Labeled PSI from Homomorphic Encryption with Reduced Computation and Communication

Kelong Cong[1], Radames Cruz Moreno[2], Mariana Botelho da Gama[1], Wei Dai[2], Ilia Iliashenko[1], Kim Laine[2], and Michael Rosenberg[3]

[1] imec-COSIC, KU Leuven, Leuven, Belgium
[2] Microsoft Research, Redmond, WA, USA
[3] University of Maryland, College Park, MD, USA

Abstract. It is known that fully homomorphic encryption (FHE) can be used to build efficient (labeled) Private Set Intersection protocols in the unbalanced setting, where one of the sets is much larger than the other (Chen *et al.* (CCS'17, CCS'18)). In this paper we demonstrate multiple algorithmic improvements upon these works. In particular, our protocol has an asymptotically better computation cost, requiring only $\mathcal{O}(\sqrt{|X|})$ homomorphic multiplications, and communication complexity sublinear in the larger set size $|X|$.

We demonstrate that our protocol is significantly better than that of Chen *et al.* (CCS'18) for many practical parameters, especially in terms of online communication cost. For example, when intersecting 2^{28} and 2048 item sets, our protocol reduces the online computation time by more than 83% and communication by more than 32%. When intersecting 2^{24} and 4096 item sets, our protocol reduces the online computation time by 50% and communication by 52%. Our comparison to other state-of-the-art unbalanced PSI protocols shows that our protocol has the best total communication complexity when $|X| \geq 2^{24}$. For labeled PSI our protocol also outperforms Chen *et al.* (CCS'18). When intersecting 2^{20} and 256 item sets, with the larger set having associated 288-byte labels, our protocol reduces the online computation time by more than 85% and communication by 36%.

Finally, we demonstrate a modification that results in nearly constant communication cost in the larger set size $|X|$, but impractically high computation complexity on today's CPUs. For example, to intersect a 210-item set with sets of size 2^{22}, 2^{24}, or 2^{26}, our proof-of-concept implementation requires only 0.76 MB of online communication, which is more than a 24-fold improvement over Chen *et al.* (CCS'18).

1 Introduction

Consider two parties, each with a private dataset of *items*. They want to learn the intersection of their respective sets without leaking any other information to each other. For example, two countries want to trace criminal suspects by finding matching identities in their police databases, while hiding other sensitive data from each other, or two banks want to identify clients with obscure financial transactions, without revealing any other data about their clients.

To solve the above problems, the two parties can engage in a *Private Set Intersection* (PSI) protocol. PSI refers to an interactive cryptographic protocol that takes two private sets as input, finds their intersection, and outputs it to one or both of the participants. If one party obtains the results of a PSI protocol (one-way PSI), we call this party *the receiver* and the other party *the sender*. General two-way PSI can be realized via two rounds of one-way PSI, where participants swap the roles of the receiver and the sender. In practice, one-way PSI stands on its own as an important primitive in various privacy-preserving protocols, including private contact discovery in mobile messengers (e.g., Signal or WhatsApp) [43,29], checking for the presence of leaked password in a breach database [38,2], and contact tracing for containing the spread of infectious diseases [61].

In this work, we focus on a variant of one-way PSI where the receiver's set is much smaller than the sender's. We also assume that the receiver has limited resources for computation and storage (e.g., a mobile phone or a wearable device), while the sender is equipped with much more powerful machinery (e.g., a server in a datacenter). This specific setting is referred to as *unbalanced*. The

goal in the unbalanced setting is thus to delegate as much computation as possible to the sender and reduce communication between the parties.

Let X and Y denote the sender's and the receiver's sets, respectively; in unbalanced PSI, we have $|Y| \ll |X|$. State-of-the-art protocols for unbalanced PSI have either $\mathcal{O}(|X|)$ communication cost [37] or $\tilde{\mathcal{O}}(|X|)$ computation complexity [14]. We will focus on the latter line of work, which allows to achieve a quasilogarithmic communication complexity. Our work uses *leveled Fully Homomorphic Encryption* (FHE), i.e., randomized encryption that allows computation on arithmetic circuits of any fixed multiplicative depth in the encrypted domain, without decryption. Leveled FHE schemes such as BGV [8] and BFV [23] induce quasipolynomial overhead both in communication and computation complexity with respect to the multiplicative depth.

Following the outline of [14], we also consider *labeled PSI*, a special type of PSI with computation. In this scenario, every element in the sender's set has associated data — a *label* — and the receiver hopes to learn the labels of the elements in the intersection. This is a generalization of the single-server *Private Information Retrieval (PIR) by keywords* problem introduced by Chor *et al.* [17]. The practical applications of labeled PSI include targeted price discrimination [46] and key retrieval in mobile messengers [43,29], where a user queries the public key of people from her contact list.

1.1 Related work

Early PSI protocols from the 1980s [44,34] were based on Diffie-Hellman (DH) key-exchange [19]. In essence, Alice and Bob perform a DH key-exchange for every element in their set; a match is found if the resulting shared secrets match. Freedman *et al.* [25] introduced a protocol based on oblivious polynomial evaluation and homomorphic encryption. Ateniese *et al.* [4] introduced a construction based on RSA accumulator. Since PSI can be thought of as a special case of secure two-party computation, it is possible to use standard techniques such as garbled circuits to construct a PSI protocol [33,50,52]. More recently many protocols [20,53,47,41,56,49,13] have been based on OT extension [35] and Oblivious Pseudo-Random Functions (OPRF).

The aforementioned work are primarily designed for the balanced setting, where the sender set is roughly the same as the receiver set. Below we zoom in on two of the most efficient paradigms in the literature in the unbalanced setting, which is the focus of this work.

Unbalanced PSI Based on OPRF A PSI protocol based on OPRF was first proposed in [24], where the OPRF was build from the Naor-Reingold (NR) pseudorandom function [45]. With the help of garbled circuits (GC) [62], later work introduced a PSI protocol from AES-based OPRF [51]. The current state-of-the-art protocol materialized in the work of Kales *et al.* [37], which is derived from [40] and [55]. The protocol has two phases, the preprocessing phase and the online phase. The authors introduced many optimizations to push as much computation and communication cost to the preprocessing phase as possible. Below we give an overview of the protocol idea from this line of work. During the preprocessing phase, the sender with a large set S generates a PRF key k and computes $\mathsf{PRF}_k(s)$ for every $s \in S$. Then it inserts every PRF output into a cuckoo filter [22] and sends it to the receiver. To compute the intersection in the online phase, the receiver computes the OPRF with the help of the sender. That is, the receiver obtains $\mathsf{PRF}_k(r)$ for every value r in its set R without the knowledge of k. Finally, the receiver locally checks whether any of its items are in the cuckoo filter.

The online phase is very efficient, and does not depend on the set size of the sender. For a receiver with 2^{10} items, the communication overhead is only a little higher than 2 megabytes [37]. The computation is also efficient since there are no expensive public-key operations when using

the GC version of the protocol.[4] The biggest bottleneck in this class of protocols is the bandwidth consumption and storage requirement during the preprocessing phase. Namely, the cuckoo filter that the sender must distribute to every receiver is linear in the size of the sender's set. When the sender set has 2^{28} items, the cuckoo filter size is more than one gigabyte [37].

Unbalanced Protocols Based on FHE On the other side of the spectrum, Chen *et al.* [15] introduced an unbalanced PSI protocol based on leveled FHE that has a communication overhead linear in the (smaller) receiver's set and logarithmic in the (larger) sender's set. Their work was later refined in [14], which strengthened the security model using OPRF, allowed arbitrary-length items, and extended the protocol to the labeled PSI setting with arbitrary-length labels.

In both works [15,14], the BFV scheme [23] is used, which has $\tilde{\mathcal{O}}(D^2)$ memory and $\tilde{\mathcal{O}}(D^3)$ running time overhead with D being the multiplicative depth of computation. The communication cost of the PSI protocol is $\mathcal{O}(\log |X|)$ ciphertexts and its computation complexity is $\mathcal{O}(|X|)$ homomorphic multiplications with multiplicative depth $D \in \mathcal{O}(\log \log |X|)$. As a result, the dependency on the sender's set size is quantified as follows: the communication complexity is $\tilde{\mathcal{O}}(C)$ with $C = \log |X|$, whereas the computation complexity is $\tilde{\mathcal{O}}(|X|)$.

The working principal is the starting point of this work. Thus we defer the detailed explanation to Section 3.

1.2 Contributions

We introduce several optimizations and improvements to the protocols of Chen *et al.* [15,14] that result in improved running time and improved communication complexity in the sender's set size. Our contributions can be summarized as follows:

- We show how to remove the slow extension field arithmetic used in [14], while still supporting arbitrary-length items and labels.
- We reduce the communication cost by using extremal postage stamp bases [12] instead of the less efficient windowing technique of [15,14].
- We show how to evaluate the intersection circuit in $\mathcal{O}(\sqrt{|X|})$ ciphertext-ciphertext multiplications using the Paterson-Stockmeyer algorithm [48], which is a quadratic improvement over [14].
- We use the techniques of [57] to achieve a very fast hash-to-curve implementation for the FourQ curve [18]. Hash-to-curve is needed for an OMGDH-based OPRF protocol [36,55,14], which is a critical component of our protocol.[5]
- We improve the communication overhead by exploiting depth-free homomorphic Frobenius automorphisms. This operation allows to compute x^t, where t is the plaintext modulus of the FHE scheme, without ciphertext-ciphertext multiplications. As a result, to compute the intersection circuit with depth $\mathcal{O}(\log \log |X|)$ the sender needs only $O(1)$ ciphertexts. This reduces the communication complexity to $\tilde{\mathcal{O}}(C^2)$ with $C = \log \log |X|$ from $\tilde{\mathcal{O}}(2^C C^2)$ given in prior work [15,14].
- We created an open-source reference implementation of our protocol.

[4] Evaluating the GC-based OPRF requires OT, but it is possible to do this in the preprocessing phase using random OT extension.

[5] OPRF was used optionally by [14] to significantly strengthen the security model.

2 Preliminaries

2.1 Notation

Throughout this paper we denote the set of integers $\{i, i+1, ..., j\}$ by $[i, j]$, and $[j]$ is shorthand for the case $i = 1$. The logarithm in base b is denoted by $\log_b x$. For the binary logarithm of x, we omit the base and write $\log x$. Vectors are denoted by lowercase Latin letters with the arrow, i.e., $\vec{a}$. For a given vector $\vec{a} = (a_1, \ldots, a_d)$, its 1-norm is defined as $|\vec{a}|_1 = \sum_{i=1}^{d} |a_i|$.

2.2 Unbalanced private set intersection

There are two parties — *the receiver* and *the sender* — having a set Y and a set X, respectively. We assume that $|X| \gg |Y|$ and the set sizes are public. Each set contains a pair of strings (a_i, L_i) where a_i is an element ID and L_i is its label. The length σ of IDs and length ℓ of labels are shared among all elements of $X \cup Y$.

A (one-way) private set intersection (PSI) protocol is an interactive protocol that outputs $\{a_i : \forall a_i \in X \cap Y\}$ to the receiver and nothing to the sender without leaking any other information about X and Y. In the labeled PSI, the protocol returns $\{(a_i, L_i) : \forall a_i \in X \cap Y\}$ to the receiver.

2.3 Fully homomorphic encryption

Fully homomorphic encryption (FHE) is a family of encryption schemes that allow arbitrary operations to be performed on encrypted data without decryption. Most existing FHE schemes [26,9,10,7,23,8,28,21,16] are based on the LWE [54] or RLWE [42] problems, which imply the presence of *noise* in encrypted messages. The noise associated with a ciphertext grows with each homomorphic operation (additively with additions and multiplicatively with multiplications). Thus, in order to avoid decryption error, one must take care to ensure that the noise does not become large enough to interfere with the underlying plaintext. Such *noise management* is realized with two frameworks.

