each $r_j^*[i] = 0$ above, output the label of the corresponding y to be $\ell_j^*[i]$.

Fig. 2. Full Labeled PSI protocol continued (online phase).

split means that after decryption the receiver holds a single vector of results $\boldsymbol{z}$, where z_i is zero if the item in the i-th position of its cuckoo table is in the intersection.

Instead of returning z_i, the sender returns a ciphertext $z_i' = z_i + r_i$, where r_i is a vector of uniformly random values. When the receiver decrypts the ciphertext, it will hold of 2-out-of-2 secret sharing of z_i, which can be forwarded to a secondary MPC protocol and be checked for zero. For example, the cardinality of the intersection can be computed by the circuit for

$$\mathsf{cardinality}(\boldsymbol{z}', \boldsymbol{r}) = \sum_i \mathbb{1}(z_i' \stackrel{?}{=} r_i)$$

In practice, we find that our protocol obtains improved performance by utilizing more than one split. This has the the effect that for each position i of the receiver's cuckoo table the sender returns s results $z_{i,1}, \ldots, z_{i,s}$, from which the receiver concludes that its item at cuckoo position i is in the intersection if $0 \in \{z_{i,1}, \ldots, z_{i,s}\}$. As a result, the cardinality can be computed as $\sum_{i,j} \mathbb{1}(z_{i,j}' \stackrel{?}{=} r_{i,j})$.

We note that while the main phase of the PSI protocol can benefit from a modest size $s = O(\log |X|)$, other parameters, e.g. the window size, can be increased to ensure s is a constant such as 4 or even 1.

In addition to simply computing on the intersection, it is possible to return secret shares of the labels. When the same basic secret sharing based approach is applied to a label ℓ held by the sender, the receiver will decrypt the share $\ell' = \ell + r$. We note that its is straightforward for the receiver to provide associated data for each item in its set. This data can simply be stored with each item in the cuckoo table, and appropriately provided as input to the MPC protocol.

In the case that the *sender* receives the output of the function f, it is important that the computation performed by f is *symmetric*. This means that the ordering of the inputs does not impact the computation. This requirement is shared with [44], and stems from the use of cuckoo hashing. In particular, the inputs to f are ordered by the receiver's cuckoo hash table, which is a function of its private input. Therefore, the output of f could reveal private information if the function is not symmetric. We note that this symmetry requirement is not needed when the receiver obtains the output of f since it already knows the ordering.

7 Malicious Security

We now show that our protocol is secure against a malicious receiver, while providing privacy [29] against a malicious sender. The ideal functionality is presented in Figure 3. Observe that the malicious party is allowed to have a larger set than if they were honest. The exact set sizes will be discussed in the proof.

Parameters: The honest sender and receiver have respective set sizes $N_{\text{X}}, N_{\text{Y}}$. If the sender or receiver is maliciously corrupt, then their set size is N'_{X} or N'_{Y} respectively.

- On input (RECEIVE, sid, Y) from the receiver where $Y \subset \{0,1\}^*$, ensure that $|Y| \leq N_{\text{Y}}$ if the receiver is honest and $|Y| \leq N'_{\text{Y}}$ otherwise. Give (RECEIVER-INPUT, sid) to the sender.
- Thereafter, on input (SEND, sid, X) from the sender where $Y \subset \{0,1\}^*$, ensure that $|X| \leq N_{\text{X}}$ if the sender is honest and $|X| \leq N'_{\text{X}}$ otherwise. Give output (OUTPUT, $\mathsf{sid}, X \cap Y$) to the receiver.

Fig. 3. Ideal PSI functionality.

7.1 One-Sided Simulation

Roughly speaking, to prove that the protocol π of Figure 2 has one-sided simulation [29] we must show that for all malicious PPT receivers $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that the interaction of $\mathcal{A}$ in the real protocol π is indistinguishable from the output of $\mathcal{S}$ when interacting with the ideal functionality of Figure 3. More formally, consider Theorem 1.

Theorem 1. *In the random oracle hybrid and in the presence of an OMGDH-hard group G, the protocol π of Figure 2 securely computes the PSI functionality (Figure 3) with one-sided simulation security [29, Definition 2.2] where $N'_{\text{Y}} = N_{\text{Y}}$.*

Proof. First, observe that steps 1 through 5 do not involve the receiver and therefore are trivial to simulate. At Step 7a the receiver performs N_{Y} OPRF invocations in the group G where the simulator $\mathcal{S}$ plays the role of the sender. In Step 7e the receiver homomorphically encrypts its queries, and sends it to the sender. At this point $\mathcal{S}$ uses the random oracle to extracts the receiver's input Y^*

from the N_Y OPRF invocations (see Section 4.3). Y^* is forwarded to the ideal functionality, which response with $X^* = X \cap Y^*$. The simulator pads X^* with random values not in Y^* to the size of N_X and then performs steps 1 through 5 using X^* as would be done by the honest sender with X. $\mathcal{S}$ completes the protocol (Step 8) playing the role of the honest sender.

The correctness of this simulator directly follows the proof of [31]. In particular, all OPRF values apart from the N_Y which were extracted are uniformly distributed in the receiver's view. As such, padding X^* with values not in Y^* induces the same distribution. Compared to [31], the primary difference in our protocol is the use of FHE to compress the amount of communication.

