Secure Two-party Computation Approach for NTRUEncrypt

Lin You*, Yan Wang*, Liang Li, Gengran Hu

[a]*School of Cyberspace Security, Hangzhou Dianzi University, Hangzhou, 310018, China*

Abstract

Secure multi-party computation can provide a solution for privacy protection and ensure the correctness of the final calculation results. Lattice-based algorithms are considered to be one of the most promising post-quantum cryptographic algorithms due to a better balance among security, key sizes and calculation speeds. The NTRUEncrypt is a lattice-based anti-quantum attack cryptographic algorithm. Since there haven't been much candidate post-quantum cryptographic algorithms for secure multi-party computation. In this paper, we propose a novel secure two-party computation scheme based on NTRUEncrypt and implement the polynomial multiplication operations under NTRUEncrypt-OT. Our secure two-party computation scheme mainly uses oblivious transfer and privacy set interaction. We prove the security of our scheme in the semi-honest model. Our scheme can be applied for multi-party computation scenarios, such as quantum attack-resisted E-votes or E-auctions.

Keywords: Secure Multi-party Computation, NTRUEncrypt, Oblivious Transfer, Privacy Set Intersection, Polynomial Multiplication

1. Introduction

Secure multi-party computation originate from Yao's [1] millionaire problem. Afterwards, in [2], Goldreich proposed a secure multi-party computation protocol, which can compute arbitrary functions [3]. The purpose of secure multi-

*Corresponding author
Email addresses: `mryoulin@gmail.com` (Lin You), `wang.yan@hdu.edu.cn` (Yan Wang)

party computation is to allow each participant to jointly calculate an objective function without revealing their private data. n the parties, holding private inputs $x_1, x_2, \ldots, x_n$, wish to compute a given function $f(x_1, x_2, \ldots, x_n)$. Secure multi-party computation can be widely used, such as electronic voting [4], sorting and searching [5] for information, deception detection [6], deep learning [7], etc. Due to the different credibility of the participants in multi-party computation, the security issues are also different. Security models are divided into ideal models, semi-honest models, and malicious models. Our scheme is based on a semi-honest [8] model, because the ideal model does not exist in reality, and malicious participants who undermine the normal operation of the protocol will be bound by the protocol to a certain extent. In the semi-honest model, each party will calculate the relevant results according to the agreement and correctly return them to other participants. There is no risk of data tampering in the intermediate process, but the participants may be based on other information entered by the participants or the intermediate results when interacting data are extra derivation of other information.

In this paper, we apply the idea of secure multi-party computation for NTRUEncrypt, and describe how to use secure two-party computation to complete various stages of operations. In fact, the essence of secure multi-party computation is the comprehensive use of cryptographic protocols with different functions. The commonly used cryptographic protocols in multi-party computation include Oblivious Transfer (OT), Garble Circuit (GC), and Secrets Share (SS) etc. In this paper, we use OT to construct our scheme. The first form of oblivious transfer was originally proposed by Michael O.Rabin in [9]. Another more practical 1-out-of-2 OT protocol, that is to get one of the two data, was proposed by Shimon Even, Oded Goldreich in [10]. There are many ways to implement oblivious transfer, which are generally implemented using public and private key cryptosystem encryption, such as RSA, ECC, and so on. In this paper, we use the NTRUEncrypt-based OT proposed in [11]. Further, we use OT extension to improve efficiency. Based on the OT extension, the efficient Privacy Set Intersection (PSI) is used in NTRUEncrypt key generation stage.

As an important part of our scheme, PSI allows two parties holding their respective sets to jointly calculate the intersection of the two sets. Freedman et al. proposed a privacy protection intersection operation based on a public key encryption system in [12]. After that, a random function combined with homomorphic encryption was proposed in [13], and the efficiency was further improved by Chen et al in [14], in which is based on fully homomorphic encryption. Then, B.Pinkas proposed a privacy set intersection based on oblivious transfer in [15]. In addition, in order to meet the intersection operation of large amounts of data, Kamaea et al. proposed a PSI scheme in [16], in which the number of set elements can reach billion times. At the end of the PSI protocol interaction, one or both parties should get the correct intersection, and will not get any information in the other party's set other than the intersection.

In this paper, we use Oblivious Pseudo-Random Functions (OPRF) to implement the PSI protocol. An OPRF is a function F: given a key k from the sender and an input element e from the receiver, computes and outputs $F_k(e)$ to the receiver. The sender obtains no output and learns no information about e while the receiver learns no information about k. We adopt the scheme proposed in [17]. Namely, by combining the Cuckoo hashing with OPRF, we limit the transmission combination of the sender to $O(n)$. Running PSI is an important part of the key generation phase in NTRUEncrypt.

Another goal of ours is to realize the multiplication [18] calculation of polynomials through secure two-party computation. The reason is that the operations involved in the NTRUEncrypt are basically multiplications of polynomials. Specifically, the sender and the receiver hold polynomials $f(x)$ and $g(x)$ respectively, and hope to jointly complete the calculation $f(x)g(x) = A(x)+B(x)$, where the sender holds $A(x)$, and the receiver holds $B(x)$. In the 3.1 section, we give the details of this scheme.

