Improved Circuit-based PSI via Equality Preserving Compression

Kyoohyung Han[1], Dukjae Moon[1], and Yongha Son[1]

Samsung SDS, Korea
{kh89.han, dukjae.moon, yongha.son}@samsung.com

Abstract. Circuit-based Private Set Intersection (circuit-PSI) enables two parties with input set X and Y to compute a function f over the intersection set $X \cap Y$, without revealing any other information. State-of-the-art protocols for circuit-PSI commonly involves a procedure that securely checks whether two input strings are equal and outputs an additive share of the equality result. More importantly, this procedure occupies the largest portion, roughly 90% computational or communication cost for circuit-PSI. In this work, we propose *equality preserving compression* (EPC) protocol that compresses the length of equality check targets while preserving equality using homomorphic encryption (HE) scheme, which is secure against the semi-honest adversary. We then apply our EPC protocol to previous circuit-PSI protocols framework and implement them. As a result, we achieve around 2x improvement on both communication and computational cost *at one stroke* than previous results.

Keywords: Private Set Intersection, Circuit-based Private Set Intersection, Homomorphic Encryption

1 Introduction

A two-party functionality of private set intersection (PSI) enables two parties P_0 and P_1 having respective input set X and Y to compute the intersection $X \cap Y$, without revealing any other information beyond the original set cardinality $|X|$ and $|Y|$ to each other.

There are many real-world applications related to PSI, and some of them only requiring the intersection set may find an efficient solution from PSI alone. However, there is another variant of PSI that outputs only $f(X \cap Y)$ for some target function f rather than the intersection set $X \cap Y$, and this would be more desirable for other applications. One typical but a popular example is PSI-Cardinality that computes cardinality of the intersection, where $f(X \cap Y) = |X \cap Y|$. Indeed these kinds of PSI are receiving growing attention from industry, for example, Google [21, 29] and Facebook [7] explored some variants PSI including PSI-Cardinality-with-Sum that computes the cardinality and the sum of associated values over the intersection set.

This PSI-with-computation notion is generalized to the *circuit-PSI* functionality, which outputs the intersection information in secret-shared form, instead of the intersection set itself. More precisely, for each element $x \in X$, circuit-PSI outputs each party random bits s_0 and s_1 respectively, such that $s_0 \oplus s_1 = 1$ if and only if $x \in X \cap Y$ (of course 0 otherwise). This is used as a general-purpose preprocessing, in the sense that two parties use the shares to perform target computation on the intersection. Notable examples would be PSI-Threshold that only reveals whether the cardinality of $X \cap Y$ is larger than some threshold, and private set union (PSU) that literally computes $X \cup Y$.

The work of Pinkas *et al.* [35] proposed a novel construction of circuit-PSI protocol which has linear communication complexity in the input set size. After that, several following works [8, 39] have proposed improved instantiation of the framework and those works indeed shows the state-of-the-art performance for circuit-PSI.

To generate final bits s_0 and s_1 in circuit-PSI, the framework involves $O(N)$ times of private *equality share generation* (ESG) that takes an input string from each party and outputs Boolean shares of the equality result of two strings. This is one of the main differences of circuit-PSI from plain PSI, where the latter one typically uses private *equality test* that simply outputs the equality result itself. For private equality test, there are many efficient method such as *oblivious pseudo-random functions* (OPRFs) [17, 24, 39]. However it is not directly applicable for ESG, and the most of circuit-PSI protocols perform ESG by other costly methods such as generic two party computation (2PC). It results in a huge performance gap between plain PSI and circuit-PSI, about $20 - 40$x. More importantly, the cost for ESG occupies the largest part of circuit-PSI, about 96% and 91% of the total communication in circuit-PSI protocols of [35] and [8] respectively. Recently reported work [39] applied Silent-OT [5] to reduce the communication burden of ESG, but this communication reduction comes at the cost of running time. Then it still takes over 20 times of running time than plain PSI protocols, which means ESG is also the most heavy part of circuit-PSI [39].

1.1 Our Contribution

Our work starts with an observation that all known methods for ESG have complexity linear in ℓ, the bit-length of strings. Some works [35, 39] simply exploited two party GMW protocol [19] by evaluating equality check circuit composed of $\ell - 1$ AND gates, and it naturally results in complexity linear in ℓ. After then [8, 16] proposed more efficient protocol that has improved communication burden, but it still suffered from linear complexity in ℓ.

- With a purpose of reducing workload of ESG, we propose a functionality what we call *equality preserving compression* (EPC) that converts two large integers into smaller integers, while preserving the equality condition. Then we construct a homomorphic encryption (HE) based efficient protocol realizing EPC functionality with semi-honest security. Asymptotically it compresses ℓ-bit input integers into $O(\log \ell)$-bits, with $\tilde{O}(\ell)$ computational and communication complexity.

- We then combine our EPC into the circuit-PSI framework of [35], which achieves semi-honest security. Our EPC protocol *perfectly* preserve equality, in other words with zero failure probability, and hence the correctness analysis for of previous circuit-PSI protocols remain exactly same. Moreover it provides concrete improvement since it changes the previously heaviest ESG part to be executed logarithmic sized input. Table 1 shows experimental results that apply our EPC protocol to previous works.

$N = 2^{20}$	Time (s)	Comms. (MB)
[39]-I	31.3	2674
w/ EPC	25.0	1019
Improve	1.25x	2.62x
[39]-II	154	255
w/ EPC	53.0	383
Improve	2.9x	0.66x
[8]	43.7	1158
w/ EPC	23.4	555
Improve	1.86x	2.08x

Table 1. Application of EPC on the previous circuit-PSI protocols for $N = 2^{20}$ size input sets. Time is measured on LAN network setting.

1.2 Related Works

Plain PSI. The early proposal of PSI is based on Diffie-Hellman (DH) [28], and this still serve as a basis of modern PSIs with considerably low communication cost but high computational cost. Recently many OPRF-based (plain) PSI protocols [9, 24, 32, 33, 39] have been reported with rather low computational cost, at the cost of communication burden.

PSI-with-functionality. Toward PSI with additional functionality, Google [21, 29] provides PSI-with-computation protocol stem from DH-based PSI, which is tailored for specific target functionality that reveals computing cardinality of the intersection and summing all associated values of the intersection sets. After then Facebook [7] further developed this to a protocol that letting two parties have additive shares of intersected elements, with a purpose of supporting general computation over the intersection set.

Circuit-PSI. As a more generalized concept, circuit-PSI is firstly proposed by [20] and then continuous improvements have been reported [12, 34, 36]. In particular [34] has a similarity with our paper, as their main idea is to cut-off the length of item while preserving equality, with a purpose of reducing the cost for equality check. However their technique is not applicable to currently best framework of circuit-PSI due to Pinkas *et al.* [35] based on *oblivious programmable PRF* (OPPRF). As OPPRF-based circuit-PSI framework shows the best performance, whose details are presented later in Section 3. We note that, despite the similarity of their names, construction of OPPRF is quite different to OPRF, and hence OPRF-based PSI protocol does not implies OPPRF-based circuit-PSI protocol. Indeed, we are aware of only one work [39] that constructs plain PSI and circuit-PSI from the same underlying idea. There is another concept of PSI-with-computation [16] different to circuit-PSI, which improves the efficiency of PSI-with-computation while additionally reveals the cardinality of intersection set as well as the desired function evaluation $f(X \cap Y)$.

HE in PSI field. There are also HE-based PSI approaches [10,11], which mainly focused on extremely unbalance-sized set cases. The first work [11] considered plain PSI, and the main idea is to solve private set membership (PSM) problem by HE after cuckoo/simple hashing, which is quite different to our use of HE. The following work [10] extended this protocol to PSI having associated value and strengthened the security to malicious setting, but HE is applied in the similar sense to the previous work. The authors of [10] leaved a short mention on circuit-PSI as a combination of their HE-based PSM protocol with the final equality share generation. As the circuit-PSI protocol was not the main interest of the paper, the authors merely mentioned that the final task can be done by 2PC without detailed analysis.

1.3 Roadmap

In Section 2, we recall the preliminaries including oblivious transfer and homomorphic encryption, and in Section 3, we present the state-of-the-art circuit-PSI framework due to [35]. In Section 4, we propose an equality preserving compression functionality concept and efficient protocol for that. Then in Section 5, we combine our proposed EPC protocol with the OPPRF-based circuit-PSI protocol to improve efficiency, and provide experimental results in Section 6.

2 Preliminary

2.1 Notations

We write vectors as bold lowercase letters, and matrices as bold uppercase letters. The i-th component of a vector $\mathbf{v}$ is denoted by v_i, and i, j-th entry of a matrix M is denoted by $m_{i,j}$. For an integer k, a set $\{1, \cdots, k\}$ is denoted by $[k]$. The logarithm function log is assumed to have base 2 unless specially denoted by $\log_w$ with base w. For any statement T that can be determined by true or false (Boolean), we denote $\mathbf{1}(T)$ be the truth value for the equality, i.e., it is 1 if T is true and 0 else.

2.2 Oblivious Transfers

An 1-out-of-n oblivious transfer (OT) of ℓ-bit input messages $(n,1)\text{-}\mathsf{OT}_\ell$ takes as input n messages $m_1,\cdots,m_n \in \{0,1\}^\ell$ from the sender and a choice index $c \in [n]$ from the receiver, and outputs m_c to the receiver and nothing to the sender. We also use a notion of 1-out-of-2 *correlated-OT* (COT) of ℓ-bit input messages $(2,1)\text{-}\mathsf{COT}_\ell$, where the sender inputs a correlation $d \in \{0,1\}^\ell$ and the receiver inputs a choice bit $b \in \{0,1\}$. Then the functionality outputs to the sender r and $d+r$ for a randomly chosen $r \in \{0,1\}^\ell$, and to the receiver $b \cdot d + r$. We write m times of $(n,1)\text{-}(\mathsf{C})\mathsf{OT}_\ell$ calls by $(n,1)\text{-}(\mathsf{C})\mathsf{OT}_\ell^m$.