7.2 Receiver Privacy

Achieving full simulation-based security against a malicious sender is extremely challenging in our setting. Arguably, the most significant barrier is that we require the communication complexity to be *sublinear* in the size of the sender's set. This makes traditional methods for extracting their set, e.g. ZK proofs [31], not viable due to the associated communication overhead being linear.

The second biggest challenge is enforcing that the sender performs the correct computation. In our protocol, the sender can deviate from the prescribed protocol in numerous ways. For instance, it is well known [23,48] that the sender can incorrectly perform simple hashing, which allows the receiver's output distribution to depend on the receiver's full set. Moreover, in our protocol, the situation is even worse in that the sender obtains a homomorphically encrypted copy of the receiver's set. Instead of using it to evaluate our PSI circuit, the sender has a large degree of freedom to compute a different circuit, which may arbitrarily depend on the receivers set.

For example, it is trivial for the sender to force the receiver to output their full set. Recall that in [12], and in our semi-honest protocol, when the returned ciphertext contains a zero, the receiver interprets this to mean their corresponding item is in the intersection. A straightforward attack for the sender would then be to simply encrypt and return a vector of zeros if it holds a public key, or otherwise return an all-zero ciphertext.[8] The receiver would then conclude that its full set is the intersection.

Faced with these significant challenges, we choose to forgo simulation based security when in the presence of a malicious sender. Instead, we show that (1) our basic protocol achieves privacy[29] against a malicious sender; (2) a simple modification of our protocol can significantly restrict the class of attacks that a malicious sender can perform.

This notion of privacy against a malicious sender and full security against the receiver has previously been considered in the context of PSI by Hazay and Lindell [29]. Informally, privacy guarantees that the transcript of the sender reveals nothing about that receiver's input. This is a relaxation of full simulation based security in that, for example, we do not achieve independence of inputs, nor guarantee output correctness. For a formal description, we refer the reader to [29, Definition 2.2].

Theorem 2. *In the presence of semantic secure fully homomorphic encryption scheme and an OMGDH-hard group G, the protocol π of Figure 2 computes the PSI functionality (Figure 3) with privacy [29, definition 2.2] against a malicious sender.*

Proof. The receiver sends two messages to the sender. First are the OPRF queries with the form $H(y)^\beta$ where $\beta \in \mathbb{Z}_q^*$ is sampled uniformly at random. By the assumption that OMGDH is a

[8] The homomorphic encryption library used may or may not reject such a ciphertext as invalid. If it does, the sender can still multiply any ciphertext obtained from the receiver by a nonzero plaintext mask that sets all but one vector slot value to zero, leading the receiver to interpret all but one item as matching.

hard problem in the group G, replacing y with a uniformly random element is indistinguishable and therefore these messages are independent of the receiver's set. The second set of messages sent by the receiver are the FHE encrypted powers which will be used to evaluate the symmetric polynomial. By the assumption that the FHE scheme is semantically secure, replacing the powers with uniformly random values is indistinguishable and therefore these messages are also independent of the receivers set.

7.3 Two-Sided Simulation with Leakage

We now present a technique for restricting the types of attacks the sender can perform. The core idea is inspired by Labeled PSI combined with the fact that evaluating a high-depth circuit using leveled FHE which supports much smaller depth is difficult—if not infeasible. First we consider the setting where the sender does not reuse the result of the pre-processing phase with multiple receivers.

For the receiver to conclude that an item y is in the intersection, we require that the sender return an encryption of the label $z = H(\mathrm{OPRF}_k(y))$, where H is a sufficiently complex hash function which we model as a random oracle, e.g. SHA256. Due to H being a random oracle the sender has two options for computing z: (1) it must know some $x = y$ and locally compute $z = H(\mathrm{OPRF}_k(x))$; (2) it must use the receiver's encrypted set of $\{\mathrm{OPRF}_k(y)\}$ to homomorphically evaluate a circuit that computes H using the given leveled FHE scheme under the fixed encryption parameters that the two parties have agreed to use.

We heuristically argue that evaluating such a circuit is extremely difficult—if not infeasible—using leveled FHE, where the parameters are chosen to support a much smaller multiplicative depth. While there is no guarantee that a high degree polynomial cannot be evaluated, we experimentally find that for our parameters the noise level always overflows when even a few extra multiplications are performed. In addition, hash functions such as SHA256 have very high depth and seem extremely difficult to evaluate using arithmetic FHE schemes due to the switching between binary and modular arithmetic (in SHA256). Moreover, the depth of H can be increased arbitrarily by repeatedly applying the hash function, and the number of repetitions can be decided after the encryption parameters have been selected. To encapsulate this assumption we define the notion of *F-limited*.

