# Efficient Threshold Private Set Intersection via BFV Fully Homomorphic Encryptions

Jingwei Hu[1] and Wangchen Dai[2]

[1] Nanyang Technological University, Singapore, {davidhu}@ntu.edu.sg
[2] Zhejiang Laboratory, China, {w.dai}@zhejianglab.com

**Abstract.** In this paper, we consider how to use fully homomorphic encryptions (FHEs) to solve the problem of private intersection cardinality and, more broadly, the problem of threshold private set intersection where one party holding a set of size $N$ and the other party holding a set of size $n$ collaboratively compute their set intersection without revealing other information about their own set if and only if the intersection size excesses the prescribed threshold. This problem has many applications for online collaboration, for example, fingerprint matching, online dating, and shareriding.

**Keywords:** Private Set Intersection, Threshold PSI, Fully Homomorphic Encryption

## 1 Introduction

This work considers how to construct threshold private set intersection (threshold PSI) from fully homomorphic encryptions. Our protocol enables two parties (*a.k.a.* sender and receiver) that each holds a set of inputs (set size is $N$ and $n$, respectively) to jointly compute the intersection if and only if the size of intersection set satisfies some specific requirement, *e.g.*, the intersection size is above a given threshold value, without leaking extra information of his own set.

### 1.1 Contributions

We study the possibility of using state-of-art fully homomorphic encryptions to solve the threshold PSI problem:

1. A private set intersection cardinality protocol for two parties against semi-honest adversaries. The communication complexity of the proposed method reaches the lower bound $\Omega(min(N, n))$.

2. A new secret token generation protocol is constructed on our private set intersection cardinality protocol. Then a PSI threshold protocol for two parties in semi-honest secure model is constructed by combining a standard PSI protocol and the new secret token generation protocol, and has communication complexity of $\Omega(N + n)$.

3. a variety of FHE-related techniques are presented to support our protocols over large sets meanwhile reducing the concrete computation/communication overheads.

4. Our protocols remain conceptually simple and modular, which simplifies both security analysis and implementations.

### 1.2 Technical Overview

Our main threshold prviate set intersection protocol can be split into two subprotocols. One for threshold token exchange protocol where a secret token is securely shared between two parties if the size of set intersection satisfies the prescribed requirement, *e.g.*, the intersection size exceeds some threshold value, and one for standard private set intersection protocol. Here we highlight some of the main ideas underlying our protocols. Our construction is based on the following assumptions which are more likely to occur in real world applications: 1) the sender's set is not smaller than the receiver's set. 2) the sender's computaitonal power is significantly than the receiver's.

**Threshold Token Exchange Protocol** The goal of threshold token exchange protocol is to enable the sender sharing a common token with the receiver if and only if the intersection set size satisfies the threshold criterion. More formally, the receiver encodes his set $C$ as a Bloom filter $\mathbf{BF}_C$ and then sends his (encrypted) Bloom filter to Sender. Sender exploits the homomorphic operations in fully homomorphic encryption to check for each element $s_i$ in his own set $S$ whether $s_i$ belongs to $\mathbf{BF}_C$ ($a.k.a$ membership test). The Bloom filter based membership test have been used in previous works [DCW13, DD15, DD15, ZC18]. However, in contrast to these works, our new solution is to utilize the FHE-based oblivious polynomial evaluation technique for directly converting the result of membership tests into intersection cardinality. Our private set intersection sub-protocol achieves communication complexity $\Omega(min(N, n))$ which is asymptotically optimal. Based on the set intersection cardinality sub-protocol, we again utilize programmable bootstrapping to construct the threshold token exchange protocol: Sender locally generates a secret token $K$ and homomorphically generates an encryption of another secret token $K'$ sent to Alice who decrypts it to obtain $K'$. The key point is that $K'$ equals to $K$ if the intersection cardinality complies with the threshold, otherwise $K'$ is completely random.

**Standard PSI Protocol** The sender who has his set $S$ and a secret token $K$ proceeds to modify his set as $S^K = \{s_i||K\}_i$ where each element in the new set $S^K$ is his original set element $s_i$ concatenated with the token $K$. Likewise, the receiver uses his token $K'$ to modify his set as $C^{K'} = \{c_i||K'\}_i$. Finally, the sender and the recevier engages in a normal PSI protocol with their updated sets. Note that there is no restriction on what PSI protocol can be used here. In our actual implementation, we choose the simple *Elliptic Curve Diffie-Hellman key exchange* based PSI protocol.

## 2 Related work

The standard PSI, *i.e.*, two parties securely computing the set intersection (does not require computing some function $f$ on the intersection) while the privacy of each party is preserved, is extensively studied in the literature and is still an active research topic. We categorize these research work into 4 classes as follows.

**Diffie-Hellman key exchange based** The earliest PSI protocols are build upon the Diffie-Hellman key exchange [Mea86, HFH99] with semi-honest security, and later extended with malicious security [CKT10, JL10]. RSA accumulator is exploited in to implement PSI protocol [ACT11]. The merit of the DH-based PSI protocols is the low communication complexity. However, these protocols reply on the complicated computation of expoentiation, which makes the computational overhead rather large. In particular, when the set held by each party is large, the computation efficiency for the DH-based PSI is low.

**oblivious transfer based** OT-extension based constructions are the mainstream of PSI protocols. These constructions typically include the hash-to-bins technique by using various hashing algorithms, *i.e.*, simple hashing, permutation-based hashing, or Cuckoo hashing, which helps reduce the computational overhead. Since the OT extension technique which relies on fast symmetric encryptions and a few asymmetric encryptions is highly efficient, the OT-extension based constructions [PSZ14, KKRT16, PSSZ15, HV17, RR17, CLR17, OOS17, PSWW18, PSZ18] are also fast. [KKRT16] also points out that the OT extensions used in this class of PSIs can be viewed as oblivious pseudorandom function (OPRF).

**oblivious key-value store based** OKVS-based construction emerges as a new branch of PSI protocols. It uses a special encoding method to encode the set into an oblivious data structure called oblivious key-value store(OKVS). The computing efficiency of OKVS-based constructions [PRTY19, CM20] is initially low since they use complicated polynomial interpolation technique. Later proposed the so called PaXoS data structure which significantly improves the computing efficiency. Improving on this result, the up-to-date fastest OKVS construction appears [RS21]. The concept and the method for OKVS is further refined in [GPR$^+$21].

**polynomial manipulation** The polynomial manipulation based constructions [FNP04, KS05, DSMRY09, HN10, Haz18, FHNP16, GS19, GN19] present the set as a polynomial, and use polynomial related operations, *e.g.*, oblivious polynomial evaluation (OPE), to equivalently perform set operations. These constructions are generally slower and also show advantages on other aspects. For example, they can argue the security in the standard model without replying on non-standard assumptions like random oracle model. They can provide more functionalities rather rhan simply computing the set intersection (refer to the threshold PSI [GS19] as one such instance). [CLR17] exploits fully homomorphic encryption to perform polynomial evaluation obliviously. Their method shows better communication complexity which only relies on the size of the smaller set if the size of set held by each party is different. [CMdG$^+$21] further optimizes [CLR17] by exploring Paterson-Stockmeyer algorithm, postage-stamp bases, and Frobenius mapping to reduce the computation and communication overhead.

**Branching program** This is a newly emerging and intriguing approach [JTKA22], with an asymptotic communication complexity sublinear to the smaller set. It exhibits an advantage when computing between non-equal sets. The core concept involves transforming set membership tests into Branching Programs (BPs), and homomorphically executing the BP. The significance of this approach lies not only in its potential for calculating PSI but also in its convenience for various operations based on PSI results, such as private computation on set intersections (PCI). Currently, this method is limited to theoretical constructs and theoretical performance estimations.

Besides the study of standard PSI, the cryptographic research community also investigate variants of standard PSI. Private intersection cardinality and threshold private set intersection, among others, are probably most important variants.

**Private Intersection Cardinality** Private intersection cardinality problem [KS05, HW06, DD15, EFG$^+$15, PSWW18], considers how two parties collaboratively compute the cardinality of the intersected set, rather than the intersection itself. [FNP04] points out the lower bound of communication complexity of private set intersection is $\Omega(n)$, where $n$ is the size of the set held by each party. This lower bound naturally applies to the private intersection cardinality problem. Google's research team explored a variant of Private Intersection Cardinality problem called the two-party intersection-sum with cardinality problem for the first time in [IKN$^+$20]. This problem involves simultaneously calculating the size of the intersection and the sum of set payloads. According to Google's paper, the most practical methods currently still rely on classical Diffie-Hellman and additive homomorphic encryption schemes. In [MPR$^+$20], research was conducted on the intersection-sum with cardinality problem under a malicious security model. Additionally, [36] proposed a novel method for estimating the size of privacy-preserving intersections in the context of designing a COVID-19 contact tracing system (utilizing a combination of Diffie-Hellman and Keyword-PIR).

**Threshold Private Set Intersection** In many applications, we do not target computing the intersection or the intersection cardinality. For example, the two parties in the fingerprint matching, *i.e.*, a client who has a small set of all his features in his fingerprint, and a server who has a large set which contains fingerprint features for multiple users. The protocol returns yes if the number of matched features excess some prescribed threshold parameter $t$ (in order to tolerate false negative errors), otherwise returns no. Another good example is the lover pairing in dating website. the two parties are the man who has a set of his preferences, and the women who also has a set of her preferences. The protocol should return the intersected preferences if the number of shared preferences are large such that this pair of man and woman can learn more about each other. Otherwise, it should return an empty set (or nothing) to avoid personal privacy leakage. These applications are abstractly referred to as threshold PSI where each of the two parties hold a set of size $n$, and engates to compute the set intersection if the number of overlapping is above a threshold value $t$. Otherwise, each of the two parties should learn nothing. The threshold PSI problem is investigated in [FNP04, HOS17, GN19, PSWW18, ZC18]. [HOS17] combines the phasing technique and the threshold key encapsulation mechanism to construct threshold PSI. [PSWW18] optimizes the generic MPC for comparsion circuit to construct threshold PSI. [GN19] constructs a malicious secure threshold PSI based on the oblivious linear function evaluation and robust secret sharing. [ZC18] presents Paillier partial homomorphic encryption based oblivious polynomial evaluation to construct semi-honest secure PSI protocol. [GS19] reveals that when the intersection is large (say the size of intersection is at least $n - 2t \approx n$), threshold PSI protocol has to have a communication complexity of $\Omega(t)$. Based on the results from [GS19], threshold Private Set Intersection for multiple parties is further studied in [BMRR21] which suggests the sublinear communication complexity still holds in the multi-party case.