The first framework consists of a *somewhat* homomorphic encryption (SHE) scheme that is capable of homomorphically computing its own decryption circuit, or perform a so-called *bootstrapping operation* that reduces the noise size. For such FHE schemes, encryption parameters can be fixed such that any circuit is computable given the bootstrapping information. However, the bootstrapping operation is much slower in comparison to other homomorphic operations; thus, it is usually avoided in practice.

The idea of the second framework is to choose large enough encryption parameters for computing a predetermined family of circuits. In this case, bootstrapping is never used, but encryption parameters grow with the depth of circuits. The SHE schemes whose encryption parameters increase polynomially with the circuit depth are called *leveled* FHE schemes. For shallow enough circuits, leveled FHE schemes are more efficient in practice.

In this work, we use two leveled FHE schemes, BGV [8] and BFV [23], implemented in the HElib [32] and SEAL [58] libraries, respectively. Both schemes are defined over a ring $R = \mathbb{Z}[X]/\langle \Phi_m(X) \rangle$ where $\Phi_m(X)$ is the cyclotomic polynomial of order m and degree n. Their ciphertext space is $R_q^2 = (R/\langle q \rangle)^2$ where q is a positive integer. The size of each ciphertext is $2n \log q$.

The plaintext space of these schemes is defined analogously as R_t for some positive integer $t \ll q$. The size of each plaintext is $n \log t$. We assume that t is a prime number. Let d be the multiplicative order of t modulo m. In this case, $\Phi_m(X)$ splits into k factors of degree d modulo t. Hence, the Chinese remainder theorem yields the isomorphism $R_t \cong \mathbb{F}_{t^d}^k$. Each copy of $\mathbb{F}_{t^d}$ is called a *slot*. Using this isomorphism, one can encode and encrypt multiple data values from $\mathbb{F}_{t^d}$ into a single ciphertext,

and have the homomorphic additions and multiplications over R_q^2 correspond to slot-wise additions and multiplications over the encoded values. Such encoding is called the SIMD packing [60]. Abusing notation, we denote a ciphertext encrypting a value $a \in \mathbb{F}_{t^d}$ in one of its slots by $[\![a]\!]$.

The noise size is defined by the standard deviation of the noise distribution, σ_e, which is typically set to 3.19.

The parameters n, q and σ_e define the security level of our schemes, where σ_e denotes the standard deviation of the noise distribution, and is ordinarily set to 3.19. The security level is estimated using the LWE estimator [1].

Homomorphic operations and their cost. The basic homomorphic operations we use are addition and multiplication, which can be performed on two ciphertexts or on a pair consisting of a ciphertext and a plaintext. We will also use a *Frobenius operation* which, given an encrypted message $[\![y]\!]$ and a positive integer i, returns $[\![y^{t^i}]\!]$ (see [27] for more details).

The cost of every homomorphic operation is defined by its running time and the amount of noise added to its output. For the cost analysis, we assume that both ciphertexts and plaintexts are represented in the Double-CRT form [27] and ciphertext noise is measured by its infinity norm. We also assume that the hybrid key-switching method is used (see [39] for more details).

Homomorphic addition is the last costly operation, taking only $\mathcal{O}(n \log q)$ integer additions. The output noise is the sum of the input noises up to some small factor about $\mathcal{O}(t)$.

Plaintext-ciphertext multiplications, which we call *scalar multiplications*, amount to $\mathcal{O}(\log q)$ coefficient-wise multiplications of n-dimensional vectors. Hence, this operation takes $\mathcal{O}(n \log q)$ integer multiplications. Scalar multiplication scales the input noise by a factor of $\mathcal{O}(t\sqrt{n})$.

Ciphertext-ciphertext *non-scalar multiplication* has two steps: vector convolution and key-switching. The first step performs $\mathcal{O}(n \log n)$ integer multiplications and the second one performs $\mathcal{O}(n \log n \log q + n(\log q)^2)$ integer multiplications. Thus, the running time of non-scalar multiplication is $\mathcal{O}(n \log n \log q + n(\log q)^2)$ integer multiplications (see [39]). If the input ciphertexts have noises of size V_1 and V_2, then the noise of their product is equal to $\mathcal{O}(n \cdot t \cdot \max(V_1, V_2))$.

The Frobenius operation first permutes the Double-CRT form of a ciphertext and then performs key-switching. The permutation does not require integer multiplications. Hence, the Frobenius operation is faster than non-scalar multiplication and its running time is dominated by key-switching; it takes $\mathcal{O}(n \log n \log q + n(\log q)^2)$ integer multiplications. The noise introduced by the Frobenius operations is also dominated by the one introduced by key-switching (see, e.g., [30]), which adds a value about $\mathcal{O}(\sqrt{n}(\log q)^2)$ to the noise of the input ciphertext.

To sum up, the most costly homomorphic operation both in running time and in noise growth is non-scalar multiplication. We therefore measure the runtime complexity of our algorithms in terms of the number of non-scalar multiplications. We also measure the depth of our circuits in relation to non-scalar multiplications. We say, then, that one (sequential) non-scalar multiplication consumes one *multiplicative level*.

Note that non-scalar multiplications can sometimes be replaced by the Frobenius operation when computing ciphertext powers. Since the Frobenius operation introduces only additive noise, we say that the Frobenius operation consumes no multiplicative levels, i.e., it is *depth-free*.

3 Unlabeled PSI

Our basic PSI protocol follows closely the frameworks of Chen *et al.* [15,14], which are based on ideas from [50].

Input. The receiver has an input set Y of size $|Y|$. The sender's input is a set X of size $|X|$. Both sets contain bit strings of length σ. The values of $|Y|, |X|$ and σ are public.

Output. The receiver outputs $X \cap Y$.

Setup. The receiver and the sender agree on an SHE scheme with the plaintext space being a finite field $\mathbb{F}$. They also publicly choose the number (typically three) of hash functions $h_i : \{0,1\}^\sigma \to [M]$, where M is a positive integer. Finally, they agree an OPRF function $F_{\mathtt{k}} : X \cup Y \to \mathbb{F}$.

The receiver generates the public and the secret keys of the SHE scheme. The sender samples an OPRF key $\mathtt{k}$ and computes $X' := \{F_{\mathtt{k}}(x) \mid x \in X\}$. Then, both parties interact to apply the OPRF to the receiver's set, whereby the receiver obtains $Y' := \{F_{\mathtt{k}}(y) \mid y \in Y\}$. Now computing $X \cap Y$ amounts to computing $X' \cap Y'$.

We note that there are multiple benefits in using OPRF values instead of the original items. Most importantly, it is necessary to provide security against a malicious receiver, because homomorphic encryption does not automatically provide input privacy for the sender's input. To address this issue, [15] used a noise flooding technique to prove security against a semi-honest receiver, but their approach does not extend to the malicious case. Another reason is discussed more below in Section 3.1: using OPRF values allows us to completely avoid costly extension field arithmetic.

Next, the receiver places each $y \in Y'$ into a cuckoo hash table B_R with bin size 1. Specifically, it will construct a table B_R such that no bin in B_R has more than one element, and for all $y \in Y'$ there is an i such that $B_R[h_i(y)] = y$. The sender creates a cuckoo hash table B_S with bin size potentially greater than 1. For all $x \in X'$ and all i, the sender places x into $B_S[h_i(x)]$, again, allowing for multiple x per bin. It is shown in [15] that if $|X| \gg M$, then each bin of the sender contains $O(|X|/M)$ values with high probability. This setup ensures that the intersection of X' and Y' is equal to the union of the respective bin intersections, namely,

$$X' \cap Y' = \bigcup_{j \in [M]} B_R[j] \cap B_S[j] = \bigcup_{j \in [M]} \{y_j\} \cap B_S[j],$$

where y_j denotes the y value at $B_R[j]$. Both parties encode their respective bins into the plaintext field $\mathbb{F}$. The receiver encrypts all its bins and sends them to the sender.

Intersection. Given the encryption $[\![y_j]\!]$ of the bin $B_R[j]$, the sender computes the intersection polynomial

$$[\![z_j]\!] := P([\![y_j]\!]) = \prod_{x \in B_S[j]} ([\![y_j]\!] - x) \tag{1}$$

If $y_j \in B_S[j]$, then $[\![z_j]\!]$ is an encryption of zero. Otherwise, $[\![z_j]\!]$ encrypts some non-zero value in $\mathbb{F}$, depending on Y'. This non-zero value does not leak any information about Y, due to the OPRF step. The sender sends $[\![z_j]\!]$ to the receiver, who decrypts it and checks whether $z_j = 0$.

Security. As shown in [14], the OPRF step makes the above PSI protocol secure against a malicious receiver and provides privacy against a malicious sender in the random oracle model. Our protocol differs from [14] in algorithmic aspects; the security guarantees and the security proof remain the same as in [14], i.e., the protocol guarantees security against a malicious receiver and privacy against a malicious sender [31]. With a small extra computational overhead the protocol can be upgraded to provide further protection against a malicious sender, as is described in [14].

3.1 Optimizations

In this section we discuss various optimization techniques to make the above protocol practical. Some of these (SIMD packing, partitioning, and windowing) have been discussed in the past in [15,14] and remain essential to our protocol. We improve the SIMD packing technique to use only more efficient and flexible prime fields, although this mainly presents challenges in the labeled case discussed below in Section 4. We utilize the Paterson-Stockmeyer algorithm [48] to improve the

computational complexity and enable new communication-computation trade-offs. We change the windowing technique to use more efficient extremal postage-stamp bases [11,12], which reduces our communication cost significantly in many cases. We show how many powers of the receiver's input can be computed with zero multiplicative depth, resulting in a variant of the protocol with extremely low communication cost. Finally, we adapt the Elligator 2 [5] map for the Four$\mathbb{Q}$ elliptic curve [18] for a fast hash-to-curve implementation, which is needed for the OMGDH-based OPRF protocol.

Permutation-based hashing [50] can be applied immediately in our work to reduce the item lengths by a few bits. But this technique will have only a marginal performance impact compared to our other techniques, so we do not include it this work.

SIMD packing. As discussed in Section 2.3, we can pack multiple data values into one ciphertext, such that these values can be simultaneously processed with homomorphic operations. Using this method, the receiver can essentially treat the slots in a ciphertext as bins of the cuckoo hash table B_R, and thus encode multiple values $y_j \in Y'$ in a single ciphertext. The sender can subsequently compute several intersection circuits (eq. (1)) in parallel, which results in a significant improvement in both computation and communication cost.

In [14] the authors were able to support arbitrary-length items by first hashing them down to a smaller domain and, using SIMD packing with extension field values that are large enough to hold the hash values. Unfortunately, extension field arithmetic can have a devastating effect on performance; this effect is particularly prominent in the labeled mode. Furthermore, the extension fields must have certain size characteristics to be useful, which in some cases leads to suboptimal parameter choices.

We observe that it is possible to not use extension fields at all, and instead use SIMD packing only over prime fields. Since the element hashes are larger than a single SIMD slot, we simply split the hash values to occupy several sequential slots. This was not possible in [14], because the authors considered the OPRF step as optional: since the individual SIMD slot values are small, they can be guessed, and even a semi-honest adversary can learn information about partially matching items with non-negligible probability. This problem is resolved by always performing the OPRF step, which randomizes the items and protects the sender's dataset from partial item leakage. A few issues remain in the labeled case, which we will discuss and resolve in Section 4.

Partitioning. To reduce the depth of computation, [15] proposed to split every sender's bin into α subsets and then compute intersection on each of these subsets separately. If B is the maximal size of a sender's bin, then this method reduces the circuit depth from $\lceil \log B \rceil$ to $\lceil \log \lceil B/\alpha \rceil \rceil$ at the cost of increasing the number of ciphertexts sent by the sender to the receiver by a factor of α.

