PSI with computation or Circuit-PSI for Unbalanced Sets from Homomorphic Encryption

Yongha Son
Samsung SDS
Seoul, South Korea
yongha.son@samsung.com

Jinhyuck Jeong
Samsung SDS
Seoul, South Korea
jhyuck.jeong@samsung.com

ABSTRACT

Circuit-based Private Set Intersection (circuit-PSI) refers to cryptographic protocols that let two parties with input set X and Y compute a function f over the intersection set $X \cap Y$, without revealing any other information. The research efforts for circuit-PSI mainly focus on the case where input set sizes $|X|$ and $|Y|$ are similar so far, and they scale poorly for extremely unbalanced set sizes $|X| \gg |Y|$. Recently, Lepoint *et al.* (ASIACRYPT'21) proposed the first dedicated solutions for this problem, which has online cost only linear in the small set size $|Y|$. However, it requires an expensive setup phase that requires huge storage of about $O(|X|)$ on the small set holder side, which can be problematic in applications where the small set holder is assumed to have restricted equipment.

In this work, we suggest new efficient proposals for circuit-PSI tailored for unbalanced inputs, which feature *zero* small set holder side storage, and comparable online phase performance to the previous work. At the technical core, we use homomorphic encryption (HE) based *plain* PSI protocols of Cong *et al.* (CCS'21), with several technically non-trivial arguments on algorithm and security.

We demonstrate the superiority of our proposals in several input set sizes by an implementation. As a representative example, for input sets of size 2^{24} and 2^{12}, our proposals require *zero* storage on the small set holder whereas Lepoint *et al.* requires over 7GB. The online phase remains similar; over LAN network setting, ours takes 7.5 (or 20.9s) seconds with 45MB (or 11.7MB) communication, while Lepoint *et al.* requires 4.2 seconds with 117MB communication.

KEYWORDS

private set intersection; homomorphic encryption

1 INTRODUCTION

In a two-party functionality called private set intersection (PSI), two parties having respective input sets X and Y compute the intersection $X \cap Y$ without revealing any other information beyond the original set cardinality $|X|$ and $|Y|$ to each other. However, PSI alone cannot solve problems where a party only wants to obtain only fractional information about the intersection or some function evaluation on the intersection set instead of the intersection set itself. We call this sort of protocol by *PSI with computation* that computes $f(X \cap Y)$ for some target function f rather than the intersection set $X \cap Y$.

As a base notion for PSI with computation, *circuit-based PSI* (circuit-PSI) [15, 30, 33] that outputs the intersect information in secret-shared form has been studied. Informally, for each $y \in Y$, it outputs two random-looking bits r and s respectively to each party such that $r \oplus s = 1$ if and only if $y \in X \cap Y$. Note that the shares provide no information about the intersection to each party. It can be used as a general-purpose preprocessing for PSI with computation, in the sense that two parties use the secret shares to perform further computation on the intersection. The most typical example would be the so-called PSI Cardinality which only reveals the cardinality of the intersection, or PSI Threshold which only reveals whether the cardinality exceeds some input threshold.

State-of-the-art circuit-PSI constructions are based on oblivious programmable PRF (OPPRF) [5, 10, 20, 30, 33], and have been rapidly improved to have both computational and communication cost linear in input set size. However, such linear dependency lies on both input sets, and hence this protocol poorly scales when two input sets have extremely *unbalanced* size.

The unbalanced case has already received much interest in PSI field [6, 7, 22, 32], since this case also finds useful real-world applications such as password breach check [19] or contact discovery [13]. However, most of the works only focused on the plain PSI functionality, although PSI with computation also deserves to be studied. Contact tracing for pandemic diseases would be a notable example. One party (typically a government institution) stores a geological database and moving routes of confirmed cases, and the other party can be each individual who has its moving routes locally. In this case, it is strongly desirable to inform only the fact of contact since detailed contact information may lead to privacy violations.

To the best of our knowledge, only one work [24] explicitly tackles this problem with dedicated protocols and sound analysis. By writing the large set size by n_x and the small set size by n_y, the first has a highly efficient online phase whose cost only depends linearly on n_y. However, such an efficient online phase comes from an expensive setup phase cost, which consists of $O(n_x)$ preprocessing on the large set side, and then transmission of the $O(n_x)$ preprocessed data to the small set side. Note that although this $O(n_x)$ data can be sent to the small side before the actual protocol starts, i.e., in the offline phase, the small side has to hold that storage during the whole online phase. The concrete amount of $O(n_x)$ storage is greater than 7 GB for $n_x = 2^{24}$, which is strongly prohibitive for applications where the small set holder could be lightweight devices with limited storage space, such as mobile.

In this regard, [24] also proposed another solution that runs without any offline preprocessing, which does not require $O(n_x)$ storage burden on the small set holder. This comes at only a small overhead in the communication cost $O(n_y \log(n_x/n_y))$, but an overhead on the computation is devastating; asymptotically it grows to $O(n_x)$. To say concrete timing, it requires over 8 hours of online running time with the large set size $n_x = 2^{24}$. This leads to a natural question:

$n_y = 2^{12}$		n_x	Ours 1	Ours 2	PJC 1	PJC 2
Online	Time (sec)	2^{20}	4.85	2.59	4.22	2,134
		2^{24}	20.9	7.54	4.22	33,997
	Comm. (MB)	2^{20}	8.1	12.2	117	213
		2^{24}	11.7	45.2	117	256
Offline	Time (sec) (Large-side Setup)	2^{20}	5.4	3.8	137	0
		2^{24}	465	68	2,207	0
	Comm. (MB) (Small-side Storage)	2^{20}	0	0	465	0
		2^{24}	0	0	7,440	0

Table 1: Comparison with previous work PJCs [24]. Timings are measured over a LAN network. For all protocols, the offline timing is dominated by large set side-alone setup timing, and the whole offline communication leads to small set side storage. For detailed explanations, see §6 and Table 4.

Is it possible to achieve sublinear complexity with respect to the large set size n_x without requiring any storage in the small set holder?

1.1 Our Contribution

This work proposes new circuit-PSI protocols tailored for extremely unbalanced input sets, affirmatively answering the question above. In particular, they require zero storage in the small set holder while providing efficient online performance sublinear to the large set size n_x. Our proposals consist of two constructions that provide computation-communication trade-offs to each other, whose underlying technique can be another interest.

The improvement in concrete performance is significant, as Table 1 shows some examples. Our proposals remove $O(n_x)$ storage requirement of the first protocol (PJC 1) of [24] while achieving similar online performance. Compared to the previous approach (PJC 2) of [24] that does not require small-side storage, our constructions show significantly faster performance. It is true that ours still needs some offline time, whereas PJC 2 removes the entire offline phase. However, the offline computations can be *locally* done on the server-side alone and hence much less problematic than offline communication that results in the storage burden on the client side. Moreover, our performances are more efficient than PJC 2 even considering the total time; offline plus online time for ours and online time for PJC 2.

Technical Overview. At the technical core of our constructions, we use homomorphic encryption (HE), which is well known to imply efficient HE-based *plain* PSI protocols [6–8] tailored for unbalanced input. Toward HE-based circuit-PSI, we first revisit a simple conversion illustrated in [6] that attaches the post-two-party computation (2PC) stage after the main HE phase. This idea was provided in a rather abstract sense, and our first contribution is a completion of this idea with a rigorous analysis of several optimization techniques proposed in the previous HE-based plain PSI [6–8]. As the most interesting one, we argue that *oblivious pseudo-random function* (OPRF) preprocessing, which provided several benefits, including malicious security in plain PSI cases, provides almost no benefit on circuit-PSI. Then we show that noise flooding [7] is rather suitable to achieve security and complete the protocol against the semi-honest adversary. This may seem to be degraded from the previous PSI works [6, 8] of malicious security, but we stress that all state-of-the-art circuit-PSIs [5, 24, 33] also assume semi-honest adversary. In the asymptotic view, this protocol requires $O(n_x)$ precomputation of the large set holder without any communication in the offline phase, and $O(\sqrt{n_x n_y})$ non-scalar HE multiplications and $O(n_y)$ 2PC-based equality test, and $O(n_y \log(n_x/n_y))$ communication in the online phase.

As the second contribution, we propose an efficient trade-off between communication and computation of the first protocol. In fact, there is a well-known trade-off from HE literature, which requires zero non-scalar HE multiplications while increasing communication to $O(\sqrt{n_x n_y})$. It works well for the plain PSI protocol [8], which consists only of the HE phase, but not for our circuit-PSI protocol, which has an additional 2PC phase. This naive trade-off leads to unsatisfactory performance, as it increases the post 2PC phase to $O(\sqrt{n_x n_y})$, which was only $O(n_y)$ before the trade-off. To overcome this, we suggest a novel method that recursively applies the HE phase to end with $O(n_y)$ 2PC phase. This leads to our second protocol of zero non-scalar HE multiplications with $O(n_y)$ 2PC phase and $O(\sqrt{n_x n_y})$ communication in the online phase.

1.2 Further Related Work

The early proposal of plain PSI is based on Diffie-Hellman (DH) [27], and this still serves as a basis for modern PSIs with considerably low communication cost but high computational cost. Many recent plain PSI proposals have been based on OPRF [28, 33], and it has a significantly low computational cost but a higher communication burden than DH-based PSI.

Unbalanced Input Sets. OPRF-based plain PSI protocols enjoy relatively fast online performance for the unbalanced set that only depends on the small set size n_y, assuming heavy offline preprocessing of cost linear in the large set size n_x. The research efforts are put to reduce the offline preprocessing [22, 32], but it still has an asymptotically linear cost. Another line of plain PSI protocols based on HE [6–8] achieves sublinear complexity, especially the logarithmic communication cost on the large set size n_x. The two protocols proposed in [24] can be understood as the first step toward both directions in the circuit-PSI functionality; the first achieves $O(n_x)$ offline and $O(n_y)$ online, and the second achieves communication cost $O(n_y \log(n_x/n_y))$ but computational cost still $O(n_x)$.

Other PSI-with-computation. [16] provides a protocol for PSI with computation in name *Private Join & Compute* based on Diffie-Hellman (DH). It aims for a specific functionality that computes the cardinality of the intersection and sums all associated values of the intersection sets. Subsequently, [4] further developed this into a protocol that provides two parties additive shares of the intersected elements between the two parties to support general computation over the intersection set. However, it has a drawback that unavoidably leaks the cardinality of the intersection. The more generalized concept of circuit-PSI was first proposed by [15], and then continuous improvements have been reported until OPPRF-based protocols [5, 10, 30, 33].

