Chapter 3
Secure Primitive for Big Data Utilization

Akinori Kawachi, Atsuko Miyaji, Kazuhisa Nakasho, Yiying Qi, and Yuuki Takano

Abstract In this chapter, we describe two security primitives for big data utilization. One is a privacy-preserving data integration among databases distributed in different organizations. This primitive integrates the same data among databases kept in different organizations while keeping any different data in an organization secret to other organizations. Another is a privacy-preserving classification. This primitive executes a procedure for server's classification rule to client's input database and outputs only the result to the client while keeping the client's input database secret to the server and server's classification rule to the client. These primitives can be executed not only independently but also jointly. That is, after we integrate databases from distributed organization by executing the privacy-preserving data integration, we can execute a privacy-preserving classification.

3.1 Privacy-Preserving Data Integration

3.1.1 Introduction

Medical organizations often store the data accumulated through medical analyses. However, detailed data analysis sometimes requires separate datasets to be integrated without violating patient or commercial privacy. Consider the scenario in which the

A. Kawachi
Mie University, 1577 Kurimamachiya-cho, Tsu City, Mie 514-8507, Japan
e-mail: kawachi@cs.info.mie-u.ac.jp

A. Miyaji (✉) · Y. Qi · Y. Takano
Osaka University, 1-1 Yamadaoka, Suita, Osaka 565-0871, Japan
e-mail: miyaji@comm.eng.osaka-u.ac.jp

Y. Takano
e-mail: ytakano@cy2sec.comm.eng.osaka-u.ac.jp

K. Nakasho
Yamaguchi University, 1677-1 Yoshida, Yamaguchi City, Yamaguchi 753-8511, Japan
e-mail: nakasho@yamaguchi-u.ac.jp

A. Miyaji and T. Mimoto (eds.), *Security Infrastructure Technology for Integrated Utilization of Big Data*,
https://doi.org/10.1007/978-981-15-3654-0_3

occurrence of similar accidents can be attributed to a particular defective product. Such defective products should be identified as quickly as possible. However, the databases related to accidents are maintained separately by different organizations. Thus, investigating the causes of accidents is often time-consuming. For example, assume child A has broken her/his leg at school, but it is not clear whether the accident was caused by defective equipment. In this case, information relating to A's injury, such as the patient's name and type of injury, is stored in hospital database S_1. Information pertaining to A's accident, such as their name and the location of the swing at the school, is stored in database S_2, which is held by the fire department. Finally, information relating to the insurance claim following A's accident, such as the name and medical costs, is maintained in the insurance company's database, S_3. Computing the intersection of these databases, $S_1 \cap S_2 \cap S_3$, without compromising privacy would enable us to combine the separate sets of information, which may allow the cause of the accident to be identified. Let us consider another situation. Several clinics, denoted as P_i, maintain separate databases, represented as S_i. The clinics wish to know the patients they have in common to enable them to share treatment details; however, P_i should not be able to access any information about patients not stored in their own dataset. In this case, the intersection of the set must not reveal private information.

These examples illustrate the need for the Multiparty Private Set Intersection (MPSI) protocol [1–4]. MPSI is executed by multiple parties who jointly compute the intersection of their private datasets. Ultimately, only designated parties can access this intersection. Previous protocols are impractical because the bulk of the computation depends on the number of players. One previous study required the size of the datasets maintained by the different players to be equal [1, 2]. Another study [3] computed only the approximate number of intersections, whereas other researchers [4] required more than two trusted third-parties.

In this section, we propose a practical MPSI with the following features:

1. The size of the datasets maintained by each party is independent of those maintained by the other parties.
2. The computational complexity for each party is independent of the number of parties. This is accomplished by introducing an outsourcing provider, O. In fact, all computations related to the number of parties are carried out by O. Thus, the number of parties is irrelevant.

3.1.2 Preliminaries

In this section, we summarize the DDH assumption, Bloom filter, and ElGamal encryption. We consider security according to the honest-but-curious model [5]: all players act according to their prescribed actions in the protocol. A protocol that is secure in an honest-but-curious model does not allow any player to gain information about other players' private input sets, besides that that can be deduced from the result of the protocol. Note that the term *adversary* here refers to insiders, i.e., protocol participants. Outsider adversaries are not considered. In fact, behavior by outsider adversaries can be mitigated via standard network security techniques.

Our protocol is based on the following security assumption.

Definition 3.1 (*DDH Assumption*) Let t be a security parameter. A decisional Diffie–Hellman (DDH) parameter generator $\mathcal{IG}$ is a probabilistic polynomial time (PPT) algorithm, a finite field $\mathbb{F}_p$, and a basepoint $g \in \mathbb{F}_p$ with prime order q. We say that $\mathcal{IG}$ satisfies the *DDH assumption* if $|p_1 - p_2|$ is negligible (in κ) for all PPT algorithms A, where $p_1 = \Pr[(\mathbb{F}_p, g) \leftarrow \mathcal{IG}(1^\kappa); y_1 = g^{x_1}, y_2 = g^{x_2} \leftarrow \mathbb{F}_p : A(\mathbb{F}_p, g, y_1, y_2, g^{x_1 x_2}) = 0]$ and $p_2 = \Pr[(\mathbb{F}_p, g) \leftarrow \mathcal{IG}(1^\kappa); y_1 = g^{x_1}, y_2 = g^{x_2}, z \leftarrow \mathbb{F}_p : A(\mathbb{F}_p, g, y_1, y_2, z) = 0]$.