2. Preliminaries

2.1. NTRUEncrypt

The National Institute of Standards and Technology (NIST) started research on post-quantum cryptography as early as 2012, and launched a global call for post-quantum cryptography standards in February 2016, of which NTRUEncrypt [19] is the algorithm submitted in the round 1. A polynomial quotient ring is applied in NTRUEncrypt. $\mathcal{R}$ is a ring, $m \in \mathcal{R}$. For any $a \in \mathcal{R}$, use $\bar{a}$ to represent the set of all a' satisfying $a' = a \pmod{m}$, then the set $\bar{a}$ is called the congruence class of a, and use $\mathcal{R}/(m)$ or $\mathcal{R}/m\mathcal{R}$ to represent the set of all congruence classes. That is:

$$\mathcal{R}/(m) = \mathcal{R}/m\mathcal{R} = \left\{ \bar{a} \colon a \in \mathcal{R} \right\}.$$

From this, we can form a polynomial ring whose coefficients are taken from $\mathcal{R}$:

$$\mathcal{R}\left[x\right] = \left\{ a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n \,:\, n \geq 0,\, a_0,\, a_1,\, \ldots,\, a_n \in \mathcal{R} \right\}.$$

Therefore, a polynomial quotient ring can be constructed by combining the polynomial ring and the quotient ring. As for the polynomial quotient ring in NTRUEncrypt, we call it a convolutional polynomial ring. Given a positive integer N, the convolutional polynomial ring with rank N is $\mathcal{R} = \mathbb{Z}\left[x\right]/(x^N - 1)$. Similarly, the convolutional polynomial ring modulo p is $\mathcal{R}_p = \mathbb{Z}_p\left[x\right]/(x^N - 1)$, and modulo q is $\mathcal{R}_q = \mathbb{Z}_q\left[x\right]/(x^N - 1)$. With the above relevant knowledge, we formally introduce the NTRUEncrypt.

Let $N \geq 1$ and select the modulus p and q, set the convolution polynomial ring as:

$$\mathcal{R} = \mathbb{Z}\left[x\right]/(x^N - 1),\, \mathcal{R}_p = \mathbb{Z}_p\left[x\right]/(x^N - 1),\, \mathcal{R}_q = \mathbb{Z}_q\left[x\right]/(x^N - 1).$$

By modulo the coefficients, the polynomial $a\left(x\right)$ in $\mathcal{R}$ is regarded as an element in $\mathcal{R}_p$ or $\mathcal{R}_q$. On the contrary, the element in $\mathcal{R}_p$ or $\mathcal{R}_q$ can be promoted to $\mathcal{R}$ by the method of center lifting. At the same time, there is a set of three-valued polynomials $\Gamma\left(d_1, d_2\right)$, and each polynomial in the set has a d_1

term coefficient of 1, and a d_2 term coefficient of -1. For example, if $a(x) = x^{17} - x^{13} + x^5 + x^3 - x$, then $a(x) \in \Gamma(3,2)$.

We are divided into three stages to introduce the NTRUEncrypt in detail:

Key generation. The receiver chooses public parameters [20] $(N,\, p,\, q,\, d)$, where N and p are prime numbers, and it needs to satisfy $q > (6d+1)\,p$ the inequality relationship can be decrypted correctly. The receiver's private key consists of two random polynomials: $f(x) \in \Gamma(d+1,\, d)$, $g(x) \in \Gamma(d,\, d)$. The receiver calculates the inverse element: $F_q(x) = f(x)^{-1} \in \mathcal{R}_q$, $F_p(x) = f(x)^{-1} \in \mathcal{R}_p$. If there is no inverse element, the receiver reselects $f(x)$. Then, the receiver calculates the public key $h(x) = F_q(x)\, g(x) \in \mathcal{R}_q$. Finally, the corresponding private key is $(f(x), F_p(x))$.

Encryption phase. The plaintext is the polynomial $m(x)$ in $\mathcal{R}$, and its coefficients are between $-\frac{1}{2}p$ and $\frac{1}{2}p$. $s(x)$ is the center lift of the polynomial in $\mathcal{R}_p$. To encrypt a message $s(x)$, the sender randomly selects the polynomial $r(x) \in \Gamma(d,\, d)$ and calculates the ciphertext $e(x)$,

$$e(x) = ph(x)\, r(x) + m(x) \pmod{q}.$$

The finally calculated ciphertext is in the ring $\mathcal{R}_q$.

Decryption phase. After obtaining the ciphertext $e(x)$, the receiver first computes $a(x) = f(x) e(x) \pmod{q}$, then raise the center of $a(x)$ to $\mathcal{R}$ and perform the modulo p operation to get the plaintext $b(x) = F_p(x) a(x) \pmod{p}$.

We can verify that the $b(x)$ calculated by the receiver and the plaintext $s(x)$ are equal, as long as the NTRUEncrypt parameter $(N,\, p,\, q,\, d)$ satisfies the following inequality:

$$q > (6d+1)\, p.$$

Then, the $b(x)$ calculated by the receiver is equal to the plaintext $m(x)$.

2.2. Oblivious Transfer

Oblivious Transfer [21] is an important module for constructing secure multi-party compution protocols, and generally can be divided into 1-ouf-of-2 OT and 1-out-of-n OT. In 1-ouf-of-2 OT, the receiver gets one of the two data from the sender. When the oblivious transfer protocol is called n times, the sender holds n element pairs $\left(x_0^i, x_1^i\right)$, $\dot{i} \in \{0,1\}^l$, and the receiver holds an n-bit vector b. After the protocol is completed, the receiver obtains each selected element $x_{b[i]}^i$, but does not know another element $x_{1-b[i]}^i$, while the sender does not know b. We denote this form of oblivious transfer as 1-out-of-2 OT_l^m, which means that the two parties involved in the oblivious transfer interaction for m times, then each time the receiver can choose one from two l-bit data. Correspondingly, 1-out-of-N OT_l^m can be interpreted as an N-to-1 oblivious transfer protocol for m calls of l-bit elements.