To compute the intersection polynomial $P([\![y]\!])$ of degree B, the sender can first compute all the monomial powers $[\![y]\!]^2, \ldots, [\![y]\!]^{\lceil B/\alpha \rceil}$ using $\lceil B/\alpha \rceil - 1$ non-scalar multiplications. These powers can be used repeatedly to compute the intersection circuits for each of the α partitions. Another advantage of partitioning: after the monomial powers have been computed once, they can be used for each partition, i.e., α times, in relatively cheap scalar multiplication operations to evaluate the intersection circuits.

Paterson-Stockmeyer algorithm. One issue with partitioning is that in many situations it is advantageous to take B to be relatively large (say, in the few thousands), requiring $\lceil B/\alpha \rceil - 1$ non-scalar multiplications, while α remains relatively small (say, 10). In such a case the computational cost of the non-scalar multiplications may dominate the online running time. We suggest applying

the Paterson-Stockmeyer algorithm [48] to compute the intersection polynomial in $\mathcal{O}(\sqrt{B})$ non-scalar multiplications instead. We now explain how this works.

First, pick positive integers L and H such that $B = LH - 1$ and $L \approx \sqrt{2(B+1)}$. The sender starts by computing the *low powers* $[\![y]\!]^2, [\![y]\!]^3, \ldots, [\![y]\!]^{L-1}$ and the *high powers* $[\![y]\!]^L, [\![y]\!]^{2L}, [\![y]\!]^{3L}, \ldots, [\![y]\!]^{(H-1)L}$ of the receiver's ciphertext $[\![y]\!]$. Then the intersection polynomial can be rewritten as

$$P([\![y]\!]) = \sum_{i=0}^{H-1} [\![y]\!]^{iL} \sum_{j=0}^{L-1} a_{iL+j} [\![y]\!]^j, \tag{2}$$

where a_k is the k-th coefficient of P. The internal sums can be computed by scalar multiplications and additions from the low powers. Non-scalar multiplications are only needed to multiply these internal sums by the high powers. The total computation complexity of computing $P([\![y]\!])$ is equal to

$$L - 2 + 2(H-1) = L + 2H - 4 \tag{3}$$

non-scalar multiplications. The minimal non-scalar complexity $\mathcal{O}(\sqrt{B})$ is achieved when $L \approx \sqrt{2(B+1)}$.

In fact, Paterson and Stockmeyer designed a slightly faster algorithm with the same asymptotic complexity. Unfortunately, it cannot be directly exploited in our work as it relies on the fact that the coefficient ring of an evaluated polynomial is a Euclidean domain. The coefficients of $P([\![y]\!])$ are plaintexts from the ring R_t, which is not Euclidean.

If the partitioning technique is used, then $P([\![y]\!])$ is replaced by α polynomials $P_i([\![y]\!]), i \in [\alpha]$ of degree $\lceil B/\alpha \rceil$. Select positive integers L_α, H_α such that $\lceil B/\alpha \rceil = L_\alpha H_\alpha - 1$. To evaluate each P_i, the sender precomputes L_α low powers and H_α high powers and computes each P_i as in eq. (2). Multiplication by the high powers is performed for each P_i, i.e., $\alpha(H_\alpha - 1)$ times. It implies that the non-scalar multiplicative complexity of evaluating every P_i is equal to

$$L_\alpha - 2 + H_\alpha - 1 + \alpha(H_\alpha - 1) = L_\alpha + (\alpha+1)H_\alpha - (\alpha+3). \tag{4}$$

Similar to eq. (3), the minimal non-scalar complexity $\mathcal{O}(\sqrt{B})$ is achieved by taking $L_\alpha \approx \sqrt{(\alpha+1)\left(\lceil B/\alpha \rceil + 1\right)}$.

Example 1. Consider a bin of size $B = 81$. Depending on the number of partitions, α, we can compute the intersection either with the Paterson-Stockmeyer method or with a naïve method, that precomputes all the powers $[\![y^2]\!], \ldots, [\![y^{\lceil B/\alpha \rceil}]\!]$. When $\alpha \leq 5$, the Paterson-Stockmeyer method requires fewer non-scalar multiplications, as demonstrated in the following table:

α	1	2	3	4	5	6	7
P.-S. mult.	23	18	16	14	14	15	16
Naïve mult.	80	40	26	20	16	13	11

In Appendix A, we show how to identify from α and B whether the Paterson-Stockmeyer method with partitioning outperforms the naïve method.

Windowing. In modern leveled FHE schemes, encryption parameters are set depending on the multiplicative depth of computation: higher multiplicative depth requires larger parameters. Unfortunately, larger parameters increase both the communication and computation complexity.

The multiplicative depth of the Paterson-Stockmeyer algorithm (as in eq. (2)) is equal to $\lceil \log((H-1)L+1) \rceil + 1$, which is at most one bigger than the depth of computing a polynomial of degree $B = LH - 1$. To reduce the depth, the receiver can send encryptions of additional precomputed powers of y. For example, if $[\![y]\!]$ and $[\![y^L]\!]$ are given to the sender, it can compute eq. (2) with a circuit of depth $\max\{\lceil \log(H-1) \rceil, \lceil \log(L-1) \rceil\} + 1$. Since the multiplicative complexity is minimal when $L > H$ (eq. (3)), the depth of computing the intersection polynomial (2) is equal to

$$\lceil \log(L-1) \rceil + 1 \approx \log \sqrt{2(B+1)} + 1 = \frac{\log(B+1)}{2} + \frac{3}{2}.$$

Sending more additional powers reduces multiplicative depth. The windowing technique, as described in [15], relies on the fact that any integer $B > 0$ can be represented uniquely in some base $b > 1$, namely $B = \sum_{i=0}^{\lfloor \log_b B \rfloor} B_i b^i$ with $B_i \in [0, b-1]$. This means that $y^B = \prod_{i=0}^{\lfloor \log_b B \rfloor} y^{B_i b^i}$. If all the powers y^{ib^j} with $i \in [b-1]$ and $j \in [0, \lfloor \log_b B \rfloor]$ are precomputed, then y^B can be obtained by a circuit of depth $\lceil \log(\lfloor \log_b B \rfloor + 1) \rceil$. As a result, the receiver can send encryptions of these $(b-1)(\lfloor \log_b B \rfloor + 1)$ additional powers such that the sender can compute all the powers $[\![y]\!]^2, \ldots, [\![y]\!]^B$ with the aforementioned depth.

In practice, it is convenient to fix a multiplicative depth D and derive b from it. Since the function $(b-1)(\lfloor \log_b B \rfloor + 1)$ is increasing with b, the smallest possible b supporting depth D results in the minimal number of ciphertexts sent from the receiver to the sender.

Let D be the target depth. This means that D should satisfy

$$D = \lceil \log \left(\lfloor \log_b B \rfloor + 1 \right) \rceil \geq \log \left(\lfloor \log_b B \rfloor + 1 \right).$$

Hence, we obtain $2^D - 1 \geq \lfloor \log_b B \rfloor > \log_b B - 1$ and thus $2^D > \log_b B$. As a result, b is the smallest integer satisfying $b > B^{2^{-D}}$, or $b = \lfloor B^{2^{-D}} + 1 \rfloor$. The number of powers that must be sent is therefore bounded by

$$(b-1)\left(\lfloor \log_b B \rfloor + 1 \right) \leq (b-1)\left(\log_b B + 1 \right) < B^{2^{-D}}(2^D + 1). \tag{5}$$

While using the Paterson-Stockmeyer algorithm, the sender should compute low and high powers and then multiply linear combinations of the low powers by the high powers as in eq. (2). This means that to achieve a target depth D while computing the intersection polynomial, the sender should be able to compute both sets of powers with depth at most $D - 1$.[6]

Following the discussion above, we obtain that the receiver should send $(b_L - 1)(\lfloor \log_{b_L}(L-1) \rfloor + 1)$ encryptions of powers of y for the sender to compute the low powers with the base $b_L = \lfloor (L-1)^{2^{-(D-1)}} + 1 \rfloor$. To compute the high powers with the same depth, the sender needs only $(b_H - 1)(\lfloor \log_{b_H}(H-1) \rfloor + 1)$ encryptions of powers of y^L with the base $b_H = \lfloor (H-1)^{2^{-(D-1)}} + 1 \rfloor$. The upper bound on the number of powers that the receiver needs to send is defined by the following lemma.

Lemma 1. *To compute a polynomial $P([\![y]\!])$ of degree $B > 0$ with the multiplicative depth $D \geq 0$ using the Paterson-Stockmeyer method, the sender needs fewer than $3(B+1)^{2^{-D}}(2^{D-1} + 1)$ powers of $[\![y]\!]$.*

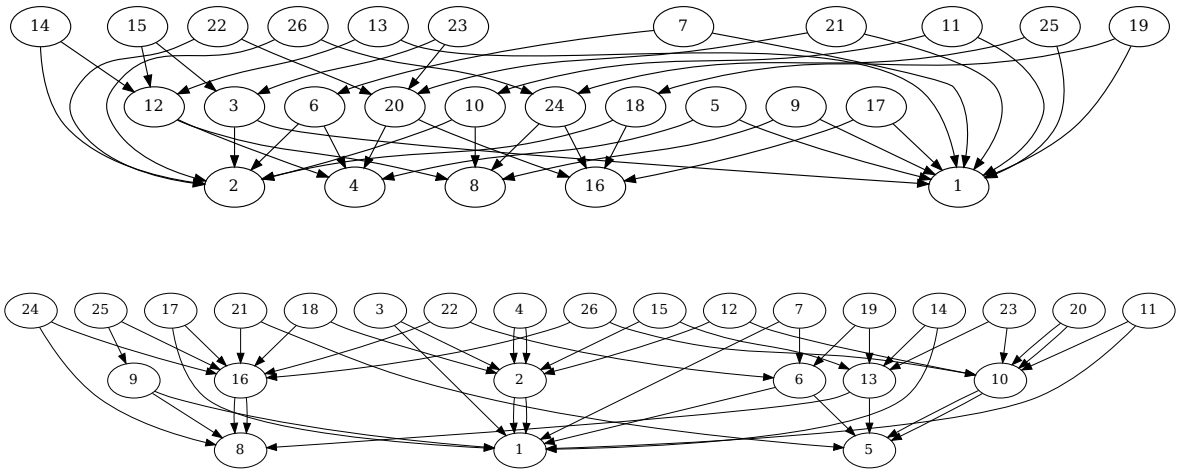
The proof can be found in Appendix B.

[6] The low powers need to be further scalar-multiplied by the polynomial coefficients, which increases noise comparably to the non-scalar multiplications for some parameterizations. Thus, in some cases we require that high powers are compted with depth $D - 1$ and low powers with depth $D - 2$.

Extremal postage-stamp bases. The windowing technique of [15] described above is easy to use: the receiver always knows exactly which powers to send. Unfortunately, it is far from optimal. To demonstrate this point, consider a case where $B = 26$. The user could choose to encrypt and send powers $\{1, 2, 4, 8, 16\}$ of their query, which the sender can use to compute all powers up to 26 in a depth-two computation, as illustrated by the first graph of Figure 1. However, the second graph of Figure 1 demonstrates an alternative computation, also of depth two, but with only three source powers: $\{1, 5, 8\}$. This immediately translates to a 40% reduction in the receiver-to-sender communication.

More generally, we would like to answer the question: which powers of the query should be sent so that the sender can compute all powers of the query up to as large of a bound B as possible, without exceeding a target depth.

Fig. 1. Graphs depicting two possible ways for the sender to compute all powers up to 26 of the receiver's query from a given set of source powers. The two arrows pointing out from a node indicate which lower powers need to be multiplied together to produce the power indicated in the node label.