Definition 1. *We say a leveled fully homomorphic encryption scheme Σ is F-limited if for all F^* the probability of the following game outputting 1 is negligible. Uniformly sample an z and output 1 if $\Sigma.\mathsf{Dec}_{sk}(F^*(\Sigma.\mathsf{Enc}_{pk}(z))) = F(z)$ and otherwise output 0.*

Under the assumption that the given (leveled) FHE scheme is H-limited, the sender must know the plaintext value $\mathrm{OPRF}_k(y)$ to be able to compute an encryption of $H(\mathrm{OPRF}_k(y))$ with non-negligible probability. As such, the sender is restricted to choosing a set X' of polynomial size, possibly larger than N_{X}, and using this in the PSI protocol. In particular, the simulator can extract X' as all x such that the sender has queried $H(\mathrm{OPRF}_k(x))$. We note that the sender is not committed to its set X', and can use an arbitrary subset of it in each protocol invocation.

On the other hand, the sender is still able to make the intersection indirectly depend on the set $Y \setminus X$. In particular, the sender can choose a circuit $\mathsf{leakage}(\cdot)$ and the ideal functionality is defined to be

$$X \cap Y \cap \mathsf{leakage}(Y)\,,$$

This leakage function models the fact that the sender can perform some malicious computation with the receiver's encrypted hash table. This allows the sender to conditionally remove items from the intersection based on items which are in $Y \setminus X$. See Appendix A for the full security proof.

While such an attack can be serious in some settings, this leakage is significantly less than in [12], where the sender can force the intersection to be Y.

Now we turn our attention to the setting where the sender reuses the pre-processing phase with multiple receivers. Here the hashing parameters have to be fixed and reused. In particular, the cuckoo hash function and the hash function H used to hash to σ bit strings are picked by the sender. This deviates from Step 1 and 2 of Figure 2, where the parties jointly sample random hash functions. This has the implication that the sender can select hash functions which conditionally fail based on the receiver's set Y. For example, if $y, y' \in Y$ both hash to cuckoo position i under all three hash function, cuckoo hashing will fail. This class of failures would be observable by the sender, and leak that the receiver's set is a member of all the sets which fail to cuckoo hash under these parameters—a single bit of information.

While such attacks can be serious, we argue that many applications can tolerate leaking a single bit. One countermeasure which can be employed is to sample the hash functions from a public reference string. This could significantly restrict the effectiveness of such selective failure attacks, as the hash functions are then fixed.

8 Experiments

We implemented our protocols (unbalanced PSI for arbitrary length items, and Labeled PSI) and benchmarked against previous methods. For unbalanced PSI with both long and short items our points of comparison are [47,5] and [12] respectively. For labeled PSI, we compare with multi-PIR by keyword from the multi-query SealPIR solution of [2].

Our implementation is built from scratch on top of the homomorphic encryption library SEAL v2.3.0-4, which is based on the BFV scheme [22]. We give a detailed report of the end-to-end and online running times along with the communication overhead of our protocol in Figure 4, both in single and multi-threaded settings. We restrict the receiver to at most 4 threads to model a low power device, while the sender utilizes up to 32 threads as denoted in the table. Figure 5 shows a comparison with the unbalanced PSI protocols of [47,5,12].

We benchmark the protocols on a 32-core Intel Xeon CPU with 256 GB of RAM. We note that this machine is similar to that utilized by [12], and that the numbers reported for their protocol are obtained directly from their paper. All protocols are ran in the LAN setting with a 10 Gbps throughput, and sub-millisecond latency.

8.1 Improved Communication from Symmetric-Key BFV

We further improve our communication cost by modifying SEAL a little bit to use a symmetric-key variant of the BFV scheme instead of the more common public-key variant. The benefit of this is that the second polynomial in a freshly encrypted BFV ciphertext (see [22]) does not have to depend on a public key and can thus be generated from a random seed. As a result, the size of every freshly encrypted ciphertext is nearly halved, significantly reducing our $R \rightarrow S$ communication. Once the sender receives the half-size ciphertexts it can expand the seeds to obtain the full ciphertexts. Unfortunately, after homomorphic multiplications there is really no way to go back to the half-size representation, so this technique cannot improve the $R \leftarrow S$ communication. We can also employ the various trade-offs in the protocol to put more communication in the $R \rightarrow S$ step which then is halved, resulting in a significant $(20 - 40\%)$ improvement in total communication with negligible computational overhead.

8.2 Unbalanced PSI

Figure 4 contains our main set of performance numbers and demonstrate a wide flexibility in set sizes. For the sender, we consider set sizes of $2^{20}, 2^{24}$ and 2^{28}, while the receiver's set sizes range between 128 and 4096. We note that for each of the receiver's set sizes, approximately 1.33 times more items could be added with no difference in performance due to extra space in the cuckoo table. However, to give a fair comparison with other protocols without parameter restrictions we round down to the nearest power of two.

We begin with our performance numbers for the smallest set sizes of $|X| = 2^{20}$ and $|Y| = 128$. For such a small intersection our protocol is extremely efficient, requiring an online time of less than a second on a single thread, and only 3.9 MB of communication. When the receiver's set is increased to 512 items we observe only a minimal increase in running time and communication. Moving to our largest receiver set size of 4096, we observe roughly a 4.7× increase in online running time and a 3× increase in communication. This sublinear growth in overhead with respect to the receiver's set size is attributed to the ability to use more efficient FHE parameters. With respect to the sender's offline running time, we observe that in almost all cases with $|X| = 2^{20}$ the running time is roughly 40 seconds on a single thread. The one exception is for $|Y| = 256$, which has double the running time. This is attributed to the FHE parameters used that allowed an efficient online time at the expense of increased computation during the offline phase.