## 3 BFV

BFV, also known as one representative scheme in the second generation of fully homomorpic encryption (FHE), is advantageous for leveled homomorphic encryptions, *i.e.*, homomorphic evaluation of a circuit with shallow multiplication depth. Theoretically speaking, by using Gentry's bootstrapping technique, BFV can support evaluating a circuit with unlimited multiplication depth. Nevertheless, the state-of-arts of bootstrapping is still far from being efficient. In this paper, since we focus on constructing efficient for the threshold PSI related computations with optimal communication complexity, we restrict the usage of BFV to leveled HE mode.

### 3.1 RLWE

BFV ciphertext is essentially a RLWE ciphertext defined over polynomial ring $R_q = \mathbb{Z}_q[X]/(X^N + 1)$:

$$(a(X), b(X)) = (a(X), a(X) \cdot s(X) + \lceil \frac{q}{t} \rfloor m(X) + e(X)) \in R_q \times R_q$$

where $s$ is a secret key, $m$ is the message defined over $\mathcal{R}_q$ (*a.k.a* BFV plaintext), $e = \sum_i e_i X^i$ is an error polynomial where each coefficient is extracted from a discrete Gaussian distribution with small variance $e_i \sim \mathcal{N}(0, \sigma^2)$. Likewise, we use $RLWE_s^{N,q}(m)$ to denote a RLWE encryption of message $m(X)$.

### 3.2 Addition and Multiplication

Homomorphic addition and multiplication are two basic homomorphic operations supported by BFV. We skip the algorithmic descriptions for these two primitives but outline the functionality abstractly. HomoAdd takes two inputs as a RLWE encryption of message $m_0$ and a RLWE encryption of message $m_1$ defined over $\mathcal{R}_q \times \mathcal{R}_q$ and outputs another RLWE encryption of $m_0 + m_1$ defined over $\mathcal{R}_q \times \mathcal{R}_q$:

$$RLWE_s^{N,q}(m_0), RLWE_s^{N,q}(m_1) \xrightarrow{\texttt{HomoAdd}} RLWE_s^{N,q}(m_0 + m_1)$$

HomoAdd consumes trivial amount of noise budget and thus BFV can perform almost unlimited number of homomorphic additions. HomoAdd can also support addition of one RLWE plaintext and one RLWE ciphertext, *i.e.*, $RLWE_s(m_0+m_1) \leftarrow \texttt{HomoAdd}(m_0, RLWE_s(m_1))$.

Likewise, HomoMul takes two inputs as a RLWE encryption of $m_0$ and a RLWE encryption of $m_1$ and outputs another RLWE encryption of $m_0 \cdot m_1$:

$$RLWE_s^{N,q}(m_0), RLWE_s^{N,q}(m_1) \xrightarrow{\texttt{HomoMul}} RLWE_s^{N,q}(m_0 \cdot m_1)$$

It is worth noting that HomoMul consumes significantly more noise budget than HomoAdd does. The typical value for the depth of multiplcation that BFV supports is 6. HomoMul also supports multiplication of one RLWE plaintext and one RLWE ciphertext, *i.e.*, $RLWE_s(m_0 \cdot m_1) \leftarrow \texttt{HomoMul}(m_0, RLWE_s(m_1))$.

## 3.3 SIMD Encoding

SIMD (single instruction multiple data) encoding is a unique feature for the second generation of FHEs which encodes a vector $\mathbf{m} = (m_0, \cdots, m_{N-1})$ over $\mathbb{F}_q$ to a polynomial over $\mathcal{R}_q$:

$$\mathbb{F}_q^N \xrightarrow{\texttt{SIMD}} p(X) \in \mathcal{R}_t$$

The key idea in SIMD is that $x^N + 1$ can be factorized into $\prod_{i=0}^{N-1}(X - \zeta^{2i+1})$. By Chinese remainder theorem (CRT), we have:

$$R_t = \frac{\mathbb{Z}_t[x]}{x^N+1} = \frac{\mathbb{Z}_t[x]}{\prod_{i=0}^{N-1}(x-\zeta^{2i+1})} \simeq^{CRT} \prod_{i=0}^{N-1} \frac{\mathbb{Z}_t[x]}{x-\zeta^{2i+1}} = \prod_{i=0}^{i=N-1} \mathbb{Z}_t[\zeta^{2i+1}]$$

In other terms, every polynomial $p(X) = \sum_i p_i X^i$ over $\mathcal{R}_t$ can be equivalently represented by its evaluations at $X = \zeta^{2i+1}$ for $i \in [N]$. The evaluations can be written in matrix form as follows:

$$\begin{bmatrix} m_0 \\ \vdots \\ m_{N-1} \end{bmatrix} = \begin{bmatrix} \zeta^{1\cdot 0} & \cdots & \zeta^{1\cdot(N-1)} \\ \vdots & \vdots & \vdots \\ \zeta^{(2N-1)\cdot 0} & \cdots & \zeta^{(2N-1)\cdot(N-1)} \end{bmatrix} \cdot \begin{bmatrix} p_0 \\ \vdots \\ p_{N-1} \end{bmatrix}$$

The vector $[m_0, \cdots, m_{N-1}]^T$ is essentially the $N$ evaluation points associated with the polynomial $p(X)$. For SIMD encoding purpose, this vector of evaluation points are interpreted as the BFV plaintext vector, and by using SIMD encoding technique, the BFV plaintext vector is transformed to BFV plaintex polynomial $p(X)$. It is easily seen that BFV SIMD encoding is the inverse process of evaluating $p(X)$ at points $X = \zeta^{2i+1}$, written in matrix form as follows:

$$\begin{bmatrix} p_0 \\ \vdots \\ p_{N-1} \end{bmatrix} = \begin{bmatrix} \zeta^{1\cdot 0} & \cdots & \zeta^{1\cdot(N-1)} \\ \vdots & \vdots & \vdots \\ \zeta^{(2N-1)\cdot 0} & \cdots & \zeta^{(2N-1)\cdot(N-1)} \end{bmatrix}^{-1} \cdot \begin{bmatrix} m_0 \\ \vdots \\ m_{N-1} \end{bmatrix}$$

According to the isomorphism between the plaintext vector and the plaintext polynomial built from CRT, the addition/multiplication of two plaintext polynomials is equivalent to the element-wise addition/multiplication of two plaintext vectors. That is,

$$\text{If } p_0(X) \leftarrow \texttt{SIMD}(\mathbf{m_0}), p_1(X) \leftarrow \texttt{SIMD}(\mathbf{m_1}),\ \text{then } p_0(X)+p_1(X) \leftrightarrow \mathbf{m_0}+\mathbf{m_1},\ \text{and}$$
$$p_0(X)\cdot p_1(X) \leftrightarrow \mathbf{m_0}\otimes\mathbf{m_1}$$

## 3.4 Left Rotation

Regarding the rotation, we view the vector after SIMD encoding as a $2 \times \frac{N}{2}$ matrix. In particular, we set $\zeta_j = \zeta^{5^j \pmod{2N}}$ where $\zeta$ is a 2N-th primitive root of unity. $\zeta_j = \zeta^{5^j \pmod{2N}}$ for all $j$ forms a multiplicative group with order $\frac{N}{2}$ and thus $p(\zeta_j)$ for $j \in [\frac{N}{2}]$returns the first row of the matrix while $p(\zeta_j^{-1} (\text{mod} t))$ for $j \in [\frac{N}{2}]$ returns the second row of the matrix.

The left cyclic rotation by $r$ slots $LRot(\mathbf{ct}, r)$ is described as follows:

1. Pre-compute the rotation key $\mathbf{rk} \leftarrow KSGen(s, s(X^{5^r}))$
2. Let $\mathbf{ct} = (a(X), b(X))$, apply Frobenius mapping to obtain $\mathbf{ct'} = (a(X^{5^r}), b(X^{5^r}))$
3. Return $KS(\mathbf{rk}, \mathbf{ct'})$

where $KSGen(s, s(X^{5^r}))$ denotes the key-switching key generation algorithm which essentially encrypts $s(X^{5^r})$ under the secret key $s$ and $KS(\mathbf{rk}, \mathbf{ct'})$ denotes the key switching algorithm which essentially converts a message encrypted under $s(X^{5^r})$ to the same message under a different secret key $s$.

It is worth mentioning that the left rotation applies to each row of the matrix separately. It is also possible to rotate the columns,*i.e.*, swap the rows by using the following algorithm $SwapRow(\mathbf{ct})$:

1. Pre-compute the rotation key $\mathbf{rk} \leftarrow KSGen(s, s(X^{-1}))$
2. Let $\mathbf{ct} = (a(X), b(X))$, apply Frobenius mapping to obtain $\mathbf{ct'} = (a(X^{-1}), b(X^{-1}))$
3. Return $KS(\mathbf{rk}, \mathbf{ct'})$

## 3.5 Oblivious Polynomial Evaluation (OPE)

Oblivious polynomial evaluation (OPE) is a simple two-party secure computation protocol where the sender who holds a secret polynomial $p(X) = \sum_{i=0}^{N} p_i X^i$ and the receiver who holds a secret input $y$ evalutes jointly the polynomial at point $x = Y$ without leaking sender's polynomial $p(X)$ and receiver's $y$.