This problem can be viewed as a variant of the *global postage-stamp problem* [11,12]:

Definition 1 (Global postage-stamp problem). *Given positive integers h and k, determine a set of k positive integers $A_k = \{a_1 = 1 < a_2 < \ldots < a_k\}$ such that all integers $1, 2, \ldots, n$ can be written as a sum of h or fewer of the a_j, and n is as large as possible. The set A_k is called an extremal postage-stamp basis.*

The connection to our problem is clear. In the notation of Definition 1, if the receiver sends encrypted powers $\{[\![y^{a_1}]\!], [\![y^{a_2}]\!], \ldots, [\![y^{a_n}]\!]\}$ to the sender, then the sender can compute all powers up to $B = n$ in multiplicative depth $\lceil \log_2 h \rceil$. Concretely, consider the powers $\{1, 5, 8\}$ used in Figure 1. Upon receiving $\{[\![y]\!], [\![y^5]\!], [\![y^8]\!]\}$, the sender iterates (in order) over all integers up to $B = 26$, and for each power that it has not yet computed (or received), it chooses a depth-optimal way of computing it as a product of two lower powers. This is exactly how the graphs in Figure 1 were generated. In fact, the basis $\{1, 5, 8\}$ is a unique extremal postage-stamp basis for $h = 4$, $k = 3$ [11].

No simple way of finding extremal postage-stamp bases is known, nor is the complexity class of the global postage-stamp problem known. Furthermore, extremal solutions are often unique (or

almost unique) and quickly become hard to find. Fortunately we only need solutions for small instances of the problem, which have been brute-forced and are presented in [12].

Extremal postage-stamp bases can be used in two ways with the Paterson-Stockmeyer algorithm. Recall that in this case the sender must compute all powers of the receiver's query up to some positive integer $L-1$, and all powers that are multiples of L not exceeding the bin size B.

Naturally, an extremal postage-stamp basis with $n = L-1$ can be used to achieve the first goal. For enabling the sender to compute as many powers of L as possible from as few source powers as possible, the receiver can apply a (possibly different) extremal postage-stamp basis, but this time multiply the exponents by L.

For example, consider again the extremal postage-stamp basis $\{1, 5, 8\}$ in Figure 1. This works great for Paterson-Stockmeyer, when $L-1 = 26$. To use Paterson-Stockmeyer, we could additionally send powers $\{L, 5L, 8L\} = \{27, 135, 216\}$, which would allow the server to compute polynomials up to degree $26L + 26 = 728$ with a depth 3 circuit.

Extremal postage-stamp bases with large h (like 2^3 or 2^4) can be hard to find, but if found, such bases allow very high-degree polynomials to be evaluated without increasing the depth, but possibly with a significant increase in online computation time due to the large number of powers needing to be computed through non-scalar multiplications.

Low depth exponentiation via the Frobenius operation. As mentioned in Section 2.3, the Frobenius operation is a cheaper alternative to non-scalar multiplication in terms of running time and added noise. Using this operation, the sender can compute powers of $[\![y]\!]$ saving multiplicative depth. Hence, the receiver can send fewer powers, decreasing the communication complexity.

Example 2. Take a plaintext modulus $t = 2$; every SIMD slot is isomorphic to $\mathbb{F}_{2^d}$ for some d and the Frobenius operation can compute $[\![x]\!] \mapsto [\![x^{2^i}]\!]$ for $i \in [d-1]$.

Suppose the sender has 255 values in its set. This means that it should compute the intersection polynomial $P([\![y]\!])$ of degree 255. Using the Paterson-Stockmeyer algorithm, it needs $[\![y^i]\!]$ and $[\![y^{16i}]\!]$ for $i \in [15]$ to compute $P([\![y]\!])$ with depth 1. In total, the receiver has to send 30 ciphertexts encrypting these powers.

However, the sender can compute $[\![y^{16i}]\!]$ from $[\![y^i]\!]$ by applying the Frobenius operation $[\![x]\!] \mapsto [\![x^{2^2}]\!]$, which is depth-free. Moreover, any even power $[\![y^{2^e a}]\!]$ with odd a can be obtained from $[\![y^a]\!]$ with the Frobenius operation $[\![x]\!] \mapsto [\![x^{2^e}]\!]$. As a result, the receiver needs to send only 8 encrypted powers, namely $[\![y]\!], [\![y^3]\!], [\![y^5]\!], [\![y^7]\!], [\![y^9]\!], [\![y^{11}]\!], [\![y^{13}]\!]$ and $[\![y^{15}]\!]$. Having these powers, the sender computes all the low and high powers for the Paterson-Stockmeyer algorithm with 22 Frobenius operations and then performs only 15 non-scalar multiplications to compute $P([\![y]\!])$ with a depth 1 circuit.

The communication complexity can be reduced even further at the cost of increased depth. If the receiver sends only $[\![y]\!]$, then the sender first computes $[\![y]\!], [\![y^2]\!], [\![y^4]\!], [\![y^8]\!]$ with depth 0 using the Frobenius operations and then obtains the powers $[\![y^3]\!] = [\![y]\!] \cdot [\![y^2]\!]$, $[\![y^5]\!] = [\![y]\!] \cdot [\![y^4]\!]$, $[\![y^7]\!] = [\![y]\!] \cdot [\![y^2]\!] \cdot [\![y^4]\!]$, $[\![y^9]\!] = [\![y]\!] \cdot [\![y^8]\!]$, $[\![y^{11}]\!] = [\![y]\!] \cdot [\![y^2]\!] \cdot [\![y^8]\!]$, $[\![y^{13}]\!] = [\![y]\!] \cdot [\![y^4]\!] \cdot [\![y^8]\!]$, $[\![y^{15}]\!] = [\![y]\!] \cdot [\![y^2]\!] \cdot [\![y^4]\!] \cdot [\![y^8]\!]$ with multiplicative binary-tree circuits of depth at most 2. The remaining low and high powers are again computed with Frobenius operations. Thus, the sender can compute $P([\![y]\!])$ with depth 3 having been given only one ciphertext, $[\![y]\!]$.

The idea above is formalized in Lemma 2 that indicates how many consecutive powers of $[\![y]\!]$ can be computed by the sender with a depth D circuit. This lemma implies that if $P([\![y]\!])$ has degree B, then the sender needs only $\mathcal{O}(1)$ encrypted powers of y to compute $[\![y^2]\!], \ldots, [\![y^B]\!]$ with a circuit of multiplicative depth $\mathcal{O}(\log \log B)$. In the prior work [14], $\mathcal{O}(\log B)$ powers are needed to perform the same task.

Lemma 2. *Let $[\![y]\!]$ be a ciphertext encrypting a plaintext message from R_t. Let D be a positive integer. Then, using a depth D circuit, the sender can compute all the powers $[\![y]\!], [\![y^2]\!], \ldots, [\![y^{e_D}]\!]$ where*

$$e_D = \left(\left(2^D \mod (t-1)\right) + 2\right) \cdot t^{\left\lfloor \frac{2^D}{t-1} \right\rfloor} - 2.$$

The proof of this lemma can be found in Appendix C.

Example 3. Let $t := 2$. In this case $2^D \mod (t-1) = 0$, so $e_D = 2^{2^D+1} - 2$. Since $e_1 = 6$, the sender can compute $\{[\![y^2]\!], \ldots, [\![y^6]\!]\}$ with a circuit of depth 1, which is supported by the second part of Example 2.

Example 4. Let $t := 3$. Since $2^D \mod (t-1) = 0$ for any $D > 0$, $e_D = 2 \cdot 3^{2^{D-1}} - 2$.

This technique requires the receiver to send additional evaluation keys to perform Frobenius operations. In particular, the receiver should send $\lfloor \log_t B \rfloor$ evaluation keys to the sender to compute powers $[\![y^2]\!], \ldots, [\![y^B]\!]$. Furthermore, the receiver can send only one evaluation key corresponding to the basic Frobenius operation $[\![x]\!] \mapsto [\![x^t]\!]$. The size of the evaluation keys can be significant (see Section 5 for more details), but they can be cached by the sender and used repeatedly for multiple executions of the protocol.

In practice, the advantage of the Frobenius operation is at odds with the SIMD packing capacity. In general, the packing capacity is equal to n/d where d is the order of the plaintext modulus t modulo the order m of the ring R, i.e $m | t^d - 1$. This implies that $d > \log_t m$ and thus $n/d < n \cdot \log_m t$. Hence, a smaller plaintext modulus yields a smaller packing capacity, but it results in a better multiplicative depth due to Lemma 2.

Fast OPRF from Four$\mathbb{Q}$ The OPRF stage is essential to our protocol, as was pointed out in Section 3.1. The sender's task is significant: it needs to choose a random number modulo the OMGDH-hard group order to act as the OPRF key, hash each of its items into a uniformly random group element, and multiply the group element with the key.

For performance reasons, we use the Four$\mathbb{Q}$ curve [18] for the group, which provides fast scalar multiplication of random points, as well as a fast implementation of hash-to-curve as follows. We apply the birational map $(u, v) \mapsto (u/v, (u-1)/(u+1))$ to modify the Elligator 2 [5] construction. A naive inspection of the required $\mathbb{F}_{p^2}$ operations suggests that this needs at least four exponentiations in $\mathbb{F}_p$, but a careful combination of the tricks from the literature (see [57]) allowed the full map to the curve to be achieved with just 3 field exponentiations (all to the power of $(p-3)/4$), and a handful of additional operations in $\mathbb{F}_p$. A mini-scalar multiplication by the cofactor moves these hashed points inside the large prime order subgroup.

We measured our implementation to take around $12,000$ clock cycles for the the hash-to-curve function, and $42,000$ clock cycles for the scalar multiplication. This is faster than GLS-254, a curve optimized for the unbalanced PSI application by Aranha and Resende [55], which takes around $50,000$ clock cycles for a scalar multiplication.

4 Labeled PSI

4.1 Labeled PSI in Chen *et al.*, 2018

In labeled PSI, the sender holds a bytestring L_i, called a *label*, of length ℓ for each of its items $x_i \in X$. The receiver learns the corresponding label L_i for each of its items $y_j = x_i \in X \cap Y$ in

the intersection. The basic idea, as introduced in [14], is for the sender to construct interpolation polynomials

$$G(y) = \begin{cases} L_i & \text{if } y = x_i; \\ \text{random field element} & \text{otherwise.} \end{cases}$$

The sender evaluates the polynomial on the receiver's encrypted input exactly like it evaluates the intersection polynomials, and sends the result back to the receiver.

It remains to construct the polynomials $G(y)$, however, the construction in [14] relies on the fact that the intepolation is done over extension fields, where non-zero field elements can be fully randomized by multiplying with a non-zero random field element.

4.2 Improvements

The above approach does not work immediately for our construction, because we break our items into prime-field element sized chunks, and compute the intersection polynomial on them independently. Thus, a partial match, which may be easy to brute-force due to the small size of the prime fields, would immediately leak the corresponding part of the label.

Fortunately, [14] already provides a partial solution to this problem: since we have to use OPRF in any case, we can extend the output of the OPRF and use one part of the OPRF value to represent the item in the intersection polynomial, and another part as a key for a symmetric cipher to encrypt the label. Then, instead of interpolating over parts of the label, we interpolate over parts of the encrypted label (see Figure 2 for an example). The receiver will be able to decrypt the result after extracting the key from the full OPRF value for the item. This technique ensures that only the receivers that know the item in the intersection can decrypt the label corresponding to that item.

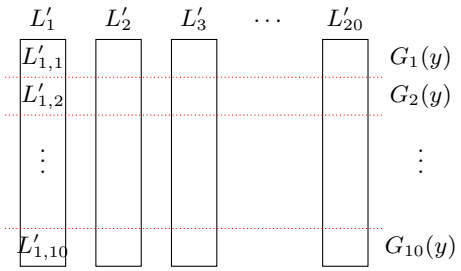


Fig. 2. In this example, there are 20 encrypted labels $\{L'_1, \ldots, L'_{20}\}$, every label is split into 10 parts. The red dotted lines indicate the different parts. A label polynomial $G_i(y)$ is created for every part.