There are protocols called OT-extension (OTe) that efficiently extend small numbers of *base* OTs to large numbers of OTs. Assuming that such small numbers of base OTs are done, the most typical IKNP OTe protocols execute $(2,1)\text{-}\mathsf{OT}_\ell$ and $(2,1)\text{-}\mathsf{COT}_\ell$ with communication $\lambda + 2\ell$ [22] and $\lambda + \ell$ [3] bits per one call, respectively, and KK OTe protocol executes $(n,1)\text{-}\mathsf{OT}_\ell$ with communication $2\lambda + n\ell$ [23]. Recently another method named Silent OT-extension [5] is proposed, which greatly reduces communication overhead of IKNP-style OT-extension, at the cost of increased running time. For sufficiently many OT and COT calls, for example more than 2^{20} calls, Silent OTe allows one to execute $(2,1)\text{-}\mathsf{OT}_\ell$ and $(2,1)\text{-}\mathsf{COT}_\ell$ with nearly $2\ell + 1$ and $\ell + 1$ bit communication per one call, respectively.

Boolean shares and Gate evaluations. For a bit $x \in \{0,1\}$, we say $x_0 \in \{0,1\}$ and $x_1 \in \{0,1\}$ satisfying $x = x_0 \oplus x_1$ be 2-party additive Boolean shares, or simply Boolean shares of x. Consider two bits x and y are shared as x_i and y_i by two party P_0 and P_1. Boolean shares of XOR evaluation $x \oplus y$ can be easily evaluated as $x_i \oplus y_i$ by each party's own, and we elaborated two methods for computing Boolean shares of AND evaluation.

In the first method [14,37], two parties execute $(4,1)\text{-}\mathsf{OT}_1$ with the sender's $k = (k_1, k_0) \in \{0,1\}^2$-th message $m_k = (x_i \oplus k_1) \wedge (y_i \oplus k_0) \oplus r$ for some random bit $r \in \{0,1\}$, and the receiver's choice index $c = (x_{1-i}, y_{1-i}) \in \{0,1\}^2$. As a result the receiver outputs $m_c = (x \wedge y) \oplus r$ and the sender outputs r, which are Boolean shares of $x \wedge y$. This can be generalized to perform ℓ parallel evaluations of AND gates with $(4^\ell,1)\text{-}\mathsf{OT}_\ell$. It can be easily calculated that $\ell = 2$ is the optimal choice, which requires $\lambda + 16$ bit communication per one AND gate evaluation.

The second method [13,19] is done by $(2,1)\text{-}\mathsf{COT}_1^2$. For the underlying idea, observe that $(2,1)\text{-}\mathsf{COT}_1$ with the sender's input correlation bit d and the receiver's input choice bit b essentially computes Boolean shares of $b \wedge d$. To evaluate AND gate, two parties execute a correlated-OT with input x_i and y_{1-i} to have Boolean shares of $a = x_i \wedge y_{1-i}$, and then with input y_i and x_{1-i} to have Boolean shares of $b = x_{1-i} \wedge y_{1-i}$. Then the party P_i outputs $x_i \wedge y_i \oplus a_i \oplus b_i$ and the other party P_{1-i} outputs $x_{1-i} \wedge y_{1-i} \oplus a_{1-i} \oplus b_{1-i}$, which are Boolean shares of $x \wedge y = (x_0 \oplus x_1) \wedge (y_0 \oplus y_1)$.

2.3 RLWE-based Homomorphic Encryption

A homomorphic encryption (HE) scheme is an encryption scheme that supports a ring-structured plaintext $\mathcal{M}$, and homomorphic arithmetic operations between ciphertexts that acts on inner plaintext. We especially exploit a ring learning with errors (RLWE) based HE scheme, BFV scheme [15].

For simplicity, we restrict our description here for RLWE-based HE using power-of-2 cyclotomic rings of integers which is sufficient for our paper. Let $\mathcal{R} := \mathbb{Z}[X]/(X^n + 1)$ be a polynomial quotient ring where n is a power-of-2 integer. This scheme supports a plaintext space $\mathcal{R}_p := \mathcal{R}/p\mathcal{R} = \mathbb{Z}_p[X]/(X^n + 1)$ for some plaintext modulus p, and the corresponding ciphertext space is $\mathcal{R}_q^2$ for some $q \gg p$.

BFV Scheme. We will briefly review the BFV homomorphic encryption scheme. The IND-CPA security of BFV is based on the hardness assumption of the RLWE problem. For more details, we refer to [4, 15].

Key Generation. Given a security parameter $\lambda > 0$, fix integers n, P (P be a positive integer that will be used in the evaluation key generation), and distributions $\mathcal{D}_{key}$, $\mathcal{D}_{err}$ and $\mathcal{D}_{enc}$ over $\mathcal{R}$ in a way that the resulting scheme is secure against any adversary with computational resource of $O(2^\lambda)$.

1. Sample $a \leftarrow \mathcal{R}_q$, $s \leftarrow \mathcal{D}_{key}$, and $e \leftarrow \mathcal{D}_{err}$. Then the secret key is defined as $\mathsf{sk} = (1, s) \in \mathcal{R}^2$, and the corresponding public key is defined as $\mathsf{pk} = (b, a) \in \mathcal{R}_q^2$, where $b = [-a \cdot s + e]_q$.
2. Sample $a' \leftarrow \mathcal{R}_q$ and $e' \leftarrow \mathcal{D}_{err}$. Then the evaluation key is defined as $\mathsf{evk} = (b', a') \in \mathcal{R}_q^2$, where $b' = [-a' \cdot s + e' + Ps']_q$ for $s' = [s^2]_q$.

Encryption. Given a public key pk and a plaintext $m \in \mathcal{R}$, Sample $r \leftarrow \mathcal{D}_{\mathsf{Enc}}$ and $e_0, e_1 \leftarrow \mathcal{D}_{err}$. Then compute $\mathsf{Enc}(\mathsf{pk}, 0) = [r \cdot \mathsf{pk} + (e_0, e_1)]_q$ and $\mathsf{Enc}^{\mathsf{BFV}}(\mathsf{pk}, m) = [\mathsf{Enc}(\mathsf{pk}, 0) + (\Delta_{\mathsf{BFV}} \cdot [m]_p, 0)]_q$, where $\Delta_{\mathsf{BFV}} = \lfloor q/p \rceil$.

Decryption. Given a secret key $\mathsf{sk} \in \mathcal{R}^2$ and a ciphertext $\mathsf{ct} \in \mathcal{R}_q^2$, $\mathsf{Dec}^{\mathsf{BFV}}(\mathsf{sk}, \mathsf{ct}) = \left\lfloor \frac{p}{q} [\langle \mathsf{sk}, \mathsf{ct} \rangle]_q \right\rceil$.

The ciphertext of BFV scheme is $(b(x), a(x))$ satisfying $b(x) = -a(x) \cdot s(x) + e(x)$. The $e(x)$ part is called as *noise term* of ciphertext. We note that infinite norm of noise term of ct in decryption function should be bounded by $\frac{q}{2p}$ for correctness.

Addition. Given ciphertexts ct_1 and ct_2 in $\mathcal{R}_q^2$, their sum is defined as $\mathsf{ct}_{\mathsf{Add}} = [\mathsf{ct}_1 + \mathsf{ct}_2]_q$.

Multiplication. Given ciphertexts $\mathsf{ct}_1 = (b_1, a_1)$ and $\mathsf{ct}_2 = (b_2, a_2)$ in $\mathcal{R}_q^2$ and an evaluation key evk, their product is defined as $\mathsf{ct}_{\mathsf{Mult}} = \left[(d_0, d_1) + \left\lfloor P^{-1} \cdot d_2 \cdot \mathsf{evk} \right\rceil\right]_q$, where (d_0, d_1, d_2) is defined by $\left[\left\lfloor \frac{p}{q}(b_1 b_2, a_1 b_2 + a_2 b_1, a_1 a_2) \right\rceil\right]_q$.

Batching. As BFV scheme support a plaintext space $\mathcal{R}_p$, multiple data in $\mathbb{Z}_p$ can be encrypted in one ciphertext. This method is called *batching*, and this method can be applied with following condition:

$$p = 1 \mod 2n. \tag{1}$$

In this case, operations between plaintext in $\mathbb{Z}_p^n$ goes to Single instruction multiple data (SIMD) operation which means component-wise addition and multiplication.

Security Notions. For security, we consider the standard IND-CPA security that requires two ciphertexts of different messages are (computationally) indistinguishable given an encryption oracle. The IND-CPA security of RLWE-based HE literally comes from the hardness of ring learning with errors (RLWE) problem. For concrete parameter setting of IND-CPA security, the bit-size of ciphertext modulus $\log q$ and polynomial ring dimension n, and error distribution $\mathcal{D}_{err}$ should be selected to secure against various lattice reduction attacks.

3 Circuit-based PSI

The definition circuit-based PSI (circuit-PSI) functionality to generate Boolean additive shares is given as Figure 1. After circuit-PSI, the results can be used for one's desired function evaluation. In the rest of this section, we describe the abstract framework of [35] which continues to the following improvements [8,39]. Then we especially review the equality share generation methods of each works which occupies the largest part of the total cost, from which we can observe the input bit-length ℓ equality share generation plays the most crucial role for complexity.

Parameters: A receiver with an input set X of size N and a sender with an input set Y of size N.

Functionality: The functionality sends to the receiver an injective indexing function $\iota : X \to [M]$ for some $M \geq N$ and a vector $\mathbf{s}_0 \in \{0,1\}^M$, and to the sender a vector $\mathbf{s}_1 \in \{0,1\}^M$ such that $s_{0,i} \oplus s_{1,i} = \mathbf{1}(\iota^{-1}(i) \in X \cap Y)$ for $i \in \iota(X)$, and $s_{0,i} \oplus s_{1,i} = 0$ for $i \notin \iota(X)$.

Fig. 1. $\mathcal{F}_{\mathsf{CPSI}}$. (Ideal) Functionality of circuit-PSI

3.1 The OPPRF-based Circuit-PSI Framework

Let the receiver $\mathcal{R}$ holds a set X and the sender $\mathcal{S}$ holds a set Y of the same size N. The framework consists of the following three main stages.