1.3 Roadmap

First, we provide some preliminaries in §2. Then in §3, we review the previous HE-based *plain* PSI protocol due to [6–8]. After then, in §4, we describe our first circuit-PSI protocol that attaches the 2PC phase after the plain PSI protocol of [8], with an argument of the irrelevance of OPRF preprocessing. In the following §5, we propose our second circuit-PSI protocol, which provides a trade-off between computation and communication with respect to our first proposal, whose technical core is a recursive call of the HE phase. Finally, we provide experimental results and comparisons in §6.

2 BACKGROUNDS

The i-th component of a vector $\vec{v}$ is denoted by v_i. For an integer k, the set $\{1, \cdots, k\}$ is denoted by $[k]$. The logarithm function log is assumed to have a base 2. For a set S, we denote a uniform distribution over S by $\mathcal{U}(S)$, and denote sampling an element s from the uniform distribution over S by $s \leftarrow S$. For any statement T that can be determined by true or false (Boolean), we denote $\mathbf{1}(T)$ as the true value of the equality, that is, it is 1 if T is true and 0 else. For a field $\mathbb{F}$, we denote $\mathbb{F}^{\times} := \mathbb{F} \setminus \{0\}$.

2.1 Semi-honest Security

We use a standard notion of security against semi-honest adversaries and provide a simplified description for the 2-party case. For a detailed explanation, we refer [25]. Consider a two-party protocol Π that computes an ideal functionality $f(x_1, x_2)$ where the party P_i has input x_i. For each party P_i, define $\mathsf{view}_i(x_1, x_2)$ as the view of party P_i during the execution of Π on input x_1, x_2. Specifically, it consists of input x_i, messages that are sent or received, random tape during the execution of the protocol, and the output of the protocol Π.

Definition 2.1. We say that a (two-party) protocol Π for P_1 and P_2 computes f against a semi-honest adversary, if there exist simulators Sim_1 and Sim_2 such that, for any inputs x_1, x_2 and $i = 1, 2$,

$$\mathsf{Sim}_i(x_i, f(x_1, x_2)) \cong_c \mathsf{view}_i(x_1, x_2),$$

where $\cong_c$ represents computational indistinguishability.

2.2 Homomorphic Encryption

Homomorphic Encryption (HE) is an encryption scheme that allows arithmetic operations in an encrypted state without decrypting the ciphertexts. More precisely, for each arithmetic operation on the plaintext, there is a corresponding operation on HE ciphertexts that outputs a ciphertext of the arithmetic operation result.

In this paper, we especially focus on a variant of BFV scheme [3] that supports a plaintext space [1] $\mathcal{R}_t := \mathbb{Z}_t[X]/(X^N + 1)$ for a power-of-two N and a prime modulus t, and the ciphertext space is $\mathcal{R}_q^2$ for a modulus $q \gg t$. To correctly evaluate an arithmetic circuit of multiplicative depth L, the ciphertext modulus q is chosen so that $\log q \approx (L + 1) \log t$. Below, we provide some properties of BFV required to understand our work.

[1] Actually BFV scheme supports more general plaintext, but we only focus on this one which is widely used in the real world.

Batching. BFV scheme can support a batching property on ciphertexts, which encrypts multiple messages as a vector over some field $\mathbb{F}$ in one ciphertext and supports *slot-wise* addition and multiplication in $\mathbb{F}$ in an encrypted state. To be precise, it exploits an isomorphism

$$\mathcal{R}_t = \mathbb{Z}_t[X]/(X^N + 1) \simeq \mathbb{Z}_t^N$$

which holds when $t = 1 \mod 2N$. Then the BFV scheme can encrypt N elements in $\mathbb{Z}_t$ in one ciphertext using this map, and support *slot-wise* addition and multiplication on $\mathbb{Z}_t^N$ in an encrypted state.

Operations and Their Costs. More precisely, the BFV scheme supports slot-wise addition and multiplications between two ciphertexts or a ciphertext and plaintext in $\mathbb{Z}_t^N$. A homomorphic addition can be operated in a significantly shorter time than multiplication. A multiplication between ciphertext and plaintext, which we call *scalar multiplication* is much faster than a multiplication between ciphertexts, say *non-scalar multiplication*. The performance difference depends on underlying parameter selection, but in our HE parameter choice, non-scalar multiplication takes at least 10x time of scalar multiplication. We refer to [21] for a detailed asymptotic cost of each operation.

Security Notions. BFV ciphertext is indistinguishable from a random distribution under the hardness assumption of Ring-Learning with Errors (RLWE) [26], and this ensures the IND-CPA security of the scheme. Furthermore, we consider a *function privacy* which requires that the output ciphertext of any homomorphic operations along with the decryption oracle leaks nothing but the inside message, in particular information about the operations processed on the ciphertext.

Although the underlying RLWE assumption naturally achieves IND-CPA, it is rather complicated to achieve the function privacy for BFV. We follow the previous approach based on *re-randomization* and *noise flooding* [2]. As the theoretic explanation requires further details of the BFV, we simply provide the resulting modifications here. To achieve function privacy, we have to prepare a marginal ciphertext modulus of σ bits where σ is a statistical security bit. After then, by adding encryption of zero having sufficiently large noise, one can convert any ciphertext to a function-privacy ciphertext that gives no more information than the internal plaintext when decrypted.

2.3 Two Party Computations

There are two popular approaches for two-party computation (2PC) provided as Boolean circuits with semi-honest security. The first one stems from Yao's garbled circuit (GC) protocol that has a strength in the number of rounds independent of the input circuit depth. The state-of-the-art protocol [36] for GC performs XOR gate with no cost and one AND gate with two symmetric-key ciphertexts communication that takes $2\lambda = 256$ bits communication. The other stems from GMW protocol [12] that also computes the XOR gate for free and one AND gate with two oblivious transfers (OT) [17]. It requires logarithmic rounds of input circuit depth, but the AND gate evaluation can be done with much lower communication cost, less than 2 bits, thanks to recent improvements on OT extensions [9, 35]. In this work, we implicitly consider the GMW protocol for 2PC as

the gain on communication is considered much bigger than the loss on the number of rounds. However, we also remark that the proposed circuit-PSI protocol can also be realized with GC.

2.4 Definition of Circuit-based PSI

Circuit-based PSI (circuit-PSI) is a formal and flexible definition of PSI with computation that generates Boolean additive shares of the intersection information. More precisely, for input sets X and Y, it outputs Boolean additive shares of $\mathbf{1}(y \in X \cap Y)$ for each $y \in Y$. They can be used for the desired function evaluation by other cryptographic primitives, typically generic 2PC. Typical examples of target functions are the cardinality (size) function, threshold function, summation of associated values, and inner product of associated values from both sides. The formal definition is given as Figure 1. The table size T and the index function ι are somewhat nonintuitive, but it is required to capture the cuckoo hashing technique, which is commonly used for current circuit-PSI protocols.

As a direct consequence of circuit-PSI, one can achieve PSI Cardinality by computing Hamming distance circuit, PSI Threshold, by further augmenting arithmetic comparison to PSI Cardinality.

Parameters: Set sizes n_x and n_y, and output size T
Input: X of size n_x from one party, and Y of size n_y from the other party.
Functionality: The functionality samples a random vector $\vec{s}_x \in \{0,1\}^T$, and a random injective function $\iota : Y \to [T]$. Then it defines a vector $\vec{s}_y \in \{0,1\}^T$ by such that

$$s_{x,i} \oplus s_{y,i} = \begin{cases} \mathbf{1}(\iota^{-1}(i) \in X \cap Y) & \text{if } i \in \iota(Y) \\ 0 & \text{otherwise.} \end{cases}$$

It sends $\vec{s}_x$ to the X holder, and $\vec{s}_y$ and the index function ι to the Y holder.

Figure 1: (Ideal) Functionality of the circuit-PSI

3 REVIEW OF PREVIOUS HE-BASED PSIS

In this section, we review a *plain* PSI protocol based on homomorphic encryption (HE) [7], along with the optimization techniques proposed in the following works [6, 8]. Let us denote one input set X with size n_x and another input set by Y with size n_y. It is widely known that HE-based PSI protocols show strength when input sizes are highly unbalanced, and hence we assume that $n_x \gg n_y$.

The small set holder S constructs a hash table C from Y with $n'_y := (1+\epsilon) \cdot n_y$ bins so that each bin contains at most one item $y \in Y$. This can be done by a cuckoo hashing with several hash functions $H_1, \cdots, H_h$ (typically $h = 3$), which inserts each element $y \in Y$ into the $H_i(y)$ -th bin $C[H_i(y)]$ for some i so that each bin of C contains at most one item. Meanwhile, the large set holder L constructs another hash table $\mathcal{S}$ with n'_y bins by inserting each $x \in X$ into $H_i(x)$-th bin $\mathcal{S}[H_i(x)]$ for all $i \in [h]$. We call C by the cuckoo table and $\mathcal{S}$ by the simple table. Then each bin of $\mathcal{S}$ contains $\beta = O(n_x/n_y)$ items with high probability. The intersection can then be derived from the union of all bin-wise intersections:

$$X \cap Y = \bigcup_k \{C[k] \cap \mathcal{S}[k]\}.$$

Since $C[k]$ is a singleton set, the goal is reduced to the membership check problem $C[k] \in \mathcal{S}[k]$. Note that this cuckoo hashing routine (with several hash functions) instead of the regular hashing routine (with only one hash function) is essential to reduce the original problem into this smaller membership check problem $C[k] \in \mathcal{S}[k]$; if one uses the regular hashing routine (with only one hash function), the small set side table C is not guaranteed to have a single element in each bin.

Now, we focus on one bin and write the cuckoo bin element by y and the simple bin set by B. The key idea is to check whether $y \in B$ by evaluating a polynomial

$$F(y) = c \cdot \prod_{x \in S} (x - y) = \sum_{i=0}^{\beta} a_i \cdot y^i$$

with a random multiplicative mask $c \leftarrow \mathbb{F}^{\times}$ where $\mathbb{F}$ is a base finite field. Note that the result is equal to 0 if y is in B, or a random element in $\mathbb{F}^{\times}$. For a private evaluation of the polynomial, the small set holder uses homomorphic encryption (HE) to encrypt y and sends it to the large set holder. Then the large set holder evaluates $F(y)$ and sends back the resulting ciphertext to the small set holder. Finally, the small set holder decrypts the ciphertext and checks whether the decrypted value is 0 or not, which corresponds to $y \in B$ or $y \notin B$ respectively.

3.1 Optimizations

There have been proposed several optimization techniques in [7, 8] on this idea. We introduce some techniques required to understand the main body of our paper.