OPE can be implemented by BFV fully homomorphic encryption as follows. Receiver encrypts his secret input $y$ as $RLWE(y)$ and sends $RLWE(y)$ to sender. Sender can compute $RLWE(p(y))$ by applying BFV `HomoAdd` and `HomoMul` repeatedly:

$$\sum_i p_i \cdot (RLWE(y))^i = RLWE(\sum_i p_i y^i) = RLWE(p(y))$$

It is easy to prove that the BFV-based OPE protocol is semi-honest secure if the RLWE ciphertext is semantically secure. A straightfoward way to compute OPE takes $\Omega(N)$ homomorphic multiplications and the multiplication depth is $\Omega(logN)$. As aforementioned, BFV at leveled HE mode can only support limited multiplication depth and thus the straightforward method is not applicable. A practical method is to let receiver prepare $logN$ ciphertexts, *i.e.*, $RLWE(y^{2^i})$ for $i \in [logN]$ and send these $logN$ ciphertexts to sender. Sender then applies homomorphic multilication algorithm to retrieve $RLWE(y^i)$ for all $i \in [N]$. Note that this new method takes $logN$ homomorphic multiplications and consumes only $\Omega(loglogN)$ multiplication depths. The price it pays is the significant increase in the communication complexity, *i.e.* the number of ciphertexts sent from receiver to sender, from $\Omega(1)$ up to $\Omega(logN)$.

## 4 The Framework for Threshold PSI via BFV Fully Homomorphic Encryption

In this section, we describe our threshold PSI at abstract level. The threshold can be one of the three types: Below-threshold (which returns the set intersection only if the intersection cardinality is below some value), Above-threshold (which returns the set intersection only if the intersection cardinality is above some value), and In-between-threshold (which returns the set intersection only if the intersection cardinality falls within some interval). As shown in Fig. 1, the client holds a set $C = \{c_i\}_i$ and the server holds a set $S = \{s_i\}_i$. The threshold PSI is divided into two subprotocols, the threshold token generation protocol and the standard PSI protocol. In the threshold token generation protocol, an encryption of secret token $K'$ and another secret token $K$ are generated from the server side based on the encrypted intersection cardinality, *i.e.*, $RLWE(|C \cap S|)$. The client receives and decrypts the encryption of $K'$ and appends $K'$ to every item in his set $C$ such that $C$ is updated to $C' = \{c_i|K'\}_i$. Meanwhile, the server also updates his set from $S$ to $S' = \{s_i|K\}_i$. In the standard PSI protocol, the client inputs his new set $C'$ and the server inputs his new set $S'$. Then the protocol proceeds exactly as a normal PSI protocol (The Diffie-Hellman based PSI is used in the figure). At the end of the protocol, the client learns nothing more than $C \cap S$ if the threshold criterion is met. Otherwise, the client receives (with an overwhelming probability) an empty set.

To be more specific, in the threshold token generation protocol, the client first encodes his set $C$ into a Bloom filter array $\mathbf{BF_c}$ and then uses fully homomorphic encryption scheme (FHE) to encrypt $\mathbf{BF_c}$ as $RLWE(\mathbf{BF_c})$. When the server receives $RLWE(\mathbf{BF_c})$, he first performs a new primitive called `ePSI-CA` to compute homomorphically an encryption of $|C \cap S|$ as $RLWE(|C \cap S|)$. The server then performs another primitive called `SecretTokenGen` to homomorphically generate an encryption of token $K'$ based on a public known threshold criterion, an encryption of $|C \cap S|$ and a self-generated secret token $K$. The relation between $K$ and $K'$ is that $K' = K$ if the intersection cardinality satisfies the threshold criterion, otherwise, $K'$ is a completely random token. The server uses $K$ to update his set to $S' = \{s_i|K\}_i$, and also sends $RLWE(K')$ back to the client. Finally, the client decrypts $RLWE(K')$ for updating his set to $C' = \{c_i|K\}_i$. The details of `ePSI-CA` and `SecretTokenGen` will be discussed later in this paper.

**Security analysis** We use the following theorem to argue that our new threshold PSI protocol is secure against semi-honest adversaries:

**Theorem 1.** *Assuming the existence of CPA-secure fully homomorphic encryption scheme and semi-honest seucre standard PSI protocol; then the protocol n Fig.1 is secure in the presence of semi-honest adversaries.*

*Proof.* Let $P_1$ play as the receiver/client, and $P_2$ play as the sender/server. Since we use a standard PSI protocol secure against semi-honest adversaries, it suffices to prove that the threshold token generation protocol is semi-honest secure. First, **we consider the case that $P_2$ is corrupted**. Thus, we merely need to show how to simulate the view of the incoming messages received by $P_2$ by constructing a simulator called $\mathcal{S}_2$. This part is easy because $P_2$'s view contains only an FHE encryption of the client's Bloom filter $\mathbf{BF}_c[\cdot]$. $\mathcal{S}_1$ chooses a uniformly distributed message whose length equals to that of the ciphertext $RLWE(\mathbf{BF}_c[\cdot])$. This simulated message cannot be distinguished from $RLWE(\mathbf{BF}_c[\cdot])$ in the real model by a probabilistic polynomial-time-bounded adversaries due to the CPA-security of FHE ciphertexts.

Next, **we proceed to the case that $P_1$ is corrupted**, and construct a simulator $\mathcal{S}_1$. The view of $P_1$ merely contains an FHE ciphertext $RLWE(K')$ sent by $P_2$. $RLWE(K')$ is the output of a series homomorphic operations including *ePSI-CA* and *SecretTokenGen* performed on $P_2$, and thus is still CPA secure. Note that in this case, $P_1$ outputs the secret

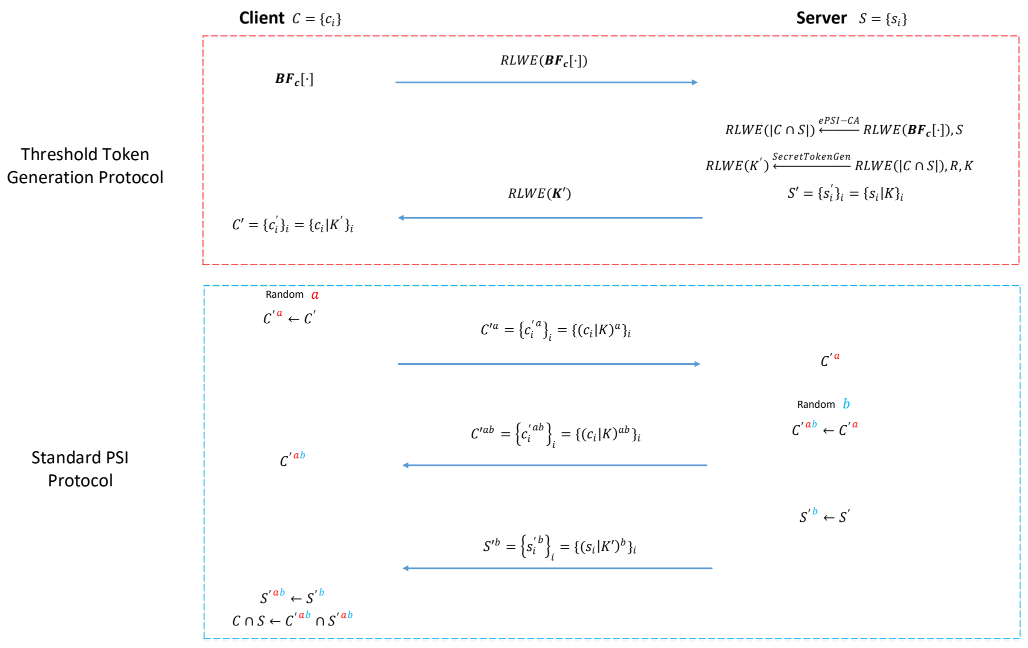
Client $C = \{c_i\}$
Server $S = \{s_i\}$
Threshold Token Generation Protocol
$BF_c[\cdot]$
$RLWE(BF_c[\cdot])$
$RLWE(|C \cap S|) \xleftarrow{ePSI-CA} RLWE(BF_c[\cdot]), S$
$RLWE(K') \xleftarrow{SecretTokenGen} RLWE(|C \cap S|), R, K$
$RLWE(K')$
$S' = \{s_i'\}_i = \{s_i|K\}_i$
$C' = \{c_i'\}_i = \{c_i|K'\}_i$
Standard PSI Protocol
Random $a$
$C'^a \leftarrow C'$
$C'^a = \{c_i'^a\}_i = \{(c_i|K)^a\}_i$
$C'^a$
Random $b$
$C'^{ab} \leftarrow C'^a$
$C'^{ab} = \{c_i'^{ab}\}_i = \{(c_i|K)^{ab}\}_i$
$C'^{ab}$
$S'^b \leftarrow S'$
$S'^b = \{s_i'^b\}_i = \{(s_i|K')^b\}_i$
$S'^{ab} \leftarrow S'^b$
$C \cap S \leftarrow C'^{ab} \cap S'^{ab}$

**Figure 1:** An abstract framework for our threshold PSI protocol

token $K'$ which is either the actual secret token $K$ or a random value $R$ depending on whether or not the threshold condition is fulfilled. Either way, $\mathcal{S}_1$ uses $P_1$'s FHE secret key to encrypt $K'$ and thus this new FHE ciphertext is indistinguishable from the incoming $RLWE(K')$ from $P_2$ in the real model. □

**Discussion** In the proof, the circuit privacy property of fully homomorphic encryption schemes must be fulfiled. In the `secretTokenGen` subprotocol, the server may interact with the client for reducing the noise in the ciphertexts. However, the noise term is related to the function that the FHE computes and the client can extract the noise term for learning information leakage in the ciphertext by decrypting it locally. To fix this circuit privacy issue, we apply the noise flooding technique, where the sender re-randomizes the output ciphertext $\mathbf{c}$ by homomorphically adding an encryption of zero with a large noise term [Gen09]. Specifically, if the error component in the output ciphertext is bounded by $B$, denoted as $|e| < B$, then the noise term in $LWE(0)$ is bounded by $\Omega(2^\lambda B)$ [AJLA+12]. By adding this large noise term, denoted as $\mathbf{c}' = \mathbf{c} + Enc(0)$, a statistical distance of $2^{-\lambda}$ (negligible) is achieved between $\mathbf{c}$ and $\mathbf{c}'$.

## 5 Challenges in the FHE-based threshold PSI

In this section, we outline the main challenges in the FHE-based threshold PSI and describe our new methods for solving these challenges.