Increasing the sender's set size to $|X| = 2^{24}$ we observe a similar trend, where a smallest set sizes for the receiver all obtain the same performance. Indeed, for $|Y| = 128, 256, 512$ the same FHE parameters were utilized, which yield a single thread online running time of 9.1 seconds, and 8.2 MB of communication. The choice to use oversized parameters for the smaller set sizes stems from the highly complex interplay between the parameters that can be optimized, while also maintaining a computational security level of 128 bits. We refer to Section 3.1 for a more detailed explanation. For a receiver's set size of 4096 we observe an online running time of 22 seconds in the single threaded setting, and 15.9 MB of communication. The sender's offline time required 806 seconds on a single thread, or 32 seconds on 32 threads. Interestingly, this offline running time is faster as the sender's set size increases. Among other things, this is the result of the sender's database having fewer items per bin as $|Y|$ increases, which in turn decreases the degree of the polynomials the sender needs to compute in pre-processing, resulting in improved performance.

In addition, we consider the case of $|X| = 2^{28}$ which, as far as we are aware of, is the largest PSI set size to have ever been considered in the two-party setting, and is only surpassed in a weaker model where an untrusted third party assists in the computation [32]. For this case we consider a receiver's set size of 1024, and observe that the online phase can be performed in just 12 seconds when the sender and receiver respectively use 32 and 4 threads. At the same time, the online phase utilizes only 18.4 MB of communication. The primary impact of such a large set size is on the sender's offline running time. However, in cases where such a large set is used, it is highly likely that the set is held by a powerful server and is relatively static, in which case the sender can amortize the cost of the pre-processing across several protocol executions. We also note that our protocol allows fast additions and deletions from the pre-processed set by updating only small targeted locations when necessary. Moreover, the current implementation of the offline phase is far from optimal in that there exists more efficient algorithms for computing coefficients of the sender's polynomials, which is the primary bottleneck.

8.3 Comparison

We now move on to the comparison with the OPRF-based protocols of [47,5] and the FHE-based protocol of [12]. First we recall the protocol paradigm of [47,5,31]. These protocols utilize the same

$\lvert X\rvert$	$\lvert Y\rvert$	Sender offline				Online				Comm.	
		T=1	4	8	32	T=1	4	8	32	$R \to S$	$R \leftarrow S$
2^{28}	2048	–	–	–	4,628	–	–	–	28.5	11.8	10.2
	1024	–	–	–	4,628	–	–	–	12.1	8.2	10.2
2^{24}	4096	806	206	111	32	22.0	5.9	3.5	2.2	8.7	7.2
	2048	747	483	58	18	12.6	3.4	2.0	1.4	8.2	8.1
	1024	1430	418	155	51	17.7	5.2	2.9	1.1	3.1	5.1
	512	1368	267	146	45	9.1	2.6	1.5	0.7	3.1	5.1
	256	1368	267	146	45	9.1	2.6	1.5	0.7	3.1	5.1
	128	1368	267	146	45	9.1	2.6	1.5	0.7	3.1	5.1
2^{20}	4096	43	6.1	3.3	1.2	4.2	1.3	1.3	1.2	4.2	7.2
	2048	39	4.9	2.7	1.1	2.1	0.7	0.6	0.6	2.9	3.6
	1024	40	5.1	2.9	1.1	2.0	0.7	0.5	0.5	2.5	2.5
	512	36	5.5	2.6	0.5	1.0	0.4	0.3	0.3	1.5	2.5
	256	87	18.4	9.5	3.3	2.4	0.8	0.6	0.3	2.1	1.0
	128	46	6.7	3.8	1.2	0.9	0.4	0.3	0.2	0.9	3.0

Fig. 4. Performance metrics of our protocol in the LAN setting for various set sizes. "Sender offline" reports the running time in seconds required to initialize the sender's set. This is non-interactive and can be reused with multiple receivers. "Online" reports the end-to-end running time in seconds required to perform the intersection, where the "Comm." columns report the directional communication requirements in MB. T is the number of threads used by the sender. The receiver uses $\max\{T, 4\}$ threads.

Diffie-Hellman based OPRF as our protocol, where the sender holds the key k and applies the OPRF to its set X to obtain $X' = \{F_k(x) : x \in X\}$. The receiver then interactively computes $Y' = \{F_k(y) : y \in Y\}$. Next the sender sends X' to the receiver, who infers the intersection from $X' \cap Y'$ [31].

The main limitation of this approach is that the receiver requires communication linear in the larger set X. The protocol above was first described in the malicious setting by [31], and later optimized for the semi-honest setting by [5]. For modest size X this approach works reasonably well. For instance, with $|X| = 2^{20}$ and a receiver's set of 5535 items, the communication of X' is 10 MB. However, increasing $|X|$ further to 2^{24} or 2^{28} starts to become less practical with 160 MB and 2.6 GB of communication, respectively.

One of the most compelling properties of this paradigm is that after the communication of X', the receiver can check the membership of a y in X using $O(\kappa)$ communication and computation—effectively a constant. Observing this, [47,35] both suggest that X' be sent during an offline or pre-processing phase, and then the online phase can have linear overhead in the smaller set, which provides extremely good performance. Moreover, many adaptive set intersections with X can be performed, where the receiver is able to use a different set each time. The online communication and running time of this approach remains linear in the receiver's set size.