One major downside of this approach is the $\mathcal{O}(B^2)$ complexity of interpolation in field operations. Using prime-fields yields a substantial improvement over the extension fields of [14], but the cost is still unfortunately high for practical parameters. To address this issue, we note that the precomputation is easily updatable: items can be added or removed, and labels can be updated. This presents a problem when the symmetric cipher is an XOR stream cipher: if the label for an item is changed and the encryption key and nonce remain unchanged, then even a semi-honest receiver can find the XOR of the labels (or label parts) before and after the change. One simple solution is to use a strictly incrementing nonce for the encryption, but this can be challenging to maintain in practice. Another solution is to always choose a random nonce for each label on each update, and append it to be a part of the label data. This way the receiver always obtains the correct nonce and can use it to decrypt the label itself. In practice, the label data, especially when

appended with a random nonce, is likely to be longer than the item. In this case the label data is simply broken up into item-size fragments and a separate interpolation polynomial is formed for each fragment. The receiver will then obtain a separate result for each fragment, reconstruct the encrypted label, and finally decrypt it.

It was pointed out in [14] that returning the intersection result to the receiver in addition to the label result is not strictly speaking necessary, but it may be necessary in practice for the receiver to know which label values hold valid data. Thus, we consider the labeled PSI protocol to return both the intersection result as well as one or more label results, depending on whether the label consists of a single or multiple item-length fragments.

Resolving a subtle issue Unfortunately, there is one remaining technicality to address with the above approach. To understand the problem, note that [14] performed interpolation over large extension fields, where an extension field element encoded an entire item. As items never repeat, interpolation would always succeed. Since we suggest to use a separate interpolation polynomial for each prime-field size part of the label, interpolation is done over a much smaller field, where it is easy to end up with item-part collisions: a single partition (recall Section 3.1) can end up with repeating item parts in the same bin. If the corresponding label parts do not match, as is likely to be the case, interpolation is impossible.

To resolve this issue, the sender must check, when inserting an item, that none of its parts do not already appear in the same locations in the partition in which the item is being inserted. If a collision is encountered, the sender must either try to insert the item in a different partition, or as a last resort create an entirely new partition where the troublesome item can be inserted.

The procedure described above results in an unfortunate phenomenon, where labeled PSI produces more partitions and with a lower 'fill rate' for the sender's data structures compared to unlabeled PSI.

Security While [14] did not explicitly discuss a security proof for the labeled case, we note that the ideal functionality ([14], Figure 3) and the security proof of the unlabeled case ([14], Theorem 1) are straightforward to extend to the labeled case.

In the ideal functionality the sender inputs both its set X and a set of labels $\{L_x \in \{0,1\}^\ell \mid x \in X\}$, and the receiver learns $\{(x, L_x) \mid x \in X \cap Y\}$. A simulator playing the role of a sender uses the random oracle to extract the malicious receiver's input Y^* from the OPRF queries it made. Then Y^* is given to the ideal functionality that responds with $\{(x, L_x) \mid x \in X \cap Y^*\}$; the simulator pads this to size $|X|$ with random pairs (x, L), where $x \in X \setminus Y^*$ and $L \in \{0,1\}^\ell$, encrypts the label values corresponding to the receiver's input with keys derived from the corresponding OPRF values, and creates the response ciphertext data. The proof is completed exactly as in Theorem 1 of [14].

5 Experiments

We tested the aforementioned techniques in two C++ libraries, HElib [32] and SEAL [58], which implement the BGV [8] and BFV [23] schemes, respectively. The encryption parameters used in this section support at least 128 bits of security unless marked otherwise (see Tables 5 and 6). The benchmarks are performed on an Intel Xeon Platinum 8272CL CPU @ 2.60 GHz, with 24 physical cores and 192 GB of RAM. We assume this is comparable to the 32-core machine used in [14]. We compiled SEAL with support for Intel HEXL [6].

The rationale behind two implementations is the following. The SEAL implementation helps us provide an adequate comparison with the prior works [15,14]. However, SEAL supports a limited set

of encryption parameters that obstructs the low-depth exponentiation technique from Section 3.1 — our most powerful tool for reducing the communication cost. In particular, the order of the ring R in SEAL is fixed such that t^d must be congruent to 1 modulo a power of two greater than or equal to 2^{12}. The multiplicative order d of the plaintext modulus must be relatively small, i.e., $80 \leq d \log t \leq 100$, to allow encoding of 80-bit strings with maximal SIMD capacity. We found that there are no such primes t in the range $[2, 3000]$. For larger t, the low-depth exponentiation technique is almost useless in practice, as the Frobenius operation yields a very sparse set of powers. To solve the above issue, we resort to the HElib library, where the cyclotomic ring R can be of any order.

In the SEAL-based implementation, we aim to show how our optimizations reduce both the running time and the communication cost over prior work, in particular [14]. This setting exploits all the techniques of Section 3.1, except for the low-depth exponentiation method. In the HElib implementation, we focus solely on the communication cost and rely on the low-depth exponentiation, the SIMD packing, and the Paterson-Stockmeyer methods.

Unlike the SEAL version, the HElib version is a proof-of-concept implementation, written only to demonstrate the low-depth exponentiation technique used to reduce the communication cost. Namely, we do not implement OPRF or networking. Nevertheless, we are able to accurately compute the communication cost since the cost of OPRF is fixed (32 bytes) per item in the receiver's set and the ciphertext sizes can be computed in HElib. We omit the computation cost of OPRF in our presentation (Appendix D) since it is harder to estimate accurately and our focus is on the communication cost.

Our SEAL-based protocol is implemented in an open-source library.[7] The protocol parameters for all experiments are included with the library to make reproducing the results easy. However, we note that there are numerous alternative ways to parameterize the protocol that may in some cases be better than what we present in our results, depending on what one is optimizing for. We chose to present results from parameterizations that in our opinion best reflect the improvements over prior work, generally optimizing more for computation than communication.

5.1 SEAL Implementation: Unlabeled Mode

The computation and communication cost of our SEAL implementation running in the unlabeled mode is given in Table 1. For each pair $(|X|, |Y|)$ we present only one result, but in reality the situation is not that simple: the protocol involves rather complex parameterization with many communication-computation trade-offs. The results we present demonstrate one setting for each size of the problem that we felt captured the overall performance best, but it is always possible to reduce the running time by increasing communication, and *vice versa*.

Table 1 includes a comparison for single-threaded execution with Chen *et al.* [14], including details for sender-to-receiver and receiver-to-sender communication. Furthermore, in Table 2 we present another a side-by-side comparison with Chen *et al.* [14], Kales *et al.* [37], and Aranha and Resende [55], which represent different state-of-the-art protocols for unbalanced PSI.

Regarding Chen *et al.* [14], it is clear that our protocol is much faster, especially for larger parameters, and has a smaller communication cost. For example, with $|X| = 2^{24}$ and $|Y| = 11041$ our computation-communication trade-offs allowed us to reduce communication by 65% and online computation by 26%. With $|X| = 2^{28}$ and $|Y| = 2048$, we reduced communication by 32% and computation by more than 83%.

The comparison between our work and Kales *et al.* [37] is more nuanced because the protocols and applications are different. For the protocol aspect, the main difference is that Kales *et al.* has

[7] https://GitHub.com/Microsoft/APSI

Set sizes		Sender offline (s)				Sender online (s)				Comm. (MB)	
$\|X\|$	$\|Y\|$	T=1	T=4	T=8	T=24 (T=32)	T=1	T=4	T=8	T=24	R → S	S → R
2^{28}	4096	-	-	-	2,131	71.2	18.6	10.0	5.1	8.3	9.5
	2048	-	-	-	2,211 (4,638)	66.7	17.5	9.4	4.8 (28.5)	5.8 (11.8)	9.3 (10.2)
	1024	-	-	-	2,487 (4,638)	63.1	16.5	8.9	4.5 (12.1)	3.4 (8.2)	9.1 (10.2)
	1	-	-	-	1,549	29.2	8.0	4.5	2.0	2.6	1.8
2^{24}	4096	864 (806)	245	147	84.3	11.0 (22.0)	3.0	1.6	0.8	5.1 (8.7)	2.5 (7.2)
	2048	871 (747)	256	157	93.2	9.8 (12.6)	2.6	1.4	0.7	2.9 (8.2)	2.3 (8.1)
	1024	873 (1,430)	271	171	108	9.1 (17.7)	2.4	1.3	0.6	1.8 (3.1)	2.1 (5.1)
	1	453	134	80.9	46.9	3.5	1.0	0.7	0.4	1.6	0.9
2^{20}	4096	29.8 (43)	9.4	5.8	4.9	2.34 (4.2)	0.68	0.37	0.26	3.4 (4.2)	2.1 (7.2)
	2048	29.1 (39)	9.1	5.6	4.1	2.59 (2.1)	0.71	0.40	0.24	2.6 (2.9)	1.5 (3.6)
	1024	28.0 (40)	8.8	5.6	4.0	2.10 (2.0)	0.57	0.32	0.17	1.3 (2.5)	1.1 (2.5)
	1	23.7	7.0	4.2	3.2	0.04	0.02	0.01	0.01	0.75	0.80

Table 1. Computation and communication cost of our SEAL implementation. 'T' denotes the thread count, and 'R→S' and 'S→ R' the receiver-to-sender and sender-to-receiver communication sizes, respectively. The times are averaged over 10 runs of the protocol. Corresponding values from Chen *et al.* (CCS'18), for single-threaded execution are in parentheses, except for the case of $|X| = 2^{28}$, where Chen *et al.* (CCS'18) uses 32 threads and we compare against our execution on 24 threads.

a significant sender-side preprocessing step and requires a possibly large cuckoo filter ($O(|X|)$) to be communicated ahead of time from the sender to the receiver, while our protocol has no such offline communication. On the other hand, the computation cost of Kales *et al.* is better in the online phase, as well as the preprocessing phase, since they do not need to perform any costly homomorphic encryption operations. In terms of applications, Kales *et al.* [37] focuses on mobile contact discovery, where the receiver is a low-powered mobile device, but our protocol does not target a specific application. For this reason Kales *et al.* may have selected protocols that require more online communication, e.g., GC-based OPRF, to reduce the computation cost of the mobile device in the online phase. Nevertheless, we would like to highlight that our communication complexity in the online phase is either comparable or a lot better than the two protocols in [37].

What cannot be seen from the table is that the cuckoo filter also causes Kales *et al.* [37] to have a non-negligible false-positive probability. In their experiments, they parameterized the cuckoo filter to have a false-positive probability of 2^{-29}. For example, if $|Y| = 11041$, this results in a probability of around $2^{-15.6}$ for the receiver to see at least one false positive result, so at least one invocation of the protocol out of less than $49,000$ invocations would report a false positive. While in some applications this may be acceptable, in others it may not be. On the other hand, we follow [14,15] and parameterize our protocol to have a false-positive probability of at most 2^{-40} per execution.[8] Our performance would be significantly improved if we allowed for a higher false-positive probability.

As Kales *et al.* [37] and Chen *et al.* [14], we also include a comparison to the work or Aranha and Resende [55]. However, we note that [55] uses extremely aggressive cuckoo filter parameters, resulting in online performance far better than any other protocol, but with an impractically high false-positive rate of 2^{-13}. Despite the aggressive cuckoo filter parameters, [55] suffers from the high offline communication complexity when the sender's set is large.

[8] In other words, this takes into account that the receives queries multiple items at a time per each execution, and each of the items may independently yield a false positive.