The first challenge is to homomorphically compute an indicator $\delta$ for the membership test, *i.e.*, the server who holds the set $S$ homomorphically decides whether an element $c_i$ from the client's set $C$ belongs to $S$:

$$\delta(c_i, S) = \begin{cases} 1 & \text{if } c_i \in S \\ 0 & \text{Otherwise} \end{cases}$$

**Table 1:** Notation symbols used in this paper

| Notations | Definition |
|---|---|
| $C$ | client set |
| $c_i \in C$ | an element in the client set |
| $S$ | server set |
| $s_i \in S$ | an element in the server set |
| $\delta(c_i, S)$ | membership test function, *i.e.*, returns 1 if the input element $c_i$ belongs to set $S$, otherwise returns 0 |
| $P_\delta(X)$ | membership test polynomial, *i.e.*, $P_\delta(c_i) = 1$ if the input element $c_i$ belongs to set $S$, otherwise $P_\delta(c_i) = 0$ |
| $RLWE_s(m) \in \mathcal{R}_q \times \mathcal{R}_q$ | an RLWE encryption of the message $m \in \mathcal{R}_q$ w.r.t. secret key $s$ |
| $i \in [N]$ | a variable $i$ is choosen from the set $\{0, 1, \cdots, N-1\}$ |

The difficulty is to convert this problem to OPE while keeping the multiplicative depth low.

The second challenge is to homomorphically sum $\delta(c_i, S)$ for all $c_i$, which leads to the intersection size $|C \cap S| = \sum_{c_i \in C} \delta(c_i, S)$. Since the SIMD coding is used in FHE, it is required to manipulate the elements in the encrypted vector, *i.e.*, to sum all elements in the SIMD slot.

The third challenge is to homomorphically compare the intersection size $|C \cap S|$ with some given threshold $t$, and then to generate a secret token based on this comparison result:

$$f_{above-threshold}(|C \cap S|) = \begin{cases} K & \text{if } |C \cap S| > threshold \\ R & \text{Otherwise} \end{cases}$$

where $K$ is generated by the server for updating his own set and $R$ is a completely random token. The difficulty is the same as the first one which includes how to keep low multiplicative depth for OPE but even more difficult since the input, *i.e.*, an encryption of $|C \cap S|$ contains relative high level of noise and the noise budget left for computing $f_{above-threshold}$ is much fewer.

## 5.1 Use OPE for membership test

The first critical task in the proposed PSI protocol is how to implement the primitive $\delta$ in the membership test. Suppose that each element in the set $C$ and $S$ is represented by a 32-bit integer. We define a super set $S^{(sup)} \overset{def}{=} \{1, 2, \cdots, 2^{32} - 1\}$ over the sender's set $S$ such that $|S^{(sup)}| = 2^{32} - 1$ and $S \subset S^{(sup)}$. Note that the element 0 is excluded from $S^{(sup)}$ and reserved as a dummy item. We use Lagrange interpolation to construct a high degree polynomial $P_\delta(X) = \sum_{i=0} P_\delta X^i$ for the set $S$ with evaluation points $(x, 1)$ for $x \in S \cap S^{(sup)}$, and $(x, 0)$ for $x \in S^{(sup)} - S$. It is easily seen that the degree of $P_\delta(X)$ is exactly $2^{32} - 1$.

Note that the above membership test must be performed homomorphically. The client encrypts every element $c_i$ in his set $C$ and sends $RLWE(c_i)$ to the server. Then the server applies OPE on $RLWE(c_i)$ and $P_\delta(X)$ to compute $RLWE(P_\delta(c_i))$. A straightforward method for computing $RLWE(P_\delta(c_i))$ consumes $logN$ multiplicative depth. However, BFV supports very limited multiplicative depth of which a typical value is 8. Therefore the straightforward method is not applicable.

**Reduce the multiplicative depth** Consider a natural method which trades communication complexity for better multiplicative depth: the receiver prepares $N = 2^{32} - 1$ ciphertexts for each $c_i$ as $RLWE(c_i), RLWE(c_i^2), \cdots, RLWE(c_i^N)$ and send these $N$ ciphertexts to server. The server can compute $RLWE(P_\delta(X))$ in depth one. However, the price is that the communication complexity increases by $N$ times. A more balanced method called windowing, is as follows: the receiver prepares $\lfloor log_2 N \rfloor + 1$ ciphertexts including $RLWE(c_i), \cdots, RLWE(c_i^j), \cdots, RLWE(c_i^{2^{\lfloor log_2 N \rfloor}})$. Now the sender can compute $RLWE(P_\delta(X))$ in depth $\lceil log(\lfloor logN \rfloor + 1) \rceil \approx loglogN$, and the price it pays is the communication complexity increased by $\lfloor logN \rfloor + 1$ times. Note that the SIMD encoding is used, therefore, the OPE is batched and the final ciphertext encrypts a vector as $\{RLWE(P_\delta(c_i))\}_i$ for $i \in [N]$. Note the noise in the ciphertext $RLWE(P_\delta(X))$ after OPE is relatively large, denoted as $Error(RLWE(P_\delta(\{c_i\})) \sim \mathcal{N}(0, \sigma_\delta^2)$.

**Reduce the number of ciphertext-ciphertext homomorphic multiplications** It is worth noting that the ciphertext-ciphertext multiplication is computationally more complex than the ciphertext-plaintext multiplication due to the relineariztion procedure. In fact, ciphertext-plaintext multiplication consumes only 2 multiplications over $R_q$ whereas ciphertext-ciphertext multiplication consumes about $4 + \Omega(\log_2 q)$ multiplications over $R_q$, which generally indicates that ciphertext-ciphertext multiplication is at least two orders of magnitude slower than ciphertext-plaintext multiplication. To conclude, the computation of OPE is bottlenecked by the number of ciphertext-ciphertext homomorphic multiplications used for homomorphically computing all powers of $X$ and it is reasonable to bound the computation overhead of membership test by the number of homomorphic multiplications occuring in the OPE protocol. Recall that if the windowing method which is used to balance between the multiplicative depth and the communication overhead, the number of ctx-ctx multiplications is upper-bounded by $B - \lfloor \log_2 B \rfloor \approx B$.

In practice, a better method we use in our software implementations for computing the OPE protocol is by applying the Paterson-Stockmeyer algorithm. In this case, $P_\delta(Enc(X))$ is rewritten as:

$$P_\delta(Enc(X)) = \sum_{i=0}^{B} P_{\delta,i} \cdot (Enc(X))^i = \sum_{i=0}^{H-1} \left( \sum_{j=0}^{L-1} P_{\delta,iL+j} \cdot Enc(X)^j \right) \cdot Enc(X)^{iL}$$

where $H \cdot L \geq B + 1$ and $H \approx L = \Omega(\sqrt{B})$. The sender first pre-computes $Enc(X)^j$ for all $j \in [L]$ and $Enc(X)^{iL}$ for all $i \in [H]$ from $Enc(X)^{2^m}$ for $m \in [\lceil log_2 L \rceil]$ and $Enc(X)^{2^n L}$ for $n \in [\lceil log_2 H \rceil]$ sent by the receiver, which consumes precisely $L + H - \lceil log_2 L \rceil - \lceil log_2 H \rceil - 2$ ciphertext-ciphertext multiplications and consumes another $H - 1$ ciphertext-ciphertext multiplications to evaluate $P_\delta(Enc(X))$. To summarize, the OPE protocol consumes about $L + 2H - \lceil log_2 H \rceil - \lceil log_2 L \rceil - 3 \approx L + \frac{2B}{L}$ ciphertext-ciphertext multiplications in total.

**Reduce the degree of $P_\delta(X)$** Consider using the classic hash-to-bins technique to further reduce the homomorphic evaluation of $P_\delta(X)$ for all $c_i \in C$. Assume that each element in the set $C$ and $S$ is represented by a 32-bit integer and we reasonably set $|S^{(sup)}| = N^{(sup)} = 2^{32} - 1, |C| = n$ where those elements $s.t.$ $s_i^{(sup)} \in S$ are called valid balls and those elements $s_i^{sup} \notin S$ are called invalid balls. The receiver invokes Cuckoo Hashing (with two hash functions $h_0, h_1$) to hash his set elements $\{c_i\}_i$ to $\Omega(n)$ bins (we prepare $1.5n$ bins in practice to achieve a negligible hash collision probability) where each bins has at most one element. The sender invokes simple hashing (with the same hash functions $h_0, h_1$) to hash his set elements $\{s_i^{(sup)}\}_i$ to $\Omega(n)$ bins (In practice, the number of bins is also set to $1.5n$) where each bin has up to $\frac{N^{(sup)}}{n} + \mathcal{O}(\sqrt{\frac{N^{(sup)} \cdot logn}{n}})$ elements with high probability. Now the sender and receiver only need to perform bin-wise PSI

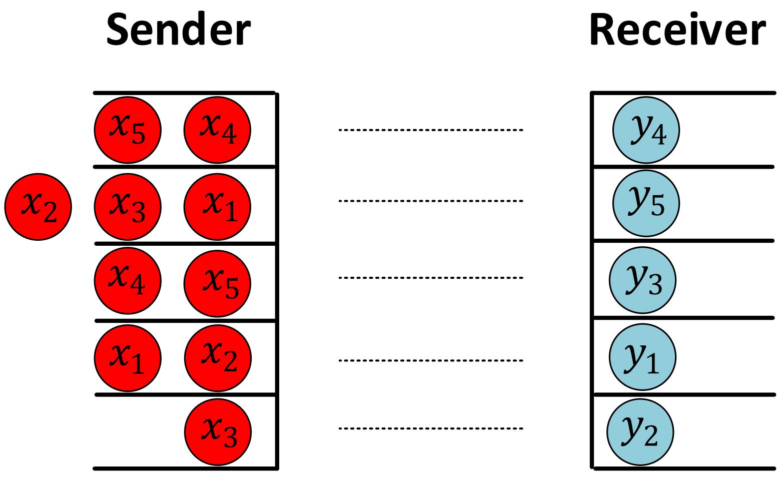

**Figure 2:** $P_\delta(X)$ by hash-to-bins technique

for each bin, in other words, the polynomial associated with each of the sender's bin has degree of $\frac{N^{(sup)}}{n} + \mathcal{O}(\sqrt{\frac{N^{(sup)} \cdot logn}{n}})$.