A Bloom filter [6], denoted by BF, consists of m arrays and has a space-efficient probabilistic data structure. The BF can check whether an element x is included in a set S by encoding S with at most w elements. The encoded Bloom filter of S is denoted by $\mathsf{BF}(S)$.

The BF uses a set of k independent uniform hash functions $\mathcal{H} = \{H_0, \ldots, H_{k-1}\}$, where $H_i : \{0,1\}^* \longrightarrow \{0, 1, \ldots, m-1\}$ for $0 \leq \forall i \leq k-1$. The BF consists of two functions: Const embeds a given set S into $\mathsf{BF}(S)$ and ElementCheck checks whether an element x is included in S. SetCheck, an extension of ElementCheck, checks whether an element x in S' is in $S' \cap S$ (see Algorithm 3.3). In Const (see Algorithm 3.1), $\mathsf{BF}(S)$ is constructed for a given set S by first setting all bits in the array to 0. To embed an element $x \in S$ into the filter, the element is hashed using k hash functions to obtain k index numbers, and the bits at these indexes are set to 1, i.e., set $\mathsf{BF}[H_i(x)] = 1$ for $0 \leq i \leq k-1$. In ElementCheck (see Algorithm 3.2), we check all locations where x is hashed; x is considered to be not in S if any bit at these locations is 0; otherwise, x is probably in S.

Some false positive matches may occur, i.e., it is possible that all $\mathsf{BF}[H_i(y)]$ are set to 1, but y is not in S. The false positive rate FPR is given by $\mathrm{FPR} = \left\{1 - \left(1 - \frac{1}{m}\right)^{kw}\right\}^k \approx \left\{1 - e^{-kw/m}\right\}^k$ [7]. However, false negatives are not possible, and so Bloom filters have a 100% recall rate.

Algorithm 3.1 Const(S)

Input: A set S
Output: A Bloom filter $\mathsf{BF}(S)$

```
for i = 0 to m − 1 do
    BF(S)[i] ← 0
end for
for all x ∈ S do
    for i = 0 to k − 1 do
        j = H_i(x)
        if BF(S)[j] = 0 then
            BF(S)[j] ← 1
        end if
    end for
end for
output BF(S). stop.
```

Algorithm 3.2 ElementCheck(BF, x)

Input: A Bloom filter $\mathsf{BF}(S)$, an element x
Output: 1 if $x \in S$ and 0 if $x \notin S$

```
for i = 0 to k − 1 do
    j = H_i(x)
    if BF(S)[j] = 0 then
        output 0. stop.
    end if
end for
output 1. stop.
```

Algorithm 3.3 SetCheck(BF, S')

Input: A Bloom filter $\mathsf{BF}(S)$, a set S'
Output: A set $S_\cap (= S \cap S')$

```
S_∩ ← {}
for all x ∈ S' do
    for i = 0 to k − 1 do
        j = H_i(x)
        if BF[j] = 0 then
            go to next x.
        end if
    end for
    add x to the set S_∩
end for
output S_∩. stop.
```

Homomorphic encryption under addition is useful for processing encrypted data. A typical homomorphic encryption under addition was proposed by Paillier [8]. However, because Paillier encryption cannot reduce the order of a composite group, it is computationally expensive compared with the following ElGamal encryption. Our protocol requires matching without revealing the original messages, for which exponential ElGamal encryption (exElGamal) is sufficient [9]. In fact, the decrypted results of exElGamal encryption can distinguish whether two messages m_1 and m_2 are equal, although the exElGamal scheme cannot decrypt messages itself. Furthermore, exElGamal can be used in (n, n)-threshold distributed decryption [10], where the decryption must be performed by *all players acting together*. An exElGamal encryption with (n, n)-threshold distributed decryption consists of three functions:

Key generation:
Let $\mathbb{F}_p$ be a finite field, $g \in \mathbb{F}_p$, with prime order q. Each player P_i chooses $x_i \in \mathbb{Z}_q$ at random and computes $y_i = g^{x_i} \pmod{p}$. Then, $y = \prod_{i=1}^{n} y_i \pmod{p}$ is a public key and each x_i is a share for each player to decrypt a ciphertext.

Encryption: $\mathsf{thrEnc}[m] \to (u, v)$
Let $m \in \mathbb{Z}_q^*$ be a message. Choose $r \in \mathbb{Z}_q$ at random, and compute both $u = g^r \pmod{p}$ and $v = g^m y^r \pmod{p}$ for the input message $m \in \mathbb{Z}_q$ and a public key y. Output (u, v) as a ciphertext of m.

Decryption: $\mathsf{thrDec}[(u, v)] \to g^m$
Each player P_i computes $z_i = u^{x_i} \pmod{p}$. All players then compute $z = \prod_{i=1}^{n} z_i \pmod{p}$ jointly.[1] Finally, each player can decrypt the ciphertext as $g^m = v/z \pmod{p}$.