On top of this general paradigm, [47,35] suggest that the encoded set X' can be compressed using techniques such as a cuckoo filter or a Bloom filter. [47] showed that for certain parameters a cuckoo filter can reduce the communication by roughly 3×, bringing the communication for $|X| = 2^{24}$ and 2^{28} down to 48 and 806 MB, respectively. This approach has the disadvantage that such a large amount of compression introduces a relatively high false positive rate of $2^{-13.4} \times |Y|$. That is, if the receiver has 1024 items outside the intersection, the probability that the receiver will output a wrong item is $2^{-3.2}$, which is extremely high for cryptographic protocols. In contrast, our protocol and [12] both provide a false positive rate of at least 2^{-40}, and this rate is independent of the receiver's set size.

In addition, the communication complexity of [47] remains linear in the larger set size. In the example of $|X| = 2^{28}$ the communication of 804 MB can be prohibitive for many applications.

Moreover, for this online-offline technique to work, the receiver must store the compressed set long-term, which can be prohibitive on mobile devices. In contrast to this paradigm, our FHE-based protocol achieves communication complexity which is sub-linear in the larger set size and requires no offline storage by the receiver. When considering $|X| = 2^{28}$ and $|Y| = 1024$, the total communication is less than 19 MB compared to 806 MB: a 43× improvement in communication and a $2^{26}\times$ improvement in the false positive rate. To achieve an equivalent false positive rate, e.g. in the [5] protocol, the communication increases to 2.6 GB—a difference of 100×.

Finally, there is a difference in the security achieved with respect to the information revealed when X is updated. In the case of [47,5,35], the receiver learns exactly how many items we inserted and removed from X. Even worse, the deletion of an item from X cannot be enforced, since the receiver can simply keep the corresponding OPRF value which they hold in X'. As such, deletions cannot be enforced even in the semi-honest model. In contrast, our protocol can easily add and remove items without leaking extra information to the receiver. In particular, the issue with deleting an item from X does not arise since only the sender holds X.

With respect to [12], we compare quite favorably in many aspects. First recall that [12] is practically restricted to 32-bit items, while we can support arbitrary length items. In particular, in [12] the actual items being compared within FHE are only 22 bits, but are extended to 32 bits using phasing [43]. Our protocol contrasts this by comparing 80-bit items within the FHE computation. Combined with the "hash to smaller domain" techniques of [45], this is sufficiently large to support our PSI set sizes, while achieving 40 bits of statistical security.

Given this significant improvement, our protocol still achieves a similar communication overhead, and often better online running times. For example, when comparing $|X| = 2^{24}$ and $|Y| = 5535$, our protocol requires 16 MB of communication and 22 seconds in the online phase on a single thread, while [12] requires 20 MB and 40 seconds. This is almost a 2× improvement in running time, a 27% improvement in the communication, and the capability to compare arbitrary length strings. Moreover, when the receiver has an even smaller set, our protocol is able to scale the FHE parameters even smaller, which allows less communication. For example, when the receiver has 512 or 1024 items, our protocol requires 9.1 and 17.7 seconds, respectively, and just 8.2 MB of communication. On the other hand, due to the noise flooding performed by [12], they are unable to take advantage of more efficient FHE parameters while also maintaining 128 bits of computational security. As such, our protocol can be 2 to 4 times faster than [12], and send almost half the amount of data, while at the same time supporting arbitrary length items.

8.4 Labeled PSI

We compare the performance of our Labeled PSI technique with the PIR by keywords utilized by the anonymous communication protocol Pung [3,2]. The Pung protocol allows a set of clients to privately send and retrieve messages through a server, without the server learning any information (including metadata) about the conversations. In each epoch of the protocol, a client wishes to privately retrieve several messages from the server from other clients using some secret keywords they share. This was achieved using a single-server PIR based on additive homomorphic encryption. In order for a client to obtain the index of messages sent to her, the server sends a Bloom filter containing the keyword-to-index mapping to each client. Pung also optimized for the multi-query using hashing techniques. In one setting, the client retrieves 256 messages in each epoch. Each message has 288 bytes, and a total number of 1 million messages is stored at the server. We used Labeled PSI to implement the retrieval process, and compared our performance to [2] in Figure 6. From the results, we see that Labeled PSI can achieve a 4.4× reduction in server's online computation time, and 6.8× reduction in communication.