Noise Flooding and OPRF. When decrypted, the noise term in the ciphertext may reveal some information about the large set holder item not in the intersection. Then the noise term should be *flooded* by a much larger error to prevent this leakage, and the original PSI proposal [7] achieved semi-honest security from this so-called *noise flooding*. Subsequently, [6] showed that such noise flooding could be removed and further achieves malicious small set holder security, with oblivious pseudo-random function (OPRF) preprocessing. OPRF is a two-party functionality that outputs a PRF key sk to the large set holder and $\mathsf{PRF}_{sk}(Y)$ to the small set holder of input Y while the large set holder without any knowledge of Y other than n_y and the small set holder without any knowledge of sk. To achieve malicious security, the large set holder randomizes items with PRF_{sk} so that any information leakage about non-intersect items is at most the PRF value of the large set holder item, which is random in the view of the small set holder. As a downside, this requires the large set holder to spend $O(n_x)$ offline time for computing OPRF. However, it can be acceptable, as it has to be done only once and can be reused for multiple small set holder sets, while X remains static[2].

Batching and Handling Long Input. One can check the membership $y \in B$ on several hash table bins at once using batching property of BFV. The most naive application would be letting the small set holder pack each cuckoo bin element into each slot of

[2]This property is only valid for Diffie-Hellman based OPRF [18], and not for OT based OPRFs.

plaintext and the large set holder can also evaluate the polynomial in a slot-wise manner.

Meanwhile, to deal with an input of arbitrary length, it is common to compress input into some collision-resistant length using a public hash function in the general context of PSI [6, 31]. The concrete length is roughly $\sigma + \log n_x + \log n_y$ for statistical security parameter σ, and it can be slightly reduced to $\ell = \sigma + \log n_x$ thanks to the permutation-based hashing [29]. Then in order to embed one item into one HE plaintext slot, HE plaintext slot field $\mathbb{F}$ should be larger than 2^ℓ. One can use algebraic extension fields of small characteristic [6, 14], or huge prime field $\mathbb{Z}_t$ [34]. However, algebraic extension fields come at an expensive computational cost, and a huge prime field has some inconvenience in implementation on a 64-bit processor.

In this regard, [8] observed that OPRF preprocessing provides another benefit on this issue, as it allows a long item y to be split into d pieces $y_1, \cdots, y_d$ of small modulus $t \approx 2^{\ell/d}$. The red dashed line in Figure 2 shows this concept. Then the inclusion polynomial is evaluated on each split, and the small set holder concludes that $y \in B$ if all polynomial evaluations are zero. This requires d times as many ciphertexts to encrypt the same number of items, but the smaller choice t brings a benefit on computational cost. On the downside, this introduces a chance of false positives, whose details are provided in Appendix E. Also, note that this split is insecure without OPRF preprocessing, as it leaks partial matching information with non-negligible probability (about $O(1/t)$).

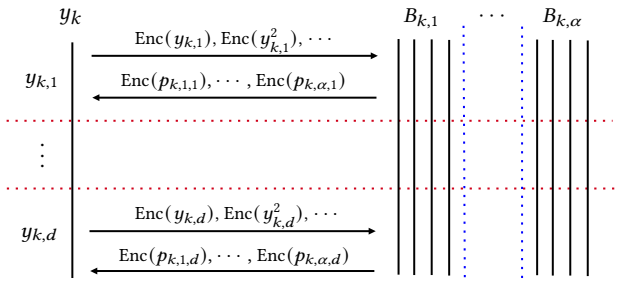

Figure 2: Figure shows k-th bins with item splits and bin partition. A cuckoo bin item y_k is split into d short items (red dash). A simple bin B_k is divided into α partitions (blue dash), and each element is also split into d pieces (red dash).

Lowering Depth. As mentioned in Section 2.2, the ciphertext modulus q of the BFV scheme is chosen linearly in the multiplicative depth L of the target computation, precisely $\log q \approx (L+1)\log t$. Since large q has a devastating effect on performance, it is crucial to lower the depth of target computation. Several techniques for this goal have been reported, and they significantly contribute to the high performance of HE-based PSI protocol. We provide only the necessary information and refer to previous works for the details.

The most important one to understand in this paper is *bin partitioning* that divides B into α partitions B_i and evaluates α polynomials for each set B_i. The blue dashed line in Figure 2 shows this concept. As a result, the small set holder obtains α polynomial evaluations and concludes that $y \in B$ if one of the decryption results p_i equals 0. Although this increases the number of returning ciphertexts by α times, this changes one polynomial evaluation of degree β into α polynomial evaluations of degree β/α.

The second is (a variant of) *Paterson-Stockmeyer* algorithm [8] that significantly improves HE polynomial evaluation. Compatible with the bin partition technique, it evaluates all α inclusion polynomials of degree β/α with $O(\sqrt{\beta})$ non-scalar multiplications, surprisingly regardless of partition number α. Note that this technique enables one to take relatively small α without increasing computational cost.

The last technique allows the small set holder to send more powers of y so that the large set holder can evaluate the polynomial $F(y) = \sum_{i=0}^{\beta} a_i \cdot y^i$ with lower depth. The early idea called *windowing* [7] sends the word powers, for example $y, y^2, \cdots, y^{2^k}$, but this is improved with the use of extremal postage-stamp bases [8].

4 A CIRCUIT-PSI EXTENSION

Using the same notation with Section 3, circuit-PSI is achieved by generating Boolean shares of $\mathbf{1}(y \in B)$ for each cuckoo bin element y and simple bin B. For this, [6] illustrated a simple modification of the plain PSI protocol in Section 3. It follows the same procedure until the large set holder computes the polynomial evaluations $p_1, \cdots, p_\alpha$ on each bin partition B_i, but then it tweaks the protocol by letting the large set holder adds random masks $r_i \leftarrow \mathbb{F}$ to each p_i so that the small set holder obtains $p_i' = p_i + r_i$. Then there exists at least one match $p_i' = r_i$ if and only if $y \in B$. In other words, it converts the original private membership test (PMT) $y \in B$ to α instances of the private equality test (PEqT) problem $p_i' = r_i$. For the final circuit-PSI output, two parties privately generate Boolean shares of $\mathbf{1}(p_i' = r_i)$ for $1 \le i \le \alpha$ and XOR all shares. PEqT is typically done by two-party computation (2PC) protocols such as GMW protocol. The idea is illustrated in Figure 3.

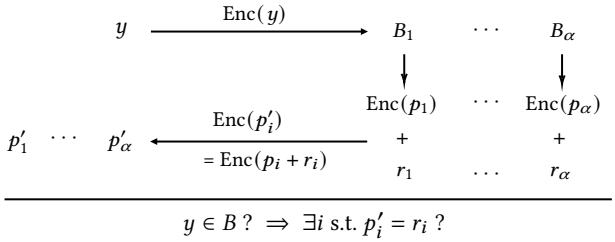

Figure 3: An illustration of extension with random mask

This idea was proposed only with an abstract description in [6], and a detailed analysis was missing. In this section, we discuss it in more depth toward a complete protocol, especially with arguments about (the irrelevance of) OPRF preprocessing and further consequences.

4.1 Revisiting OPRF and Noise Flooding

Recall from Section 3 that OPRF preprocessing provides several nice properties for plain PSI purposes. The most appealing one would be secured against the malicious small set holder, and the next would be secure item splits. However, we argue below that for the illustration above of the circuit-PSI, OPRF preprocessing provides no help on security even against a semi-honest adversary, and

rather noise flooding is mandatory. We also discuss other benefits of OPRF preprocessing almost vanish, and finally, there is a slight advantage to spending long offline time on OPRF preprocessing in circuit-PSI extension.

Irrelevance of OPRF on Security. We first explain that OPRF preprocessing is irrelevant to the security requirement of the circuit-PSI. For that, we review how OPRF provides the security of the *plain PSI*. Recall that in the OPRF preprocessing phase, the sender obtains $F_{sk}(X)$ and the receiver obtains $F_{sk}(Y)$ where the corresponding PRF key sk is only known to the sender. Then as the core argument, observe that even if the receiver knows the whole sender OPRF values, $F_{sk}(X)$, it does not lead to any information leakage with the *plain PSI* respect: The sender only knows PRF key sk, and hence the PRF value $F_{sk}(x)$ for a non-matching $x \in X \backslash Y$ is totally randomized in the view of the receiver. In short, OPRF preprocessing ensures that the receiver has no further knowledge than the intersection $X \cap Y$.

We stress that this argument is valid only when the intersection set $X \cap Y$ is not considered as an information leakage. This is quite a natural requirement for the plain PSI, but the circuit-PSI definition even prohibits the knowledge of the intersection to none of the two parties; In circuit-PSI, two parties should learn only Boolean vectors that *secret-share* the matching result. As the OPRF preprocessing argument cannot prevent the receiver from knowing the intersection set, we conclude that OPRF preprocessing has nothing with the security of our circuit-PSI protocol.

Other Benefits of OPRF. Although the largest benefit of OPRF (malicious security) turns out to be vain for circuit-PSI, other benefits of OPRF may still be valid. However, we explain that most of the other benefits of OPRF also vanish due to the additive mask r_i on polynomial evaluations. First, OPRF helped to remove the multiplicative random mask c_i in $F_i(y) = c_i \cdot \prod_{x \in B_i}(x - y)$. Second, OPRF made it secure to split an item into several HE slots by making partial matching irrelevant to the original input item.

Meanwhile, in our protocol, additive mask r_i perfectly hides polynomial evaluation and blocks the small set holder from recognizing such partial leakage, even without OPRF. The only remaining benefit of OPRF processing is to remove the necessity of padding dummy elements on each simple bin. This is still valid even with an additive mask and noise flooding, but this benefit is too marginal to spend a significant time on OPRF processing.

Revival of Noise Flooding. What we explain above is the irrelevance of OPRF to circuit-PSI security. To complete the security argument, it remains to explain how our construction hides each matching result. Recall that the polynomial evaluation p_i corresponds to the matching result in our construction. Since we add random mask r_i for each p_i, it may seem that the evaluations p_i are perfectly hidden, and hence the receiver learns nothing from the received ciphertext $\mathrm{Enc}(p'_i)$. However, it implicitly assumes that the resulting HE ciphertext only reveals the inner message when decrypted. For example, the ciphertext decryption procedure may reveal some information about the coefficients of the inclusion polynomial p_i. This translates into the underlying HE scheme should provide *function privacy*, which can be concretely achieved by noise flooding for RLWE-HE schemes.

4.2 Protocol Description

Summarizing the arguments of the previous section, we describe the circuit-PSI protocol with additive mask and noise flooding in Figure 4. Other techniques such as batching, extremal postage-stamp bases, and the Paterson-Stockmeyer method are not presented for simplicity. However, we emphasize that these techniques are essential for efficiency and are indeed applied in our implementation in Section 6.

Inputs: A set $X \subset \{0,1\}^*$ of size n_x from one party L and a set $Y \subset \{0,1\}^*$ of size n_y from the other party S.