ExElGamal encryption with (n, n)-threshold decryption has the following features:
(1) homomorphic under addition: $\mathsf{Enc}(m_1)\mathsf{Enc}(m_2) = \mathsf{Enc}(m_1 + m_2)$ for messages $m_1, m_2 \in \mathbb{Z}_p$.
(2) homomorphic under scalar operations: $\mathsf{Enc}(m)^k = \mathsf{Enc}(km)$ for a message m and $k \in \mathbb{Z}_q$.

3.1.3 Previous Work

This section summarizes prior works on PSI between a server and a client and MPSI among n players. In PSI, let $S = \{s_1, \ldots, s_v\}$ and $C = \{c_1, \ldots, c_w\}$ be server and client datasets, respectively, where $|S| = v$ and $|C| = w$. In MPSI [1], we assume that each player holds the same number of datasets.

PSI protocol based on polynomial representation: The main idea is to represent the elements in C as the roots of a polynomial. The encrypted polynomial is sent to the server, where it is evaluated on the elements in S, as originally proposed by

[1] The computational complexity of z for each player can be made independent of the number of players in various ways. For example, set $z = 1$. P_1 computes $z = z \cdot z_1$ and sends z to P_2, P_2 computes $z = z \cdot z_2$ and sends z to P_3, and, finally, P_n computes $z = z \cdot z_n$ and shares z among all players. If we place all players in a binary tree, the communication complexity can be reduced, but each player's computational complexity is still independent of the number of players.

Freedman [11]. This is secure against honest-but-curious adversaries under secure public key encryption. The computational complexity is $O(vw)$ exponentiations, and the communication overhead is $O(v + w)$. The computational complexity can be reduced to $O(v \log \log w)$ exponentiations using the balanced allocation technique [12]. Kissner and Song extended this protocol to MPSI [1], which requires $O(nw^2)$ exponentiations and $O(nw)$ communication overhead. The MPSI version is secure against honest-but-curious and malicious adversaries (in the random oracle model) using generic zero-knowledge proofs.

PSI protocol based on DH-key agreement: The main objective here is to apply the DH-key agreement protocol [13]: after representing the server and client datasets as hash values $\{h(s_i)\}$ and $\{h(c_i)\}$, respectively, the client encrypts the dataset as $\{h(c_i)^{r_i}\}$ using a random number r_i and sends the encrypted set to the server. The server encrypts the client set $\{h(c_i)^{r_i}\}$ and the server set $\{h(s_i)\}$ using a random number r, which gives $\{h(c_i)^{rr_i}\}$ and $\{h(s_i)^r\}$, respectively, and returns these sets to the client. Finally, the client evaluates $S \cap C$ by decrypting to $\{h(c_i)^r\}$. This is secure against honest-but-curious adversaries under the DDH assumption. The total computational complexity is $O(v + w)$ exponentiations, and the total communication overhead is $O(v + w)$. The security of this approach can be enhanced against malicious adversaries in the random oracle model [14] by using a blind signature. However, no extensions to MPSI based on the DH-key agreement protocol have been proposed.

PSI protocol based on BF: This protocol was originally proposed in [4]. As the Bloom filter itself reveals information about the other player's dataset, the set of players is separated into two groups: input players who have datasets and privacy players who perform private computations under shared secret information. In [15], the privacy of each player's dataset is protected by encrypting each array of the Bloom filter using Goldwasser–Micali encryption [16]. In an honest-but-curious version, the computational complexity is $O(kw)$ hash operations and $O(m)$ public key operations, and the communication overhead is $O(m)$, where m and k are the number of arrays and hash functions, respectively, used in the Bloom filter. The Bloom filter is used in the Oblivious transfer extension [17, 18] and the newly constructed garbled Bloom filter [19]. The main novelty in the garbled Bloom filter is that each array requires λ bits rather than the single bit needed for the conventional Bloom filter. To embed an element $x \in S$ to a garbled Bloom filter, x is split into k shares with λ bits using XOR-based secret sharing ($x = x_1 \bigoplus \cdots \bigoplus x_k$). The x_i are then mapped to an index of $H_i(x)$. An element y is queried by subjecting all bit strings at $H_i(y)$ to an XOR operation. If the result is y, then y is in S; otherwise, y is not in S. The client uses a Bloom filter $\mathsf{BF}(C)$, and the server uses a garbled Bloom filter $\mathsf{GBF}(S)$. If x is in $C \cap S$, then for every position i it hashes to, $\mathsf{BF}(C)[i]$ must be 1 and $\mathsf{GBF}(S)[i]$ must be x_i. Thus, the client can compute $C \cap S$. The computational complexity of this method is $O(kw)$ hash operations and $O(m)$ public key operations, and the communication overhead is $O(m)$. The number of public key operations can be changed to $O(\lambda)$ using the Oblivious transfer extension. This is secure against honest-but-curious adversaries if the Oblivious transfer protocol is secure. Finally, some researchers have computed the approximate number of multiparty set unions [3].