2.2.1. NTRUEncrypt-OT

OT is usually implemented using public and private key encryption. In this paper, for the sake of anti-quantum, our underlying specific OT implementation utilizes the NTRUEncrypt-based OT solution proposed in [11]. Specific steps are as follows:

1. The sender holds the polynomial as $n_1(x)$, $n_2(x)$, … , $n_t(x)$, and chooses $h_1(x) \in \mathcal{R}_q$ as the public key during oblivious transfer. The receiver chooses $h_2(x) \in \mathcal{R}_q$.

2. The sender randomly chooses three-valued polynomials $r_1(x)$, $r_2(x)$, …, $r_t(x)$ in order to encrypt $n_1(x)$, $n_2(x)$, …, $n_t(x)$, then calculates:

$$e_i(x) = r_i(x)\, h_1(x) + n_i(x) \pmod{q}.$$

3. After the receiver receives all $e_i(x)$, the receiver select one of $e_\mu(x)$, $\mu \in \{1, \ldots, t\}$, then randomly chooses small coefficients polynomial $r'(x)$, and calculates:

$$c'_\mu(x) = r'(x)\, h_2(x) + e_i(x) \pmod{q},$$

then send the result to the sender.

4. After the sender receives $c'_\mu(x)$, and calculates:

$$e''_i(x) = c'_\mu(x) - r_i h_1(x).$$

5. The receiver receives all $e''_i(x)$ and decrypts it with own private key to get $n_\mu(x)$.

Using the above process, the two parties involved first randomly select their own public key, and then complete the oblivious transfer. Specifically in our scheme, in the use of NTRUEncrypt-based oblivious transfer, the interactive data is converted into a binary form, and then further expressed as a polynomial. When describing our scheme, we treat OT as a black box. In the specific scheme, we will no longer elaborate on the detailed process of OT. All OTs in the scheme use NTRUEncrypt-based OT by default.

2.2.2. Construct OPRF by Extension OT

Based on the idea of OPRF, we introduce the OT extension method proposed in [17]. Because OT is a time-consuming operation, the use of OT extension [22] is to reduce the use of OT to improve algorithm efficiency. A large number of OPRF are constructed by transforming the OT extension. We use S to represent the sender and R to represent the receiver. The specific description is as follows.

R has m selection strings $\boldsymbol{r}=(\boldsymbol{r}_1, \boldsymbol{r}_2, \ldots, \boldsymbol{r}_m)$, $1 \leq i \leq m$, $\boldsymbol{r}_i \in \{0,1\}^*$. R and the S have a common random encoding function $C\colon \{0,1\}^* \to \{0,1\}^k$ and a common hash function $H\colon \{0,1\}^k \to \{0,1\}^l$.

1. First, R randomly initializes a random matrix T with m rows and k columns, where each element in the matrix is either 0 or 1. S randomly selects the bit string $\boldsymbol{s}=(s_1, s_2, \ldots, s_k)$ of length k.

2. R constructs a matrix U, $\boldsymbol{u}_j = \boldsymbol{t}_j \oplus C(\boldsymbol{r}_j)$, where $\boldsymbol{u}_j$ and $\boldsymbol{t}_j$ are represented as the j-th row of U and T respectively. $\boldsymbol{r}_j$ is an arbitrary length input data.

3. R as the receiver and S as the sender, execute 1-out-of-2 OT_m^k. For the i-th oblivious transfer of length m, the input of R is $(\boldsymbol{t}^i, \boldsymbol{u}^i)$, $\boldsymbol{t}^i$ and $\boldsymbol{u}^i$ are represented as the i-th column of the matrix T and U respectively. The input of S is s_i, when s_i=0, S gets $\boldsymbol{t}^i$, s_i=1, S gets $\boldsymbol{u}^i$. S will be arranged into a matrix Q in the order of the received columns.

4. Finally, R as the receiver and S as the sender can perform m times OPRF. R outputs $H(j,\ \boldsymbol{t}_j)$, and S outputs $H(j,\ \boldsymbol{q}_j \oplus (C(\boldsymbol{r}') \cdot \boldsymbol{s}))$, $1 \leq j \leq m$, $\boldsymbol{r}' \in \{0,1\}^*$. $\boldsymbol{q}_j$ is represented as the j-th row of matrix Q.

In this way, a large number of OPRF can be realized, and only a fixed number of 1-out-of-2 oblivious transfer are required to realize m times OPRF. Therefore, the pseudo-random function $F(\cdot)$ is generated by the above method. $\boldsymbol{r}_j$ and $\boldsymbol{r}$ denote the data to be compared between R and S, S outputs $H(j,$ $\boldsymbol{q}_j \oplus (C(\boldsymbol{r}') \cdot s))$, and R outputs $H(j,\ \boldsymbol{t}_j)$. Only need to compare the output results of R and S to achieve privacy comparison. That is, in our scheme, $F(\cdot)$ is a hash function, and "$\cdot$" represents the selected randomization seed and the specific element that needs to complete the intersection operation.

2.3. Privacy Set Intersection

Private Set Intersection (PSI) [23] can be regarded as a secure multi-party computation that takes the participants' respective private information as a set, and the function implemented by the objective function is the set intersection. We use OPRF and Cuckoo hashing [24] to construct PSI. Using the Cuckoo hashing to map n pieces of data to b bins, we first select 3 hash functions h_1, h_2, h_3: $\{0,\ 1\}^* \rightarrow a$, $1 \leq a \leq b$, and b empty bins $B[1,\ 2,\ \ldots,\ b]$. Before putting data x into the bin, need to check whether the three buckets $B[h_1(x)]$, $B[h_2(x)]$, $B[h_3(x)]$ have empty bins. If there is, put the data x into an empty bin, if not, randomly select a bin from the three bins, propose the original element x' in this bin, and put x into the bin. This operation is performed recursively for the elements that are kicked out, until the elements are put into a bin. If after a certain number of rounds, there are still elements that are not found in the bin, then the element is put into a special bin, which we call a bucket.