Protocol:

1. [**Offline Phase**] L performs the followings. This can be done beforehand S is determined.
   (1) Initial hashing X to $X' \subset \{0,1\}^\ell$ that provides $2^{-\sigma}$ false positive.
   (2) Construct simple hash table $\mathcal{S}$ of size $n'_y = O(n_y)$ with input X'.
   (3) Choose number of partitions α and item splits d. Then for each simple table bin $\mathcal{S}[k] = B_k$:
   - Divide B_k into α partitions $B_{k,i}$. Then split each item of $B_{k,i}$ into d pieces of the same bit-length to have $B_{k,i,j}$. Finally compute and store the coefficients of inclusion polynomial $F_{k,i,j}(y) := \prod_{x \in B_{k,i,j}}(x - y)$.
2. [**Online HE Phase**] Two parties execute a HE-based polynomial evaluation and obtain PEqT instances.
   (1) S performs initial hashing Y to $Y' \subset \{0,1\}^\ell$.
   (2) S constructs cuckoo hash table C of size $n'_y = O(n_y)$ with input Y'.
   (3) For each cuckoo bin item $C[k] = y_k$:
   - Split y_k into d pieces of the same bit-length $y_{k,j}$. For each piece $y_{k,j}$, encrypt and send to L homomorphic encryptions of some powers of $y_{k,j}$.
   (4) L evaluates inclusion polynomial $F_{k,i,j}(y_{k,j})$ of simple bin B_k, to have HE encryptions of results $\{p_{k,i,j}\}_{i \in [\alpha], j \in [d]}$.
   (5) L samples random mask $r_{k,i} \in \{0,1\}^\ell$, split it into d pieces and mask encryptions of $p_{k,i,j}$ using them.
   (6) L adds an encryption of zero having sufficiently large noise to the resulting ciphertexts, in order to provide function privacy.
   (7) L sends back the resulting ciphertexts to S, and S decrypts the received ciphertexts and restore $p'_{k,i} \in \{0,1\}^\ell$.
3. [**Online 2PC Phase**] For each hash bin index k:
   - Two parties execute 2PC-based PEqT to generate Boolean share of $\mathbf{1}(p'_{k,i} = r_{k,i})$. Then each party locally XORs all α Boolean shares to output the result.

Figure 4: Circuit-PSI protocol with random masks

Correctness and Security. The correctness (false positive rate) is exactly the same as the base plain PSI protocol [8], assuming that the 2PC phase has no false positive. For completeness, we provide the details in Appendix E.

The following theorem shows the security argument. Note that the protocol description of Figure 4 has no OPRF preprocessing phase, but has a noise-flooding procedure in Step 2-(6). In this respect, the core of the following theorem is to argue that noise-flooding is sufficient for semi-honest security.

Due to the space limit, we place the proof in Appendix F.

Theorem 4.1. *The protocol in Figure 4 realizes circuit-PSI functionality against a semi-honest adversary, assuming a semi-honest secure 2PC protocol.*

Remark on Item Splits d. The use of noise flooding introduces another consideration on the choice of the number of the item splits d, which was not in the previous work [8]. More precisely, when encrypting the same length item with d splits, it takes a total of $d \cdot \sigma$ bits of ciphertext modulus overhead due to noise flooding. Moreover, considering the false positive probability of item splitting (see Appendix E), smaller split d leads to a smaller total item length. Therefore, it is definitely better to take d as small as possible when noise flooding is applied. However, we also note that it is not always possible to take $d = 1$ due to HE parameter constraints, especially batching slots n and the ciphertext modulus q.

Handling Associated Values. There are many applications of PSI-with-computation where intersection target sets X and Y are a sort of identifiers, and each identifier has an additional associated value. Notable examples would be PSI-Sum [16] that outputs the total sum of associated value over intersected identifiers, or more generally, one can compute the inner product of associated values from both sides over intersection [24]. Our protocol can be easily extended to handle associated value using known techniques, whose details in Appendix A.

5 TRADE-OFF FROM HE RECURSION

In this section, we propose another circuit-PSI protocol, which provides a trade-off between computation and communication compared to the protocol of the previous section.

We first review the naive trade-off for the *plain* PSI. There is an easy way to reduce the computational burden by evaluating inclusion polynomials with only scalar multiplications. This clearly increases communication costs because it requires the small set holder to send HE encryptions of all powers. It is asymptotically optimal to divide one simple bin of size β into $\alpha = O(\sqrt{\beta})$ partitions, and let the small set holder send $\beta/\alpha = O(\sqrt{\beta})$ powers. As the plain PSI only consists of HE phase, this really works and provides an efficient trade-off between computation and communication.

However, such an advantage becomes useless for circuit-PSI due to the post-2PC phase, which takes $O(\alpha)$ cost. When non-scalar multiplications are allowed, α can be taken by $O(1)$ (less than 10) thanks to Paterson-Stockmeyer strategy. However, when the naive (all-power-sending) trade-off is applied, we have $\alpha = O(\sqrt{\beta})$ and hence 2PC phase cost becomes much larger than this trade-off. This leads to poor trade-off efficiency.

To solve this problem, we observe that the HE phase can be tweaked to a procedure that reduces the set size of the private membership test (PMT). Then the main protocol recursively applies the HE subroutine from the initial PMT (from cuckoo/simple hashing), which enables one to end with the $O(1)$ 2PC phase. In the rest of this section, we provide the details of tweaks to the HE phase and discuss the total cost of such a recursive call.

5.1 HE Subroutine with Identical Mask

Recall that the HE phase of the previous protocol adds the resulting ciphertexts of p_i with independently sampled random mask r_i to end with α equality check instance, whose security is quite straightforward. We consider a variant where the large set holder adds *identical random mask* $r \leftarrow \mathbb{F}$ to all ciphertexts of p_i. Then the small set holder obtain $t_i := p_i + r$ and then it holds that $y \in B$ if and only if $r \in T := \{t_1, \cdots, t_\alpha\}$. As a result, the output of the HE phase becomes one instance of PMT for α size set (say α-PMT), not α instances of private equality test (PEqT). The idea is illustrated in Figure 5, whose correctness is immediate. However, we need two slight modifications to p_i for the security proof, which are elaborated on below.

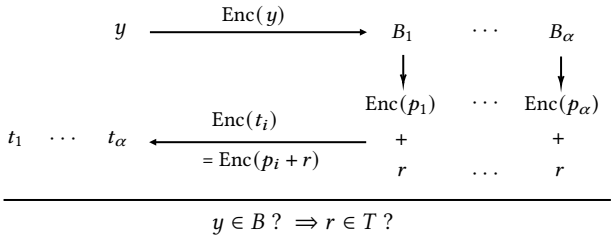

Figure 5: An illustration of the HE subroutine with identical mask

First, we have to ensure that there is at most one item in B equals to y, which implies there exists at most one i such that $p_i = 0$. We remark that this property, which we call the *unique intersection* property, is naturally ensured when the entire item is inserted into one slot, that is, $d = 1$, since the large set holder can eliminate duplication when constructing a simple table. However, for the $d > 2$ case, the unique intersection property should be valid for each HE plaintext slot split rather than the entire items, but this duplication cannot be easily removed anymore, since each duplicated split comes from a different item[3]. Thus, we assume that this protocol always uses parameterization with $d = 1$.

Second, we modify the protocol to sample the multiplicative mask c_i in the polynomial $p_i = c_i \cdot \prod(x - y)$ from $\mathbb{F}$, rather than $\mathbb{F}^\times$. This makes the nonzero p_i is uniformly distributed over $\mathbb{F}$, which makes the distribution of

$$\{p_1, \cdots, p_\alpha\} \sim \begin{cases} \{0\} \times \mathcal{U}(\mathbb{F}^{\alpha-1}) & \text{if } y \in B \\ \mathcal{U}(\mathbb{F}^\alpha) & \text{if } y \notin B, \end{cases}$$

up to re-ordering. Then identical mask r makes $T = \{t_1, \cdots, t_\alpha\}$ to be uniform over $\mathbb{F}^\alpha$ for both cases, and this completes the security. This modification definitely introduces a new false positive probability, because now p_i can be zero although there is no match in that partition. Since there are $O(n_y)$ hash-bins and each hash-bin is divided into α partitions, we have $O(\alpha n_y)$ polynomials to evaluate. Then the probability of all c_i are nonzero is $1 - (1 - 1/t)^{O(\alpha n_y)} \geq O(\alpha n_y / t)$ thanks to Bernoulli's inequality.

As the main usage of this subroutine is the recursive reduction of the bin size, it is given two hash tables (simple and cuckoo) as

[3] We leave as future work to design a clever way to resolve this duplication, if possible. This will make it possible to use $d \geq 2$ with an identical mask that enables much more flexible parameterizations.

input, rather than input sets X and Y. You can see the detailed description in Figure 6. We also summarize the security argument by the following lemma for later usage, whose proof is already discussed above.

Lemma 5.1. *Each bin of the output hash table $\mathcal{D}$ of the Figure 6 protocol is uniformly distributed on $\mathbb{F}^{\alpha}$.*

Inputs: A hash table $\mathcal{S}$ of size n that contains β elements in each bin from one party L, and another hash table C of size n that contains one element in each bin from the other party S.

Outputs: A hash table $\mathcal{R}$ of size n that contains one element in each bin to L and another hash table $\mathcal{T}$ of size n that contains $\alpha(\leq \beta)$ elements in each bin to S.

Protocol:
1. [**Preprocessing**] Perform Step 1-(3) of Figure 4; L precomputes coefficients of the inclusion polynomial $F_{k,i,j}(y)$ for $k \in [n], i \in [\alpha]$ and $j \in [d]$.
2. [**HE Phase**]
   (1) Perform Step 2-(4) of Figure 4 with item split $d = 1$; S sends homomorphic encryptions of powers of y_k.
   (2) L samples multiplicative mask c_k and homomorphically evaluates polynomial $c_k \cdot F_{k,i}(y_k)$ to have HE encryptions of results $\{p_{k,i}\}_{i\in[\alpha]}$.
   (3) L samples random mask $r_k \in \{0,1\}^{\ell}$ and masks HE encryption of $p_{k,i}$, and sends back the resulting ciphertexts. Let $\mathcal{R}[k] = r_k$.
   (4) S decrypts the received ciphertexts and append the decrypted message $t_{k,i} \in \{0,1\}^{\ell}$ to $\mathcal{T}[k]$.

Figure 6: HE-subroutine with Identical Mask

5.2 Circuit-PSI from Recursive HE-subroutine

We proceed to the full circuit-PSI protocol using the HE subroutine of the identical mask, as described in Figure 7. As a rough overview, two parties initialize PMT instances with cuckoo/simple hashing respectively. Then HE-subroutine of Figure 5 enables us to recursively reduce the PMT set size. Finally, it solves PMT using the 2PC-based method after reaching a sufficiently small set.