3.1.4 Practical MPSI

This section presents a practical MPSI that is secure under the honest-but-curious model.

3.1.4.1 Notation and Privacy Definition

In the remainder of this paper, the following notations are used.

- P_i: ith player, $i = 1, \ldots, n$
- O: outsourcing provider with no knowledge of the inputs or outputs
- $S_i = \{s_{i,1}, s_{i,2}, \ldots, s_{i,w_i}\}$: dataset held by P_i, where $|S_i| = \omega_i$
- $\cap S_j$: intersection of all n players
- thrEnc and thrDec: (n, n)-threshold exElGamal encryption and decryption, respectively
- m and k: number of arrays and hashes used in BF
- $\boldsymbol{\ell} = [\ell, \ldots, \ell]$ $(1 \leq \ell \leq n)$: an n-dimensional array, where all strings in the array are set to ℓ
- $\mathsf{BF}(S_i) = [\mathsf{BF}_i[0], \ldots, \mathsf{BF}_i[m-1]]$: Bloom filter applied to a set S_i
- $\mathsf{IBF}(\cap S_i) = [\sum_{i=1}^{n} \mathsf{BF}_i[0], \ldots, \sum_{i=1}^{n} \mathsf{BF}_i[m-1]]$: integrated Bloom filter of n sets $\{S_i\}$, where $\sum_{i=1}^{n} \mathsf{BF}_i[j]$ is the sum of all players' arrays

We introduce an outsourcing provider O to reduce the computational burden on all players. The dealer has no information regarding the elements of any player's set. The privacy issues faced by MPSI with an outsourcing provider can be informally written as follows.

Definition 3.2 (*MPSI privacy*) An MPSI scheme with an outsourcing provider O is player-private if the following two conditions hold:

- P_i does not learn anything about the elements of other players' datasets except for the elements that P_i originally possesses.
- the outsourcing provider O does not learn anything about the elements of any player's set.

3.1.4.2 Proposed MPSI

Our MPSI comprises four phases: (i) initialization, (ii) Bloom filter construction and the encryption of P_i data, (iii) the O's randomization of $\mathsf{thrEnc}(\mathsf{IBF}(\cup S_i) - \mathbf{n})$, and (iv) the computation of $\cap \mathsf{P}_i$. The computation of $\cap \mathsf{P}_i$ consists of three steps: (a) joint decryption of an (n, n)-threshold exElGamal among n players, (b) Bloom filter check, and (c) output intersection.

Figure 3.1 shows an overview of our protocol after the initialization phase. The system parameters of a finite field $\mathbb{F}_p$ and a basepoint $g \in \mathbb{F}_p$ with order q for an

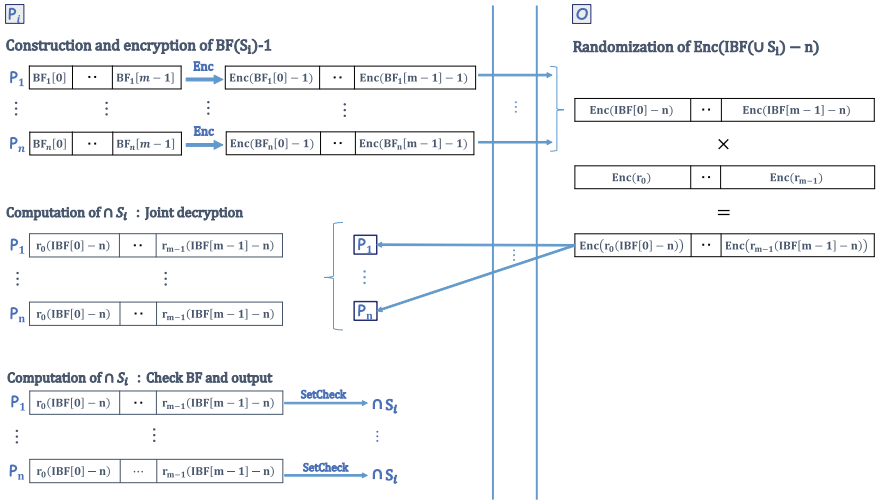

Fig. 3.1 Overview of our MPSI

(n, n)-threshold exElGamal encryption (thrEnc, thrDec) are provided to both P_i and O. For the Bloom filter, $\mathsf{Const}(S)$ and $\mathsf{SetCheck}(\mathsf{BF}, S')$ are only provided to P_i, where the array size is m and k independent hash functions are used.

To encrypt, randomize, or subtract a vector such as a Bloom filter $\mathsf{BF} = [a_0, \ldots, a_{m-1}]$, each location is encrypted, randomized, or subtracted independently:

$$\begin{aligned} \mathsf{thrEnc}(\mathsf{BF}) &= [\mathsf{thrEnc}(a_0), \ldots, \mathsf{thrEnc}(a_{m-1})], \\ \mathbf{r}\mathsf{BF} &= [r_0 a_0, \ldots, r_{m-1} a_{m-1}], \text{ or} \\ \mathsf{BF} - \mathbf{r} &= [a_0 - r_0, \ldots, a_{m-1} - r_{m-1}] \end{aligned}$$

for $\mathbf{r} = [r_0, \ldots, r_{m-1}] \in \mathbb{Z}_q^m$.