In the construction of the private set intersection, we take the communication between Alice and Bob as an example. Alice has n pieces of data, $1.2n$ bins and one bucket are constructed, and the size of the bucket is s. For Bob, there is now at most one element in each bin, and the bucket has at most s elements. Bob can forge other data to fill these bins and bucket.

As the sender, Alice generates $1.2n+s$ random seeds k, which are used as random seeds for $1.2n+s$ oblivious pseudo-random functions. As the receiver, Bob computes an OPRF for the data it holds. At the same time, Alice can arbitrarily compute a pseudo-random function, compute two sets of its own input data:

$$H = \left\{ F\left(k_{h_z(x)},\, x_i\right) | x_i \in X, z \in (1,\, 2,\, 3) \right\},$$

$$S = \left\{ F\left(k_{1.2n+j},\, x_{1.2n+j}\right) | x_{1.2n+j} \in X, j \in (1,\, \ldots,\, s) \right\}.$$

$F(\cdot)$ is OPRF constructed by extension OT that we introduced in section 2.2.2. Next, Alice sends H and S to Bob in a random order, and Bob compares it with the result of the function computed by itself, if any the same value indicates that the corresponding data is shared data. So far, a secure privacy set intersection method has been constructed using Cuckoo hashing.

3. Secure Two-party Computation Scheme

In this section, we introduce our two-party computation solution in detail. This scheme is used in the key generation and decryption phases in NTRUEncrypt. For the key generation stage, our main work is to generate a three-valued polynomial that satisfies the conditions. In the introduction of the NTRUEncrypt, it is mentioned that the selection of the private key is two random polynomials, which belong to $\Gamma(d+1,\, d)$ and $\Gamma(d,\, d)$ respectively. We use a PSI scheme based on extension OT to construct OPRF, which is used in the private key generation phase of the NTRUEncrypt. We describe the private key generation phase in detail in Section 3.2. In addition, since both the private key generation phase and the decryption phase involve polynomial multiplication, we first describe how the two parties involved complete the polynomial multiplication in Section 3.1.

3.1. Polynomial Multiplication in NTRUEncrypt

We take the communication between the two participants Alice and Bob as an example. Alice holds a and Bob holds b. They wish to perform a computation by which Alice obtains x and Bob obtains y such that x+y=ab. In [25], this method is described in detail. We propose a new scheme that allows this idea to be applied to the multiplication calculation of polynomials, that is, Alice holds

$f(x)$ and Bob holds $f(x)$, they wish to perform a computation by which Alice obtains $A(x)$ and Bob obtains $B(x)$ such that $A(x)$+$B(x)$=$g(x)f(x)$. Alice and Bob perform the following operations.

1. Alice claims that the polynomial has N terms, and N is a public parameter of NTRUEncrypt, which is expressed as:

$$f(x) = a_0 + a_1x^1 + a_2x^2 + \cdots + a_Nx^N.$$

It should be noted that even if the coefficients of most terms are 0, they are still expressed in this form. The purpose of this is to hide the number of terms in the polynomial.

2. Bob randomly generates $(N+1)^2$ polynomials $C_q(x)$ locally, $1 \le q \le (N+1)^2$ and does not impose any restrictions on the number of terms, coefficients and exponents of these polynomials.

3. Alice and Bob perform NTRU-Based OT operations, for $f(x)$, Alice expresses the coefficient a_i of each term as $a_{i0}a_{i1}\ldots a_{iN}$. If a_{iu}=0, Alice gets: $c_q(x)$. If a_{iu}=1, Alice gets: $g(x)+c_q(x), 0 \le u \le N$.

4. For each coefficient a_i, after Alice and Bob perform N OT operations, Alice obtains:

$$A_i(x) = \sum_{i=0,u=0}^{N} 2^i\left(a_{iu}g(x) + C_q(x)\right).$$

Bob calculates :

$$B_i(x) = -\sum_{i=0}^{N} 2^i C_q(x).$$

Obviously, Alice and Bob can complete the calculation of $a_ig(x)x^i$ after performing $(N+1)$ times OT on each coefficient, $g(x)a_ix^i$=$A_i(x)$+$B_i(x)$. Similarly, after OT for each coefficient, we can easily get:

$$g(x)\left(a_0 + a_1x^1 + a_2x^2 + \cdots + a_Nx^N\right) = \sum_{i=0}^{N} A_i(x) + \sum_{i=0}^{N} B_i(x).$$

Alice holds $A(x)$=$\sum_{i=0}^{N} A_i(x)$ and Bob holds $B(x)$=$\sum_{i=0}^{N} B_i(x)$. We also briefly describe our solution in Figure 1. Each time the two parties calculate a polynomial multiplication, $(N+1)^2$ times of OT are required.