As said before, this protocol is particularly useful when combined with the naive trade-off strategy that sends all power to evaluate polynomials with only scalar multiplications. It further reduces 2PC inputs than $O(\sqrt{\beta})$; possibly up to $O(1)$ size. Furthermore, as each HE subroutine cost largely depends on the input set size, the additional cost of recursive application would be quite small: By denoting the cost of the initial call by C, all additional calls require only $O(\sqrt{C})$ cost. Figure 8 shows an overview of the protocols.

We note that the recursive HE subroutine definitely increases interaction rounds, but also remark that the 2PC phase already requires additional rounds. In particular, when the 2PC phase is executed by the GMW protocol, $O(\log \ell)$ additional rounds are required, which is at least 5 in our interest parameter regime. Meanwhile, the number of HE subroutines to reach $O(1)$ size is $O(\log\log\beta)$ (assuming square root decreasing), which is typically not greater than 3.

Inputs: A set $X \subset \{0,1\}^*$ of size n_x from one party L and a set $Y \subset \{0,1\}^*$ of size n_y from the other party S.

Protocol:
1. [**Offline Phase**] Same with Figure 4-1; L performs initial hashing, constructs simple hash table $\mathcal{S}$, and precomputes coefficients for the initial inclusion polynomial.
2. [**Online HE Phase**] Two parties recursively call the HE subroutine to have small-sized membership check instances.
   (1) Same with Figure 4 until 2-(3); S performs initial hashing, constructs cuckoo hash table C.
   (2) L and S recursively runs several HE-subroutine of Figure 6 on initial input $\mathcal{S}$ and C, and end with $\mathcal{R}$ and $\mathcal{T}$.
3. [**Online 2PC Phase**] For each hash bin index k:
   - Execute 2PC-based PMT to generate Boolean share of $\mathbf{1}(\mathcal{R}[k] \in \mathcal{T}[k])$.

Figure 7: Circuit-PSI protocol with recursive HE-subroutines

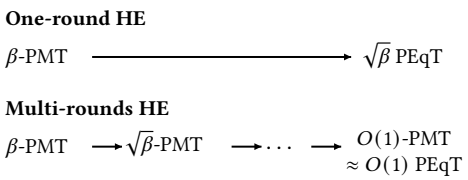

Figure 8: The upper one comes from the naive application of scalar multiplication-only strategy. Our HE-subroutine with an identical mask enables the lower one.

We finally remark that this new approach has almost no benefit when we evaluate polynomial evaluation using non-scalar multiplications, unfortunately. Recall that Paterson-Stockmeyer method evaluates the polynomial with $O(\sqrt{\beta})$ non-scalar multiplications, surprisingly independent of the choice of α. Therefore, the original HE phase can already set very small $\alpha(< 10)$, and then further HE subroutines provide little performance gain.

5.2.1 Reusable Offline Phase. The preprocessing phase of the first HE-subroutine call can be done offline, but the preprocessing phase of the following calls should be counted as online time as their input is determined by the previous HE-subroutine calls. However, we would also like to remark that the offline phase can be reused as long as the large set is static: The large set holder can compute polynomial coefficients without the multiplicative mask offline, and process it in the online phase with freshly sampled $c_i \leftarrow \mathbb{F}$. Note that such multiplication of c_i can be done as plain integers rather than HE scalar multiplication, and hence takes a tiny computational cost.

5.2.2 Unique Intersection Condition. There remains one subtle issue. Recall that an identical mask is secure only when there is at most one match, and we ensure the initial simple table satisfies that property. However, outputs of the first HE subroutine, say r and D, are not guaranteed to satisfy such conditions. Then any duplication in D should be removed and padded with dummy elements. This removal of duplication does not harm the correctness, and we simply take one random element from $\mathbb{F}$ and fill it in all removed

positions. Note that the unique intersection condition still holds although we use the same element to pad unless the dummy element is the same with r, which happens with a small probability of $1/t$ per bin. Since we have $O(n_y)$ hash bins, this padding process introduces another chance of false positive about $1-(1-1/t)^{O(n_y)} = O(n_y/t)$.

5.2.3 Correctness and Security. For correctness, it can be easily checked that there are no false negatives, and we only need to consider the false positives. First, the initial (permutation-based) hashing into $\log t$ bits has $O(n_x/t)$ chance of a false positive from a hash collision. Second, the zero coefficient sampling of §5.1 can occur with probability $O(\alpha n_y/t)$. Lastly, the dummy padding of §5.2.2 fails with probability $O(n_y/t)$. Recall first that we assume that $n_x \gg n_y$, and second the number of partitions α is much smaller than the bin size $O(n_x/n_y)$ by definition. Thus we conclude the first term $O(n_x/t)$ is dominant and the final false positive probability is also $O(n_x/t)$. Detailed computation without $O(\cdot)$ notation can be found in Appendix E.

As we can ensure the unique intersection condition for all HE-subroutine calls, the security proof can be easily adapted from Theorem 4.1. Due to the space limit, we place the proof in Appendix F.

Theorem 5.2. *The protocol in Figure 7 realizes circuit-PSI functionality against a semi-honest adversary, assuming a semi-honest secure 2PC.*

6 PERFORMANCE EVALUATION

In this section, we provide implementation results and a comparison with previous works. We use machines equipped with 3.50GHz Intel Xeon processors with 128GBs of RAM. The network settings are simulated on localhost using the Linux `tc` command, and we denote 30Gbps bandwidth and 0.05ms RTT by LAN, and 100Mbps bandwidth and 80ms RTT by WAN.

6.1 Evaluation of Our Proposals

We refer to our first proposal of Figure 4 of Section 4 by 'Construction 1' and our second proposal of Figure 7 of Section 5 by 'Construction 2'. We also refer to the naive trade-off protocol that consists of a one-round HE subroutine with only scalar multiplications (the upper one in Figure 8) by 'Naive Tradeoff' (or 'Naive').

We first compare the asymptotic behaviors of our proposals, summarized by Table 3. We also implement our protocols and then provide some discussion based on the experimental results. Our HE phase implementation is mainly adapted from Microsoft APSI library [1]. For the 2PC phase, we implement the GMW protocol [12] to evaluate the equality circuits and test membership. GMW protocol requires several calls of *oblivious transfer* (OT), and we use the Ferret OT library [35].

6.1.1 Asymptotic Evaluations. Clearly, all protocols require $O(n_x)$ computational cost for the first offline phase, and we focus on the online cost. Note that the cuckoo hashing splits the original problem into $O(n_y)$ subproblems of the private membership test (PMT) with $\beta = O(n_x/n_y)$ size sets. Thus, it suffices to focus on the computation and communication cost for each PMT.

In 'Construction 1', the small set holder sends $O(\log(\beta/\alpha))$ encryptions of powers using windowing [7] or extremal postage-stamp base [8], where α is the number of partitions. Then the large set holder homomorphically evaluates the inclusion polynomial using $O(\sqrt{\beta})$ non-scalar multiplications and $O(\beta)$ scalar multiplications, using the Paterson-Stockmeyer algorithm [8]. Note that again the computational cost for polynomial evaluations is independent of the number of partitions α. As a result, the large set holder sends back α resulting ciphertexts, and then two parties execute the α PEqTs to have Boolean shares of the membership results, which are the output of the circuit-PSI. Then the total communication is $O(\log(\beta/\alpha) + \alpha)$ for each bin, which is $O(\log \beta)$ assuming the partition size $\alpha = O(1)$.

In 'Construction 2', the small set holder sends $O(\sqrt{\beta})$ encryptions for each bin of size β. Then the large set holder uses only $O(\beta)$ scalar multiplications to evaluate the inclusion polynomial and sends back $O(\sqrt{\beta})$ encryptions. The next HE calls are executed with bin size less than $O(\sqrt{\beta})$, and then we can conclude that the first HE call is dominant in recursive calls. Then the two parties recursively reduce the bin size until $O(1)$, and finally perform PMT on $O(1)$-sized set, which has the same asymptotic cost with PEqTs $O(1)$ times.

6.1.2 Experimental Evaluations. Table 2 shows comparison based on concrete experiments. Note that the 'Construction 2' row and the 'Naive' row show the concrete advantage of our recursion idea. We also remark that the HE phase of our proposals requires very complicated parameterization related to several optimization techniques. We detail the whole parameterization and underlying rationale in Appendix B.

Offline Setup Cost. As expected, the second construction shows faster performance than the first one in the online phase while increasing communication. However, it would seem weird that the offline phase also speeds up in the second construction, since the removal of non-scalar multiplication only affects to online phase.

We provide a brief explanation about this. The dominant process in the offline phase is the coefficient computation that evaluates elementary symmetric polynomials[4] on each partition. Computation of elementary symmetric polynomials takes $O(|P|^2)$ for each partition P. Now recall that the size of each partition is $O(\beta/\alpha)$ where $\beta = O(n_x/n_y)$ in our construction. Note that the first construction has to take quite a small α so that the cost of the online 2PC phase is reasonable. On the other hand, the second construction can set much larger $\alpha = O(\sqrt{\beta})$ for the first HE call, because it can be further reduced by following HE subroutines. Then each size of the partition is much smaller in the second construction, and this leads to the speed-up of the offline phase.

This also explains another seemingly weird point; the offline cost of Construction 1 drops along with increased n_y for fixed n_x, which is because the size of each partition is likely to decrease when n_y grows.

6.2 Comparison to Previous Works

We also provide comparisons with other protocols for several input set sizes in Table 4, including state-of-the-art related works; Private

[4]The polynomials of the form $\sum_{1\le i_1<i_2<\cdots<i_k\le n} x_{i_1}\cdots x_{i_k}$.

n_x	n_y	Protocol	Offline	Online								
			Setup (s)	Comm. (MB)			LAN (s)			WAN (s)		
				Total	HE	2PC	Total	HE	2PC	Total	HE	2PC
2^{20}	5535	Construction 1 (§4)	5.42	8.06	6.75	1.31	4.85	3.07	1.78	8.45	4.18	4.27
		Construction 2 (§5)	3.79	12.15	11.23	0.92	2.59	0.98	1.61	8.28	4.30	3.98
		Naive (§5)	3.79	13.94	9.33	4.61	4.75	0.5	4.25	9.26	2.31	6.95
	11041	Construction 1 (§4)	4.07	10.89	9.51	1.38	5.04	3.13	1.91	8.60	4.55	4.05
		Construction 2 (§5)	4.27	19.27	17.43	1.84	3.44	1.39	2.1	9.48	5.14	4.34
		Naive (§5)	4.27	21.54	14.63	6.91	6.75	0.67	6.08	12.0	2.90	9.10
2^{24}	5535	Construction 1 (§4)	351	11.7	10.3	1.45	20.92	18.9	2.02	22.2	17.8	4.39
		Construction 2 (§5)	61.1	40.3	39.6	0.69	6.53	4.75	1.78	13.6	10.1	3.49
		Naive (§5)	61.1	52.4	34.4	17.9	19.32	3.62	15.7	26.6	5.89	20.7
	11041	Construction 1 (§4)	320	16.4	14.6	1.76	24	25.8	1.85	30.4	25.9	4.56
		Construction 2 (§5)	62.8	62.5	61.1	1.38	7.7	5.53	2.15	16.3	12.0	4.29
		Naive (§5)	62.8	78	50.4	27.7	28.7	3.82	24.9	39.3	6.98	32.3

Table 2: Comparisons between our proposals, where the best item for each pair of n_x, n_y is marked with blue. 'Naive' rows and 'Construction 2' rows show the effect of the recursive HE calls. All experiments are executed with a single thread and satisfy at least 128 bit security and at most 2^{-40} false positive probability.