$\|X\|$	$\|Y\|$	**Protocol**	**Sender offline (s)**	**Offline comm. and receiver storage (MB)**	**Sender online (s)**	**Online comm. (MB)**
2^{28}	2048	[14] (T=32)	4,628	0	28.5	22.28
		Ours (T=24)	2,211	0	4.75	15.09
	1024	[55] (T=32)	182	806	0.16	0.07
		LowMC-GC-PSI [37]	1,869	1,072	0.93	24.01
		ECC-NR-PSI [37]	52,332	1,072	1.34	6.06
		[14] (T=32)	4,628	0	12.1	18.57
		Ours (T=24)	2,487	0	4.54	12.56
2^{24}	11041	[55]	342	48	0.71	0.67
		LowMC-GC-PSI [37]	117	67	12.51	258.79
		ECC-NR-PSI [37]	3,298	67	11.94	65.24
		[14]	656	0	20.10	41.48
		Ours	832	0	14.90	14.58
	5535	[55]	342	48	0.35	0.34
		LowMC-GC-PSI [37]	117	67	5.63	129.73
		ECC-NR-PSI [37]	3,298	67	5.93	32.71
		[14]	806	0	22.01	16.39
		Ours	871	0	11.77	8.49
2^{20}	11041	[55]	22	3	0.71	0.67
		LowMC-GC-PSI [37]	7.3	4.2	12.51	258.79
		ECC-NR-PSI [37]	242	4.2	11.94	65.24
		[14]	43	0	4.47	14.34
		Ours	28	0	2.46	8.95
	5535	[55]	22	3	0.35	0.34
		LowMC-GC-PSI [37]	7.3	4.2	5.63	129.73
		ECC-NR-PSI [37]	242	4.2	5.93	32.71
		[14]	43	0	4.23	11.50
		Ours	27	0	1.91	5.39

Table 2. Comparison to prior work. All executions are with a single thread, except those where the thread count is explicitly marked with T=thread count.

5.2 Very Large Senders

It was pointed out by Kales *et al.* [37] that in practical applications, such as mobile contact discovery, it is realistic to encounter sender's set sizes of more than a billion. While our protocol easily extends to this case, running our implementation on such a large set requires significantly more RAM than our system had available, resulting in extensive paging and a massive slowdown in online computation.

Therefore, for very large senders a more viable solution is to partition the dataset into smaller parts and run the protocol multiple times with smaller parameters. We note that the receiver-to-sender communication needs to be done only once, so for example running our protocol against a sender with $|X| = 2^{31}$ would require only $3.4 + 8 \times 9.1 = 76.2$ MB of communication, with $|Y| = 1024$, whereas [37] would require a cuckoo filter of ≈ 8 GB to be communicated from the sender to the receiver, which is clearly impractical.

For very large senders, instead of using the parameterizations used to produce Table 1, it may be beneficial to use parameterizations that minimize the sender-to-receiver communication, potentially at the expense of increased computational cost.

5.3 SEAL Implementation: Labeled Mode

We demonstrate a few examples in the labeled mode as well. The label size matters a lot, namely, recall from Section 4.2 that if labels are longer than the item length (more correctly, the length of the OPRF value used to represent the item), then the label must be broken into multiple item-length chunks and separate interpolation polynomials must be formed, and later evaluated, per each chunk. Since the same encrypted query data is used to evaluate the interpolation polynomials for each chunk, the label size has no direct impact on the receiver-to-sender communication. However, the sender-to-receiver communication increases linearly with the number of chunks. Thus, if the labels are very long, it is beneficial to parameterize the protocol in a way that minimizes the sender-to-receiver communication.

Unfortunately there are not many meaningful points of comparison. Chen *et al.* [14] presented a single datapoint comparing to a PIR paper [3], where the label size was $\ell = 288$ B. We replicated this experiment using our optimizations; the results are presented in Table 3. Our protocol clearly outperforms both points of comparison in all measured aspects. More examples for label sizes $\ell = 16$ B and $\ell = 32$ B are in Table 4.

ℓ	$\lvert X\rvert$	$\lvert Y\rvert$	**Protocol**	**Sender online (s)**	**Client encrypt (s)**	**Comm. (MB)**
288	2^{20}	256	[3]	20.5	4.92	120
			[14]	4.6	0.77	17.6
			Ours	0.66	0.05	11.2

Table 3. Comparison to prior work in the labeled mode for label byte-length $\ell = 288$. The sender uses 16 threads and the receiver a single thread.

5.4 HElib Implementation: Optimizing for Communication Complexity

Our proof-of-concept implementation in HElib aims to illustrate that unbalanced PSI can achieve a sublogarithmic communication complexity in $|X|$. We conducted experiments with X and Y containing elements of arbitrary bit length. These elements are hashed to 80-bit strings in the

Set sizes			Sender offline (s)	Sender online (s)				Comm. (MB)	
ℓ	$\lvert X\rvert$	$\lvert Y\rvert$	T=24	T=1	T=4	T=8	T=24	R $\rightarrow$ S	S $\rightarrow$ R
32	2^{22}	4096	882	11.5	3.7	2.0	1.8	5.1	4.2
		1	198	4.4	1.6	1.5	1.5	1.6	1.7
	2^{20}	4096	174	5.3	1.5	1.1	1.1	4.2	3.1
		1	18.5	1.7	0.46	0.29	0.21	1.2	1.3
16	2^{22}	4096	467	9.4	2.9	1.7	1.4	5.1	3.2
		1	114	3.0	1.0	0.98	0.92	1.6	1.0
	2^{20}	4096	114	4.3	1.2	0.92	0.86	4.2	2.1
		1	10.9	1.6	0.42	0.25	0.17	1.2	0.79

Table 4. Computation and communication cost of our SEAL implementation in the labeled setting. The label byte-size is ℓ while the number of threads is denoted by T. The times are averaged over 10 runs of the protocol.

cuckoo hashing stage of the protocol. The set sizes are $|X| \in \{2^{20}, 2^{22}, 2^{24}, 2^{26}\}$ for the sender and $|Y| \in \{126, 210, 341, 558, 1245\}$ for the receiver. To get the smallest possible communication cost, we avoid partitioning (i.e., we take $\alpha = 1$). Neither the windowing method, nor the extremal postage-stamp bases are exploited.

As mentioned in Section 3.1, before executing our PSI protocol the receiver sends $O(\log |X|)$ evaluation keys to the sender. The number of keys can be reduced to one, at the cost of additional Frobenius operations. Since the evaluation keys are independent of Y, they can be sent only once and cached for repeated executions of the protocol. Thus, this communication overhead Table 5 can be amortized over multiple receiver's requests.

	Key size (MB)			
$\lvert Y\rvert$	$\lvert X\rvert = 2^{20}$	2^{22}	2^{24}	2^{26}
1245*	5.72	6.88	7.50	8.12
1024 [14]	1.05	-	2.11	-
558	5.49	6.18	6.87	8.13
512 [14]	0.43	-	0.43	-
341	7.71	9.90	11.1	12.2
256 [14]	0.43	-	0.21	-
210	6.22	7.44	8.31	9.17
128 [14]	0.03	-	0.03	-
126	7.97	9.02	10.6	-

Table 5. Evaluation key size of our HElib implementation and of the prior work [14] in the offline stage. The security level of all the parameters used in these experiments is at least 128 bits except for the parameters with '*' where it is at least 106 bits.

The communication cost of our PSI protocol is shown in Table 6. For comparison, we also included the communication cost of [14]. Note that in [14], evaluation keys and ciphertexts are generated in the symmetric-key mode, which allows us to almost halve their size. Unfortunately, we do not have access to this mode in HElib, but we requested the corresponding communication size [14] in the public-key mode from the authors.

As shown in Table 6, the communication cost of our implementation grows very slowly with the sender's set size. For example, it remains constant for $|Y| \in \{210, 341, 1245\}$ from $|X| = 2^{22}$ to $|X| = 2^{26}$. Furthermore, a 64-fold increase of the sender's set results in a 6-20% larger communication size for all the parameters we tested. This is a much smaller growth in comparison to [14], where a

16-fold increase of $|X|$ leads to a 40-291% rise in communication. More results with corresponding encryption parameters and running time can be found in Appendix D.

	Communication (MB)			
$\|Y\|$	$\|X\| = 2^{20}$	2^{22}	2^{24}	2^{26}
1245*	2.09	2.28	2.28	2.28
1024 [14]	6.45	-	9.02	-
558	1.27	1.27	1.27	1.36
512 [14]	5.01	-	10.64	-
341	1.10	1.32	1.32	1.32
256 [14]	4.73	-	13.58	-
210	0.72	0.76	0.76	0.76
128 [14]	4.69	-	18.32	-
126	0.63	0.63	0.66	-

Table 6. Communication cost of our HElib implementation and of the prior work [14] in the online stage. The security level of all the parameters used in these experiments is at least 128 bits except for the parameters with '*' where it is at least 106 bits.

6 Conclusions

We have demonstrated several improvements to the protocol of [14], reducing the communication and online computation cost significantly. Our improvements enable very powerful communication-computation trade-offs that can be leveraged to make the protocol practical and scalabe in various scenarios. Finally, we showed that homomorphic encryption can be used to enable PSI in the unbalanced setting, with sublogarithmic communication cost in the larger set, although this protocol is not practical to use today. In the future, hardware acceleration to homomorphic encryption may change the situation.

Acknowledgments

We would like to thank Craig Costello and Patrick Longa (Microsoft Research) for significant help and advise regarding the hash-to-curve algorithm for the FourQ curve and support with the FourQlib library, as well as Hao Chen (Facebook) for helpful discussions in preliminary phases of this work.

This work is supported by CyberSecurity Research Flanders with reference number VR20192203. Additionally, the first author is supported by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract number FA8750-19-C-0502. The third author is supported by ERC Advanced Grant ERC-2015-AdG-IMPaCT and by the Flemish Government through FWO SBO project SNIPPET S007619N. The fifth author is supported by a Junior Postdoctoral Fellowship from the Research Foundation – Flanders (FWO).

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of CyberSecurity Research Flanders, DARPA, the US Government, the ERC or the FWO. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

Finally, we would like to thank the anonymous reviewers for their helpful comments.

References

1. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Mathematical Cryptology **9**(3), 169–203 (2015), http://www.degruyter.com/view/j/jmc.2015.9.issue-3/jmc-2015-0016/jmc-2015-0016.xml
2. Ali, J.: Validating leaked passwords with k-anonymity. https://blog.cloudflare.com/validating-leaked-passwords-with-k-anonymity/ (2018), accessed: 2021-04-26
3. 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)
4. Ateniese, G., De Cristofaro, E., Tsudik, G.: (If) size matters: Size-hiding private set intersection. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 156–173. Springer, Heidelberg (Mar 2011). https://doi.org/10.1007/978-3-642-19379-8_10
5. Bernstein, D.J., Hamburg, M., Krasnova, A., Lange, T.: Elligator: Elliptic-curve points indistinguishable from uniform random strings. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 967–980 (2013)
6. Boemer, F., Kim, S., Seifu, G., de Souza, F.D., Gopal, V., et al.: Intel HEXL (release 1.2). https://github.com/intel/hexl (2021)
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. Challis, M.F.: Two new techniques for computing extremal h-bases ak. The Computer Journal **36**(2), 117–126 (1993)
12. Challis, M.F., Robinson, J.P.: Some extremal postage stamp bases. Journal of Integer Sequences **13**(2), 3 (2010)
13. Chase, M., Miao, P.: Private set intersection in the internet setting from lightweight oblivious PRF. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2020, Part III. pp. 34–63. LNCS, Springer, Heidelberg (Aug 2020)
14. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled PSI from fully homomorphic encryption with malicious security. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 1223–1237. ACM Press (Oct 2018). https://doi.org/10.1145/3243734.3243836
15. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1243–1255. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3134061
16. 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)
17. Chor, B., Gilboa, N., Naor, M.: Private information retrieval by keywords. Citeseer (1997)
18. Costello, C., Longa, P.: Fourq: four-dimensional decompositions on a q-curve over the mersenne prime. Cryptology ePrint Archive, Report 2015/565 (2015), https://eprint.iacr.org/2015/565
19. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Transactions on Information Theory **22**(6), 644–654 (1976)
20. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013. pp. 789–800. ACM Press (Nov 2013). https://doi.org/10.1145/2508859.2516701
21. Ducas, L., Micciancio, D.: FHEW: Bootstrapping homomorphic encryption in less than a second. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 617–640. Springer, Heidelberg (Apr 2015)
22. Fan, B., Andersen, D.G., Kaminsky, M., Mitzenmacher, M.D.: Cuckoo filter: Practically better than bloom. In: Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies. p. 75–88. CoNEXT '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2674005.2674994
23. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144 (2012), http://eprint.iacr.org/
24. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 303–324. Springer, Heidelberg (Feb 2005)
25. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (May 2004). https://doi.org/10.1007/978-3-540-24676-3_1

26. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC. vol. 9, pp. 169–178 (2009)
27. Gentry, C., Halevi, S., Smart, N.P.: Homomorphic evaluation of the AES circuit. In: Advances in Cryptology–CRYPTO 2012, pp. 850–867. Springer (2012)
28. 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
29. Hagen, C., Weinert, C., Sendner, C., Dmitrienko, A., Schneider, T.: All the numbers are US: Large-scale abuse of contact discovery in mobile messengers. In: 28th Annual Network and Distributed System Security Symposium, NDSS. The Internet Society (2021)
30. Halevi, S., Shoup, V.: Design and implementation of helib: a homomorphic encryption library. Cryptology ePrint Archive, Report 2020/1481 (2020), https://eprint.iacr.org/2020/1481
31. 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)
32. HElib: An implementation of homomorphic encryption (2.1.0). https://github.com/homenc/HElib (Mar 2021), iBM
33. Huang, Y., Evans, D., Katz, J.: Private set intersection: Are garbled circuits better than custom protocols? In: NDSS 2012. The Internet Society (Feb 2012)
34. Huberman, B.A., Franklin, M., Hogg, T.: Enhancing privacy and trust in electronic communities. In: Proceedings of the 1st ACM Conference on Electronic Commerce. p. 78–86. EC '99, Association for Computing Machinery, New York, NY, USA (1999). https://doi.org/10.1145/336992.337012, https://doi.org/10.1145/336992.337012
35. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_9
36. Jarecki, S., Liu, X.: Fast secure computation of set intersection. In: International Conference on Security and Cryptography for Networks. pp. 418–435. Springer (2010)
37. Kales, D., Rechberger, C., Schneider, T., Senker, M., Weinert, C.: Mobile private contact discovery at scale. In: Heninger, N., Traynor, P. (eds.) USENIX Security 2019. pp. 1447–1464. USENIX Association (Aug 2019)
38. Kannepalli, S., Laine, K., Moreno, R.C.: Password monitor: Safeguarding passwords in microsoft edge. https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/ (2021), accessed: 2021-04-26
39. Kim, A., Polyakov, Y., Zucca, V.: Revisiting homomorphic encryption schemes for finite fields. Cryptology ePrint Archive, Report 2021/204 (2021), https://eprint.iacr.org/2021/204
40. Kiss, Á., Liu, J., Schneider, T., Asokan, N., Pinkas, B.: Private set intersection for unequal set sizes with mobile applications. PoPETs **2017**(4), 177–197 (Oct 2017). https://doi.org/10.1515/popets-2017-0044
41. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 818–829. ACM Press (Oct 2016). https://doi.org/10.1145/2976749.2978381
42. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. Journal of the ACM (JACM) **60**(6), 43 (2013)
43. Marlinspike, M.: The difficulty of private contact discovery. A company sponsored blog post (2014), https://signal.org/blog/contact-discovery/
44. 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
45. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. Journal of the ACM **51**(2), 231–262 (2004)
46. 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)
47. Orrù, M., Orsini, E., Scholl, P.: Actively secure 1-out-of-N OT extension with application to private set intersection. In: Handschuh, H. (ed.) CT-RSA 2017. LNCS, vol. 10159, pp. 381–396. Springer, Heidelberg (Feb 2017). https://doi.org/10.1007/978-3-319-52153-4_22
48. Paterson, M.S., Stockmeyer, L.J.: On the number of nonscalar multiplications necessary to evaluate polynomials. SIAM Journal on Computing **2**(1), 60–66 (1973)
49. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: PSI from PaXoS: Fast, malicious private set intersection. In: Rijmen, V., Ishai, Y. (eds.) EUROCRYPT 2020, Part II. pp. 739–767. LNCS, Springer, Heidelberg (May 2020)
50. 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)

51. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (Dec 2009)
52. Pinkas, B., Schneider, T., Weinert, C., Wieder, U.: Efficient circuit-based PSI via cuckoo hashing. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 125–157. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78372-7_5
53. Pinkas, B., Schneider, T., Zohner, M.: Faster private set intersection based on OT extension. In: Fu, K., Jung, J. (eds.) USENIX Security 2014. pp. 797–812. USENIX Association (Aug 2014)
54. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Gabow, H.N., Fagin, R. (eds.) Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005. pp. 84–93. ACM (2005). https://doi.org/10.1145/1060590.1060603, http://doi.acm.org/10.1145/1060590.1060603
55. Resende, A.C.D., Aranha, D.F.: Faster unbalanced private set intersection. In: Meiklejohn, S., Sako, K. (eds.) FC 2018. LNCS, vol. 10957, pp. 203–221. Springer, Heidelberg (Feb / Mar 2018)
56. 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
57. Scott, M.: A note on the calculation of some functions in finite fields: Tricks of the trade. Cryptology ePrint Archive, Report 2020/1497 (2020), https://eprint.iacr.org/2020/1497
58. Microsoft SEAL (release 3.7). https://github.com/Microsoft/SEAL (Sep 2021), microsoft Research, Redmond, WA.
59. Shoup, V.: NTL: A library for doing number theory (11.4.3). https://libntl.org/ (Jan 2021)
60. Smart, N.P., Vercauteren, F.: Fully homomorphic simd operations. Designs, codes and cryptography **71**(1), 57–81 (2014)
61. Trieu, N., Shehata, K., Saxena, P., Shokri, R., Song, D.: Epione: Lightweight contact tracing with strong privacy (2020)
62. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS. pp. 162–167. IEEE Computer Society Press (Oct 1986). https://doi.org/10.1109/SFCS.1986.25

A Complexity of the Paterson-Stockmeyer algorithm with partitioning

Let α be the number of partitions as in Section 3.1. The Paterson-Stockmeyer algorithm achieves the minimal non-scalar multiplicative complexity when $L_\alpha = \left\lfloor \sqrt{(\alpha+1)\left(\lceil B/\alpha \rceil + 1\right)} \right\rfloor$. By taking $L_\alpha = \left\lfloor \sqrt{(\alpha+1)\left(\lceil B/\alpha \rceil + 1\right)} \right\rfloor$, we obtain $H_\alpha = \left\lceil \frac{\lceil B/\alpha \rceil + 1}{L_\alpha} \right\rceil$. Thus, the minimal complexity from (4) is bounded as follows

$$\begin{aligned} & L_\alpha + (\alpha+1)H_\alpha - (\alpha+3) \\ & \quad < \sqrt{(\alpha+1)\left(\lceil B/\alpha \rceil + 1\right)} + (\alpha+1)\left(\frac{\lceil B/\alpha \rceil + 1}{L_\alpha} + 1\right) - (\alpha+3) \\ & \quad < \sqrt{(\alpha+1)\left(\lceil B/\alpha \rceil + 1\right)} + \frac{(\lceil B/\alpha \rceil + 1)(\alpha+1)}{\sqrt{(\alpha+1)\left(\lceil B/\alpha \rceil + 1\right)} - 1} - 2 \end{aligned}$$

Let $C = \sqrt{(\alpha+1)\left(\lceil B/\alpha \rceil + 1\right)}$. Then, the above upper bound turns into

$$\begin{aligned} C + \frac{C^2}{C-1} - 2 = \frac{C^2 - C + C^2 - 2C + 2}{C-1} &= \frac{2C^2 - 3C + 2}{C-1} \\ &= 2C - 1 + \frac{1}{C-1} < 2C = 2\sqrt{(\alpha+1)\left(\lceil B/\alpha \rceil + 1\right)} \\ &< 2\sqrt{(\alpha+1)B/\alpha + 2\alpha + 2} = 2\sqrt{B + B/\alpha + 2\alpha + 2} \end{aligned}$$

As a result, we obtain that the optimal number of non-scalar multiplications in the Paterson-Stockmeyer algorithm with partitioning is around $2\sqrt{B + B/\alpha + 2\alpha + 2}$ where α is the number of partitions.

If α is big enough, this complexity can exceed $\lceil B/\alpha \rceil - 1$, the multiplicative complexity of evaluating all the powers of $[\![y^2]\!], \ldots, [\![y^{\lceil B/\alpha \rceil}]\!]$ to compute all the polynomials $P_i([\![y]\!])$'s.

Let us demonstrate when the Paterson-Stockmeyer method with partitioning should be used instead of computing all the powers of $[\![y]\!]$. Assume that $\alpha = \sqrt{B}/r$. If $r > 4$, then substituting α in the above upper bound yields

$$\begin{aligned} L_\alpha + (\alpha+1)H_\alpha - (\alpha+3) &< 2\sqrt{B + B/\alpha + 2\alpha + 2} \\ &< 2\sqrt{B + \left(r + \frac{2}{r}\right)\sqrt{B} + \left(\frac{r + \frac{2}{r}}{2}\right)^2} \\ &= 2\left(\sqrt{B} + \frac{r + \frac{2}{r}}{2}\right) = 2\sqrt{B} + r + \frac{2}{r} \\ &= 2\sqrt{B} + \frac{\sqrt{B}}{\alpha} + \frac{2\alpha}{\sqrt{B}}. \end{aligned}$$

This upper bound is lower than $B/\alpha - 1$, when

$$2\alpha^2 + \left(2B + \sqrt{B}\right)\alpha + B - B\sqrt{B} < 0.$$

It follows that if the number of partitions satisfies

$$\alpha < \frac{-2B - \sqrt{B} + \sqrt{4B^2 + 12B\sqrt{B} - 7B}}{4}, \tag{6}$$

then the Paterson-Stockmeyer method is more efficient than computing all the powers of $[\![y]\!]$.

B Proof of Lemma 1

Recall from Section 3.1 that the windowing base for the low powers is equal to $b_L = \lfloor (L-1)^{2^{-(D-1)}} + 1 \rfloor$ and the base for the high powers is set to $b_H = \lfloor (H-1)^{2^{-(D-1)}} + 1 \rfloor$. Note that $b_H \leq b_L$ when L is optimally chosen since $L > H$. In this case, the communication complexity is equal to

$$\begin{aligned} &(b_L - 1)\left(\left\lfloor \log_{b_L}(L-1) \right\rfloor + 1\right) + (b_H - 1)\left(\left\lfloor \log_{b_H}(H-1) \right\rfloor + 1\right) \\ &\quad < (b_L - 1)\left(\log_{b_L}(L-1) + 1\right) + (b_H - 1)\left(\log_{b_H}(H-1) + 1\right) \end{aligned}$$

Since $\log_{\lfloor b+1 \rfloor} x \leq \log_b x$ for any real $b > 1$, we have

$$\begin{aligned} &(b_L - 1)\left(\log_{b_L}(L-1) + 1\right) + (b_H - 1)\left(\log_{b_H}(H-1) + 1\right) \\ &\quad < (b_L - 1)\left(2^{D-1} + 1\right) + (b_H - 1)\left(2^{D-1} + 1\right) \\ &\quad \leq \left((L-1)^{2^{-(D-1)}} + (H-1)^{2^{-(D-1)}}\right)\left(2^{D-1} + 1\right) \end{aligned}$$

Recall that $LH \geq B + 1$ and $L = \lfloor \sqrt{2(B+1)} \rfloor$ in the optimal case. Then, $H = \lceil (B+1)/L \rceil$ and $H - 1 < (B+1)/L < \sqrt{B+1}$. Hence, we can rewrite the above bound as follows

$$\begin{aligned} &\left((L-1)^{2^{-(D-1)}} + (H-1)^{2^{-(D-1)}}\right)\left(2^{D-1} + 1\right) \\ &\quad < \left(2^{2^{-D}}(B+1)^{2^{-D}} + (B+1)^{2^{-D}}\right)\left(2^{D-1} + 1\right) \\ &\quad < (B+1)^{2^{-D}}\left(2^{2^{-D}} + 1\right)\left(2^{D-1} + 1\right) \\ &\quad < 3(B+1)^{2^{-D}}\left(2^{D-1} + 1\right) \end{aligned}$$

The last inequality holds as $D \geq 0$.