An illustrative toy example is shown in Fig. 2 where $|S^{(sup)}| = N^{(sup)} = 5, |C| = n = 5$ and the number of bins is set to 5. The sender uses simple hashing to hash his set elements $x_1, \cdots, x_5$ to 5 bins: for example, $x_1$ is inserted into bin#2 and bin#4 since $h_0(x_1) = 2, h_1(x_1) = 4$. The receiver uses Cuckoo hashing to hash his set elements $y_1, \cdots, y_5$ to 5 bins without hash collision (with a high probability). After hashing, the sender and the receiver performs the bin-wise comparison to check whether $y_i$ belongs to set $S$: for example, the receiver compares his bin#1 which contains $y_4$ with the sender's bin#1 which contains $x_4, x_5$, and so on so forth.

**Batching the computation of OPE** Consider exploiting the SIMD encoding to parallelize the computation of OPE. Note that in practice, the elements in the sender's bins are represented by membership test polynomial denoted as $P_{\delta_i}(X)$ for bin#$i$. The basic idea goes as follows: the receiver batches $N$ items (in $N$ distinct bins) into one ciphertext sent to the sender. The sender batches $N$ coefficients from his membership test polynomials $\{P_{\delta_i}(X)\}_i$ (one coefficient from each membership test polynomial) into one plaintext, and therefore $\frac{N^{(sup)}}{n} + \mathcal{O}(\sqrt{\frac{N^{(sup)} \cdot logn}{n}})$ such plaintexts suffice representing $N$ distinct membership test polynomials. Finally, the sender performs OPE on the ciphertext sent by the receiver and $\frac{N^{(sup)}}{n} + \mathcal{O}(\sqrt{\frac{N^{(sup)} \cdot logn}{n}})$ plaintexts of his own to obtain an encryption of $N$ membership test results.

An illustrative toy example for batching OPE is shown in Fig. 3 with the same configuration from Fig. 2. For security reasons, the degree for each membership test polynomial $P_{\delta_i}(X)$ assoicated to bin#$i$ on sender's side must be equal and therefore, these bins are inserted by dummy items. For example, sender's bin#1 has $x_5, x_4$ and a direct Lagrange interpolation yields a polynomial $P^*_{\delta_1}(X)$ with degree 2. The sender inserts a dummy item, *i.e.*, $X = 0$ and makes $P_{\delta_1}(X) = X \cdot P^*_{\delta_i}(X)$ with degree 3. The sender repeats the interpolation polynomial construction process for all bins to create $P_{\delta_i}(X) = \sum_{j=0}^{3} P_{\delta_i,j} X^j$ for all $i \in \{1,2,3,4,5\}$. Then, the sender batches $P_{\delta_i,j}$ for all $i$, *i.e.*, the $j$-th coefficient of the polynomial $P_{\delta_i}(X)$ for all $i$ into one plaintext denoted as $pt^{(j)}(X)$. It is easily seen that 3 such plaintexts batch all the coefficients $P_{\delta_i,j}$ for all $i$ and all $j$. Finally, the sender inputs the ciphertext **ct** which encrypts $\mathbf{y} = (y_4, y_5, y_3, y_1, y_2)$ sent by the receiver, and his own plaintexts $pt^{(j)}(X)$ for $j \in \{1,2,3\}$ for performing an

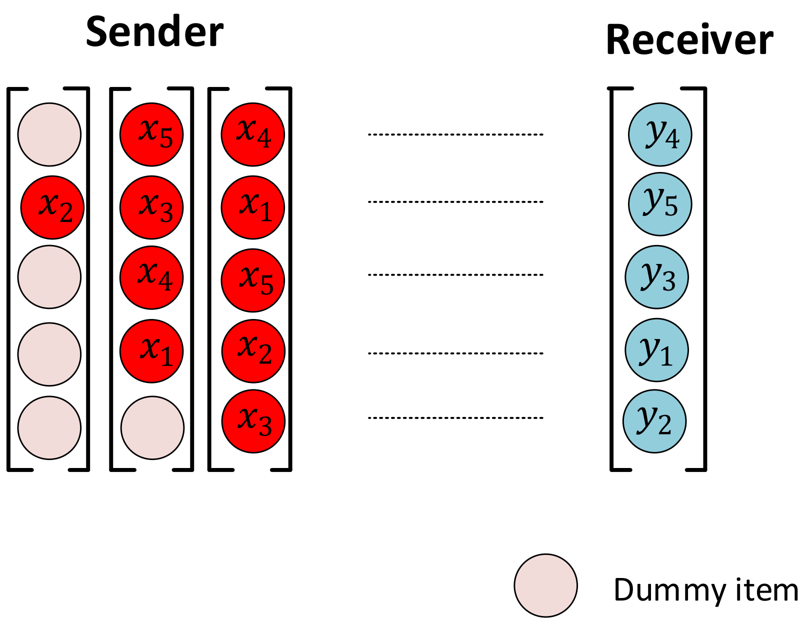

**Figure 3:** Batching the computation of $P_\delta(X)$ together with the hash-to-bins technique

OPE protocol to batch the computation of $\mathbf{P}_\delta(\mathbf{y})$ where $\mathbf{P}_\delta(\cdot) = \{P_{\delta_1}(\cdot), \cdots, P_{\delta_5}(\cdot)\}$.

**Computation overhead** We set the size of the hash table (number of bins used) equal to the FHE parameter $N$. We use $h = 3$ hash functions for simple hashing for the sender and Cuckoo hashing scheme for the receiver. Therefore, the maximum load in the bin for the server' set satisfies $B \approx h \cdot |S^{(sup)}|/N$, and the maximum load in the bin for the receiver is strictly 1. Since we use batching, one OPE operation suffices to complete the membership test. The OPE operation is implemented by the Paterson-Stockmeyer algorithm. This algorithm requires precisely $L + 2H - \lceil log_2 L \rceil - \lceil log_2 H \rceil - 3$ ctx-ctx multiplications and $B - 1$ ptx-ctx multiplications where we set $L = 65, H = \lceil B/L \rceil = \lceil 1024/65 \rceil = 16$ for easier implementations.

**Noise growth** If the windowing method and Paterson's method are applied, the critical path for the OPE-based membership consists of two parts. The first part is for the preprocessing for $Enc(x^j)$ for $j \in [L]$ and $Enc(x^{iL})$ for $i \in [H]$ from $Enc(x), Enc(x^2), \cdots, Enc(x^{2^{\log_2 L}})$ and $Enc(x^L), Enc(x^{2L}), \cdots, Enc(x^{2^{\log_2 H} L})$. Then it is easy to see that it consumes $\lceil \log_2 \lceil \log_2 (max(H, L)) \rceil \rceil$ layers of multiplicative depth.

The second part is for the Paterson-Stockmeyer method where one layer of ptx-ctx multiplication and one layer of ctx-ctx multiplication are consumes. To summarize, the total multiplicative depth used in the OPE protocol is bounded by $\lceil \log_2 \lceil \log_2 (max(H, L)) \rceil \rceil + 2$. The noise bound for the result of OPE is estimated as:

$$||e_{\mathtt{OPE}}||^{can} \le \left(\frac{32\sqrt{Nt}}{\sqrt{12}}\right)^D \cdot \frac{16\sqrt{Nt}}{\sqrt{12}} \cdot ||e_0||^{can} + \frac{32}{6} Nt^2 \cdot \frac{\left(\frac{32\sqrt{Nt}}{\sqrt{12}}\right)^D - 1}{\frac{32\sqrt{Nt}}{\sqrt{12}} - 1} \cdot \frac{16\sqrt{Nt}}{\sqrt{12}} + \frac{16}{12} Nt^2$$

where the initial noise in the fresh BFV ciphertext is bounded by $||e_0||^{can} \le \frac{Nt}{2}$. The estimation above exploits the general term formula $a_n = k^n \cdot a_0 + b \cdot \frac{k^n - 1}{k - 1}$ for the linear recursive relation $a_n = k \cdot a_{n-1} + b$.

If the Paterson-Stockmeyer method is applied, the critical path is changed to $D = \lceil \log_2 (\lfloor \log_2 B \rfloor + 1) \rceil \approx \log_2 \log_2 B$ ciphertext-ciphertext multiplication depth for computing

$Enc(X)^j$ and $Enc(X)^{iL}$ plus 1 plaintext-ciphertxt multiplicative depth and 1 ciphertxt-ciphertxt multiplicative depth. Therefore, the noise bound is estimated as:

$$||e_{\mathtt{OPE}}||^{can} \leq \frac{32\sqrt{N}t}{\sqrt{12}} \left( (\frac{32\sqrt{N}t}{\sqrt{12}})^D \cdot \frac{16\sqrt{N}t}{\sqrt{12}} \cdot ||e_0||^{can} + \frac{32}{6}Nt^2 \cdot \frac{(\frac{32\sqrt{N}t}{\sqrt{12}})^D - 1}{\frac{32\sqrt{N}t}{\sqrt{12}} - 1} \cdot \frac{16\sqrt{N}t}{\sqrt{12}} + \frac{16}{12}Nt^2 \right) + \frac{32}{12}Nt^2$$

## 5.2 Homomorphic Accumulation

In the previous subsection, $\frac{|C|}{N}$ ciphertexts are computed where each ciphertex encrypts a binary vector. The next target is to (homomorphically) accumulate all ciphertexts. This accumulation is performed in two phases.

In the first phase, the normal homomorphic addition is performed as $\sum_{i=0}^{\frac{|C|}{N}-1} RLWE((P_\delta(c_{iN}), \cdots, P_\delta(c_{iN+N-1})$ which eventually returns a single ciphertext denoted as $RLWE(P_{\delta,0}, \cdots, P_{\delta,N-1})$.