Our protocol proceeds as follows.

Initialization:

1. P_i generates $x_i \in \mathbb{Z}_q$, computes $y_i = g^{x_i} \in \mathbb{Z}_q$, and publishes y_i to the other players as a public key, where the corresponding secret key is x_i.
2. P_i computes $y = \prod_i y_i$, where y is the n-player public key. Note that no player knows the corresponding secret key $x = \sum x_i$ before executing the joint decryption.

Construction and encryption of $\mathsf{BF}(S_i) - \mathbf{1}$:

1. P_i executes $\mathsf{Const}(S_i) \longrightarrow \mathsf{BF}(S_i) = [\mathsf{BF}_i[0], \ldots, \mathsf{BF}_i[m-1]]$ (Algorithm 3.1).
2. P_i encrypts $\mathsf{BF}(S_i) - \mathbf{1}$ using thrEnc_y:

$$\mathsf{thrEnc}_y(\mathsf{BF}(S_i) - \mathbf{1}) = [\mathsf{thrEnc}_y(\mathsf{BF}_i[0] - 1), \ldots, \mathsf{thrEnc}_y(\mathsf{BF}_i[m-1] - 1)],$$

where y is an n-player public key.
3. P_i sends $\mathsf{thrEnc}_y(\mathsf{BF}(S_i) - \mathbf{1})$ to O.

Randomization of $\mathsf{thrEnc}(\mathsf{IBF}(\cap S_i) - \mathbf{n})$**:**

1. O encrypts $\mathsf{IBF}(\cap S_i) - \mathbf{n}$ without knowing $\mathsf{IBF}(\cap S_i)$ using an additive homomorphic feature and multiplying by $\mathsf{thrEnc}_y(\mathsf{BF}(S_i) - \mathbf{1})$ as follows:

$$\mathsf{thrEnc}_y(\mathsf{IBF}(\cap S_i) - \mathbf{n}) = \prod_{i=1}^{n} \mathsf{thrEnc}_y(\mathsf{BF}(S_i) - \mathbf{1}).$$

2. O randomizes $\mathsf{thrEnc}_y(\mathsf{IBF}(\cap S_i) - \mathbf{n})$ by $\mathbf{r} = [r_0, \ldots, r_{m-1}] \in \mathbb{Z}_q^m$:

$$\mathsf{thrEnc}_y(\mathbf{r}(\mathsf{IBF}(\cap S_i) - \mathbf{n})) = (\mathsf{thrEnc}_y(\mathsf{IBF}(\cup S_i) - \mathbf{n}))^{\mathbf{r}}.$$

3. O broadcasts $\mathsf{thrEnc}_y(\mathbf{r}(\mathsf{IBF}(\cap S_i) - \mathbf{n}))$ to P_i.

Computation of $\cap S_i$**:**

1. All players decrypt $\mathsf{thrEnc}_y(\mathbf{r}(\mathsf{IBF}(\cap S_i) - \mathbf{n}))$ jointly.
2. P_i computes $\mathsf{SetCheck}(\mathbf{r}(\mathsf{IBF}(\cap S_i) - \mathbf{n}), S_i)$ and obtains $\cap S_i$.

The above protocol satisfies the correctness requirement. This is because each array position of $\mathsf{thrEnc}_y(\mathbf{r}(\mathsf{IBF}(\cap S_i) - \mathbf{n}))$ is decrypted to 1, where $x \in \cap S_i$ is embedded by each hash function; however, each array position for which $x \notin \cap S_i$ is embedded by each hash function is decrypted to a random value.

3.1.4.3 Security Proof

The security of our MPSI protocol is as follows.

Theorem 3.1 *For any coalition of fewer than n players, the MPSI is player-private against an honest-but-curious adversary under the DDH assumption.*

Proof The views of P_i and O, that is,

$$\mathsf{thrEnc}_y(\mathsf{BF}_{m,k}(S_i)) = [\mathsf{thrEnc}_y(\mathsf{BF}_i[0]), \ldots, \mathsf{thrEnc}_y(\mathsf{BF}_i[m-1])],$$

are shown to be indistinguishable from a random vector $\mathbf{r} = [r_0, \ldots, r_{m-1}] \in \mathbb{Z}_q^m$. Assume that a polynomial-time distinguisher $\mathcal{D}$ outputs 0 when the views are presented as a random vector and outputs 1 when they are constructed in MPSI, $\mathsf{thrEnc}(\mathsf{BF}_i[0]), \ldots, \mathsf{thrEnc}(\mathsf{BF}_i[m-1])$. We show that a simulator $\overline{\mathrm{SIM}}$ that solves the DDH assumption can be constructed as follows.

Upon receiving a DDH challenge $(\overline{g}, \overline{g}^{\alpha}, \overline{g}^{\beta}, \overline{g}^{\gamma})$, $\overline{\mathrm{SIM}}$ executes the following:

1. Set n-player public key $y = \overline{g}^{\beta}$ and choose random numbers $d_0, \ldots, d_{m-1}$ and $r_1, \ldots, r_{m-1}$ from $\mathbb{Z}_q$.