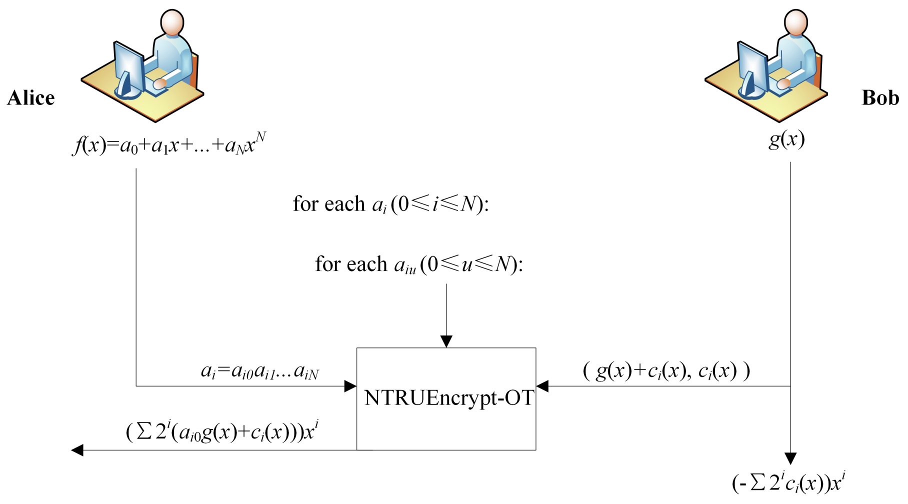

Figure 1: Two-party Polynomial Multiplication

Table 1: Reversal

```
Algorithm 1: Inversion Operation
Input: $[a(x)]$, $[\cdot]$ expressed as secret data
Output: $[a^{-1}(x)]$
1. Choose randomly $[b(x)]$
2. Compute $[c(x)] <- [a(x)][b(x)]$
3. Open $[c(x)]$ , and reverse it
4. $[a^{-1}(x)] = c^{-1}(x)[b(x)]$
```

3.2. Private Key Generation

According to the public parameter d in NTRUEncrypt, we stipulate that if d is an odd number, then for the random polynomial $f(x)$, Alice generates $\frac{d+1}{2}$ positive monomials and $\frac{d-1}{2}$ negative monomials locally, while Bob generates $\frac{d+1}{2}$ positive monomials and $\frac{d+1}{2}$ negative monomials locally. For $g(x)$, Alice generates $\frac{d+1}{2}$ positive monomials and $\frac{d+1}{2}$ negative monomials locally, while Bob generates $\frac{d-1}{2}$ positive monomials and $\frac{d-1}{2}$ negative monomials locally. Similarly, if d is an even number, for the random polynomial $f(x)$, Alice generates $\frac{d+2}{2}$ positive monomials and $\frac{d}{2}$ negative monomials locally, while Bob generates $\frac{d}{2}$ positive monomials and $\frac{d}{2}$ negative monomials locally. For $g\left(x\right)$, Alice generates $\frac{d}{2}$ positive monomials and $\frac{d}{2}$ negative monomials locally, while Bob generates $\frac{d}{2}$ positive monomials and $\frac{d}{2}$ negative monomials locally. It should be noted that we stipulate that the exponents of the polynomials generated locally by Alice and Bob must satisfy the characteristics of the set, that is, the same elements cannot exist.

Further, we consider that the exponents of the monomials selected by both parties may overlap, which leads to the fact that the coefficients of the polynomials combined by the two parties do not meet the conditions of three-value polynomials. For the coefficients, in order to avoid the number of terms is reduced, we stipulated that participants cannot choose two elements with the same absolute value. Both parties combine it with the corresponding index and take its absolute value. Then both parties obtain a set of positive integers, calculate the intersection elements based on PSI, and loop this operation until the intersection is an empty set.

Alice and Bob respectively hold private sets X and Y. For $f\left(x\right)$, $g\left(x\right)$, we need to perform this operation twice. In order to calculate whether there is an intersection between the elements of the two parties, we use the PSI protocol introduced in section 2.3. Under the premise of no intersection, Alice and Bob have jointly completed the generation of two random polynomials $f\left(x\right)\in\Gamma\left(d+1,\,d\right)$ and $g\left(x\right)\in\Gamma\left(d,\,d\right)$. Among them, Alice holds $f_A\left(x\right)$ and $g_A\left(x\right)$, Bob holds $f_B\left(x\right)$ and $g_B\left(x\right)$, so that, $f\left(x\right)=f_A\left(x\right)+g_A\left(x\right)$, $g\left(x\right)=g_A\left(x\right)+g_B\left(x\right)$. Next, Alice and Bob calculate the inverse element $F_q\left(x\right)=f\left(x\right)^{-1}\in\mathcal{R}_q$, $F_p\left(x\right)=f\left(x\right)^{-1}$ $\in\mathcal{R}_p$. We describe the method of calculating the inverse element in Algorithm 1. Under the condition of mutual calculation by both parties, based on the polynomial multiplication scheme proposed in Section 3.1, we perform the following operations:

1. Alice and Bob each randomly generate polynomials locally, and then add them to form $b\left(x\right)$. That is, $b\left(x\right)=b_A\left(x\right)+b_B\left(x\right)$. Among them, Alice holds $b_A\left(x\right)$ and Bob holds $b_B\left(x\right)$.

2. Both sides calculate the inverse element of $f(x)$:

$$\begin{aligned} f(x)\,b(x) &= \left(f_A(x)+f_B(x)\right)\left(b_A(x)+b_B(x)\right) \\ &= f_A(x)\,b_A(x)+f_A(x)\,b_B(x)+f_B(x)\,b_A(x)+f_B(x)\,b_B(x) \end{aligned}$$

For cross terms: $f_A(x)\,b_B(x)$, $f_B(x)\,b_A(x)$, we complete the two-party computation through the polynomial multiplication scheme proposed in Section 3.1. After the calculation of $f(x)\,b(x)$ is completed, the result of $f(x)\,b(x)$ is marked as $c(x)$, and the inverse element of $c(x)$ can be calculated publicly by using the extended Euclidean algorithm. Finally, after both parties have performed the inversion computation, Alice holds $F_{Aq}(x)$, and Bob holds $F_{Bq}(x)$. Similarly, for inverse element of $g(x)$, Alice holds $F_{Ap}(x)$, and Bob holds $F_{Bp}(x)$.