**Method-I for Phase Two** There exsit two methods to implement the second phase. We first introduce the first method as shown in the total sum algorithm (see Alg. 1), which exploits the homomorphic cyclic rotation to accumulate all the elements in the encrypted vector $(P_{\delta,0}, \cdots, P_{\delta,N-1})$, and eventually returns $RLWE(\sum_{i=0}^{N-1})$ where $\sum_{i=0}^{N-1} P_{\delta,i} = |C \cap S|$.

```
  Input: RLWE ciphertext $\mathbf{ct}$ which encrypts a vector $\mathbf{v} = (v_0, v_1, \cdots, v_{N-1})$
  Output: RLWE ciphertext $\mathbf{ct}'$ which encrypts a vector $\mathbf{v}' = (\sum_i v_i, \cdots, \sum_i v_i)$
1 $\mathbf{ct}' \leftarrow \mathbf{ct}, e \leftarrow 1$
2 for $j \leftarrow numBits(N) - 2$ down to $0$ do
3     $\mathbf{ct}' \leftarrow \mathbf{ct}' + LeftRotate(\mathbf{ct}', -e)$
4     $e \leftarrow 2 \cdot e$
5     if $bit_j(N) == 1$ then
6         $\mathbf{ct}' \leftarrow \mathbf{ct} + LeftRotate(\mathbf{ct}', -1)$
7         $e \leftarrow e + 1$
8 return $\mathbf{ct}'$
```

**Algorithm 1:** `Total_Sum` algorithm which accumulates every element in the encrypted vector

The function $numBits(\cdot)$ in step 2, Alg. 1 denotes the number of bits used to represent the input integer. Alg. 1 is highly efficient regarding the computational complexity. Assume that $N$ is $2^n - 1$ for some $n$ which is exactly the worst case since the if condition (step 10-step 12) is always performed. In this case, it is readily seen the number of homomorphic rotations required in Alg. 1 is:

$$2(logN - 1) = \Omega(logN)$$

Nevertheless, the problem with `Total_Sum` described in Alg. 1 is that the storage for rotation key might be huge: $log_2 N$ rotation keys are needed to fulfil each rotation mentioned in step 3. To solve this problem, a storage efficient version of `Total_Sum` is given in Alg. 2. The basic idea is to factor the right rotation by $e$ slots into a combination of at most $k$ types of right rotaions. In this case, we choose $k < log_2 N$ rotation keys which correspond to right rotation by $2^{\ell_i}$ slots for $i \in [k]$ to save storage for rotation keys. A natural selection for $\{\ell_i\}_{i \in [k]}$ is that $\{\ell_i\}_{i \in [k]}$ evenly divides the interval $[0, 1, \cdots, log_2 N]$. For easy of analysis, we assume that $N$ is $2^n - 1$ for some $n$ which is exactly the worst case since $e$ starts with $1_2$ and grows up $11_2, 111_2 \cdots$. For $e$ falls in the interval-$i$ with

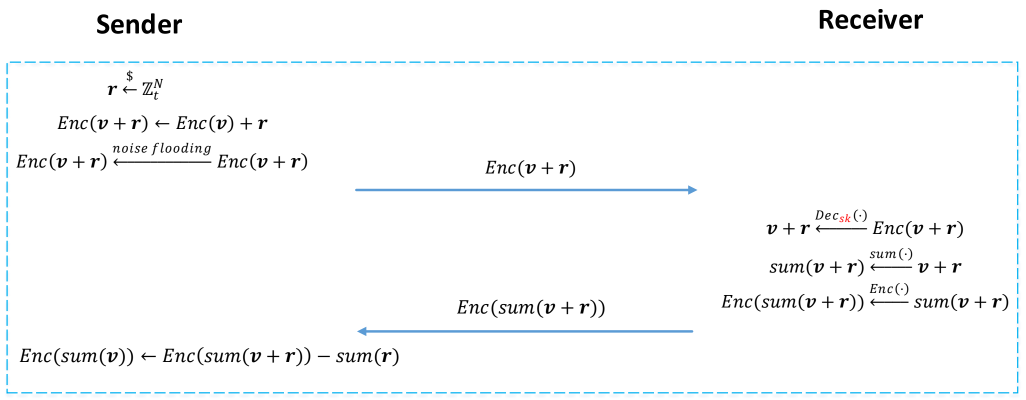
Sender
Receiver
$r \xleftarrow{\$} \mathbb{Z}_t^N$
$Enc(v+r) \leftarrow Enc(v)+r$
$Enc(v+r) \xleftarrow{noise\ flooding} Enc(v+r)$
$Enc(v+r)$
$v+r \xleftarrow{Dec_{sk}(\cdot)} Enc(v+r)$
$sum(v+r) \xleftarrow{sum(\cdot)} v+r$
$Enc(sum(v+r)) \xleftarrow{Enc(\cdot)} sum(v+r)$
$Enc(sum(v+r))$
$Enc(sum(v)) \leftarrow Enc(sum(v+r)) - sum(r)$

**Figure 4:** A fast homomorphic accumulation for all elements in an encrypted vector by interaction

$[2^{i \cdot logN/k}, 2^{(i+1) \cdot logN/k} - 1]$ and $i \in [k]$, the number of cyclic rotations is:

$$(2^{\frac{logN}{k}} - 1)i + \sum_{j=1}^{\frac{logN}{k}} (2^j - 1) = (2^{\frac{logN}{k}} - 1)i + 2^{\frac{logN}{k}+1} - \frac{logN}{k} - 2$$

Therefore, the total number of cyclic rotations (in the worst case) is:

$$(2^{\frac{logN}{k}-1} \cdot \frac{k-1}{2} \cdot k) + 2^{\frac{logN}{k}+1} \cdot k - (\frac{logN}{k} + 2)k = \Omega(k^2 2^{\frac{logN}{k}})$$

```
   Input: RLWE ciphertext $\mathbf{ct}$ which encrypts a vector $\mathbf{v} = (v_0, v_1, \cdots, v_{N-1})$
   Output: RLWE ciphertext $\mathbf{ct}'$ which encrypts a vector $\mathbf{v}' = (\sum_i v_i, \cdots, \sum_i v_i)$
1  $\mathbf{ct}' \leftarrow \mathbf{ct}, e \leftarrow 1$
2  for $j \leftarrow numBits(N) - 2$ down to $0$ do
3      rewrite $e$ w.r.t the base $\{2^{\ell_1}, \cdots, 2^{\ell_k}\}$ such that $e = \sum_i e_i \cdot 2^{\ell_i}$
4      $\mathbf{ct}'' \leftarrow \mathbf{ct}'$
5      for $i \leftarrow k$ down to $1$ do
6          for $j \leftarrow e_i - 1$ down to $0$ do
7              $\mathbf{ct}'' \leftarrow LeftRotate(\mathbf{ct}'', -2^{\ell_i})$
8      $\mathbf{ct}' \leftarrow \mathbf{ct}' + \mathbf{ct}''$
9      $e \leftarrow 2 \cdot e$
10     if $bit_j(N) == 1$ then
11         $\mathbf{ct}' \leftarrow \mathbf{ct} + LeftRotate(\mathbf{ct}', -1)$
12         $e \leftarrow e + 1$
13 return $\mathbf{ct}'$
```

**Algorithm 2:** `Total_Sum` algorithm (storage efficient version) which accumulates every element in the encrypted vector

**Method-II for Phase Two** Method-I for homomorphic accumulation of the encrypted vector extensively exploits the cyclic rotation which bottlenecks the computational performance of the entire threshold PSI protocol, though it avoids the interaction between the sender and the receiver completely. Here we introduce a more practical method as shown in Fig. 4 to solve this computational inefficiency by only one round of light communication between the sender and the receiver.

In Fig. 4, the sender randomizes the encryption of a sensitive and secret vector $\mathbf{v} \overset{def}{=} (P_{\delta,0}, \cdots, P_{\delta,N-1})$ by adding a random vector $\mathbf{r}$ to it for creating a new ciphertext $Enc(\mathbf{v}+\mathbf{r})$. It is worth noting that the noise term of the input ciphertext $Enc(\mathbf{v})$ may contain extra sensitive information of the sender's set and thus an operation of noise flooding is necessary to hide this undesirably leakage before being sent to the receiver. The receiver locally decrypts $Enc(\mathbf{v}+\mathbf{r})$ to $\mathbf{v}+\mathbf{r}$ and sums all elements in the vector $\mathbf{v}+\mathbf{r}$ to obtain $sum(\mathbf{v}+\mathbf{r})$. The receiver re-encrypts $sum(\mathbf{v}+\mathbf{r})$ and sends this ciphertext to the sender. Finally, the sender de-randomizes $Enc(sum(\mathbf{v}+\mathbf{r}))$ by subtracting $\mathbf{r}$ to obtain $Enc(sum(\mathbf{v}))$ without knowing the exact $sum(\mathbf{v}) \overset{def}{=} \sum_i P_{\delta,i}$.

Regarding the performance, the sender performs two homomorphic additions and one noise flooding, and the receiver performs one encryption, one decryption and one sum of all elements in the plaintext, the computational overheads of which are almost negligible if compared with homomorophic cyclic rotations used in method-I. The price that this new method pays is an extra round of communication includes two BFV ciphertexts, which is about a few megabytes.

## 5.3 Homomorphic Comparison

We again apply OPE for the specific homomorphic comparison. Differing from the normal comparison functionality, our purpose is to output a secret token by checking whether the input is smaller or bigger than a threshold value. We denote this new functionality as `SecretTokenGen`. For example, for the above-threshold criteria, *i.e.*, it outputs a valid secret token $K$ if and only if the cardinality is above some given threshold $t$, the polynomial we construct has the following form:

$$f_{th}(X) = r \cdot \prod_{i=t}^{N} (X-i) + K$$

where $r$ is a random number and $k$ is the target secret token. It is seen that the multiplicative depth for computing $f_{th}(X)$ is $loglog(N-t+1)$. Likewise, for the below-threshold criteria, *i.e.*, it outputs a valid secret token $k$ if and only if the cardinality is below some given threshold $t$, the polynomial can be constructed as follows:

$$f_{th}(X) = r \cdot \prod_{i=0}^{t} (X-i) + K$$

The challenage part is that the initial noise in the ciphertext $RLWE(|C \cap S|)$ is too large to further evaluate $f_{th}(X)$. Bootstrapping is a theoretic approach to reduce the noise level but in practice we should avoid using this method. We propose using the following method to fast reduce the noise in $RLWE(|C \cap S|)$ at the price of another round of communication between the client and server.