	Cons. 1 (§4)	Cons. 2 (§5)	Naive
Non-scalar Mult.	$O(\sqrt{n_x n_y})$	0	0
Scalar Mult.	$O(n_x)$	$O(n_x)$	$O(n_x)$
PEqTs	$O(n_y)$	$O(n_y)$	$O(\sqrt{n_x n_y})$
Comm.	$O(n_y \log(n_x/n_y))$	$O(\sqrt{n_x n_y})$	$O(\sqrt{n_x n_y})$

Table 3: The first column corresponds to our first proposal of §4, where $\alpha = O(1)$. The second column corresponds to our second proposal of §5. The last column corresponds to the naive trade-off version. Compared to the second column, it requires asymptotically many PEqTs.

Join & Compute (PJC) [24] and OPPRF-based Circuit-PSI [5, 11, 33]. Below is a summary, and then details follow.

- Our constructions have definite advantages over PJC 1 to have no small set holder side storage and over PJC 2 to have much faster performance (even considering offline and online timing).
- PJC 1 and Construction 2 generally show similar online performance, where PJC 1 has better computation cost and our constructions have better communication cost.
- As shown in $n_y = 2^8$ columns, our constructions poorly scale to smaller $n_y < 2^{12}$, due to a technical issue related batching property of HE.
- As shown in $n_x = 2^{20}, n_y = 2^{16}$ case, OPPRF-based Circuit-PSI shows quite decent performance for some degree of unbalanced n_x/n_y.

6.2.1 Comparison to Private Join & Computes. There is another family of PSI-with-computation protocols named Private Join & Compute (PJC), which is built on the Diffie-Hellman assumption. The first proposal [16] had no special interest in an unbalanced setting and in fact, had linear complexity $O(n_x + n_y)$. Furthermore, the coefficients on n_x and n_y are similar, since their protocol is somewhat *symmetric*; both parties perform $O(n_x + n_y)$ encryptions. Recently, [24] proposed two protocols for asymmetric cases, which we named PJC 1 and PJC 2. Their purpose is closest to ours, to the best of our best knowledge, and they would be the main comparison target of our work.

We note that PJCs further compute arithmetic shares of the associated value of the large set holder beyond circuit-PSI Boolean shares, what the authors call PIR-with-default. We cannot find in [24] the performance reports of the exact counterpart of circuit-PSI, and Table 4 numbers correspond to PIR-with-default cost. However, remark that our protocols can also be efficiently modified to have the same functionality as PIR-with-default, whose additional cost is just one more polynomial evaluation. See A for details.

As we found no publicly available implementation of PJCs, we simply take the numbers in the original paper for $n_x = 2^{20}$, and *compute* other numbers based on asymptotic complexity. Moreover, we presume that the original reports are conducted on a LAN network simulated by localhost[5]. Then we simply estimate the WAN timings of the PJCs by $T + C/x$, where T stands for the reported time, C for communications (in MB), and x for bandwidth (in MBps). Note that this estimation is quite advantageous for PJCs, since it does not reflect real network latency, and the actual running time on WAN would take more than our estimation.

Comparison to PJC 2 is immediate since our protocols show sharply improved performance, even if we count offline and online costs both. Compared to PJC 1 which has online cost $O(n_y)$, we have to concede that our protocols are asymptotically worse since ours anyhow has a dependency with n_x. However, our experiments show that our protocols have comparable concrete performances. Moreover, the linear dependency on n_y is quite heavy in PJC 1, and it rapidly becomes inefficient for larger n_y, for example, $n_y = 2^{16}$.

We emphasize the huge difference in the offline phase. Our protocols have a significantly shorter setup time and require *zero* communication, whereas [24] protocol requires a much longer processing time and a huge amount of communication. The timing for setup is probably not so critical because it can be done on the large set

[5] We found no specific description of the network environment in the original paper.

		n_x	2^{20}			2^{24}			2^{26}	
		n_y	2^8	2^{12}	2^{16}	2^8	2^{12}	2^{16}	2^{12}	2^{16}
Online	LAN Time (s)	Construction 1 (§4)	4.96	4.85	24	20.9*	20.9	61.7	55.6	192
		Construction 2 (§5)	2.59*	2.59	11	7.54*	7.54	19.2	18.9	36.2
		PJC 1 [24]	0.62	4.22	47.2	0.62	4.22	47.2	4.22	47.2
		PJC 2 [24]	3,026	2,134	2,228	48,384	33,997	35,389	-	-
		OPPRF-CPSI	3.3	3.31	4.28	35.42	35.88	38.1	159	166
	WAN Time (s)	Construction 1 (§4)	7.98	8.43	32.2	27.9*	27.9	74.9	63.8	209
		Construction 2 (§5)	7.8*	7.8	21.3	15.1*	15.1	38.7	28.8	60.2
		PJC 1 [24]	1.18	13.6	190.7	1.18	13.6	191	13.6	191
		PJC 2 [24]	3,028	2,151	2,374	48,384	34,017	35,564	-	-
		OPPRF-CPSI	5.5	5.89	7.77	75	76.7	79.4	312	329
	Comm. (MB)	Construction 1 (§4)	4.8	8.1	61	11.7*	11.7	94.2	22.5	242
		Construction 2 (§5)	12.2*	12.2	66.3	45.2*	45.2	187.3	84.8	323
		PJC 1 [24]	7	117	1,794	7	117	1,794	117	1,794
		PJC 2 [24]	29	213	1,821	34.6	256	2,185	-	-
		OPPRF-CPSI	23.4	25.7	32.8	374	406	442	1,623	1,752
Offline	Time (s) (Large-side Setup)	Construction 1 (§4)	7.64	5.4	7.4	465*	465	101	1,916	440
		Construction 2 (§5)	3.8*	3.8	7	68*	68	80	323	315
		PJC 1 [24]	137	137	137	2,207	2,207	2,207	8,828	8,828
		PJC 2 [24]	0	0	0	0	0	0	0	0
		OPPRF-CPSI	0	0	0	0	0	0	0	0
	Comm. (MB) (Small-side Storage)	Construction 1 (§4)	0	0	0	0	0	0	0	0
		Construction 2 (§5)	0	0	0	0	0	0	0	0
		PJC 1 [24]	465	465	465	7,440	7,440	7,440	29,760	29,760
		PJC 2 [24]	0	0	0	0	0	0	0	0
		OPPRF-CPSI	0	0	0	0	0	0	0	0

Table 4: Comparison to prior works. All executions are executed with a single thread and satisfy at least 128 bit security and 2^{-40} false positive probability. For PJCs [24] whose implementation is not publicized, we take some numbers in the original paper and compute the others based on asymptotic behavior. For each pair (n_x, n_y) and each feature, the best item is colored blue. We also highlight in red that PJC 1 has large offline communication.
(*) Our constructions cannot have efficient parameterizations for $n_y = 2^8$ case, due to the usage of batching. We run the experiment by padding the small set with dummy items up to size 2^{12}. For more explanation, and §6.2.3.

side alone. However, offline communication really matters, because the small set holder has to store the received data for the online phase. Such a burden on storage could be a significant obstacle when the small set holder is a mobile or IoT device that probably has insufficient storage. Moreover, although offline storage can be reused for multiple calls, it is not totally offline in the sense that the small set holder starts from the download (or access) of such a large database when it initiates circuit-PSI. For these reasons, our protocols of zero offline communication have quite appealing features.

We found that the authors of [24] already compared PJC with the random mask idea of [6] where this work starts, but it has some points to be fixed or updated. See Appendix D for details.

As a final remark, one may think that our efficiency gain mainly comes from the batching property of HE. However, PJC [24] schemes also utilize HE to some extent to enjoy the benefit of the batching property of HE, which says that our efficiency gain rather comes from better logic based on a polynomial evaluation to check membership.

6.2.2 Comparison to OPPRF-based Circuit-PSIs. OPPRF-based protocols [5, 10, 30, 33] are the most efficient family of circuit-PSI protocols for balanced input sets. As we found no performance reports on unbalanced sets from the original papers [5, 33], we provide numbers obtained from our own implementation that combines the state-of-the-art OPPRF-based protocol [5, 33]. We would like to remark that OPPRF-based protocols also require $O(n_y)$ times of PEqT, and we also use the fast OT extension library Ferret [35] for that.

Asymptotically, they have both computational and communication complexity linear on n_x and n_y, say $O(n_x + n_y)$. However, we observed that OPPRF-based protocols have a quite smaller coefficient on the large set size n_x than n_y, whose details can be found in Appendix C. This implies that they are also expected to remain efficient to some extent of unbalance. Our experiments indeed indicate that OPPRF-based protocols already have a decent performance to some degree of unbalance. Based on our experiments in Table 4, they show even better performance when the ratio is small (see $n_x = 2^{20}$ and $n_y = 2^{16}$ case). Roughly judging from our experiments, the break-even point is about $n_x/n_y \approx 2^8$.

This protocol actually has some procedures that can be done offline, especially random OTs for equality checks. However, the portion is not so large as it depends on the small set size n_y, and more importantly, such offline computations cannot be reused while

offline computations of our and PJC can be reused for multiple calls. Therefore, we include them in the online cost.