$\|X\|$	$\|Y\|$	Protocol	Sender Offline	Offline Comm. & Receiver Storage	Online Time	Online Comm.
2^{28}	2048	Ours*	4,628	0	28.5	22.28
		[47]*	182	806	0.1	0.13
		[5]*	182	2,684	0.1	0.13
	1024	Ours*	4,628	0	12.1	18.57
		[47]*	182	806	0.16	0.07
		[5]*	182	2,684	0.16	0.07
2^{24}	11041	Ours	656	0	20.1	41.48
		[12]	71	0	44.70	23.20
		[47]	342	48	0.71	0.67
		[5]	342	160	0.71	0.67
	5535	Ours	806	0	22.01	16.39
		[12]	64	0	40.10	20.10
		[47]	342	48	0.35	0.34
		[5]	342	160	0.35	0.34
2^{20}	11041	Ours	43	0	4.49	14.34
		[12]	6.4	0	6.40	11.50
		[47]	22	3	0.71	0.67
		[5]	22	10	0.71	0.67
	5535	Ours	43	0	4.23	11.50
		[12]	4.3	0	4.30	5.60
		[47]	22	3	0.35	0.34
		[5]	22	10	0.35	0.34

Fig. 5. Our PSI protocol compared with [12,47,5] in the LAN setting for various set sizes. All executions are with a single thread with the exception (*) of $|X| = 2^{28}$, which is performed with 32 threads by the sender, and 4 threads by the receiver. Communication/storage is in MB and running time is in seconds. The "Sender offline" column is running time required by the sender to initialize their database. It can be reused and is non-interactive.

$\|X\|$	$\|Y\|$	ℓ (Bytes)	Method	Server online	Client encrypt	Comm.
2^{20}	256	288	[2]	20.5 s	4.92 s	120 MB
			Ours	4.6 s	0.77 s	17.6 MB

Fig. 6. Peformance of Labeled PSI applied to the Pung anonymous communication protocol.

References

1. Validating leaked passwords with k-anonymity. https://blog.cloudflare.com/validating-leaked-passwords-with-k-anonymity/ (2018), accessed: 2018-05-08
2. Angel, S., Chen, H., Laine, K., Setty, S.: Pir with compressed queries and amortized query processing. In: 2018 IEEE Symposium on Security and Privacy (SP). pp. 962–979. IEEE (2018)
3. Angel, S., Setty, S.: Unobservable communication over fully untrusted infrastructure. In: OSDI. pp. 551–569 (2016)
4. Bajard, J.C., Eynard, J., Hasan, M.A., Zucca, V.: A full RNS variant of FV like somewhat homomorphic encryption schemes. In: International Conference on Selected Areas in Cryptography. pp. 423–442. Springer (2016)
5. Baldi, P., Baronio, R., De Cristofaro, E., Gasti, P., Tsudik, G.: Countering gattaca: efficient and secure testing of fully-sequenced human genomes. In: Proceedings of the 18th ACM conference on Computer and communications security. pp. 691–702. ACM (2011)
6. Boneh, D., Gentry, C., Halevi, S., Wang, F., Wu, D.J.: Private database queries using somewhat homomorphic encryption. In: International Conference on Applied Cryptography and Network Security. pp. 102–118. Springer (2013)
7. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical GapSVP. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO. Lecture Notes in Computer Science, vol. 7417, pp. 868–886. Springer (2012)
8. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. In: Proceedings of the 3rd Innovations in Theoretical Computer Science Conference. pp. 309–325. ACM (2012)
9. Brakerski, Z., Vaikuntanathan, V.: Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: Advances in Cryptology–CRYPTO 2011, pp. 505–524. Springer (2011)
10. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) LWE. SIAM Journal on Computing **43**(2), 831–871 (2014)
11. Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Hoffstein, J., Lauter, K., Lokam, S., Moody, D., Morrison, T., Sahai, A., Vaikuntanathan, V.: Security of homomorphic encryption. Tech. rep., HomomorphicEncryption.org, Redmond WA (July 2017)
12. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1243–1255. ACM (2017)
13. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 409–437. Springer (2017)
14. Cheon, J.H., Kim, M., Kim, M.: Search-and-compute on encrypted data. In: International Conference on Financial Cryptography and Data Security. pp. 142–159. Springer (2015)
15. Cheon, J.H., Kim, M., Kim, M.: Optimized search-and-compute circuits and their application to query evaluation on encrypted data. IEEE Transactions on Information Forensics and Security **11**(1), 188–199 (2016)
16. Cheon, J.H., Kim, M., Lauter, K.: Homomorphic computation of edit distance. In: International Conference on Financial Cryptography and Data Security. pp. 194–212. Springer (2015)
17. Chillotti, I., Gama, N., Georgieva, M., Izabachene, M.: Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 3–33. Springer (2016)
18. Chor, B., Gilboa, N., Naor, M.: Private information retrieval by keywords. Citeseer (1997)
19. Ciampi, M., Orlandi, C.: Combining private set-intersection with secure two-party computation. Tech. rep., Cryptology ePrint Archive, Report 2018/105 (2018)
20. De Cristofaro, E., Gasti, P., Tsudik, G.: Fast and private computation of cardinality of set intersection and union. In: International Conference on Cryptology and Network Security. pp. 218–231. Springer (2012)
21. Dowlin, N., Gilad-Bachrach, R., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Manual for using homomorphic encryption for bioinformatics. Proceedings of the IEEE **105**(3), 552–567 (2017)
22. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012), http://eprint.iacr.org/
23. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Theory of Cryptography Conference. pp. 303–324. Springer (2005)
24. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC. vol. 9, pp. 169–178 (2009)
25. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the aes circuit. In: Advances in cryptology–crypto 2012, pp. 850–867. Springer (2012)
26. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO (1). Lecture Notes in Computer Science, vol. 8042, pp. 75–92. Springer (2013). https://doi.org/10.1007/978-3-642-40041-4, http://dx.doi.org/10.1007/978-3-642-40041-4