Step 1. Hashing. For $\varepsilon > 0$, each party creates a hash table with $M = (1+\varepsilon)\cdot N$ bins, but with different hashing method. The receiver applies cuckoo hashing with d hash functions $h_1, \cdots, h_d : \{0,1\}^* \to [M]$ on input X. More precisely, for a suitable choice of ε, there is a cuckoo hashing algorithm that stores every element $x \in X$ in $h_j(x)$-th bin for some $j \in [d]$ with overwhelming probability, while ensuring that at most one element is stored in each bin. This yields a simple representation of the cuckoo hash table: $T_X[h_j(x)] = x$. Note that the mapping from $x \in X$ to $h_j(x)$ determines the indexing function ι in the circuit-PSI definition of Figure 1.

On the other hand, the sender creates a simple hash table with the same hash functions on input Y, which stores each $y \in Y$ in every bin $h_j(x)$ for every $j \in [d]$. Naturally each bin can hold more than one element, and hence the i-th bin of the simple hash table $T_Y[i]$ is indeed a set. It is known that that for $M = O(N)$ hash table size, the number of elements in each bin is $O(\log(N))$.

Since $h_j(x) \neq h_j(y)$ for some j implies $x \neq y$, two parties only need to compare each elements of the same bin of each hash tables. Since the cuckoo hash table T_X ensures at most one element of $x \in X$ per each bin, circuit-PSI reduces to the problem that securely outputs an additive share of $\mathbf{1}(T_X[i] \in T_Y[i])$ for each bin i, which is essentially a private set membership (PSM) problem. Here the receiver has to fill the empty bin in T_X with dummy value to prevent additional information leakage.

Step 2. Bin Tagging. This step further reduces the aforementioned PSM problem into an equality share generation (ESG) problem between two parties, where each party inputs a vector $\mathbf{v}$ and $\mathbf{v}^*$ of length M respectively, and is given as output a Boolean vector of additive share of $\mathbf{1}(v_i = v_i^*)$.

This is realized by a functionality called *oblivious programmable pseudo-random function* (OPPRF) [25] where the sender obliviously computes a PRF F on receiver's input while the sender can *program* F with values (y_i, z_i) so that $F(y_i) = z_i$. The formal definition of OPPRF is given as Figure 2. [35] is the first work that applies OPPRF functionality for this purpose, and then [8] and [39] developed more efficient OPPRF protocols to improve the performance of circuit-PSI.

Parameters: A sender with input $L = \{(y_i, z_i)\}$ where $y_i \in \{0,1\}^*$ and $z_i \in \{0,1\}^\ell$, and a receiver with input $X = \{x_i\}$ with $x_i \in \{0,1\}^*$.

Functionality: The functionality samples a random function $F : \{0,1\}^* \to \{0,1\}^\ell$ such that $F(y) = z$ for each $(y, z) \in L$, and sends $F(X) := \{F(x) : x \in X\}$ to the receiver.
After then, upon an input y of the sender, the functionality outputs $F(y)$ to the sender.

Fig. 2. $\mathcal{F}_{\mathsf{OPPRF}}$. (Ideal) Functionality of oblivious programmable PRF

To convert PSM problem to ESG problem, two parties execute a protocol for OPPRF functionality with the following input. The sender who has a simple table samples a random tag value $v_i \in \{0,1\}^\ell$ for each i-th bin, and generate the input set L obtained by concatenating each $y \in Y$ with the tag of the bins where y is stored, namely

$$L = \left\{ \left(y || h_j(y), v_{h_j(y)}\right) \right\}_{y \in Y, j \in [d]} = \left\{ (y' || i, v_i) \right\}_{i \in [M], y' \in T_Y[i]} .$$

The receiver feeds its input set by $\tilde{T}_X = \{T_X[i] || i\}_{i \in [M]}$. After the execution of OPPRF protocol, the receiver assigns

$$v_i^* = F\left(T_X[i] || i\right) \in \{0,1\}^\ell$$

in each hash address i to construct a vector $\mathbf{v}^*$ of length M. From the definition of OPPRF functionality, it holds that $v_i = v_i^*$ if the element $T_X[i]$ is in the set $T_Y[i]$, otherwise v_i^* is a random element. Therefore the original PSM-related problem is translated into equality share generation problem between $\mathbf{v}$ from the sender and $\mathbf{v}^*$ from the receiver.

Failure Probability of OPPRF. Note that there is a failure probability of $2^{-\ell}$ where the random element v_i^* is same to v_i despite $T_X[i]$ is not in $T_Y[i]$. The length of tag ℓ should be chosen so that the overall failure probability is smaller than $2^{-\sigma}$ where σ is statistical security parameter. Since there are M bins, it should hold that $2^{-\sigma} > 1 - (1 - 2^\ell)^M$, which is sufficient with $\ell > \sigma + \lceil \log M \rceil$. One exception is OPPRF of [8] that requires $\ell > \sigma + \lceil \log 4M \rceil$, and this comes from different structure of their OPPRF.

Step 3. Equality Share Generation. In this step two parties finally generate Boolean shares of $\mathbf{1}(v_i = v_i^*)$, whose definition is formally given as Figure 3.

Parameters: A sender with an input string a and a receiver with an input string b.

Functionality: The functionality outputs bits s_0 and s_1 such that $s_0 \oplus s_1 = \mathbf{1}(a = b)$ to each party respectively.

Fig. 3. $\mathcal{F}_{\mathsf{ESG}}$. (Ideal) Functionality of equality share generation

There are several protocols performing this step in semi-honest model. As our main interest is to improve this step, we review them in the next subsection. We remark that this step occupies the most of computational or communication cost among three steps, such as 96% of the total protocol cost in Table 3 of [35], 91% in Table 2 of [8]. The breakdown for each step is not given in [39], but as it used the same method to generate equality share generation with [35], we strongly predict that Step 3 occupies the largest part of the total cost among three steps also for [39].

3.2 Protocols for Equality Share Generation

Our paper aims to reduce the huge burden of equality share generation (ESG). For that, we briefly review known methods for generate equality shares below, and Table 2 shows the summary. We would like to remark that the bit-length ℓ plays crucial role for complexity.

GMW Protocol. One can directly use evaluate the equality check circuit on ℓ-bit string composed of $\ell - 1$ AND gate evaluations. Circuit-PSI protocols of [35] and [39] exploited this method, while performing AND gate evaluations by $(2, 1)\text{-}\mathsf{COT}_1^{2\ell-2}$ using IKNP OTe or Silent OTe. IKNP OTe comes at the fastest execution time but huge communication cost, and Silent OTe shows opposite performance; slow but light.

CGS Protocol. Proposed in [8], this method uses oblivious transfer in more direct way than GMW protocol. Roughly speaking, it divides given ℓ-bit input strings into d-length substrings, and perform $(2^d, 1)\text{-}\mathsf{OT}_1$ to generate equality share of each substring. Then the final equality result is obtained by evaluating $\ell/d - 1$ AND gates of all equality share. The circuit-PSI protocol of [8] takes $d = 4$ and exploites KK OTe with $n = 16$, where equality shares of substring is generated by $(16, 1)\text{-}\mathsf{OT}_1$ and AND gate evaluation is done with $(16, 1)\text{-}\mathsf{OT}_2$. A detailed description of the protocol including $\ell \neq 0 \mod d$ case can be found in Appendix A.1.

OSN Protocol. Recently, [16] proposed another method for ESG which has a downside that it always reveals the cardinality of the intersection set $|X \cap Y|$. It is based on oblivious switching network (OSN) protocol [30] that requires $(2, 1)\text{-}\mathsf{OT}_{2\ell}^{M \log M}$. The authors of [16] exploited IKNP OTe to execute such OT calls. We note that although OSN protocol seems to require less numbers of $(2, 1)\text{-}\mathsf{OT}$ calls than GMW protocol ($2M\ell$ vs. $M \log M$) the running time of OSN protocol is quite slower than GMW protocol due to its complicated structure [16].

	Required OT	Approx Comm.	Running Time
IKNP-GMW	$(2, 1)\text{-}\mathsf{COT}_1^{2\ell}$	$2\lambda\ell$	Fast
KK-CGS	$(16, 1)\text{-}\mathsf{OT}_{\leq 2}^{3\ell/8}$	$0.75(\lambda + 12)\ell$	Medium
IKNP-OSN	$(2, 1)\text{-}\mathsf{OT}_{2\ell}^{\log M}$	$\log M(\lambda + 4\ell)$	Medium
Silent-GMW	$(2, 1)\text{-}\mathsf{COT}_1^{2\ell}$	4ℓ	Slow

Table 2. Known methods for ESG and (amortized) costs for one ℓ-bit input.

Remark 1. One may wonder whether other combination of OT extension and ESG protocol (e.g. KK-GMW or Silent-OSN) is possible or even better. Regarding this we show that other combinations are possible, but Table 2 is the best four combinations *regardless of input length* ℓ. See Appendix A.3 for more details.

3.3 Applications of Circuit-PSI

Below we present some typical but popular applications of circuit-PSI. We would like to remark that the overheads for these applications are significantly small compared to circuit-PSI cost, as also remarked in [35].

Private Intersection Cardinality and Threshold. These applications would be most direct consequences of circuit-PSI. The cardinality of intersection set (*PSI-Ca*) can be obtained by evaluating a Hamming distance circuit that requires less than M AND gates on circuit-PSI outputs. Moreover, by augmenting one comparison circuit to the Hamming distance circuit (less than $M + \log M$ AND gates), we can let the parties know whether the cardinality is larger than some threshold t (*PSI-Th*).

Private Sum over Intersection. Assume the sender having set X additionally holds an associated values $\{v_x \in \mathbb{G} : x \in X\}$ for some additive group $\mathbb{G}$, and we want to let the receiver having set Y knows the sum of associated values over the intersection set, namely $V = \sum_{x \in X \cap Y} v_x$. This is sometimes called *PSI-Sum*[1]. For that we adapt a method of [16]: The sender samples $\mathbf{r} \in \mathbb{G}^M$ that sums to $\sum r_i = 0$. Then two parties execute OT upon the choice bit $s_{1,i}$ from the receiver, and two messages r_i for $s_{0,i}$ choice and $r_i + v_{\iota^{-1}(i)}$ for $1 - s_{0,i}$ choice from the sender, where $v_{\iota^{-1}(i)} = 0$ for $i \notin \iota(X)$. The receiver adds all received value to have $\sum r_i + V = V$, without knowing any other information since each summand is masked by random value r_i. This can be easily tweaked to let the sender know $V = \sum_{x \in X \cap Y} v_x$, by letting the sender samples $\mathbf{r}$ such that $\sum r_i = R$ for a sender-side chosen $R \in \mathbb{G}$. Then from the same protocol the receiver ends with $R + V$, and finally sends back the value to the sender so that the sender recover $V = (R + V) - R$.