6.2.3 Further Set Sizes. We first consider further smaller n_y. Note that our protocols insert n_y elements into $O(n_y)$ size cuckoo table, and split each item into d HE plaintext items. Then total $O(d \cdot n_y)$ items are batched by batching technique. However, for extremely small n_y, the total number of items is smaller than the batching packing slots. In this case, we have to pad empty batching slots, and this means that the cost of our protocol cannot be continuously smaller along with n_y. The batching slot number N is taken at least 2^{13} in our parameterization, which implies that our protocol performance remains similar for $n_y \leq 2^{12}$. This problem was already in the plain PSI protocol [8], but it could be partially compensated by taking a larger d while having a smaller plaintext slot $\mathbb{Z}_t$. However, in our case, the ciphertext modulus margin σ should be added in all d splits, and such compensation becomes harder. We note that two $n_y = 2^8$ columns of Table 4 explicitly show this drawback. Our constructions have no much difference from $n_y = 2^{12}$, but PJC enjoys linearly decreasing online performance than $n_y = 2^{12}$.

Next, we consider much larger n_x than 2^{26}. Note that the numbers in Table 4 quite faithfully follow asymptotic cost, so that the concrete performances can be easily extrapolated beyond tables. Then we can expect that OPPRF-based protocols still show decent performance for some degree of unbalance, for example, $n_x/n_y \leq 2^8$. We then focus on a larger ratio of $n_x/n_y \geq 2^8$ where PJC and our protocols become better than OPPRF-based ones. Since the dependency on n_y is heavier for PJC than ours, PJC would be concretely better when n_y itself is small. Such a cross-point of n_y where PJC is better would get larger along with n_x because ours anyhow have a dependency with n_x, whereas PJC has no dependency. See $n_x = 2^{26}$ cases in Table 4; PJC 1 is better in some aspects when $n_y = 2^{12}$, but our Construction 2 is better when $n_y = 2^{16}$.

However, we again remark that for both smaller n_y and larger n_x cases, our proposals always have the strong advantage of zero offline storage. We believe there would be many applications or environments where this strength could be much useful.

REFERENCES

[1] APSI 2021. APSI: C++ library for Asymmetric PSI. https://github.com/microsoft/APSI. (2021).

[2] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. 2012. Multiparty Computation with Low Communication, Computation and Interaction via threshold FHE. In *EUROCRYPT*. Springer, 483–501.

[3] Jean-Claude Bajard, Julien Eynard, M Anwar Hasan, and Vincent Zucca. 2016. A full RNS variant of FV like somewhat homomorphic encryption schemes. In *SAC*. Springer, 423–442.

[4] Prasad Buddhavarapu, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Vlad Vlaskin. 2020. Private Matching for Compute. *IACR Cryptol. ePrint Arch.* 2020 (2020), 599.

[5] Nishanth Chandran, Divya Gupta, and Akash Shah. 2022. Circuit-PSI with linear complexity via relaxed batch OPPRF. *Proceedings on Privacy Enhancing Technologies* 2022, 1 (2022), 353–372.

[6] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. 2018. Labeled PSI from fully homomorphic encryption with malicious security. In *ACM CCS*. 1223–1237.

[7] Hao Chen, Kim Laine, and Peter Rindal. 2017. Fast Private Set Intersection from Homomorphic Encryption. In *ACM CCS*. 1243–1255.

[8] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. 2021. Labeled PSI from Homomorphic Encryption with Reduced Computation and Communication. *ACM CCS* (2021).

[9] Geoffroy Couteau, Peter Rindal, and Srinivasan Raghuraman. 2021. Silver: Silent VOLE and Oblivious Transfer from Hardness of Decoding Structured LDPC Codes. In *CRYPTO*. Springer, 502–534.

[10] Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. 2021. Private Set Operations from Oblivious Switching. (2021).

[11] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2021. Oblivious key-value stores and amplification for private set intersection. In *CRYPTO*. Springer, 395–425.

[12] O. Goldreich, S. Micali, and A. Wigderson. 1987. How to Play ANY Mental Game. In *STOC*. Association for Computing Machinery, New York, NY, USA, 218–229. https://doi.org/10.1145/28395.28420

[13] Christoph Hagen, Christian Weinert, Christoph Sendner, Alexandra Dmitrienko, and Thomas Schneider. 2021. All the Numbers are US: Large-scale Abuse of Contact Discovery in Mobile Messengers. *NDSS* (2021).

[14] HElib 2021. HElib: An implementation of homomorphic encryption. https://github.com/homenc/HElib. (2021).

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

[16] Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. 2020. On Deploying Secure Computing: Private Intersection-Sum-with-Cardinality. In *Euro S&P*. IEEE, 370–389.

[17] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending Oblivious Transfers Efficiently. In *CRYPTO*. Springer, 145–161.

[18] Stanisław Jarecki and Xiaomin Liu. 2010. Fast Secure Computation of Set Intersection. In *SCN*. Springer Berlin Heidelberg, 418–435.

[19] Sreekanth Kannepalli, Kim Laine, and Radames Cruz Moreno. 2021. Password Monitor: Safeguarding passwords in Microsoft Edge. https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/. (2021).

[20] Ferhat Karakoç and Alptekin Küpçü. 2020. Linear complexity private set intersection for secure two-party protocols. In *CANS*. Springer, 409–429.

[21] Andrey Kim, Yuriy Polyakov, and Vincent Zucca. 2021. Revisiting Homomorphic Encryption Schemes for Finite Fields.. In *ASIACRYPT*. Springer.

[22] Ágnes Kiss, Jian Liu, Thomas Schneider, N Asokan, and Benny Pinkas. 2017. Private Set Intersection for Unequal Set Sizes with Mobile Applications. *Proc. Priv. Enhancing Technol.* 2017, 4 (2017), 177–197.

[23] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. 2016. Efficient Batched Oblivious PRF with Applications to Private Set Intersection. In *ACM CCS*. 818–829.

[24] Tancrede Lepoint, Sarvar Patel, Mariana Raykova, Karn Seth, and Ni Trieu. 2021. Private join and compute from PIR with default. In *ASIACRYPT*. Springer, 605–634.

[25] Yehuda Lindell. 2017. How to simulate it–a tutorial on the simulation proof technique. *Tutorials on the Foundations of Cryptography* (2017), 277–346.

[26] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On ideal lattices and learning with errors over rings. In *EUROCRYPT*. Springer, 1–23.

[27] Catherine Meadows. 1986. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *S&P*. IEEE, 134–134.

[28] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2020. PSI from PaXoS: fast, malicious Private Set Intersection. In *EUROCRYPT*. Springer, 739–767.

[29] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. 2015. Phasing: Private Set Intersection Using Permutation-based Hashing. In *USENIX Security*. USENIX Association, Washington, D.C., 515–530. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/pinkas

[30] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. 2019. Efficient Circuit-based PSI with Linear Communication. In *EUROCRYPT*. Springer, 122–153.

[31] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2014. Faster Private Set Intersection Based on OT Extension. In *USENIX*. USENIX Association, USA, 797–812.

[32] Amanda C Davi Resende and Diego F Aranha. 2018. Faster unbalanced private set intersection. In *International Conference on Financial Cryptography and Data Security*. Springer, 203–221.

[33] Peter Rindal and Phillipp Schoppmann. 2021. VOLE-PSI: Fast OPRF and Circuit-PSI from Vector-OLE. In *EUROCRYPT*. Springer.

[34] SEAL 2021. Microsoft SEAL (release 3.7). https://github.com/microsoft/SEAL. (2021). Microsoft Research, Redmond, WA.

[35] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. 2020. Ferret: Fast extension for correlated OT with small communication. In *ACM CCS*. 1607–1626.

[36] Samee Zahur, Mike Rosulek, and David Evans. 2015. Two halves make a whole. In *EUROCRYPT*. Springer, 220–250.

A HANDLING ASSOCIATE VALUES

First, it is rather easy to handle the associated value on the side of the small set holder, since the final output of circuit-PSI is uniquely bound with each small set holder side item by the cuckoo table. For further computations, the small set holder can feed the corresponding associated value along with a Boolean share of $\mathbf{1}(y \in X \cap Y)$ derived from circuit-PSI.

Handling large set holder side associated value is non-trivial since the large set holder cannot know which element to bind with the associated value from the circuit-PSI output. Here, *labeled PSI* idea [6, 8] can be efficiently applied to this problem. The original purpose of labeled PSI is to let the small set holder learns the associated value v_i for each of its items $y_j = x_i \in X \cap Y$. The basic idea is to let the large set holder construct a polynomial G such that $G(y) = v_i$ if $y = x_i$. Then the large set holder can HE evaluate $G(y)$ in the similar way to evaluate inclusion polynomial $F(y)$, and send back the encryption of $G(y)$ so that the small set holder learns the associated value v_i for $y = x_i$.

This idea can be efficiently adapted to our situation, but with a slight modification. Since the associated value is likely to be easily distinguished from the random value, the naive application of the idea may let the small set holder infer the matching result. This can be remedied by allowing the large set holder to sample the additive mask r and construct the polynomial so that $G(y) = v_i - r$ if $y = x_i$. As a result, the small set holder and the large set holder end with an additive share of v_i if $y = x_i$. This additive share can be fed into further computation along with a Boolean share of $\mathbf{1}(y \in X \cap Y)$ derived from circuit-PSI.

B DETAILED PARAMETERIZATIONS

Table 5 provides full details for our parameterizations, which can be easily converted to `json` format for [1]. We also find better parameterizations for the plain PSI [8] by choosing a smaller item split d.

For the first proposal, we found several parameterizations provided in the previous work [8] available in [1], which are tailored for the plain PSI. We slightly change some parameters considering the additional cost of the 2PC phase and additional ciphertext modulus margin for noise flooding. As an underlying rationale, the number of item splits d is chosen as small as possible, and we also take smaller α so that the post-2PC phase is not so heavy. All parameters satisfy false-positive probability at most 2^{-40}, and the underlying HE parameters satisfy 128-bit security.