2. Send $[(\overline{g}^{\alpha}, \overline{g}^{d_0} \cdot \overline{g}^{\gamma}), ((\overline{g}^{\alpha})^{r_1}, \overline{g}^{d_1} \cdot (\overline{g}^{\gamma})^{r_1}), \ldots, ((\overline{g}^{\alpha})^{r_{m-1}}, \overline{g}^{d_{m-1}} \cdot (\overline{g}^{\gamma})^{r_{m-1}})]$ as $\overline{\mathsf{thrEnc}_y(\mathsf{BF}_{m,k}(S_i))}$ to $\mathcal{D}$.

If $(\overline{g}, \overline{g}^{\alpha}, \overline{g}^{\beta}, \overline{g}^{\gamma})$ is a DH-key-agreement-protocol element, i.e., $\gamma = \alpha\beta$, then $\overline{\mathsf{thrEnc}_y(\mathsf{BF}_{m,k}(S_i))}$ is distributed in the same way as when constructed by the MPSI scheme. Thus, $\mathcal{D}$ must output 1. If $(\overline{g}, \overline{g}^{\alpha}, \overline{g}^{\beta}, \overline{g}^{\gamma})$ is not a DH tuple, then $\overline{\mathsf{thrEnc}_y(\mathsf{BF}_{m,k}(S_i))}$ is randomly distributed, and $\mathcal{D}$ has to output 0. Therefore, $\overline{\mathsf{SIM}}$ can use the output of $\mathcal{D}$ to respond to the DDH challenge correctly. Therefore, $\mathcal{D}$ can answer correctly with negligible advantage over random guessing. Furthermore, as all inputs of each player are encrypted until the decryption is performed, and decryption cannot be performed by fewer than n players, nothing can be learned by any player prior to decryption.

As for the views of $\mathsf{thrEnc}_y(\mathbf{r}(\mathsf{IBF}_{m,k}(\cap S_i) - \mathbf{n}))$, the same argument holds. Therefore, for any coalition of fewer than n players, MPSI is player-private under the honest-but-curious model.

Next, we present d-and-over MPSI. The procedures of d-and-over MPSI are the same as those of MPSI until O computes $\mathsf{thrEnc}_y(\mathsf{IBF}(\cap S_i))$. Thus, we describe the procedure after O computes $\mathsf{thrEnc}_y(\mathsf{IBF}(\cap S_i))$.

Encryption of ℓ-subtraction of $\mathsf{IBF}(\cap S_i)$: O executes the following:

1. Encrypt $\mathsf{IBF}(\cap S_i) - \boldsymbol{\ell}$ randomized by $\mathbf{r} = [r_0, \ldots, r_{m-1}] \in \mathbb{Z}_q^m (d \leq \ell \leq n)$: $\mathsf{thrEnc}_y(\mathbf{r}(\mathsf{IBF}(\cap S_i) - \boldsymbol{\ell})) = (\mathsf{thrEnc}_y(\mathsf{IBF}(\cap S_i)) \cdot \mathsf{thrEnc}_y(-\boldsymbol{\ell}))^{\mathbf{r}}$.
2. Broadcast $\{\mathsf{thrEnc}_y(\mathbf{r}(\mathsf{IBF}(\cap S_i) - \boldsymbol{\ell}))\}_{\ell}$ $(d \leq \ell \leq n)$ to P_i.

d-and-over MPSI computation: P_i executes the following:

1. All P_i jointly decrypt $\{\mathsf{thrEnc}_y(\mathbf{r}(\mathsf{IBF}(\cap S_i) - \boldsymbol{\ell}))\}_{\ell}$.
2. Let CBF_{ℓ} be an m-array for $d \leq \ell \leq n$, where an array is set to 1 if and only if the corresponding array of $\mathbf{r}\mathsf{IBF}(\cap S_i) - \boldsymbol{\ell}$ is 1, and others are set to 0.
3. Set $\mathsf{CBF} = \mathsf{CBF}_{\ell} \vee \cdots \vee \mathsf{CBF}_n$.
4. Execute $\mathsf{SetCheck}_{m,k}(\mathsf{CBF}, S_i) \longrightarrow \cap^{\geq d} S[i]$ and output $\cap^{\geq d} S[i]$.

The correctness of d-and-over MPSI follows from the fact that if an element $x \in \cap^{\ell} S$ for $d \leq \exists \ell \leq n$, the corresponding array locations in $\mathsf{IBF}(\cap S_i) - \mathbf{j}$ for $\ell \leq \exists j \leq n$, where x is mapped by k hashes, are an encryption of 0, which are decrypted to 1; otherwise, it is an encryption of randomized value.