Remark 2. Circuit-PSI can handle the case where both parties hold associated value sets so that parties perform further computations over those sets. However it is somewhat complicated as it requires some modification of OPPRF application (of Step 2. Bin Tagging). Thus we simply refer Section 6 of [35] for details.

[1] Some protocols [16, 21] outputs both cardinality and summation. It should be remarked that circuit-PSI based protocol can selectively exposes cardinality or summation, or even both.

Private Set Union. Circuit-PSI also leads to private set union (PSU), where the receiver obtains $X \cup Y$. Note that by assuming the receiver holds Y (and the sender holds X), PSU is equivalent to let the receiver know $X \backslash Y$. We can also adapt a method of [16] for PSU as follows: Two parties first run circuit-PSI on each input X and Y so that the result equality share is related to X, i.e., two parties obtain $s_{0,x}$ and $s_{1,x}$ for each $x \in X$. Then the set $X \backslash Y$ can be obtained from OT between two parties, with a choice bit $s_{1,x}$ from the receiver, and two messages x for $s_{0,x}$ choice and $\perp$ for $1 - s_{0,x}$ choice from the sender. Note that the receiver obtains x if $x \in X \backslash Y$, and $\perp$ otherwise, and the sender knows no information about which element of X is sent to the receiver.

4 Equality Preserving Compression

The final equality share generation procedure occupies the largest part of the total cost, and the input bit-length ℓ of equality share generation plays an important role. In this section, we present a procedure that converts the equality share generation target inputs into another values whose size is asymptotically logarithm of the original input bit-length, while the equality results remain unchanged. More formally, we define the 2-party functionality *equality preserving compression* (EPC) $\mathcal{F}_{\mathsf{EPC}}$ that takes an integer $v \in \mathbb{Z}_t$ from the sender and another integer $v^* \in \mathbb{Z}_t$ from the receiver. The functionality outputs each party a random integer r and r^* in another modulus ring $\mathbb{Z}_p$, where it holds that $v = v^*$ in $\mathbb{Z}_t$ *if and only if* $r = r^*$ in $\mathbb{Z}_p$ for $p < t$.

Parameters: A sender with an input $v \in \mathbb{Z}_t$ and a receiver with an input $v^* \in \mathbb{Z}_t$, and the target size p.

Functionality: The functionality sends a random $r \in \mathbb{Z}_p$ and $r^* \in \mathbb{Z}_p$ to the sender and receiver respectively, such that $v = v^*$ in $\mathbb{Z}_t$ if and only if $r = r^*$ in $\mathbb{Z}_p$.

Fig. 4. $\mathcal{F}_{\mathsf{EPC}}$. (Ideal) Functionality of equality preserving compression

4.1 A Basic Protocol

Our protocol starts from the following simple observation on word decomposition. For any base w, the w-base decomposition of v and v^* by $v = \sum_{i=0}^{u-1} v_i \cdot w^i$ and $v^* = \sum_{i=0}^{u-1} v_i^* \cdot w^i$ where $u := \lceil \log_w t \rceil$ and $v_i, v_i^* \in [0, w)$ satisfies

$$v = v^* \iff D := \sum_{i=0}^{u-1} (v_i - v_i^*)^2 = 0 \text{ in } \mathbb{Z}. \qquad (2)$$

Note that $D \leq u \cdot (w-1)^2 \approx \log_w t \cdot (w-1)^2$, which has much smaller size than the original size t.

Based on this idea, we consider a simple protocol that privately computes D and output a random element $r \in \mathbb{Z}_p$ and $r^* := r + D \in \mathbb{Z}_p$ by Figure 5. However the correctness may fails without any condition on the word base w, since it may happens that $r = r^* \in \mathbb{Z}_p$ despite of $v \neq v^*$ if D is divisible by p. To avoid this, the word base w has to be chosen so that D is always less than p, namely

$$p > u \cdot (w-1)^2. \tag{3}$$

We note that $u \cdot (w-1)^2 = O(\log t)$, this protocol asymptotically realizes $\mathcal{F}_{\mathsf{EPC}}$ for $p = O(\log t)$.

Parameters: A sender with input $v \in \mathbb{Z}_t$ and a receiver with input $v^* \in \mathbb{Z}_t$ and the target size p.

Protocol:

1. Sender generates a HE secret key sk, and decomposes in w-base $v \in \mathbb{Z}_t$ to $\{v_i\}_{0 \le i < u}$ for $u = \lceil \log_w t \rceil$. After that sender encrypts each v_i using sk, and sends them to receiver.
2. Receiver picks a random integer $r \in \mathbb{Z}_p$, and decomposes $v^* \in \mathbb{Z}_t$ to $\{v_i^*\}_{0 \le i < u}$. Then receiver homomorphically compute $r + \sum_{i=0}^{u-1}(v_i - v_i^*)^2$, and sends the resulting ciphertext back to sender.
3. Sender decrypts the received ciphertext using sk, to obtain $r^* = r + \sum_{i=0}^{u-1}(v_i - v_i^*)^2 \in \mathbb{Z}_p$.

Fig. 5. A basic protocol for $\mathcal{F}_{\mathsf{EPC}}$ functionalities

4.2 Optimizations and Full Protocol

We provide several optimizations of the basic protocol of Figure 5. A full protocol description that puts everything together is presented by Figure 6.

Batching by RLWE-HE. As RLWE-based HE supports batching several elements into one ciphertext and SIMD operation, we can batching n times of $\mathcal{F}_{\mathsf{EPC}}$ calls. As a downside, this requires a restriction on the choice of target size p by $p = 1 \mod 2n$ (1), which necessarily requires $p > 2n$.

Removing Ciphertext Multiplications. In RLWE-based HE, homomorphic multiplication takes much larger time than scalar multiplication. To remove homomorphic multiplications, we let the sender additionally sends one more ciphertext which is an encryption of $\sum_{i=0}^{u-1} v_i^2$. In this case, the receiver can computes D by

$$D = \sum_{i=0}^{u-1} v_i^2 - 2 \cdot \sum_{i=0}^{u-1} v_i \cdot v_i^* + \sum_{i=0}^{u-1} v_i^{*2}.$$

As the receiver knows v_i^* values, it can compute $\sum_{i=0}^{u-1} v_i^{*2}$ part and then the receiver only needs to perform scalar multiplications to obtain an encryption of D.

Remark 3. This opens possibility to apply *additive homomorphic encryption* (AHE) schemes such as Paillier scheme [31], but RLWE-based AHE is still better for our PSI usage. See Appendix B for more detailed argument.

Realizing Function Privacy. For the security proof, we need to ensure the function privacy from the return ciphertext from receiver to sender. For that we apply randomization and noise flooding method, whose detail will be presented in the next subsection. Concretely this can be realized by letting receiver randomize the resulting ciphertext by homomorphically adding a fresh encryption of zero, and add large enough error to apply noise flooding method before send the computation result back to sender.

4.3 Security and Cost Analysis

In this section, we will discuss about security of our protocol with correctness proof. We also analyze the computational and communication costs. Before that, we need to recall some details of RLWE-based HE scheme. We will focus on BFV scheme [15], but it does not mean that our method is restricted to this scheme.

Randomizing BFV Ciphertexts. Recall that a BFV encryption of a message $m(x)$ is of the form

$$\left(-a(x) \cdot s(x) + \frac{q}{p} \cdot m(x) + e(x), a(x)\right) \in \mathcal{R}_q^2.$$

As secret key owner can recover not only $m(x)$ but also $e(x)$. For this reason, we need to add additional noise $e^*(x)$ such that $|e_i^*| > 2^\sigma \cdot B$ for the function privacy of homomorphic encryption scheme. Here B is upper bound of $e(x)$'s coefficients and σ is the statistical security parameter. This method is called *noise flooding* and this idea is firstly proposed by [18].

Noise Analysis. For the concrete choice of homomorphic encryption parameter, we need to analyze the noise term in our HE-based EPC protocol. Here we will consider the infinity norm $||f(x)||$ which is defined as $\max_i |f_i|$ and the expansion factor of ring $\mathcal{R}$ is defined as $\delta_R = \max\{||f(x) \cdot g(x)||/(||f(x)|| \cdot ||g(x)||) : f(x), g(x) \in \mathcal{R}\}$. In addition, we assume that the noise term of $\mathsf{ctxt}_{k,i}$ in Figure 6 is bounded by B_{fresh}.

Lemma 1 (Noise growth during homomorphic constant multiplication). *For the given BFV ciphertext $(b(x), a(x))$ with noise term $e(x)$ such that $||e(x)|| < B$, the result ciphertext of homomorphic constant multiplication has noise term $e^*(x)$ such that $||e^*(x)|| < \delta_R \cdot p \cdot B + \delta_R \cdot p^2$.*

Parameters: A sender with input $\mathbf{v} \in \mathbb{Z}_t^M$ and a receiver with input $\mathbf{v}^* \in \mathbb{Z}_t^M$ and the target size p.