27. Halevi, S., Shoup, V.: Algorithms in helib. In: International cryptology conference. pp. 554–571. Springer (2014)
28. Hart, W., Johansson, F., Pancratz, S.: FLINT: Fast Library for Number Theory (2013), version 2.4.0, http://flintlib.org
29. Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In: Canetti, R. (ed.) Theory of Cryptography. pp. 155–175. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
30. Ion, M., Kreuter, B., Nergiz, E., Patel, S., Saxena, S., Seth, K., Shanahan, D., Yung, M.: Private intersection-sum protocol with applications to attributing aggregate ad conversions. Tech. rep., Cryptology ePrint Archive, Report 2017/738 (2017)
31. Jarecki, S., Liu, X.: Fast secure computation of set intersection. In: International Conference on Security and Cryptography for Networks. pp. 418–435. Springer (2010)
32. Kamara, S., Mohassel, P., Raykova, M., Sadeghian, S.: Scaling private set intersection to billion-element sets. In: Christin, N., Safavi-Naini, R. (eds.) Financial Cryptography and Data Security. pp. 195–215. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
33. Khedr, A., Gulak, G., Vaikuntanathan, V.: Shield: scalable homomorphic implementation of encrypted data-classifiers. IEEE Transactions on Computers **65**(9), 2848–2858 (2016)
34. Kim, A., Song, Y., Kim, M., Lee, K., Cheon, J.H.: Logistic regression model training based on the approximate homomorphic encryption. Cryptology ePrint Archive, Report 2018/254 (2018), https://eprint.iacr.org/2018/254
35. Kiss, Á., Liu, J., Schneider, T., Asokan, N., Pinkas, B.: Private set intersection for unequal set sizes with mobile applications. Proceedings on Privacy Enhancing Technologies **2017**(4), 177–197 (2017)
36. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 818–829. ACM (2016)
37. Marlinspike, M.: The difficulty of private contact discovery. A company sponsored blog post (2014), https://whispersystems.org/blog/contact-discovery/
38. Meadows, C.: A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In: 1986 IEEE Symposium on Security and Privacy. pp. 134–134 (April 1986). https://doi.org/10.1109/SP.1986.10022
39. Nagy, M., De Cristofaro, E., Dmitrienko, A., Asokan, N., Sadeghi, A.R.: Do i know you?: efficient and privacy-preserving common friend-finder protocols and applications. In: Proceedings of the 29th Annual Computer Security Applications Conference. pp. 159–168. ACM (2013)
40. Odlyzko, A.: Privacy, economics, and price discrimination on the internet. In: Proceedings of the 5th international conference on Electronic commerce. pp. 355–366. ACM (2003)
41. Olumofin, F., Goldberg, I.: Privacy-preserving queries over relational databases. In: International Symposium on Privacy Enhancing Technologies Symposium. pp. 75–92. Springer (2010)
42. Orrù, M., Orsini, E., Scholl, P.: Actively secure 1-out-of-n ot extension with application to private set intersection. In: Cryptographers Track at the RSA Conference. pp. 381–396. Springer (2017)
43. Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: Private set intersection using permutation-based hashing. In: 24th USENIX Security Symposium (USENIX Security 15). pp. 515–530 (2015)
44. Pinkas, B., Schneider, T., Weinert, C., Wieder, U.: Efficient circuit-based psi via cuckoo hashing. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 125–157. Springer (2018)
45. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: USENIX Security Symposium. vol. 14, pp. 797–812 (2014)
46. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on OT extension. ACM Transactions on Privacy and Security (TOPS) **21**(2), 7 (2018)
47. Resende, A.C.D., Aranha, D.F.: Faster unbalanced private set intersection. Journal of Internet Services and Applications **9**(1), 1–18 (2018)
48. Rindal, P., Rosulek, M.: Malicious-secure private set intersection via dual execution. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1229–1242. CCS '17, ACM, New York, NY, USA (2017). https://doi.org/10.1145/3133956.3134044, http://doi.acm.org/10.1145/3133956.3134044
49. Rivest, R.L., Adleman, L., Dertouzos, M.L.: On data banks and privacy homomorphisms. Foundations of secure computation **4**(11), 169–180 (1978)
50. Smart, N.P., Vercauteren, F.: Fully homomorphic SIMD operations. Designs, codes and cryptography **71**(1), 57–81 (2014)
51. Wang, F., Yun, C., Goldwasser, S., Vaikuntanathan, V., Zaharia, M.: Splinter: Practical private queries on public data. In: NSDI. pp. 299–313 (2017)

A Security Proof for Leaky PSI

Parameters: The honest sender and receiver have respective set sizes $N_{\text{X}}, N_{\text{Y}}$. If the sender or receiver is maliciously corrupt, then their set size is N'_{X} or N'_{Y} respectively.