3.1.5 Efficiency

Although many PSI protocols have been proposed, to the best of our knowledge, relatively few consider the multiparty scenario [1–4]. Our target is multiparty private set intersection, and the final result must be obtained by *all* players acting together, without a trusted third-party (TTP). Among previous MPSI protocols, the approach in [3] computes only the approximate number of intersections, and that in [4] requires

Table 3.1 Efficiency of [1] and the proposed protocol

	[1]	Ours
Computational complexity	$O(n\omega^2)$	$\mathsf{P}_i : O(\omega_i), O : O(n\omega)$
Communication overhead	$O(n\omega)$	$\mathsf{P}_i : O(\omega + n), O : O(n\omega)$
Restriction on set size	$\lvert S_1\rvert = \cdots = \lvert S_n\rvert$	None
Protected values	$S_i (\forall i \in [1, n])$	$S_i, \lvert S_i\rvert (\forall i \in [1, n])$

more than two TTPs. In contrast, [2] follows almost the same method as [1] and thus has a similar complexity. The only difference exists in the security model. Hence, we only compare our scheme with that of [1].

The computational and communication efficiency of the proposed protocol and [1] are compared in Table 3.1. These approaches are secure against honest-but-curious adversaries without a TTP under exElGamal encryption (DDH security) and Paillier encryption (Decisional Composite Residue (DCR) security), respectively. The Bloom filter parameters (m, k) used in our protocol are set as follows: $k = 80$ and $m = 80\omega / \ln 2$, where ω is the maximum $|S_i| = \omega_i$. Then, the probability of false positives is given by $p = 2^{-80}$.

Our MPSI uses the Bloom filter for the computations performed by P_i and the integrations performed by the O. The use of a Bloom filter eliminates the restriction on set size. Thus, in our MPSI, the set size of each player is flexible. However, P_i's computations consist of Bloom filter construction, joint decryption, and Bloom filter check. Neither the computations related to the Bloom filter nor the joint decryption depends on the number of players, as shown in Sect. 3.1.2. In summary, the computational complexity of operations performed by P_i is $O(\omega_i)$. All player-dependent data are sent to O, who integrates $\prod_{i=1}^{n} \mathsf{thrEnc}_y(\mathsf{IBF}(\cap S_i))$ without decryption. Therefore, the computational complexity of operations performed by O is $O(n\omega)$.

3.1.6 System and Performance

PSI or MPSI implicitly assumes that every attendee can provide data, any attendee can retrieve data from the shared data, and all attendees can communicate with each other. If PSI or MPSI is implemented straightforwardly, such implementation should become a system like a peer-to-peer (P2P) network system. Although a fully distributed system like P2P network has attractive features, such as high availability and scalability, it incurs some unfavorable features.

The network address and port translation (NAPT) is a major obstacle for P2P network systems. Modern P2P network systems take advantage of NAPT traversal technologies to overcome NAPT, but it should be costly to make the architecture complex. The absence of trusted node is also an obstacle for attendee or group management. Making consensus on a P2P network system is difficult or highly

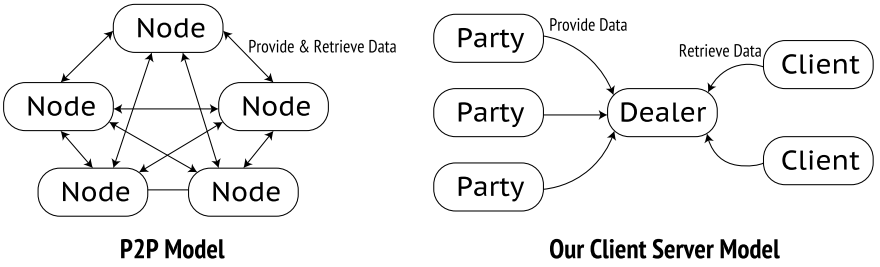

Fig. 3.2 P2P and client server model

costly. Additionally, unpredictable node joining and leaving are reasons that make the P2P network systems complex. To avoid the complexities of P2P networks, we designed a system based on the client server model.

Then, we discuss the design of PSI or MPSI's client server model. There are 2 main functionalities of PSI or MPSI: (1) First, the data sharing is a functionality for sharing data among attendees. (2) Next, the data retrieving from the shared data is a functionality. Any attendee can retrieve data from the shared data, but the retrieving avoids correcting privacy sensitive data by using privacy preserving techniques described above.

However, we do not assume that every attendee provides and retrieves data. Imagine that an incident analysis situation in which data are provided by several organizations which employ labor and operate some machines, and a research institute collects data from the organizations and analyzes it. In such a situation, data providers do not need the data retrieving functionality, and data analysts do not need the data sharing functionality.

Therefore, we define 3 roles for our MPSI application design as follows.

- Parties: entities for data providing
- Clients: entities for data retrieving
- Dealer: an entity for forwarding requests between parties and clients

From the perspective of privilege separation, defining and separating roles are significant. Figure 3.2 shows a P2P network model and our client server model. As show in this figure, every P2P network node is connected to each other and can provide and retrieve data, but parties only provide data and clients only retrieve data in the client server model. The dealer forwards requests from parties and clients and provides other functionalities that are not specified by PSI or MPSI. For example, attendee or group management, user authentication, and data logging should be performed by the dealer.