Protocol:

1. [**Setup**] Two parties agree on a proper HE parameter (n, q) that supports plaintext space $\mathbb{Z}_p^n$, and satisfies IND-CPA security. Then the sender samples a key pair $(\mathsf{sk}, \mathsf{pk})$, and sends the public key pk to the receiver. The sender pads $\mathbf{v}$ by 0 and the receiver pads $\mathbf{v}^*$ by 1, until they have length divisible by n, say $\gamma \cdot n$. Two parties also agree on word base w satisfying $p > \lceil \log_w t \rceil \cdot (w-1)^2$, and define $u = \lceil \log_w t \rceil$.
2. [**Encryption**] Sender performs the following for $0 \le k < \gamma$:
   (a) Decompose each v_{nk+j} into $\sum_{i=0}^{u-1} v_{j,i} \cdot w^i$ for $1 \le j \le n$.
   (b) Batch them into $\mathbf{m}_{k,i} = (v_{j,i})_{1 \le j < n} \in \mathbb{Z}_p^n$ for $0 \le i < u$.
   (c) Define $\mathbf{m}_{k,u} = (\sum_{i=0}^{u-1} v_{j,i}^2)_{1 \le j < n} \in \mathbb{Z}_p^n$
   (d) Encrypt $\{\mathbf{m}_{k,i}\}$ into $\{\mathsf{ctxt}_{k,i}\}$ using sk and send those ciphertexts to the receiver.
3. [**Compute D and Masking**] Receiver performs the following for $0 \le k < \gamma$:
   (a) Decompose each v^*_{nk+j} into $\sum_{i=0}^{u-1} v^*_{j,i} \cdot w^i$ for $1 \le j \le n$.
   (b) Batch them into $\mathbf{m}^*_{k,i} = (v^*_{j,i})_{1 \le j < n} \in \mathbb{Z}_p^n$ for $0 \le i < u$.
   (c) Define $\mathbf{m}^*_{k,u} = (\sum_{i=0}^{u-1} v^{*2}_{j,i})_{1 \le j < n} \in \mathbb{Z}_p^n$
   (d) Compute a ciphertext $\mathsf{ctxt}_{k,d} = \mathsf{ctxt}_{k,u} \oplus \sum_{i=0}^{u-1} \left(\mathsf{ctxt}_{k,i} \odot 2\mathbf{m}^*_{k,i}\right) \oplus \mathbf{m}^*_{k,u}$
   (e) Sample a random vector $\mathbf{r}^*_k \in \mathbb{Z}_p^n$.
   (f) Generate an encryption $\mathsf{ctxt}_{fp,k}$ (using pk) of zero of error size B_{fp} which is large enough for function privacy.
   (g) Send back $\mathsf{ctxt}_k := \mathsf{ctxt}_{k,d} \oplus \mathsf{ctxt}_{fp,k} \oplus \mathbf{r}^*_k$ to the sender.
4. [**Decryption**] Sender decrypts ctxt_k to have $\mathbf{r}_k \in \mathbb{Z}_p^n$ for $0 \le k < \gamma$.
5. [**Finalize**] Sender outputs $\mathbf{r} \in \mathbb{Z}_p^M$ by concatenating every $\mathbf{r}_k$ and cutting the last $\gamma \cdot n - M$ dummy elements. Receiver outputs $\mathbf{r}^* \in \mathbb{Z}_p^M$ by performing the same with $\mathbf{r}^*_k$.

Fig. 6. A full protocol Π_{BEPC} for M batch calls of $\mathcal{F}_{\mathsf{EPC}}$ functionalities

Proof. In case of homomorphic constant multiplication, it can be done by multiplying a constant polynomial $c(x)$ to each $a(x)$ and $b(x)$. And each coefficient of $c(x)$ is bounded by the plaintext modulus p. For the $a^*(x) = c(x) \cdot a(x)$ and $b^*(x) = c(x) \cdot b(x)$,

$$\begin{aligned}b^*(x) + a^*(x)\cdot s(x) &= \left\lfloor \frac{q}{t} \right\rceil \cdot (m(x)\cdot c(x)) + e(x)\cdot c(x)\\ &= \left\lfloor \frac{q}{p} \right\rceil \cdot ([m(x)\cdot c(x)]_p + p\cdot I(x)) + e(x)\cdot c(x)\\ &= \left\lfloor \frac{q}{p} \right\rceil \cdot [m(x)\cdot c(x)]_p + \left(\frac{q}{p} + \epsilon\right)\cdot p \cdot I(x) + e(x)\cdot c(x)\\ &= \left\lfloor \frac{q}{p} \right\rceil \cdot [m(x)\cdot c(x)]_p + \epsilon\cdot p\cdot I(x) + e(x)\cdot c(x) \mod q\end{aligned}$$

Therefore, $||e^*(x)|| = ||\epsilon \cdot p \cdot I(x) + e(x)\cdot c(x)|| \le \delta_R \cdot p^2 + \delta_R \cdot p \cdot B$. □

Furthermore, homomorphic addition between two ciphertext with noise bound B_1 and B_2 returns ciphertext with noise bound $B_1 + B_2 + 2p$. Finally, homomorphic addition between ciphertext with noise bound B and plaintext returns ciphertext with noise bound $B + 2p$.

From now on, we can analyze the noise term in our HE-based EPC protocol. This analysis gives a method for concrete HE parameter choices. If we see Figure 6, the receiver have to compute following (at 3-(d)):

$$\mathsf{ctxt}_{k,d} = \mathsf{ctxt}_{k,u} \oplus \sum_{i=0}^{u-1} \left(\mathsf{ctxt}_{k,i} \odot 2\mathbf{m}^*_{k,i}\right) \oplus \mathbf{m}^*_{k,u}.$$

By Lemma 1, the noise term of output ciphertext $\mathsf{ctxt}_{k,d}$ will be bounded by $B^* = 2u \cdot (\delta_R \cdot p \cdot B_{\mathsf{fresh}} + \delta_R \cdot p^2) + B_{\mathsf{fresh}} + 4p$. After that we need to add encryption of zero of error size $B_{fp} = 2^\sigma B^*$ for statistic security parameter σ for the function privacy. At last, receiver needs to add random vector $\mathbf{r}^*_k$ to the ciphertext. So, for the correct BFV decryption at the decryption phase, the ciphertext modulus q should satisfies the following inequality:

$$\frac{q}{p} > (2^\sigma + 1)\cdot \left(2u \cdot (\delta_R \cdot p \cdot B_{\mathsf{fresh}} + \delta_R \cdot p^2) + B_{\mathsf{fresh}} + 4p\right) + 2p.$$

Recall that we have $u = O(\log t)$ for the target size $p = O(\log t)$, and therefore we asymptotically have $q = O(\log^4 t)$ where t is input size.

Theorem 1. *The protocol Π_{BEPC} of Figure 6 realizes M times of $\mathcal{F}_{\mathsf{EPC}}$ functionality calls in a semi-honest model if*

$$q > p \cdot (B_{fp} + B^* + 2p)$$

where $B^* = 2u \cdot (\delta_R \cdot p \cdot B_{\mathsf{fresh}} + \delta_R \cdot p^2) + B_{\mathsf{fresh}} + 4p$ and $B_{fp} = 2^\sigma B^*$ for a statistical security parameter σ.

Proof. It is already explained that the condition for q provides the correctness and the function privacy required for our protocol.

For the sake of simplicity, we forget batching for a while and assume each parties has integer v and v^* in $\mathbb{Z}_t$. During the protocol execution, the receiver has input v^* and a random output r^*, and its view consists of the public key pk and the ciphertexts of v_i (decomposed value) and $\sum v_i^2$. This can be simulated by replacing all ciphertexts to encryptions of zero, which is indistinguishable from the real execution thanks to the IND-CPA security of HE.

The sender has input v and its view is a ciphertext of $D + r$ and it outputs the plaintext $D + r \in \mathbb{Z}_p$ by decrypting the ciphertext. This can be simulated by encrypting the output $r' \in \mathbb{Z}_p$ of ideal functionality, since from the function privacy the sender cannot know any other information than the decryption result, and the distribution of $D + r$ is identical to the distribution of r' (uniform over $\mathbb{Z}_p$). $\square$

Asymptotic cost analysis. As the ciphertext modulus q is determined as above, we can estimate the total costs. Let $\gamma = \lceil M/n \rceil$ and $u = \log_w t$ by following notations of Π_{BEPC}. For computational cost, our protocol requires $\gamma(u + 2)$ encryptions, γu homomorphic constant multiplications, $2\gamma(u + 3)$ homomorphic additions, and γ decryptions for M numbers of EPC calls. Such HE operations including homomorphic constant multiplication can be done by $O(1)$ numbers of $\mathcal{R}_q$ operations that is roughly translates into $O(n \log n \log q)$ bit operations [4]. By approximating $\gamma n \approx M$, we conclude that amortized computational cost per EPC call is $O(\log t \cdot \log n \cdot \log q)$ bit operations as $u = O(\log t)$. In case of secure RLWE parameters, $n \propto \log q$ roughly holds for the fixed computational security parameter λ. Since $q = O(\log^4 t)$, we conclude that the computational cost per one EPC is $\tilde{O}(\log t)$. Toward communication cost, the sender sends $\gamma(u + 1)$ fresh ciphertexts to the receiver and the receiver returns γ ciphertexts after HE oepration to the sender. The size of fresh ciphertexts is $\gamma(u + 1)(n \log q + \lambda)$ and the size of returned ciphertexts is $2\gamma n \log q$. Then the total communication cost is $\gamma n(u + 3) \log q + \gamma(u + 1)\lambda$ bits. We again approximate $\gamma n \approx M$ and divide the total cost by M to see amortized cost for one EPC call. Then it results in approximately $(u + 3) \log q \approx (\log_w t + 3) \log q$ bits communication and asymptotically $\tilde{O}(\log t)$ for one EPC call.

5 Application to Circuit-PSI Framework

Our equality preserving compression (EPC) of the previous section can be seamlessly augmented to the OPPRF-based circuit-PSI framework described in Section 3 as Figure 7.

Since EPC perfectly preserve equality (without failure probability), all previous works' analysis for correctness (or failure probability) are still valid. Moreover, as Theorem 1 shows that EPC is secure against semi-honest adversary, the semi-honest security Π_{CPSI} is also guaranteed.

Theorem 2. *The protocol Π_{CPSI} of Figure 7 realizes the $\mathcal{F}_{\mathsf{CPSI}}$ functionality in a semi-honest model in the hybrid model of $\mathcal{F}_{\mathsf{OPPRF}}, \mathcal{F}_{\mathsf{EPC}}$ and $\mathcal{F}_{\mathsf{ESG}}$.*

Parameters: A receiver with an input set X of size N and a sender with an input set Y of size N, and compression target bit-length ℓ_{c}.