Next, we calculate the public key $h(x){=}F_q(x)g(x)\in\mathcal{R}_q$. Similarly, for the cross term, we use the polynomial multiplication scheme in Section 3.1 to complete the calculation. Since then, we have implemented the key generation phase in NTRUEncrypt through two-party computations.

3.3. Encryption and Decryption Stage

In the encryption phase, we only need to calculate according to the process of NTRUEncrypt itself. Suppose a participant encrypts the plaintext $m(x)$, first randomly select the polynomial $r(x)\in\Gamma(d,\,d)$, and then use the public key $h(x)$ generated in the key generation stage to calculate the ciphertext:

$$e(x) = ph(x)\,r(x) + m(x)\,(\operatorname{mod} q)\,.$$

In the decryption phase, when Alice and Bob receive the ciphertext $e(x)$, both parties complete the decryption together. They first compute:

$$a(x) = f(x)\,e(x) = f_A(x)\,e(x) + f_B(x)\,e(x) = a_A(x) + a_B(x)\,.$$

Among them, Alice holds $a_A(x)$, and Bob holds $a_B(x)$. Then compute the plaintext $b(x)$, $b(x){=}F_p(x)a(x)$, where $F_p(x){=}f^{-1}(x)\in\mathcal{R}_p$. Next, Alice and Bob use the two-party polynomial multiplication calculation for the cross term. So that $b(x){=}b_A(x){+}b_B(x)$, where Alice holds $b_A(x)$ and Bob holds $b_B(x)$.

Similarly, $F_p(x)=F_{Ap}(x)+F_{Bp}(x)$, where Alice holds $F_{Ap}(x)$ and Bob holds $F_{Bp}(x)$.

DECRYPTION PROCESS

Initialization:

When Alice and Bob receive the ciphertext $e(x)$, both parties complete the decryption together, that is, complete two polynomial multiplication calculations

1. In the first stage, polynomial multiplication is done modulo q:

Alice		Bob
computes: $a_A(x)=f_A(x)e(x)$		computes: $a_B(x)=f_B(x)e(x)$
Output: $a_A(x)$	$\longleftrightarrow$	Output: $a_B(x)$

Therefore, Alice and Bob complete $a(x) = f(x)\,e(x)$ together.

2. In the second stage, polynomial multiplication is done modulo p:

Alice		Bob
computes: $b_A(x)=F_{Ap}(x)a_A(x)$		computes: $b_B(x)=F_{Bp}(x)a_B(x)$
Output: $b_A(x)$	$\longleftrightarrow$	Output: $b_B(x)$

4. Complete Implementation Overview

Initialization phase

In this section, we describe the scheme as a whole. Alice and Bob each randomly select a polynomial as their public key for the default NTRUEncrypt-OT. Then, in the process of constructing OPRF with OT extension, Alice acts as the sender and Bob acts as the receiver. Both parties construct the oblivious pseudo-random function $F(\cdot)$ according to the method we introduced in section 2.2.2. Nextly, Alice and Bob randomly select three hash functions together, and use the Cuckoo hashing principle and OPRF $F(\cdot)$ to construct a PSI protocol.

Key generation phase:

1. Alice and Bob implement the PSI protocol to determine whether the monomial exponent sets they generate have intersections. If there is an intersection,

the set is regenerated, otherwise, the monomials are added to generate the corresponding polynomials $f\,(x) \in \Gamma(d+1,\,d)$, $g\,(x) \in \Gamma(d,\,d)$.

2. Using Algorithm 1 to calculate the inverse element $F_q\,(x) = f^{-1}\,(x) \in \mathcal{R}_q$, $F_p\,(x) = f^{-1}\,(x) \in \mathcal{R}_p$. Note that the algorithm involves polynomial multiplication, so we can use the scheme proposed in Section 3.1 to complete the polynomial multiplication computation.

3. Similarly, complete the polynomial multiplication $h\,(x) = F_q\,(x)g\,(x) \in \mathcal{R}_q$, the corresponding private key is $(f\,(x),\,F_p\,(x))$.

Encryption and decryption stage:

According to the public key generated in the key generation stage, any party or other participant can choose a polynomial in $\mathcal{R}$ with a coefficient between $-\frac{1}{2}p$ and $\frac{1}{2}p$ for encryption.

After encryption, both parties complete the decryption together. For the ciphertext $e\,(x)$, both parties first calculate $a\,(x)$, and then perform further calculations to obtain the plaintext $b\,(x) = F_p\,(x)a\,(x)(\text{mod}\,p)$. In Figure 2, we describe the general process of our scheme.