For the second proposal, we let the HE subroutines reduce the input set sizes to square root until the set size is sufficiently small; precisely, we stop after reaching ≤ 4. The corresponding HE parameters can be set to support one-depth multiplication. Recall that it was essential to use $d = 1$ (embed one item in one slot), and hence batching cannot be fully exploited for extremely small sets like $n_y = 2^8$ since it typically batches at least 2^{12} items. Thus we cannot find efficient parameterization for those cases in this protocol, and omit this case in our experiments. There remains one more subtle issue with respect to implementation. For very large inputs, the plaintext modulus t should satisfy $\log t \approx 40 + \log n_x$ to have a false positive probability less than 2^{-40}. However, our implementation base HE library SEAL only supports plaintext modulus up to 60 bits,

	n_x	n_y	T	m	α	d	$\log t$	N	$\log q$
Plain PSI [8]	2^{20}	2^{12}	2^{13}	98	5	2	34	2^{13}	204
		2^{13}	2^{14}	98	3	2	35	2^{13}	208
	2^{24}	2^{12}	2^{13}	1304	5	2	38	2^{14}	288
		2^{13}	2^{14}	1304	3	2	39	2^{14}	292
Cons. 1 §4	2^{20}	2^{8}	2^{12}	228	4	2	33	2^{13}	197
		2^{12}	2^{13}	98	5	2	34	2^{14}	246
		2^{13}	2^{14}	98	3	1	60	2^{14}	352
		2^{16}	$3 \cdot 2^{15}$	40	2	2	35	2^{13}	200
	2^{24}	2^{12}	2^{13}	1304	5	2	38	2^{14}	316
		2^{13}	2^{14}	1304	3	2	39	2^{14}	320
		2^{16}	$3 \cdot 2^{15}$	98	7	2	36	2^{14}	255
	2^{26}	2^{12}	2^{13}	1304	20	2	38	2^{14}	292
		2^{16}	$3 \cdot 2^{15}$	98	24	2	36	2^{14}	255
Cons. 2 §5	2^{20}	2^{12}	2^{13}	24	4	1	60	2^{13}	177
		2^{13}	2^{14}	18	4	1	60	2^{13}	177
		2^{16}	$3 \cdot 2^{15}$	8	3	1	60	2^{13}	177
	2^{24}	2^{12}	2^{13}	83	3	1	60	2^{13}	177
		2^{13}	2^{14}	55	3	1	60	2^{13}	177
		2^{16}	$3 \cdot 2^{15}$	24	2	1	60	2^{13}	177
	2^{26}	2^{12}	2^{13}	158	4	1	60	2^{13}	177
		2^{16}	$3 \cdot 2^{15}$	55	2	1	60	2^{13}	177

Table 5: Parameters used in our experiments. T: hash table size. m: the size of the partition (of the first HE call). α: the partition numbers per bin (of the last HE call). d: number of splits of one item. t: HE plaintext modulus. N: HE batching slots. q: HE ciphertext modulus.

and we cannot conduct experiments with the required plaintext modulus size $\log t \approx 40 + \log n_x$ for $n_x > 2^{20}$ cases. For that cases, we use the plaintext modulus $\log t = 60$, but the ciphertext modulus q is taken assuming correct plaintext modulus $\log t = 40 + \log n_x$. Precisely, we first take an initial HE ciphertext modulus q by the minimal one where the protocol without noise flooding is correct, and then augment $\sigma = 40$ bits margin on them.

We finally remark that Construction 2 is much easier to parameterize in several respects, which can be an implicit strength. It simply takes the partition size by the square root of the current bin size and has no additional false positive probability from item splitting that makes it easier to take the plain modulus t, and only requires depth 1 scalar multiplication that makes it easier to take the ciphertext modulus q.

C OPPRF-BASED CIRCUIT-PSI

OPPRF-based circuit-PSI protocols follow the framework due to Pinkas *et. al.* [30], which consists of three consecutive stages: hashing, OPPRF, and 2PC. We investigate the communication cost of each stage. The hashing phase is the same as ours; cuckoo and simple hashing incurs no communication. Then in the OPPRF phase, two parties execute OPPRF that starts with an initial OPRF call, and one party (typically a large set holder) computes some hint (of programmable PRF) and sends the hint to the other party. There are many OPRF constructions [11, 23, 28, 33], but they commonly require $C_{\mathsf{oprf}} \cdot 1.3\lambda n_y$ bits communication where 1.3 comes from the size of the hash table. The state-of-the-art C_{oprf} is 1.3 due to [11]. The hint of programmable PRF typically has size $3C_{\mathsf{pprf}}(\sigma + \log n_y)n_x$ bits [5, 30, 33], where 3 is the number of hash functions

and σ is for the probability of failure $2^{-\sigma}$. The state-of-the-art C_{pprf} is also 1.3 due to [11]. Finally, two parties compute a programmable PRF to have a pair of strings of length $\ell = \sigma + \log n_y$ for each hash table bin and perform an equality check by 2PC. This can be done by generic 2PC protocol with $O(n_y \cdot \ell)$ bit communications, and concretely GMW protocol requires $\approx 2n_y \cdot \ell$ bit communications using state-of-the-art OT extensions [9, 35]. Putting everything together, the total bit communication is

$$(1.7\lambda + 2\sigma + 2\log n_y) \cdot n_y + 3(\sigma + \log n_y) \cdot n_x,$$

which becomes under our choice of $\lambda = 128$ and $\sigma = 40$

$$\approx (297 + 2\log n_y)n_y + (156 + 3.9\log n_y)n_x.$$

D UPDATES ON COMPARISON IN PJC

We provide several update points from the comparison of the PJC and the HE-based protocol in [24]. First, it estimated the running time for the final 2PC phase of [6] by 40% of [30], based on the fact that the same 2PC procedure occupies 40% timing of [30]. However, such 40% portion is only valid for a balanced input case, and the final 2PC portion becomes much smaller for an unbalanced case since it depends on the small set size (precisely, the hash table size). Second, it estimated the communication cost of the final 2PC phase also following [30], which used a somewhat classic OT extension protocol [17]. There has been a significant improvement in OT extensions, especially the communication cost, which is much lower than [17]. This leads to a smaller communication cost for our proposals. Meanwhile, such improvement of OT extensions seems to not have a critical effect on PJC protocols, especially in offline setup cost that mainly consists of the (garbled) bloom filter construction. Finally, they estimated the offline cost of [6] including OPRF preprocessing, but we argue that such OPRF can be skipped. This reduces the offline cost of HE-based protocols, as shown in Table 2.

E DETAILS ON FALSE POSITIVES

False Positives from Item Splits. The original paper [8] does not provide a detailed discussion of false positive probability from item splits, but we found a relevant computation in the implementation code [1] of [8], and we provide the contents here.

The idea of item split is to evaluate inclusion polynomials in d splits of long items. Then the large set holder sends d resulting ciphertexts that correspond to each substring, and the small set holder concludes $y \in X$ if all ciphertexts are decrypted into zero. It introduces some chance of false positive where each substring matches with different items of X despite $y \notin X$, so that the small set holder falsely concludes $y \in X$. Toward detailed computation, we have to consider bin partition α. By denoting each partition size by $m := \beta/\alpha$, the probability of a false partial match is $1 - (1 - 1/t)^m \approx m/t$. For a false positive on this partition, such partial match has to occur on every substring, whose probability is $p = (m/t)^d$. The probability p should not happen for all α partitions and all $(1+\varepsilon) \cdot n_y$ hash bins, and therefore the final false positive probability is $1 - (1-p)^{(1+\varepsilon)\alpha n_y} \approx (1+\varepsilon)m^d \alpha n_y / t^d$. It holds that $2^\ell \approx t^d$ for the original item length ℓ, and by substituting $m = \beta/\alpha$ and $\beta \approx n_x/n_y$, we have

$$(1+\varepsilon)(n_x/\alpha n_y)^d \alpha n_y / 2^\ell.$$

We remark that, although the formula seems to make sense for $d = 1$, there is no need to consider this false positive in this case because there would be no false matching.

False Positives of Construction of §4. Prior to constructing hash tables, it performs an initial hashing into some fixed-length ℓ. Assuming permutation-based hashing, the false positive probability of hash collision would be $n_x/2^\ell$. We then have additional false positives from the item split as elaborated above. Therefore, to ensure false positive less than $2^{-\sigma}$, it should hold that

$$\log(n_x + (1+\varepsilon)(n_x/\alpha n_y)^d \alpha n_y) + \sigma \le \ell.$$

False Positives of Construction of §5. In this case, we also perform an initial hashing of false positive probability $2^{-\ell} \cdot n_x$. This construction is only possible with $d = 1$ case, and then the HE plaintext is chosen $t \approx 2^\ell$. We do not have to consider false positives from item split since $d = 1$, but it has additional false positives from algorithmic modifications. More precisely, the zero coefficient sampling introduces $(1+\varepsilon)n_y\alpha/2^\ell$, and dummy padding introduces $(1+\varepsilon)n_y/2^\ell$ additional false positive probability. These new false positives can occur for every HE-subroutine call, and hence the probabilities should be multiplied by the number of subroutine calls, say $\log\log(n_x/n_y)$. To summarize, in order to ensure a false positive less than $2^{-\sigma}$, it should hold that

$$\log(n_x + \log\log(n_x/n_y)(1+\varepsilon)(\alpha+1)n_y) + \sigma \le \ell.$$

F MISSING PROOFS

Proof for Theorem 4.1. It suffices to consider only the HE phase; Step 1 and 2, since we assume that the 2PC protocol is semi-honest secure.

We start with a simulation of the view of S in the HE phase, which is the important part related to the argument of Section 4.1. The only part we have to care about is Step 2-(7), where S receives ciphertexts of $p'_{k,i}$ and then outputs the decrypted result $p'_{k,i}$. Since the simulator knows the outputs $p'_{k,i}$, we can simulate the ciphertexts of $p'_{k,i}$ by fresh encryptions of $p'_{k,i}$. Note that S is only able to know the inner plaintext $p'_{k,i}$ from the ciphertexts in the real execution, thanks to the noise-flooding procedure of Step 2-(6). This makes our simulation that gives fresh encryptions of $p'_{k,i}$ indistinguishable from the real execution.

The simulation of the view L is quite straightforward and not so different from the previous proof for the plain PSI [8]. For this case, we only have to care about the fresh ciphertexts of $y_{k,j}$ sent from S in Step 2-(3). This can be easily simulated by random ciphertexts, which are indistinguishable from the ciphertexts of $y_{k,j}$ of real execution, thanks to IND-CPA security. Note that this replacement of ciphertexts has no effect on the correctness, as L outputs random masks $r_{k,i}$.

Proof for Theorem 5.2. Since the final 2PC phase is assumed to be semi-honest, we only need to consider the HE phase, which consists of recursive calls of the HE subroutine of Figure 6. Thus it suffices to show for each HE-subroutine call, the view of each party can be easily simulated.

We start from the simulation of the view of S. Note that the only part we have to simulate is the ciphertexts of $t_{k,i}$ in Step 2-(4) of Figure 6. To simulate this, we sample $t'_{k,i}$ from $\mathcal{U}(\mathbb{F}^\alpha)$ and define the ciphertext by fresh encryptions of $t'_{k,i}$. We then show that this simulation is indistinguishable from the real execution. First, note that S only obtains the decryption results from the ciphertexts thanks to noise flooding, and we only need to see whether the distributions of $t_{k,i}$ is also $\mathcal{U}(\mathbb{F}^\alpha)$. As Lemma 5.1 ensures that the outputs of the real execution $t_{k,i}$ are also distributed on $\mathcal{U}(\mathbb{F}^\alpha)$, this ends the proof.

For the simulation of the view of L, we simply simulate the ciphertexts sent from S in Step 2-(1) by random ciphertexts. This is indistinguishable from the real execution, thanks to the IND-CPA of the underlying HE.