Protocol:

1. [**Hashing**] Both parties agree on hash functions $h_1, \cdots, h_d$, and table size parameter ε. The receiver construct a cuckoo hash table T_X from X, and the sender constructs a simple hash table T_Y from Y using hash functions $h_1, \cdots, h_d$ into $M = (1+\varepsilon) \cdot N$ bins. The receiver define the address mapping X to T_X by ι.
2. [**Bin Tagging**] The sender samples uniformly random tags $\mathbf{v} \in \mathbb{Z}_{2^\ell}^M$ and sends $L = \{(y' \| i, v_i)\}_{i \in [M], y' \in T_Y[i]}$ to $\mathcal{F}_{\mathsf{OPPRF}}$. The receiver sends $\tilde{T}_X = \{T_X[i] \| i\}_{i \in [M]}$ to $\mathcal{F}_{\mathsf{OPPRF}}$, and receives $\mathbf{v}^* \in \mathbb{Z}_{2^\ell}^M$ from $\mathcal{F}_{\mathsf{OPPRF}}$.
3. [**Equality Preserving Compression**] The sender sends $\mathbf{v}$ and the receiver sends $\mathbf{v}^*$ to $\mathcal{F}_{\mathsf{EPC}}$, and receives $\mathbf{r} \in \mathbb{Z}_{2^{\ell_{\mathsf{c}}}}^M$ and $\mathbf{r}^* \in \mathbb{Z}_{2^{\ell_{\mathsf{c}}}}^M$ from $\mathcal{F}_{\mathsf{EPC}}$ respectively.
4. [**Equality Share Generation**] For $1 \le i \le M$, the sender sends r_i and the receiver sends r_i^* to $\mathcal{F}_{\mathsf{ESG}}$, and receives $\mathbf{s}_{0,i} \in \{0,1\}$ and $\mathbf{s}_{1,i} \in \{0,1\}$ from $\mathcal{F}_{\mathsf{ESG}}$ respectively.

Fig. 7. Π_{CPSI}. Protocol of our circuit-PSI: OPPRF-based framework + EPC

For the asymptotic complexity view, the overall cost remains same since EPC itself takes $\widetilde{O}(\ell)$ complexities. However, EPC changes the input bit-length of the most heavy equality share generation (ESG) part from ℓ to $\ell_{\mathsf{c}} = O(\log \ell)$, which brings concrete performance gain.

About the choice of ℓ_{c}. Observe that EPC cost increases with smaller the output length ℓ_{c} but ESG cost decreases with ℓ_{c}, and hence there would be a balancing point of ℓ_{c}.

For more detailed discussion, we have to consider the concrete cost since the asymptotic total cost is always $O(\ell)$ regardless of the choice of ℓ_{c}. One plausible way is to measure the communication costs along with ℓ_{c} that can be objectively calculated than computational costs. However it may leads to inadequate conclusion, in particular when ESG is performed by Silent OT which has extremely low communication cost but high computational cost.

There is another point that makes the problem difficult. Observe that EPC output length ℓ_{c} is determined by HE plaintext modulus p as $\ell_{\mathsf{c}} = \lceil \log p \rceil$. However plaintext modulus p has to satisfy HE batching condition $p = 1 \mod 2n$, and the choice of ℓ_{c} is somewhat restrictive.

For reasons mentioned above, we postpone the discussion of ℓ_{c} choice to later experimental Section 6 where we discuss with the concrete numbers.

Offline tag encryption. The tag vector $\mathbf{v}$ sampled by the sender in the bin tagging step is independent to the input set of the protocol, so it can be sampled before the input set is known, in other words in offline phase. This observation brings negligible improvement in the original framework without equality preserving compression, as it only shifts the random $\mathbf{v}$ sampling time to offline. Meanwhile, it has a notable effect when combined with our equality preserving compression, as the server can perform the encryption phase of Π_{BEPC} in offline phase. Then the online phase of the protocol performs only HE operations, which leads to faster online execution.

6 Performance Evaluation

In this section, we consider concrete instantiation of our circuit-PSI protocol of Section 5 and evaluate the performance. More precisely, we first discuss concrete parameter selections of sub-protocols, especially with respect to the compression target ℓ_{c}. Then we evaluate the performances of several combinations of our EPC protocol and previous ESG protocols reviewed in Section 3.2. Finally we provide full circuit-PSI protocol costs evaluation by attaching previous hashing and OPPRF steps, and some consequences of our protocols.

Throughout this section, we assume computational security parameter $\lambda = 128$ and statistical security parameter $\sigma = 40$. For experiments, we use a single machine equipped with 3.50GHz Intel Xeon processors with 128GBs of RAM and the network setting is simulated via local host that has 30Gbps bandwidth in default (denoted by LAN). All experiments are executed with a single thread on each party in order to be consistent with previous works.

6.1 Parameter Selections

OPPRF output length (EPC input length) $\boldsymbol{\ell}$. For circuit-PSI on input set size N, OPPRF output strings will be fed as input of EPC. We take the length $\ell = \sigma + \lceil \log M \rceil$ for failure probability less than $2^{-\sigma}$ (See Section 3) where $M = (1+\varepsilon) \cdot N$ is cuckoo/simple hash table size with d hash functions. We use $\varepsilon = 0.27$ and $d = 3$ by following previous works [35, 39], and then OPPRF output length is given by $\ell = \sigma + 1 + \lceil \log N \rceil$.

HE Parameters for EPC for output length $\boldsymbol{\ell_{\mathsf{c}}}$. First of all we fix HE ring dimension $n = 2^{12}$ which is the minimal one supporting depth-1 scalar multiplication. Upon the choice of n, HE plaintext modulus p is taken by some prime integer that satisfying $p = 1 \mod 2n$ (1) for batching. Here note that HE plaintext modulus p determines compression target length $\ell_{\mathsf{c}} = \lceil \log p \rceil$. The minimal prime satisfying $p = 1 \mod 2n$ is $p = 40961$, and hence the minimal possible ℓ_{c} is $\lceil \log(40961) \rceil = 16$. For larger $\ell_{\mathsf{c}} > 16$, there would be several primes p such that $p = 1 \mod 2n$ having bit-length ℓ_{c}, and we choose maximal p among them for each ℓ_{c}. HE ciphertext modulus q is determined as following: An initial modulus is taken q' by the minimal one where our protocol is correct, and then the final modulus q is augmented by σ-bit margin on q' for function privacy. It empirically holds that $\log q \approx \sigma + 2\log p + \log n$. Finally we take the word-base w by the maximal one satisfying the correctness condition $p > u \cdot (w-1)^2$ (3)

where $u = \lceil \ell / \log w \rceil$. Table 3 shows some resulting parameters for some input size N and ℓ_{c}. All HE parameters (n, q) satisfies at least $\lambda = 128$-bit security according to [1].

		EPC output ℓ_{c}				
		16	18	20	22	28
	p	40961	188417	1032193	4169729	268369921
	$\log q$	84	88	92	96	108
$N = 2^{16}$	w	65	154	385	834	7327
$(\ell = 57)$	u	10	8	7	6	5
$N = 2^{20}$	w	62	145	360	772	7327
$(\ell = 61)$	u	11	9	8	7	5

Table 3. EPC Parameters for input set size N and EPC output length ℓ_{c}: w is the word-decompose base, and u is the length of decomposition. Note that HE parameters are independent to set size N.

6.2 Combinations of EPC and ESG

Recall that we review ESG protocols as Table 2 in Section 3.2: IKNP-GMW, KK-CGS, IKNP-OSN, Silent-GMW[2]. In this section we discuss applications of EPC on top of them[3]. More precisely, for an input set size N, we have $M = (1+\varepsilon) \cdot N$ numbers of bit strings of length $\ell = \sigma + 1 + \lceil \log N \rceil$ to generate equality share. We then apply EPC protocol with compression target ℓ_{c}, and then execute ESG protocol with ℓ_{c} bit strings.

Communication costs. Table 4 and Table 5 shows communication costs of each EPC and ESG combination with several choice of ℓ_{c}, where 'None' rows mean previous approaches that ESG protocol is directly fed with ℓ bit strings.

As the most interesting consequence, note that IKNP-OSN and KK-CGS shows similar communication cost before applying EPC, but EPC makes huge difference between them. It is natural because IKNP-OSN enjoys less benefit from EPC than other protocols as it has linear cost in $\lambda + 4\ell_{\mathsf{c}}$ rather than ℓ_{c} alone. We also note that EPC rather increases communication cost for Silent-GMW, due to extremely small communication cost of Silent OTe. However this does not mean EPC is useless for Silent-GMW, as it provides much faster running time.

[3] We emphasize again that, although it is possible to consider other OTe-ESG combination, our choices are the best one regardless of input length ℓ_{c}. See Appendix A.3 for detailed argument.

	ℓ_c	EPC	IKNP-GMW	KK-CGS	IKNP-OSN
$N = 2^{16}$ ($\ell = 57$)	16	11.20	49.60 (22%)	26.28 (42%)	41.68 (26%)
	18	9.92	53.44 (18%)	29.01 (34%)	41.67 (23%)
	20	9.43	58.07 (16%)	28.64 (32%)	42.45 (22%)
	None	-	143.4	60.34	56.51
$N = 2^{20}$ ($\ell = 61$)	16	187.3	801.7 (23%)	428.6 (43%)	797.2 (23%)
	18	168.1	864.4 (19%)	473.5 (35%)	803.1 (21%)
	20	161.1	939.3 (17%)	468.4 (34%)	821.5 (19%)
	None	-	2457	1032	1181

Table 4. Communication cost of EPC and ESG combinations in MB. For each column named with ESG protocol, the numbers are the sum of EPC and ESG protocol cost. The percentage in parenthesis is the portion of EPC cost.

	ℓ_c	EPC	Silent-GMW
$N = 2^{16}$ ($\ell = 57$)	16	11.20	11.80 (95%)
	22	9.231	10.14 (91%)
	28	8.861	9.933 (90%)
	None	-	2.222
$N = 2^{20}$ ($\ell = 61$)	16	187.3	196.8 (95%)
	22	152.8	166.1 (92%)
	28	137.6	154.6 (89%)
	None	-	38.10

Table 5. Communication cost of EPC and Silent-GMW combinations in MB. For 'Silent-GMW' column, the numbers are the sum of EPC and Silent-GMW protocol cost. The percentage in parenthesis is the portion of EPC cost.

Timing costs. We proceed to compare timing costs of EPC and ESG combinations based on implementation results. Note that from communication table 4, IKNP-OSN even shows similar cost with IKNP-GMW when $\ell_c = 16$, where both protocol shows the smallest communication cost. As IKNP-GMW is expected to show much faster running time than IKNP-OSN, IKNP-OSN seems no advantage in both communication and timing cost. Therefore we can rule out IKNP-OSN for the final candidates of ESG.