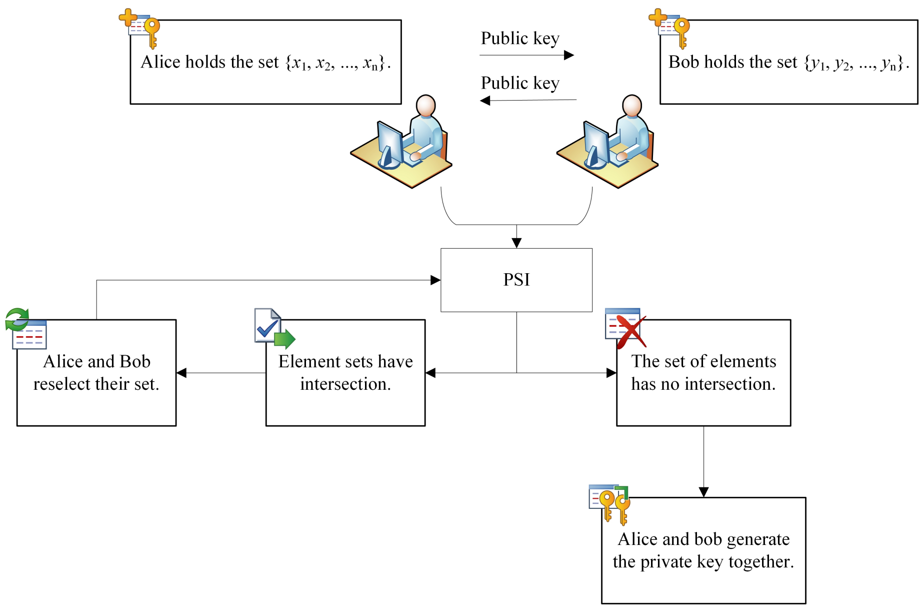
Alice holds the set $\{x_1, x_2, ..., x_n\}$.
Public key
Public key
Bob holds the set $\{y_1, y_2, ..., y_n\}$.
PSI
Alice and Bob reselect their set.
Element sets have intersection.
The set of elements has no intersection.
Alice and bob generate the private key together.

Figure 2: The Overall Process of the Program

5. Security Analysis

The security of our two-party computation scheme is based on the security of the PSI and the security of the NTRUEncrypt.

Theorem : Our two-party computation scheme is safe in the semi-honest model.

Proof : In the semi-honest model, if the receiver R try to derive the information of the sender S during the execution of the protocol as much as possible. The information sent by S to R is $H\left(q_j \oplus \left(C\left(r'\right) \cdot s\right)\right)$, $r' \neq r_j$. If R solves r' by brute force, however, we observe the following equation:

$$q_j \oplus \left(C\left(r'\right) \cdot s\right) = t_j \oplus \left(C\left(r'\right) \cdot s\right) \oplus \left(C\left(r'\right) \cdot s\right) = t_j \oplus \left(\left(C\left(r'\right) \oplus C\left(r_j\right)\right) \cdot s\right).$$

In the above equation, only s is unknown to R. To ensure safety, we need to make the hamming weight of $C\left(r'\right) \oplus C\left(r_j\right)$ greater than the safety parameter k'. Under this premise, R needs to guess at least k' bits in s before it is possible. For a successful attack, we usually set k' to be greater than or equal to 128, which is a symmetric password security parameter. We know that the output length of the random coding function C is k. If the Hamming weight of $C\left(r'\right) \oplus C\left(r_j\right)$ is greater than k' , $k > k'$ is required. According to the conclusion of [17], only is needed to ensure that the hamming weight of $C\left(r'\right) \oplus C\left(r_j\right)$ is less than the probability of k' is negligible.

The security of OT based on NTRUEncrrypt is proven in [11]. In Table 2, we give the security parameters of NTRUEncrypt. If p is fixed to 3 and q is fixed to 2048, according to the relationship of $q > (6d+1)\,p$, the maximum positive integer that d can take is 113, and the maximum positive integer that $\frac{d}{2}$ can take is 56. In our two-party computations scheme, $f\left(x\right)$ and $h\left(x\right)$ are jointly completed by both parties. The adversary can search for possible keys by verifying whether $f\left(x\right)h\left(x\right)$ is a three-valued polynomial. The adversary needs to calculate the size of the three-valued polynomial set. We use $N_{\Gamma(d_1,d_2)}$ to represent the size of the three-valued polynomial set, so:

$$N_{\Gamma(d_1,d_2)} = C_N^{d_1} C_{N-d_1}^{d_2} = \frac{N!}{d_1! d_2! \left(N - d_1 - d_2\right)!}.$$

When both d_1 and d_1 get $\frac{N}{3}$, $N_{\Gamma(d_1,d_2)}$ can have the maximum value. Since all cycles of $f(x)$ can be decryption keys, there are N options. Therefore, the adversary needs to try about $\frac{N_{\Gamma(d_1,d_2)}}{N}$ times. In Table 3, we obtain the degree of difficulty for the adversary to crack a polynomial held by a participant, and in our two-party computation scheme, the number of calculations will increase exponentially. For higher security levels, the number of times the adversary needs to try to crack will be a more difficult value to estimate. Moreover, OT based on NTRUEncrypt is proved is secure under the universal composability framework in [11].

For our proposed two-party polynomial multiplication scheme, in order to hide the number of polynomial terms, we express each coefficient as a bit string, even if the coefficient is 0. Due to the nature of oblivious transfer, the sender cannot obtain the specific bits transmitted by the receiver. At the same time, if the adversary wants to brute force a certain coefficient of the polynomial, each coefficient of the polynomial needs to be calculated 2^{N+1} times, and the brute-force cracking of the entire polynomial needs to be calculated $\left(2^{N+1}\right)^{N+1}$ times, which is even far greater than the number of attempts required by the adversary to brute force the NTRUEncrypt private key.