The basic idea is essentially a client-assisted bootstrapping where the server resends the ciphertext $RLWE(|C \cap S|)$ back to the client who can decrypt $RLWE(|C \cap S|)$ and then re-encrypt $|C \cap S|$. The new ciphertext $RLWE(|C \cap S|)$ has the minimum amount of noise since it is newly generated. However, the client learns the exact value of $|C \cap S|$ when he decrypts $RLWE(|C \cap S|)$. This extra learned knowledge viloates the security definition of threshold PSI. Therefore, the server must randomize the ciphertext before sending it to the client. The client receives the randomized message which leaks no information to him. More concretely, the sender performs $RLWE(|C \cap S| + r') = RLWE(|C \cap S|) + (0, r')$ and send it to receiver. The receiver re-encrypts it to have a noise-less ciphertext $RLWE(|C \cap S| + r')$.

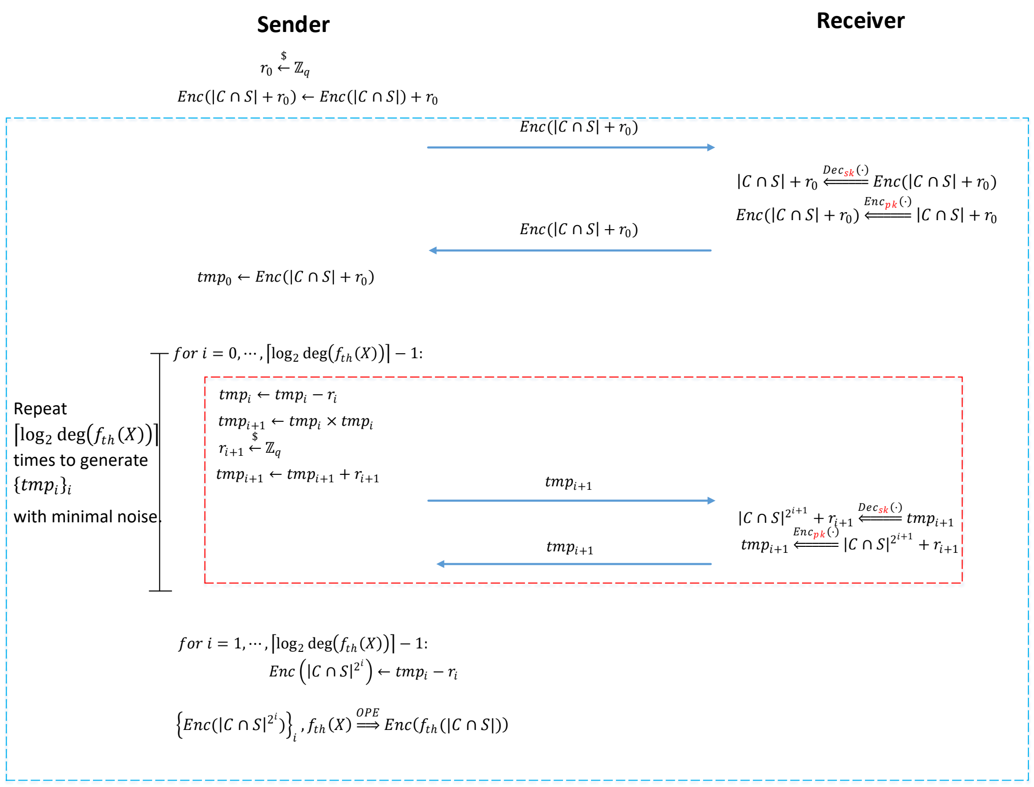
Sender
Receiver
$r_0 \xleftarrow{\$} \mathbb{Z}_q$
$Enc(|C \cap S| + r_0) \leftarrow Enc(|C \cap S|) + r_0$
$Enc(|C \cap S| + r_0)$
$|C \cap S| + r_0 \xLeftarrow{Dec_{sk}(\cdot)} Enc(|C \cap S| + r_0)$
$Enc(|C \cap S| + r_0) \xLeftarrow{Enc_{pk}(\cdot)} |C \cap S| + r_0$
$Enc(|C \cap S| + r_0)$
$tmp_0 \leftarrow Enc(|C \cap S| + r_0)$
$for\ i = 0, \cdots, \lceil \log_2 \deg(f_{th}(X)) \rceil - 1:$
Repeat $\lceil \log_2 \deg(f_{th}(X)) \rceil$ times to generate $\{tmp_i\}_i$ with minimal noise
$tmp_i \leftarrow tmp_i - r_i$
$tmp_{i+1} \leftarrow tmp_i \times tmp_i$
$r_{i+1} \xleftarrow{\$} \mathbb{Z}_q$
$tmp_{i+1} \leftarrow tmp_{i+1} + r_{i+1}$
$tmp_{i+1}$
$|C \cap S|^{2^{i+1}} + r_{i+1} \xLeftarrow{Dec_{sk}(\cdot)} tmp_{i+1}$
$tmp_{i+1} \xLeftarrow{Enc_{pk}(\cdot)} |C \cap S|^{2^{i+1}} + r_{i+1}$
$tmp_{i+1}$
$for\ i = 1, \cdots, \lceil \log_2 \deg(f_{th}(X)) \rceil - 1:$
$Enc\left(|C \cap S|^{2^i}\right) \leftarrow tmp_i - r_i$
$\left\{Enc(|C \cap S|^{2^i})\right\}_i, f_{th}(X) \overset{OPE}{\Longrightarrow} Enc(f_{th}(|C \cap S|))$

**Figure 5:** A client-assisted bootstrap framework for our secret token generation subprotocol `SecretTokenGen`

Finally the sender de-randomize $RLWE(|C \cap S| + r')$ by performing $RLWE(|C \cap S|) = RLWE(|C \cap S| + r') - (0, r')$.

Fig. 5 describes in detail the process for executing the `secretTokenGen` subprotocol where server inputs an RLWE encryption of the intersection cardinality, *i.e.*, $Enc(|C \cap S|)$ and finally outputs a RLWE encryption of $f_{th}(|C \cap S|)$ after $\lceil log_2(\texttt{deg}(f_{th}(X))) \rceil + 1$ rounds of interactions with the client. In the first round of interaction, the sender reduces the large noise in $Enc(|C \cap S|)$ to much smaller one by exploiting the client-assisted bootstrapping technique described above. Then in the next $\lceil log_2(\texttt{deg}(f_{th}(X))) \rceil$ rounds, the sender locally generates $Enc(|C \cap S|^{2^j})$ for $j = 0, 1, \cdots, \lceil log_2(\texttt{deg}(f_{th}(X))) \rceil - 1$ from $Enc(|C \cap S|)$ by performing homomorphic multiplications. Note that the noise increased by the operation of homomorohic multiplications is reduced to minimal by the client-asisted bootstrapping. Finally, these noise-suppressed $Enc(|C \cap S|^{2^j})$ (for all $j$) are utilized in the next `OPE` protocol (described previously in 'Reduce multiplicative depth' paragraph in Section 5.1) to generate $Enc(f_{th}(|C \cap S|))$. Assume the threshold $t = \mathcal{O}(N)$, and it is readily seen that the communication overhead for `secretTokenGen` subprotocol is $2 + 2\lceil log_2\texttt{deg}(f_{th}(X)) \rceil = \mathcal{O}(logN)$ RLWE ciphertexts where the sender transmits $1 + \lceil log_2\texttt{deg}(f_{th}(X)) \rceil$ ciphertexts to the receiver and vice versa.

**Computation overhead** Analogous to the computation overhead for the OPE-based membership, the homomorphic comparison is also bottlenecked by the number of ciphertext-ciphertext homomorphic multiplications used. Let $\texttt{deg}(f_{th}(X)) = \varepsilon N$ with $0 < \varepsilon < 1$. If the combination of the windowing method and the Paterson-Stockmeyer algorithm is applied, the number of ctx-ctx multiplications is reduced to $L + 2H - 3$ where $L = 65, H = \lceil \frac{\texttt{deg}(f_{th}(X))}{L} \rceil$.

### 5.4 Put all pieces together

We formally describe the BFV-based threshold PSI protocol in Alg. 3. In line 1, the client initializes the FHE scheme and then shares it with the server. In line 2, the client encrypt his own set $C$. In line 3, the server receives the encrypted client's set and then performs the `ePSI-CA` primitive to obtain an RLWE encryption of $|C \cap S|$ as $RLWE(|C \cap S|)$. In line 4, the server continues to generate the secret token $K'$ homomorphically based on the threshold criterion and $RLWE(|C \cap S|)$ an also augments his set as $S^K = \{s_i|K\}_i$ with a random and secret token $K$. In line 5, the client augments his set as $C^{K'} = \{c_i|K'\}_i$ with the decrypted token $K'$. In line 6, a standard PSI between the set $S^K$ and $C^{K'}$ is performed which returns the intersection $C \cap S$ if $|C \cap S| > t$.

```
Input: On the client side: a set $C$, the threshold value $t$; On the server side: a set $S$
Output: The client side receives $C \cap S$ if $|C \cap S|$ satisfies the threshold requirement
1 Initialise system parameters for BFV, i.e. $RLWE_s^{n,q}(\cdot)$, relinearization key
    relinKey and rotation key rotKey
2 The client encrypts his set (in SIMD manner) as
    $RLWE(\mathbf{c}_i) = RLWE(c_{i\cdot N}, c_{i\cdot N+1}, \cdots, c_{i\cdot N+N-1})$ for all $c_{N\cdot i+j} \in C$ to the server.
3 The server runs the ePSI-CA primitive to obtain $RLWE_s(|C \cap S|)$
4 The server runs the SecretTokenGen primitive to obtain an encrypted token
    $RLWE(K')$ and an augmented set $S^K = \{s_i|K\}_i$
5 The client receives and then decrypts $RLWE(K')$ to obtain $K'$. The client
    appends $K'$ to every element in his set $C$ as $C^{K'} = \{c_i|K'\}_i$
6 Finally, the client and server engages in a normal PSI protocol with set $S^K$ and set
    $C^{K'}$
7 return $C^{K'} \cap S^K$
```

**Algorithm 3:** Framework for threshold PSI via FHE

We implement Alg. 3 in C++, based on SEAL and APSI lib. In particular, the primitive `ePSI-CA` in step 3 consists of two algorithms: 1. the batched OPE using Pateerson-Stockmeyer algorithm 2. the homomorphic accumulation where phase two is implemented by method-II shown in Fig. 4. The primitive `SecretTokenGen` in step 4 is essentially the OPE-based homomorphic comparison variant shown in Fig. 5.