To implement our EPC protocol, we make use BFV [6,15] implementation of Microsoft SEAL [40]. To implement ESG protocols, we use an open source OT library libOTe [38][4]. Table 6 shows the experimental results. As EPC reduces total number of OTs for ESG (which was linear in ESG input length ℓ), our EPC combination reduces timing costs for all ESG protocols, especially also for Silent-GMW.

[4] Actually libOTe does not support KK OT extension [23], and we adapt the library.

	ℓ_c	EPC	IKNP-GMW	KK-CGS	Silent-GMW	EPC	ℓ_c
$N = 2^{16}$ ($\ell = 57$)	16	0.383	0.798 (47%)	1.076 (35%)	2.130 (18%)	0.383	16
	18	0.351	0.797 (44%)	1.155 (30%)	2.127 (12%)	0.257	22
	20	0.301	0.809 (37%)	1.119 (27%)	3.515 (6%)	0.211	28
	None	-	1.161	2.310	6.698	-	None
$N = 2^{20}$ ($\ell = 61$)	16	6.391	10.81 (59%)	15.04 (42%)	38.24 (17%)	6.391	16
	18	5.510	10.44 (53%)	15.47 (36%)	38.78 (11%)	4.318	22
	20	4.929	10.48 (47%)	15.87 (31%)	68.25 (5%)	3.351	28
	None	-	17.07	35.18	139.4	-	None

Table 6. Timing cost of EPC and ESG combinations in second. For each column named with ESG protocol, the numbers are the sum of EPC and ESG protocol cost. The percentage in parenthesis is the portion of EPC cost.

Best ℓ_c for each ESG protocol. IKNP-GMW and KK-CGS has best performance for both communication and computational cost when $\ell_c = 16$, which is the minimal possible choice for ℓ_c. The communication cost of Silent-GMW rather decreases with larger ℓ_c, but the running time increases with larger ℓ_c. As the running time of $\ell_c = 16$ and 22 are similar for Silent-GMW, we conclude $\ell_c = 22$ is the best EPC output length for Silent-GMW.

6.3 Circuit-PSI

To complete circuit-PSI protocols, we only need to attach hashing and OPPRF protocols before EPC and ESG protocols. We use Microsoft Kuku [27] library for cuckoo/simple hashing, and we find that OPPRF proposals of [39] and [8] showed almost same best performance. Since the works' implementations are not publicized, we implement [8]'s OPPRF using libOTe [38], and implement [39]'s OPPRF by adapting [33] implementation[5]. For underlying OPRF for OPPRF, we implement [24] OPRF, which is also based on libOTe [38].

Previous best protocols. The previous best protocols are also due to [39] and [8] what we denote RS-C and CGS-C circuit-PSI protocol, respectively. [39] performs ESG in two ways by IKNP-GMW and Silent-GMW, and we distinguish two circuit-PSI protocols by IKNP-RS-C and Silent-RS-C, respectively. [8] performs ESG by KK-CGS-C.

Performance comparison. We provide results of applying EPC to the previous best protocols by Table 7. Note that previous best protocols already used the best candidates of ESG discussed in Section 6.2; IKNP-GMW, KK-CGS and Silent-GMW, and hence we cannot make further improvement from selecting other ESG protocol combination. 'Improve' rows show EPC brings overall

[5] Available : `https://github.com/cryptobiu/PaXoS_PSI`

improvement of all protocols except communication cost of Silent-RS-C [39]. It implies that EPC always provides better performance for IKNP-RS-C and KK-CGS-C *regardless of network environment*. For Silent-RS-C, it originally takes $154+255\cdot 8/x$ seconds and Silent-RS-C with EPC takes $53+383\cdot 8/x$ on x Mbps bandwidth. Then EPC improves Silent-RS-C for bandwidth larger than around 10 Mbps.

$N = 2^{16}$	Time (s)			Comm. (MB)		
	OPPRF	ESG	Total	OPPRF	ESG	Total
IKNP-RS-C [39]	0.82	1.16	1.98	13.6	143	157
w/ EPC ($\ell_c = 16$)	0.82	0.78	1.60	13.6	49.6	63.2
Improve	-	1.48x	1.23x	-	2.88x	2.48x
Silent-RS-C [39]	0.82	6.70	7.52	13.6	2.22	15.8
w/ EPC ($\ell_c = 22$)	0.82	2.13	2.95	13.6	10.1	23.7
Improve	-	3.14x	2.54x	-	0.21x	0.66x
KK-CGS-C [8]	0.58	2.31	2.89	7.92	60.3	68.2
w/ EPC ($\ell_c = 16$)	0.58	1.08	1.66	7.92	26.3	34.2
Improve	-	2.13x	1.74x	-	2.29x	2x
$N = 2^{20}$	Time (s)			Comm. (MB)		
	OPPRF	ESG	Total	OPPRF	ESG	Total
IKNP-RS-C [39]	14.2	17.1	31.3	217	2457	2674
w/ EPC ($\ell_c = 16$)	14.2	10.8	25.0	217	802	1019
Improve	-	1.58x	1.25x	-	3.06x	2.62x
Silent-RS-C [39]	14.2	139.4	154	217	38.1	255
w/ EPC ($\ell_c = 22$)	14.2	38.8	53.0	217	166	383
Improve	-	3.59x	2.9x	-	0.22x	0.66x
KK-CGS-C [8]	8.45	35.2	43.7	126	1032	1158
w/ EPC ($\ell_c = 16$)	8.45	15.0	23.4	126	429	555
Improve	-	2.34x	1.86x	-	2.4x	2.08x

Table 7. Application of EPC on the previous best circuit-PSI protocols.

Performance of circuit-PSI applications with EPC. Circuit-PSI leads to several useful protocol: PSI-Ca, PSI-Th, PSI-Sum and PSU (See Section 3.3 for definitions and details). We remark that they already show highly efficient performance for each functionality even before our EPC optimization, and our EPC brings further improvement on the performances. Table 8 provides resulting performances of circuit-PSI based protocols and some comparisons with non-circuit-PSI based protocols. Compared with the Diffie-Hellman based protocol called *Private Join & Compute (PJC)* [21] that supports PSI-Ca and PSI-Sum, our circuit-PSI based protocols have significantly faster running time but shows latter larger communication burden. For PSU, circuit-PSI based protocol (with our EPC) has definitely better performance than non-circuit-PSI based one [26]. Note that from '(Base) Circuit-PSI' row, the base circuit-PSI consists of the most of portion of total cost as also pointed out in [35].

		$N = 2^{16}$		$N = 2^{20}$	
		Time (s)	Comm. (MB)	Time (s)	Comm. (MB)
PSI-Ca	[21] (PJC)	48.9	5.1	776	84
	Ours	1.88	35.8	24.8	582
PSI-Sum	[21] (PJC)	48.9	5.1	776	84
	Ours	1.85	36.1	24.7	585
PSU	[26]	12.8	131	243	2476
	Ours	1.85	36.6	24.8	594
(Base) Circuit-PSI		1.66	34.2	23.4	555

Table 8. Cost of circuit-PSI application with EPC optimization, denoted by 'Ours', where the base circuit-PSI protocol is chosen by KK-CGS-C. In PSI-Sum, the associate values are assumed by 32-bit, same to [21].

References

1. Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic Encryption Security Standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.
2. Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty Computation with Low Communication, Computation and Interaction via threshold FHE. In *EUROCRYPT*, pages 483–501. Springer, 2012.
3. Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More Efficient Oblivious Transfer and Extensions for Faster Secure Computation. In *ACM CCS*, pages 535–548, 2013.
4. Jean-Claude Bajard, Julien Eynard, M Anwar Hasan, and Vincent Zucca. A full RNS variant of FV like somewhat homomorphic encryption schemes. In *International Conference on Selected Areas in Cryptography*, pages 423–442. Springer, 2016.
5. Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation. In *ACM CCS*, pages 291–308, 2019.

6. Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO*, pages 868–886. Springer, 2012.
7. Prasad Buddhavarapu, Andrew Knox, Payman Mohassel, Shubho Sengupta, Erik Taubeneck, and Vlad Vlaskin. Private Matching for Compute. *IACR Cryptol. ePrint Arch.*, 2020:599, 2020.
8. Nishanth Chandran, Divya Gupta, and Akash Shah. Circuit-PSI with Linear Complexity via Relaxed Batch OPPRF. Cryptology ePrint Archive, Report 2021/034, 2021. https://eprint.iacr.org/2021/034.
9. Melissa Chase and Peihan Miao. Private Set Intersection in the Internet Setting From Lightweight Oblivious PRF. In *CRYPTO*, pages 34–63. Springer, 2020.
10. Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In *ACM CCS*, pages 1223–1237, 2018.
11. Hao Chen, Kim Laine, and Peter Rindal. Fast Private Set Intersection from Homomorphic Encryption. In *ACM CCS*, pages 1243–1255, 2017.
12. Michele Ciampi and Claudio Orlandi. Combining Private Set-Intersection with Secure Two-party Computation. In *SCN*, pages 464–482. Springer, 2018.
13. Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY-A framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.
14. Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. Pushing the Communication Barrier in Secure Computation using Lookup Tables. In *NDSS*, 2017.
15. Junfeng Fan and Frederik Vercauteren. Somewhat Practical Fully Homomorphic Encryption. *IACR Cryptol. ePrint Arch.*, 2012:144, 2012.
16. Gayathri Garimella, Payman Mohassel, Mike Rosulek, Saeed Sadeghian, and Jaspal Singh. Private Set Operations from Oblivious Switching, 2021.
17. Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In *CRYPTO*, pages 395–425. Springer, 2021.
18. Craig Gentry et al. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Standford University, 2009.
19. O. Goldreich, S. Micali, and A. Wigderson. How to Play ANY Mental Game. In *STOC*, page 218–229, New York, NY, USA, 1987. Association for Computing Machinery.
20. Yan Huang, David Evans, and Jonathan Katz. Private Set Intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.
21. Mihaela Ion, Ben Kreuter, Ahmet Erhan Nergiz, Sarvar Patel, Shobhit Saxena, Karn Seth, Mariana Raykova, David Shanahan, and Moti Yung. On Deploying Secure Computing: Private Intersection-Sum-with-Cardinality. In *EuroS&P*, pages 370–389. IEEE, 2020.
22. Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending Oblivious Transfers Efficiently. In *CRYPTO*, pages 145–161. Springer, 2003.
23. Vladimir Kolesnikov and Ranjit Kumaresan. Improved OT extension for transferring short secrets. In *CRYPTO*, pages 54–70. Springer, 2013.
24. Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient Batched Oblivious PRF with Applications to Private Set Intersection. In *ACM CCS*, pages 818–829, 2016.
25. Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical Multi-party Private Set Intersection from Symmetric-key Techniques. In *ACM CCS*, pages 1257–1272, 2017.