Table 2: NTRU Security Parameters

Security Level	N	q	p
Moderate Security	401	2048	3
Standard Security	439	2048	3
High Security	593	2048	3
Highest Security	743	2048	3

6. Conclusion

Post-quantum cryptography is not only more secure but also has wider application scenarios. As for our proposed secure two-party computation scheme NTRUEncrypt, is not only suitable for E-vote or E-auction against quantum

Table 3: Crack Difficulty

Security Level	Number of calculations
Moderate Security	2^{445}
Standard Security	2^{461}
High Security	2^{515}
Highest Security	2^{556}

attacks, but also can be used to implement homomorphic encryption, attribute encryption, function encryption and indistinguishable obfuscation. In the field of cryptography, the protection of private keys is always the first issue. As far as the NTRUEncrypt is concerned, the primary condition for the key recovery problem to become a difficult mathematical problem is that this problem cannot be broken by exhausting or colliding search methods. Nowadays, The best known method to recover the private key from the NTRU public key is the lattice reduction algorithm. But from another perspective, we have made the private key generated by multiple parties. Hence, even if the private key is recovered, the adversary cannot determine which part of the private key each participant holds.

References

[1] Yao, A.: 'Protocols for secure computations', 23rd annual symposium on foundations of computer science (sfcs 1982), IEEE, 1982, pp. 160–164

[2] Goldreich, O., Micali, S., Wigderson, A.: 'How to play any mental game, or a completeness theorem for protocols with honest majority', Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali, 2019, pp. 307–328

[3] Burra, S., Larraia, E.: 'High Performance Multi-Party Computation for Binary Circuits Based on Oblivious Transfer', IACR Cryptol. ePrint Arch, 2015, pp. 472

[4] Gang, C.: 'An electronic voting scheme based on secure multi-party computation', 2008 International Symposium on Computer Science and Computational Technology, IEEE, 2008, pp. 292–294

[5] Rao, C., Singh, K., Kumar, A.: 'Oblivious stable sorting protocol and oblivious binary search protocol for secure multi-party computation', Journal of High Speed Networks, IOS Press, 2021, pp.1–16

[6] Seo, M.: 'Fair and Secure Multi-Party Computation with Cheater Detection', Cryptography, Multidisciplinary Digital Publishing Institute, 2021, pp. 19

[7] Tran, A., Karnjana, J.: 'An efficient approach for privacy preserving decentralized deep learning models based on secure multi-party computation', Neurocomputing, Elsevier, 2021, pp. 245–262

[8] Resende, A.: 'Faster unbalanced Private Set Intersection in the semi-honest setting', Journal of Cryptographic Engineering, Springer, 2021, pp. 21–38

[9] Rabin, M.: 'How To Exchange Secrets with Oblivious Transfer', Information Security Applications, IACR Cryptol. ePrint Arch, 2005

[10] Even, S., Goldreich, O., Lempel, A.: 'A randomized protocol for signing contracts', Communications of the ACM, ACM New York, NY, USA, 1958, pp. 637–647

[11] Bi, B., Huang, D., Mi, B., et al.: 'Efficient LBS security-preserving based on NTRU oblivious transfer', Wireless Personal Communications, Springer, 2019, pp. 2663–2674

[12] Freedman, M., Nissim, K., Pinkas, B.: 'Efficient private matching and set intersection', International conference on the theory and applications of cryptographic techniques, Springer, 2004, pp. 1–19

[13] Freedman, M., Ishai, Y., Pinkas, B., et al.: 'Keyword search and oblivious pseudorandom functions', Theory of Cryptography Conference, Springer, 2005, pp. 303–324

[14] Chen, H., Laine, K., Rindal, P.: 'Fast private set intersection from homomorphic encryption', Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, 2017, pp. 1243–1255

[15] Pinkas, B., Schneider, T., Zohner, M.: 'Faster private set intersection based on OT extension', 23rd USENIX Security Symposium (USENIX Security 14), 2014, pp. 797–812

[16] Kamara, S., Mohassel, P., Raykova, M., et al.: 'Scaling private set intersection to billion-element sets', International Conference on Financial Cryptography and Data Security, Springer, 2014, pp. 195–215

[17] Kolesnikov, V., Kumaresan, R., Rosulek, M., et al.: 'Efficient batched oblivious PRF with applications to private set intersection', Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 818–829

[18] Camacho-Ruiz, E.: 'Timing-Optimized Hardware Implementation to Accelerate Polynomial Multiplication in the NTRU Algorithm', ACM Journal on Emerging Technologies in Computing Systems (JETC), ACM New York, NY, 2021, pp. 1–16

[19] Ahmad, K., Kamal, A.: 'Fast hybrid-MixNet for security and privacy using NTRU algorithm', Journal of Information Security and Applications, Elsevier, 2021, pp. 102872

[20] Kirchner, P.: 'Revisiting lattice attacks on overstretched NTRU parameters', Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer, 2017, pp. 3–26

[21] Costa, B., Branco, P.: 'Randomized Oblivious Transfer for Secure Multiparty Computation in the Quantum Setting', Entropy, Multidisciplinary Digital Publishing Institute, 2021, pp. 1001

[22] Pinkas, B., Schneider, T., Zohner, M.: 'Scalable private set intersection based on OT extension', ACM Transactions on Privacy and Security (TOPS), ACM New York, NY, USA, 2018, pp. 1–35

[23] Kissner, L., Song, D.: 'Privacy-preserving set operations', Annual International Cryptology Conference, Springer, 2005, pp. 241–257

[24] Pagh, R., Rodler, F.: 'Cuckoo hashing', Journal of Algorithms, Elsevier, 2004, pp. 122–144

[25] Gilboa, N.: 'Two party RSA key generation', Annual International Cryptology Conference, Springer, 1996, pp. 116–129