C Proof of Lemma 2

As in Example 2, the sender can compute $[\![y^{t^i}]\!]$ for any i with depth-free Frobenius operations. It means that any encrypted power $[\![y^a]\!]$ can be obtained by non-scalar multiplications of elements from the set $\{[\![y^{t^i}]\!]\}_i$ and Frobenius operations. In particular, the sender finds the biggest i such that $t^i | a$ and then sets $a' = a/t^i$. This means that $[\![y^a]\!]$ can be computed from $[\![y^{a'}]\!]$ with one Frobenius operation.

Next, the sender decomposes a' in base t and obtains $a' = \sum_{i=0}^{\ell-1} a'_i t^i$ with $a'_i \in [0, t-1]$ for any i and $\ell = \lfloor \log_t a' \rfloor + 1$. This decomposition indicates that to obtain $[\![y^{a'}]\!]$ the sender can compute a product with a'_i copies of $[\![y^{t^i}]\!]$ for any $i \in [0, \ell-1]$. Let $\vec{a}'$ denote a vector $(a'_0, \ldots, a'_{\ell-1}) \in \mathbb{Z}^\ell$. Note that $|\vec{a}'|_1 = |\vec{a}|_1$. Hence, $[\![y^a]\!]$ can be computed with a binary tree circuit containing $|\vec{a}|_1$ nodes and of depth $\lceil \log_2 |\vec{a}|_1 \rceil$.

Consider the minimal a such that $[\![y^a]\!]$ can be computed with a circuit of depth $D+1$. In other words, a is the minimal positive integer that satisfies $|\vec{a}|_1 = 2^D + 1$. Let a have the following base-t decomposition

$$\vec{a} = (t-1, t-1, \ldots, t-1, r) \tag{7}$$

where $r \leq t-1$ and $|\vec{a}|_1 = 2^D + 1$. For any positive $v < a$, it holds that $v_i \leq a_i$ for any i and $v_j < a_j$ for some j. Hence, $|\vec{v}|_1 < |\vec{a}|_1$, which results in $|\vec{v}|_1 \leq 2^D$. Thus, all the powers $\{[\![y]\!], [\![y^2]\!], \ldots, [\![y^{a-1}]\!]\}$ can be computed with a depth D binary-tree circuit, which implies that a satisfies the above property. Hence, $e_D = a - 1$.

From (7), it follows that $|\vec{a}|_1 = \sum_{i=0}^{\ell-1}(t-1) + r$. Hence, $|\vec{a}|_1 = \ell(t-1) + r = 2^D + 1$, which leads to $r - 1 = 2^D - \ell(t-1)$. Since $r - 1 < t - 1$, this implies that $r = (2^D \mod (t-1)) + 1$ and $\ell = \frac{2^D - (r-1)}{t-1} = \left\lfloor \frac{2^D}{t-1} \right\rfloor$.

Since $a = \sum_{i=0}^{\ell-1}(t-1)t^i + rt^\ell$, we obtain

$$\begin{aligned} a &= (t-1)\frac{t^\ell - 1}{t-1} + rt^\ell = (r+1)t^\ell - 1 \\ &= \left(\left(2^D \mod (t-1)\right) + 2\right) \cdot t^{\left\lfloor \frac{2^D}{t-1} \right\rfloor} - 1. \end{aligned}$$

Thus, $e_D = ((2^D \mod (t-1)) + 2) \cdot t^{\left\lfloor \frac{2^D}{t-1} \right\rfloor} - 2$, which concludes the proof.

D Experimental results in HElib

To test our PSI protocol in the unlabeled mode using the HElib library, we fixed the plaintext parameters corresponding to different receiver's set size (see Table 7). The windowing and the partitioning techniques are ignored (i.e. $\alpha = 1$). The bit-length of hashed items is equal to $\sigma = 80$ as in [14].

The experiments were conducted with 12 and 24 threads (Table 8). The security parameter λ is defined by the LWE estimator [1] with relation to the modulus qP, the ring dimension n and the error standard deviation $\sigma_e = 3.19$. The secret keys have ternary coefficients and their Hamming weight is arbitrary.

We were not able to run our experiment for the parameter set with $|X| = 2^{26}$ and $|Y| = 126$ because the NTL library [59] could not handle very large polynomials by default and returned the "Polynomial too big for FFT" error.

$\lvert Y\rvert$	t	m	n	k
1337	2789	18719	18718	2674
1245	1033	21923	19920	2490
819	107	19939	19656	1638
746	41	22381	22380	1492
672	53	19517	18816	1344
558	23	21223	20088	1116
419	11	20113	20112	838
341	7	21143	21142	682
210	5	14981	14700	420
126	3	12853	12852	252

Table 7. Plaintext parameter sets fixed for different sender's set sizes $|Y|$. t is the plaintext modulus; m is the order of the cyclotomic ring R; n is the dimension of R; k is the number of SIMD slots.

Parameters						Running time (s)							Communication (MB)				
$\|X\|$	$\|Y\|$	q	qP	c	λ	Offline		Online		Encryption		Dec.	Offline	Online		Total	Total (online)
						$T=12$	24	12	24	12	24			R $\to$ S	S $\to$ R		
2^{26}	1337	471	635	3	100	421	268	1190	783	0.264	0.128	1.14	10.5	2.20	0.224	12.9	2.42
	1245	412	635	2	106	466	296	1310	889	0.240	0.124	1.24	8.12	2.04	0.233	10.4	2.28
	819	299	470	2	145	642	406	1420	884	0.196	0.105	1.04	7.91	1.46	0.217	9.59	1.68
	746	269	425	2	188	807	511	2120	1200	0.218	0.115	1.21	9.71	1.50	0.219	11.4	1.71
	672	269	425	2	155	770	480	2050	1210	0.187	0.112	0.918	7.81	1.26	0.186	9.26	1.44
	558	231	341	2	213	1030	638	2210	1310	0.202	0.123	0.959	8.13	1.15	0.208	9.49	1.36
	419	215	317	2	232	1390	866	2530	1480	0.154	0.010	0.882	9.60	1.07	0.189	10.9	1.26
	341	215	317	2	246	1990	1270	2570	1650	0.217	0.112	0.944	12.2	1.12	0.195	13.5	1.32
	210	170	280	2	187	2260	1450	3050	1640	0.147	0.083	0.533	9.17	0.619	0.143	9.93	0.762
	126	170	280	2	160	–	–	–	–	–	–	–	–	–	–	–	–
2^{24}	1337	471	635	3	100	92.9	59.9	336	276	0.105	0.089	1.14	9.75	2.20	0.224	12.2	2.42
	1245	412	635	2	106	102	64.7	400	338	0.112	0.104	1.23	7.50	2.04	0.233	9.78	2.28
	819	323	475	2	143	140	89.3	395	307	0.105	0.087	1.03	7.32	1.58	0.198	9.09	1.77
	746	269	425	2	188	176	111	432	298	0.112	0.103	1.18	8.84	1.50	0.219	10.6	1.71
	672	269	425	2	155	168	106	446	295	0.109	0.08	0.887	7.13	1.26	0.186	8.58	1.44
	558	215	317	2	232	217	138	466	314	0.118	0.081	0.955	6.87	1.07	0.193	8.14	1.27
	419	215	317	2	232	298	185	597	376	0.095	0.079	0.876	8.70	1.07	0.189	9.96	1.26
	341	215	317	2	246	423	264	610	382	0.088	0.072	0.925	11.1	1.12	0.195	12.4	1.32
	210	170	280	2	187	484	305	588	354	0.065	0.05	0.491	8.31	0.619	0.143	9.07	0.762
	126	170	280	2	160	774	509	738	469	0.057	0.043	0.369	10.6	0.540	0.123	11.3	0.663
2^{22}	1337	471	635	3	100	20.2	13.4	103	92.9	0.105	0.089	1.11	8.99	2.20	0.224	11.4	2.42
	1245	412	635	2	106	21.7	14.1	149	140	0.107	0.102	1.21	6.88	2.04	0.233	9.16	2.28
	819	323	475	2	143	30.3	19.3	139	121	0.083	0.069	1.01	6.64	1.58	0.198	8.42	1.77
	746	269	425	2	188	38.1	24.2	133	110	0.084	0.071	1.15	7.98	1.50	0.219	9.69	1.71
	672	269	425	2	155	36.5	23.1	132	108	0.072	0.074	0.862	6.45	1.26	0.186	7.89	1.44
	558	215	317	2	232	46.9	30.1	134	111	0.088	0.067	0.939	6.18	1.07	0.193	7.45	1.27
	419	215	317	2	232	63.9	39.6	155	117	0.068	0.069	0.850	7.80	1.07	0.189	9.06	1.26
	341	215	317	2	246	90.1	56.5	161	120	0.073	0.068	0.904	9.90	1.12	0.195	11.2	1.32
	210	170	280	2	187	104	65.2	150	105	0.05	0.049	0.463	7.44	0.619	0.143	8.20	0.762
	126	161	265	2	171	164	106	179	112	0.043	0.036	0.380	9.02	0.511	0.116	9.65	0.628
2^{20}	1337	419	586	3	108	4.22	2.79	26.1	23.9	0.075	0.079	1.12	7.60	1.96	0.224	9.78	2.18
	1245	377	580	2	117	4.56	2.88	45.8	43.4	0.087	0.08	1.24	5.72	1.87	0.213	7.81	2.09
	819	269	421	2	164	6.34	4.10	49.9	48.1	0.073	0.074	1.02	5.29	1.32	0.198	6.81	1.52
	746	269	425	2	188	8.13	5.18	52.5	49.3	0.073	0.068	1.15	7.11	1.50	0.219	8.82	1.71
	672	269	425	2	155	7.88	4.97	52.7	49.6	0.066	0.064	0.861	5.77	1.26	0.186	7.21	1.44
	558	215	317	2	232	10.0	6.44	49.5	46.1	0.065	0.069	0.915	5.49	1.07	0.193	6.76	1.27
	419	215	317	2	232	13.5	8.30	53.9	45.3	0.063	0.062	0.844	6.90	1.07	0.189	8.16	1.26
	341	170	280	2	284	19.2	11.9	45.6	38.0	0.058	0.060	0.890	7.71	0.892	0.206	8.81	1.10
	210	161	265	2	199	22.2	14.0	45.1	38.7	0.04	0.047	0.467	6.22	0.587	0.136	6.95	0.722
	126	161	265	2	171	34.5	22.3	51.7	40.6	0.043	0.033	0.363	7.97	0.511	0.116	8.60	0.628

Table 8. Complexity of our HElib implementation using 12 and 24 threads on the sender and 1 thread on the receiver. q denotes the ciphertext modulus of the BGV scheme; qP is the modulus of key-switching keys in the BGV scheme; c is the key-switching parameter in HElib; λ is the security parameter; T is the number of threads; 'R $\to$ S' and 'S $\to$ R' denote the communications from receiver to sender, and from sender to receiver. The receiver's communication consists of ciphertexts (ciph.) and key-switching keys (keys). The plaintext parameters for a given $|Y|$ can be found in Table 7.