26. Vladimir Kolesnikov, Mike Rosulek, Ni Trieu, and Xiao Wang. Scalable private set union from symmetric-key techniques. In *ASIACRYPT*, pages 636–666. Springer, 2019.
27. Microsoft Kuku. https://github.com/microsoft/Kuku. Microsoft Research, Redmond, WA.
28. Catherine Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *S&P*, pages 134–134. IEEE, 1986.
29. Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. Two-sided Malicious Security for Private Intersection-Sum with Cardinality. In *CRYPTO*, pages 3–33. Springer, 2020.
30. Payman Mohassel and Saeed Sadeghian. How to hide circuits in MPC an efficient framework for private function evaluation. In *EUROCRYPT*, pages 557–574. Springer, 2013.
31. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238. Springer, 1999.
32. Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight Private Set Intersection from Sparse OT Extension. In *CRYPTO*, pages 401–431. Springer, 2019.
33. Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXoS: fast, malicious Private Set Intersection. In *EUROCRYPT*, pages 739–767. Springer, 2020.
34. Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private Set Intersection Using Permutation-based Hashing. In *USENIX Security*, pages 515–530, Washington, D.C., August 2015. USENIX Association.
35. Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient Circuit-based PSI with Linear Communication. In *EUROCRYPT*, pages 122–153. Springer, 2019.
36. Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient Circuit-based PSI via Cuckoo Hashing. In *EUROCRYPT*, pages 125–157. Springer, 2018.
37. Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow2: Practical 2-party secure inference. In *ACM CCS*, pages 325–342, 2020.
38. Peter Rindal. libOTe: an efficient, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.
39. Peter Rindal and Phillipp Schoppmann. VOLE-PSI: Fast OPRF and Circuit-PSI from Vector-OLE. In *EUROCRYPT*. Springer, 2021.
40. Microsoft SEAL (release 3.5). https://github.com/microsoft/SEAL, April 2020. Microsoft Research, Redmond, WA.

A Details of ESG Protocols

A.1 CGS Protocol

Given two ℓ-bit strings $\mathbf{a}$ and $\mathbf{b}$, [8] proposed another protocol for generating equality share of $\mathbf{a}$ and $\mathbf{b}$. The full detail is given by Figure 8. It requires $(2^d, 1)\text{-}\mathsf{OT}_1^1$ for generating equality shares of substrings, and then AND gate evaluations of them.

Parameters: A receiver with an ℓ-bit string $\mathbf{a}$ and a sender with an ℓ-bit string $\mathbf{b}$, and an optimization parameter d.

Protocol:

1. Two parties agree on ℓ_1 and ℓ_2 such that $\ell = d\ell_1 + \ell_2$ with $0 < \ell_2 \leq d$.
2. For $0 \leq i < \ell_1$, two parties run $(2^d, 1)\text{-}\mathsf{OT}_1^1$ as following
   (a) The sender computes $b_i = \sum_{k=1}^{d} b_{di+k} \cdot 2^k$ and samples a random bit $r_i \in \{0,1\}$. Then defines j-th message $m_j = \mathbf{1}(b_i = j) \oplus r_i$ for $0 \leq j < 2^d$.
   (b) The receiver defines the choice index $a_i = \sum_{k=1}^{d} a_{di+k} \cdot 2^k$, and obtain the corresponding message $s_i \in \{0,1\}$.
3. Two parties run $(2^{\ell_2}, 1)\text{-}\mathsf{OT}_1^1$ as following
   (a) The sender computes $b_{\ell_1} = \sum_{k=1}^{\ell_2} b_{d\ell_1+k} \cdot 2^k$, and samples a random bit $r_{\ell_1} \in \{0,1\}$. Then defines j-th message $m_j = \mathbf{1}(b_{\ell_1} = j)$ for $0 \leq j < 2^{\ell_2}$.
   (b) The receiver defines the choice index $a_{\ell_1} = \sum_{k=1}^{\ell_2} a_{d\ell_1+k} \cdot 2^k$, and obtain the corresponding message $s_i \in \{0,1\}$.
4. Two parties privately compute Boolean shares of $\prod_{i=0}^{\ell_1}(r_i \oplus s_i)$.

Fig. 8. Π_{CGS}. Protocol of equality share generation from [8]

A.2 OSN Protocol

The original description of the protocol in [16] does not exactly correspond to circuit-PSI definition in rigorous sense: Instead of Boolean shares, it outputs a permutation π over one party's set X, and a vector $\mathbf{e} \in \{0,1\}^N$ such that $e_i = 1$ if and only if $x_{\pi(i)} \in X \cap Y$ to the other party. Note that the cardinality of intersection is unavoidably revealed as a Hamming weight of $\mathbf{e}$. Nevertheless, one can privately convert the outputs into equality shares using $(2,1)\text{-}\mathsf{OT}_1^M$ (by Figure 9), and hence it is fine to understand this protocol as a sort of (cardinality-revealing) ESG.

A.3 Combination of ESG Protocol and OT Extension

Table 9 lists up possible combinations of ESG protocols and OT extensions, with required number of OT and communication cost where some small terms

Parameters: A receiver with a set X of size n and a permutation π, and a sender with a vector $\mathbf{e} \in \{0,1\}^n$
Protocol:

1. For $0 \leq i < n$, two parties run $(2,1)$-OT_1^1 as following
   (a) The receiver samples a random bit $s_{0,i} \in \{0,1\}$ and defines $m_0 = s_{0,i}$ and $m_1 = 1 - s_{0,i}$.
   (b) The sender uses a choice bit e_i and obtain the corresponding message $s_{1,i} := m_{e_i} \in \{0,1\}$.

Fig. 9. Conversion of [16]'s output to equality shares

are omitted. Below we explain some candidates are strictly worse than the other *regardless of* input bit-length ℓ. This justifies that we only have to consider four ESG protocols in Table 2.

IKNP OTe. GMW protocol is strictly better than CGS protocol. However it is ambiguous to say which one is better between GMW and OSN protocol, and we leave both IKNP-GMW and IKNP-OSN protocol as possible candidates.

KK OTe. CGS protocol (with $d = 4$) is strictly better than GMW protocol, and we can rule out KK-GMW protocol. OSN protocol can be executed with KK OTe with *parallel* $(2,1)$-OT_ℓ^m call by $(2^d,1)$-$\mathsf{OT}_{d\ell}^{m/d}$. However, the communication cost becomes larger when OSN protocol is executed by IKNP OTe. Thus we also rule out KK-OSN candidate.

Silent OTe. Same to IKNP OTe case, GMW protocol is strictly better than CGS protocol. Different to IKNP OTe case, GMW protocol is strictly better than OSN protocol, since OSN protocol has $\log M$ times larger communication cost. Thus we only leave Silent-GMW protocol.

B Comparison with Paillier Additive HE

As our protocol only perform scalar multiplications, one may consider to use another *additive HE* (AHE), for example Paillier [31] scheme. Paillier scheme supports plaintext space $\mathbb{Z}_P$ for some integer P, and the corresponding ciphertext space is $\mathbb{Z}_{P^2}$. Here P is typically taken quite large ($\geq 2^{1024}$) to ensure certain security level, and a naive application of Protocol Π_{EPC} outputs huge random numbers in $\mathbb{Z}_P$. This can be circumvented by applying well-known *smudging* technique [2] where we take a sufficiently large random masking r so that r statistically hides the information of d, and each party take the final modulus reduction by p on each output $d + r$ and r.

OT extension	Base Protocol	Required OT	Comm.
IKNP	GMW	$(2,1)\text{-}\mathsf{COT}_1^{2\ell}$	$2\lambda\ell$
	$\text{CGS}_{d=1}$	$(2,1)\text{-}\mathsf{OT}_1^{\ell} + (2,1)\text{-}\mathsf{COT}_1^{2\ell}$	$3\lambda\ell$
	OSN	$(2,1)\text{-}\mathsf{OT}_{2\ell}^{\log M}$	$\log M(\lambda + 4\ell)$
KK	GMW	$(16,1)\text{-}\mathsf{OT}_2^{\ell/2}$	$(\lambda + 12)\ell$
	$\text{CGS}_{d=4}$	$(16,1)\text{-}\mathsf{OT}_1^{\ell/4} + (16,1)\text{-}\mathsf{OT}_2^{\ell/8}$	$0.75(\lambda + 12)\ell$
	OSN	$(4,1)\text{-}\mathsf{OT}_{4\ell}^{\log M/2}$	$\log M(\lambda + 8\ell)$
Silent	GMW	$(2,1)\text{-}\mathsf{COT}_1^{2\ell}$	4ℓ
	$\text{CGS}_{d=1}$	$(2,1)\text{-}\mathsf{OT}_1^{\ell} + (2,1)\text{-}\mathsf{COT}_1^{2\ell}$	7ℓ
	OSN	$(2,1)\text{-}\mathsf{OT}_{2\ell}^{\log M}$	$4\ell \log M$

Table 9. Cost for ESG Candidates for several OT extensions.

However, we argue that RLWE-based AHE is still better for circuit-PSI purpose, where the encryption target message size is much less than 32-bit. RLWE-based AHE can supports plaintext space $\mathbb{Z}_p^N$ for rather small p, and the corresponding ciphertext space is taken $\mathcal{R}_q^2$ where $\log q = O(\log p)$. Then the amortized encryption cost per one message is $2 \log q$. For our interest message size, RLWE ciphertext modulus $q \approx 2^{100}$ suffices so that one message is encrypted into less than 200 bits,. However Paillier AHE encrypts a message into a quite large ciphertext of $2 \log P \geq 2048$ bits, and the amortized cost is less inefficient than RLWE-base AHE.