## 6 Experimental Results and Comparisons

**Table 2:** Details of Computation and communication cost of our threshold PSI implementation

| \|X\| | \|Y\| | sender offline | | | sender online | | | | comm.(MB) | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | total | `interpolate_set` | `cmp_poly` | total | `membership_test` | `homo_add` | `homo_cmp` | $C \rightarrow S$ | $S \rightarrow C$ |
| $2^{20}$ | 4096 | | | | | | | | | |
| | 2048 | | | | | | | | | |
| | 1024 | | | | | | | | | |
| | 1 | | | | | | | | | |

Our works extends the FHE-based standard PSI [CLR17, CMdG+21] to PSI-cardinality and threshold PSI and thus we first compare with theirs. Compared with [CLR17,

**Table 3:** Comparisons between our threshold PSI implementation and others

| \|X\| | \|Y\| | Protocol | sender offline | sender online | comm.(MB) |
|---|---|---|---|---|---|
| $2^{10}$ | $2^{10}$ | ours, PSI-cardinality | | | |
| $2^{12}$ | $2^{12}$ | ours, tPSI | | | |
| $2^{12}$ | $2^{12}$ | [JTKA22], PSI | — | 7.396s | 141 |
| $2^{20}$ | 5535 | ours, tPSI | | | |
| | | [CLR17], PSI | 43s | 4.23s | 11.50 |
| | | [CMdG+21], PSI | 28s | 3.23s | 5.39 |
| | 11041 | ours, tPSI | | | |
| | | [CLR17], PSI | 43s | 4.47s | 14.34 |
| | | [CMdG+21], PSI | 29s | 4.23s | 8.94 |
| $2^{20}$ | 1024 | ours, PSI-cardinality | | | |
| $\approx 2^{22}$ | 1120 | [TSS+20], PSI-cardinality | — | 35.2s | 126.7 |
| $2^{20}$ | $2^{20}$ | [IKN+20], PSI-sum-cardinality | — | 776.4s | 84 |
| $2^{20}$ | $2^{20}$ | [MPR+20], PSI-sum-cardinality | — | 35583s | 436.7 |
| $2^{20}$ | $2^{20}$ | [PSWW18], PSI-cardinality-threshold | — | 86.6s | 6950.6 |

CMdG+21], our works realizes computations over private set intersection with acceptable commputation and communication overhead.

[JTKA22] proposes a new idea of constructing PSI and its variant protocols by homomorphically evaluating a branching program. Compared with [JTKA22], our works is advantageous in running time when the server'set is relatively large.

[IKN+20, MPR+20] study a particular PSI variant called private intersection-sum with cardinality (`PSI-sum-cardinality`). Compared with theirs, it is easy to modify our design to implement `PSI-sum-cardinality` with little performance penalty.

[TSS+20] proposes building PSI-cardinality protocol based on keyword PIR. [TSS+20] does not implement their protocol but provides performance estimates. Compared with [TSS+20] which uses the single-server PIR, our performance is more or less the same.

**Theorem 2.** *Let* $||e_0||^{can}$ *and* $||e_1||^{can}$ *denote the canonical norm of the noise in* $\mathbf{ct}_0$ *and* $\mathbf{ct}_1$*, respectively. The noise after a ciphertext-ciphertext homomorphic multiplication* $\mathbf{ct}_0 \cdot \mathbf{ct}_1$ *is upper-bounded by* $16(\frac{\sqrt{N}t||e_0||^{can}}{\sqrt{12}} + \frac{\sqrt{N}t||e_1||^{can}}{\sqrt{12}}) + \frac{32Nt^2}{12}$

**Theorem 3.** *Let* $||e_1||^{can}$ *denote the canonical norm of the noise in* $\mathbf{ct}_0$ *and* $\mathbf{ct}_1$ *and* $m_1(X) \in R_t$ *denotes a plaintext polynomial, respectively. The noise after a ciphertext-plaintext homomorphic multiplication* $\mathbf{ct}_0 \cdot m_1(X)$ *is upper-bounded by* $16(\frac{\sqrt{N}t||e_0||^{can}}{\sqrt{12}}) + \frac{16Nt^2}{12}$

**Theorem 4.** *Let* $||e_0||^{can}$ *and* $||e_1||^{can}$ *denote the canonical norm of the noise in* $\mathbf{ct}_0$ *and* $\mathbf{ct}_1$*, respectively. The noise after a ciphertext-ciphertext homomorphic addition* $\mathbf{ct}_0 + \mathbf{ct}_1$ *is upper-bounded by* $||e_0||^{can} + ||e_1||^{can} + 12 \cdot \frac{\sqrt{N}t}{\sqrt{12}}$

## References

[ACT11] Giuseppe Ateniese, Emiliano De Cristofaro, and Gene Tsudik. (if) size matters: size-hiding private set intersection. In *International Workshop on Public Key Cryptography*, pages 156–173. Springer, 2011.

[AJLA+12] 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 *Advances in Cryptology–EUROCRYPT 2012: 31st Annual International Conference on* *the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings 31*, pages 483–501. Springer, 2012.

[BMRR21] Saikrishna Badrinarayanan, Peihan Miao, Srinivasan Raghuraman, and Peter Rindal. Multi-party threshold private set intersection with sublinear communication. In *IACR International Conference on Public-Key Cryptography*, pages 349–379. Springer, 2021.

[CKT10] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 213–231. Springer, 2010.

[CLR17] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1243–1255, 2017.

[CM20] Melissa Chase and Peihan Miao. Private set intersection in the internet setting from lightweight oblivious prf. In *Annual International Cryptology Conference*, pages 34–63. Springer, 2020.

[CMdG$^+$21] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled psi from homomorphic encryption with reduced computation and communication. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1135–1150, 2021.

[DCW13] Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 789–800, 2013.

[DD15] Sumit Kumar Debnath and Ratna Dutta. Secure and efficient private set intersection cardinality using bloom filter. In *International Conference on Information Security*, pages 209–226. Springer, 2015.

[DSMRY09] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In *International Conference on Applied Cryptography and Network Security*, pages 125–142. Springer, 2009.

[EFG$^+$15] Rolf Egert, Marc Fischlin, David Gens, Sven Jacob, Matthias Senker, and Jörn Tillmanns. Privately computing set-union and set-intersection cardinality via bloom filters. In *Australasian Conference on Information Security and Privacy*, pages 413–430. Springer, 2015.

[FHNP16] Michael J Freedman, Carmit Hazay, Kobbi Nissim, and Benny Pinkas. Efficient set intersection with simulation-based security. *Journal of Cryptology*, 29(1):115–155, 2016.

[FNP04] Michael J Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *International conference on the theory and applications of cryptographic techniques*, pages 1–19. Springer, 2004.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. Stanford university, 2009.

[GN19] Satrajit Ghosh and Tobias Nilges. An algebraic approach to maliciously secure private set intersection. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 154–185. Springer, 2019.

[GPR$^+$21] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Oblivious key-value stores and amplification for private set intersection. In *Annual International Cryptology Conference*, pages 395–425. Springer, 2021.

[GS19] Satrajit Ghosh and Mark Simkin. The communication complexity of threshold private set intersection. In *Annual International Cryptology Conference*, pages 3–29. Springer, 2019.

[Haz18] Carmit Hazay. Oblivious polynomial evaluation and secure set-intersection from algebraic prfs. *Journal of Cryptology*, 31(2):537–586, 2018.

[HFH99] Bernardo A Huberman, Matt Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 78–86, 1999.

[HN10] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. In *International Workshop on Public Key Cryptography*, pages 312–331. Springer, 2010.

[HOS17] Per Hallgren, Claudio Orlandi, and Andrei Sabelfeld. Privatepool: Privacy-preserving ridesharing. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 276–291. IEEE, 2017.

[HV17] Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Scalable multi-party private set-intersection. In *IACR international workshop on public key cryptography*, pages 175–203. Springer, 2017.

[HW06] Susan Hohenberger and Stephen A Weis. Honest-verifier private disjointness testing without random oracles. In *International Workshop on Privacy Enhancing Technologies*, pages 277–294. Springer, 2006.

[IKN$^+$20] 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 *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 370–389. IEEE, 2020.

[JL10] Stanisław Jarecki and Xiaomin Liu. Fast secure computation of set intersection. In *International Conference on Security and Cryptography for Networks*, pages 418–435. Springer, 2010.

[JTKA22] Jonas Janneck, Anselme Tueno, Jörn Kußmaul, and Matthew Akram. Private computation on set intersection with sublinear communication. *Cryptology ePrint Archive*, 2022.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious prf with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 818–829, 2016.

[KS05] Lea Kissner and Dawn Song. Privacy-preserving set operations. In *Annual International Cryptology Conference*, pages 241–257. Springer, 2005.

[Mea86] Catherine Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *1986 IEEE Symposium on Security and Privacy*, pages 134–134. IEEE, 1986.

[MPR$^+$20] Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. Two-sided malicious security for private intersection-sum with cardinality. In *Annual International Cryptology Conference*, pages 3–33. Springer, 2020.

[OOS17] Michele Orrù, Emmanuela Orsini, and Peter Scholl. Actively secure 1-out-of-n ot extension with application to private set intersection. In *Cryptographers' Track at the RSA Conference*, pages 381–396. Springer, 2017.

[PRTY19] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. Spot-light: Lightweight private set intersection from sparse ot extension. In *Annual International Cryptology Conference*, pages 401–431. Springer, 2019.

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 515–530, 2015.

[PSWW18] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based psi via cuckoo hashing. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 125–157. Springer, 2018.

[PSZ14] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on {OT} extension. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 797–812, 2014.

[PSZ18] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on ot extension. *ACM Transactions on Privacy and Security (TOPS)*, 21(2):1–35, 2018.

[RR17] Peter Rindal and Mike Rosulek. Malicious-secure private set intersection via dual execution. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1229–1242, 2017.

[RS21] Peter Rindal and Phillipp Schoppmann. Vole-psi: fast oprf and circuit-psi from vector-ole. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 901–930. Springer, 2021.

[TSS$^+$20] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. Epione: Lightweight contact tracing with strong privacy. *arXiv preprint arXiv:2004.13293*, 2020.

[ZC18] Yongjun Zhao and Sherman SM Chow. Can you find the one for me? In *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*, pages 54–65, 2018.