- On input (RECEIVE, sid, Y) from the receiver where $Y \subset \{0,1\}^*$, ensure that $|Y| \leq N_{\text{Y}}$ if the receiver is honest and $|Y| \leq N'_{\text{Y}}$ otherwise. Give (RECEIVER-INPUT, sid) to the sender.
- Thereafter, on input (SEND, sid, X, leakage) from the sender where $Y \subset \{0,1\}^*$ and leakage is the leakage function, ensure that $|X| \leq N_{\text{X}}$ if the sender is honest and $|X| \leq N'_{\text{X}}$ otherwise.
  The functionality computes $Y^* = \mathsf{leakage}(Y)$ and gives output (OUTPUT, sid, $X \cap Y \cap Y^*$) to the receiver.

Fig. 7. Ideal leaky PSI functionality.

Input: Receiver inputs set $Y \subset \{0,1\}^*$ of size N_{Y}; sender inputs set $X \subset \{0,1\}^*$ of size N_{X}. $N_{\text{X}}, N_{\text{Y}}$ are public. κ and λ denote the computational and statistical security parameters, respectively.
Output: The receiver outputs $Y \cap X$ if the sender is honest and $Y \cap X \cap \mathsf{leakage}(Y)$ otherwise; the sender outputs $\perp$.

1. The parties run steps 1 through 7 of Figure 2.
   (a) In Step 5(b)ii the sender defines the label for each $x' \in X'$ to be $H(x')$ where $H : \{0,1\}^\sigma \to \{0,1\}^\sigma$ is a random oracle which is sufficiently complex such that for $H'(\{z^{i \cdot w^j} \mid 1 \leq i < w, 0 \leq j \leq \lfloor \log_w(B) \rfloor\}) = H(z)$ the FHE scheme is H'-limited (Definition 1). Note that w is the windowing parameter of Step 7d.
   (b) In Step 7a the sender must provide a zero knowledge proof that all OPRF instances used the same key k.
2. The parties perform Step 8 except that the symmetric polynomials $\overline{S}_{b,j}$ are not evaluated or sent back.
3. Instead of performing Step 9, the receiver does the following. For the b-th batch, the receiver decrypts the ciphertexts $q_{b,1}, ..., q_{b,\alpha}$ to obtain $r_{b,1}, ..., r_{b,\alpha}$, which are interpreted as vectors of n/d elements in $\mathbb{F}_{t^d}$.
   Let $r^*_1, ..., r^*_\alpha$ be vectors of m elements in $\mathbb{F}_{t^d}$ obtained by concatenating $r^*_j = r^*_{1,j} || ... || r^*_{md/n,j}$. For all $y' \in Y'$, output the corresponding $y \in Y$ if

$$\exists j : r^*_j[i] = H(y')\,,$$

   where i is the index of the bin that y' occupies in $\mathcal{C}$

Fig. 8. Full Malicious PSI protocol with leakage.

Theorem 3. *In the random oracle hybrid and in the presence of an OMGDH-hard group G along with a semantic secure H-limited leveled fully homomorphic encryption scheme Σ, the protocol π of Figure 8 securely computes the leaky PSI functionality (Figure 7) with (non-black box) simulation security where $N'_{\text{Y}} = N_{\text{Y}}$ and $N'_{\text{X}} = \mathsf{poly}(\kappa)$.*

Proof (Proof Sketch). Proving that the protocol is secure against a malicious receiver is a straight forward extension of Theorem 1.

In the case of a malicious sender we must describe a non-black box simulator which extracts the senders set and leakage function which are forwarded to the ideal functionality of Figure 7. The simulator plays the role of the receiver using a dummy set and at the end of the protocol extracts the senders set and function as follows.

First, the simulator extracts the OPRF key k from the zero knowledge proof and uses the random oracle to extract all of the OPRF inputs which the sender has queried with key k and

defines this as their set $X*$. By Definition 1 it follows that all of the labels for items in $\{0,1\}^*\backslash X^*$ are uniformly distributed and that the sender can not construct a ciphertext which decrypts to one of these labels with non-negligible probability. If this does not hold then this sender can be used to construct a contradiction. Namely, since all OPRF outputs for inputs in $\{0,1\}^*\backslash X^*$ are uniformly distributed in $\{0,1\}^{\mathsf{poly}(\kappa)}$, we can program an output of the OPRF/random oracle on an input $y \in Y$ to be the $z \in \{0,1\}^{\mathsf{poly}(\kappa)}$ from Definition 1 and the code of the sender to be F^*. If the sender (F^*) was able to generate a ciphertext of $H(z)$ then this ciphertext is a contradiction since the FHE scheme is assumed to be H-limited.

The leakage function is then defined as follows. On input Y the function applies the OPRF to Y and performs cuckoo hashing on the result. By inspecting the code of the sender, it is determined if the correct label is computed for the given cuckoo location. If so then the input represented by this location is output by the leakage function. The simulator can then send the sender's set and this leakage function to the ideal functionality (Figure 7).

Remarks: A zero knowledge proof that the sender uses the same key to all of the OPRF invocations is added to Figure 8. This is primarily to ease the exposition of the proof. Instead of zero knowledge proofs it is sufficient for the receiver to abort if any of the OPRF response messages are $0, 1 \in G$. The proof must then handle the case that multiple OPRF keys are used by the sender and the case where the the key is unknown, e.g. sent a random value back instead of $H(y)^{\beta\alpha}$. These cases can be handled by adding addition logic to the leakage function.