Figure 3.3 shows an example sequence diagram of our MPSI application. In this figure, there are 2 parties, 1 client, and 1 dealer. First of all, parties 1 and 2 join the dealer (join p1 and p2). A party must join before providing data, and it must be performed only once at initialization. After that, the client sends a request of data retrieval to the dealer (cl req), and parties send a request to confirm whether the dealer

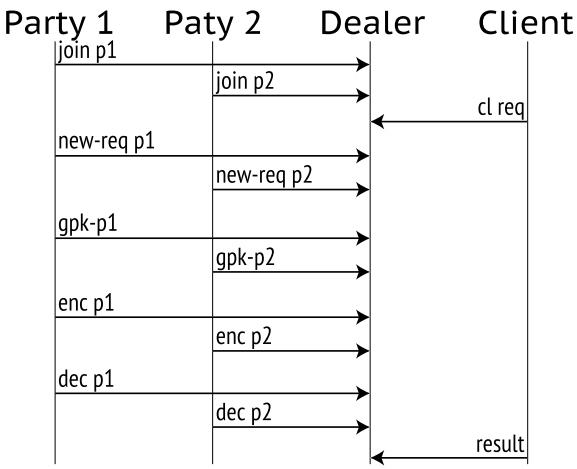

Fig. 3.3 Sequence diagram of MSPI application

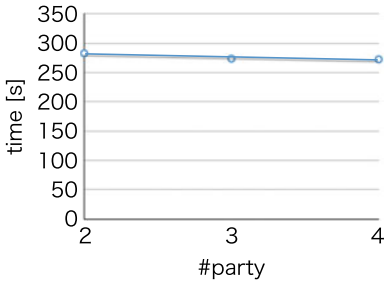

Fig. 3.4 Performance

received data retrieval requests by clients (new-req p1 and p2). Then, the parties and the dealer generate keys, share the keys, encrypt data, and decrypt data (gpk p1 and p2, enc p1 and p2, and dec p1 and p2). Finally, the client gets the result from the dealer.

We measured performance of our MPSI application written in Python language on an Amazon's EC2 server (2.4 GHz CPU, 1 GB Memory). Figure 3.4 shows the results when there are from 2 to 4 parties which provide data including 10,000 entries. The results show that it takes approximately 280 s to accomplish data retrieval and that the computational amount does not depend on the number of parties.

3.2 Classification

In this section, we present a secure classification protocol, a type of secure computation protocols. We assume two participants Alice and Bob of the protocol. Alice has private data x, and Bob has a classification model C. The task is that Alice learns $C(x)$ at the end of the protocol while preserving the privacy of x and C. That is, Alice can learn only $C(x)$ and Bob can learn nothing. Our construction is based on a code-based public-key encryption scheme called HQC [20], which is a candidate of NIST's Post-Quantum Cryptography standardization [21].

3.2.1 Error-Correcting Code

We start with several fundamental notions for error-correcting codes.

Definition 3.3 (*Linear code*) A code $\mathbb{C}$ such that $c_1 + c_2 \in \mathbb{C}$ always holds for any codeword $c_1, c_2 \in \mathbb{C}$ is called a linear code. The code $\mathbb{C}$ of code length n and information bit number k is described as "a" code.

Definition 3.4 (*Generation matrix*) For matrices $\mathbb{G} \in \mathbb{F}^{k\times n}$,$\mathbb{G}$ that satisfy

$$\mathbb{C} = \{\boldsymbol{m} \cdot \mathbb{G} | \boldsymbol{m} \in \mathbb{F}^k\} \tag{3.1}$$

is called a generator matrix. The generator matrix is the basis of linear codes and generates all codewords.

Definition 3.5 (*Parity check matrix*) For a matrix $\mathbf{H} \in \mathbb{F}^{(n-k)\times n}$, $\mathbf{H}$ that satisfies

$$\mathbb{C} = \{\boldsymbol{x} \in \mathbb{F}^n | \mathbf{H} \cdot \boldsymbol{x}^\top = \mathbf{0}\} \tag{3.2}$$

is called a parity check matrix.

Definition 3.6 (*Cyclic matrix*) When $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{F}^n$, the circulant matrix for $\boldsymbol{x}$ is defined as

$$\mathbf{rot}(\boldsymbol{x}) = \begin{pmatrix} x_1 & x_n & \cdots & x_2 \\ x_2 & x_1 & \cdots & x_3 \\ \vdots & \vdots & \ddots & \vdots \\ x_n & x_{n-1} & \cdots & x_1 \end{pmatrix} \in \mathbb{F}^{n\times n} \tag{3.3}$$

In addition, the multiplication of two polynomials x, y has the following properties:

$$\begin{aligned} \boldsymbol{x} \cdot \boldsymbol{y} &= \boldsymbol{x} \times \mathbf{rot}(\boldsymbol{y})^\top \\ &= (\mathbf{rot}(\boldsymbol{x}) \times \boldsymbol{y}^\top)^\top \\ &= \boldsymbol{y} \times \mathbf{rot}(\boldsymbol{x})^\top \\ &= \boldsymbol{y} \cdot \boldsymbol{x}. \end{aligned} \tag{3.4}$$

Definition 3.7 (*Cyclic shift*) The operation of shifting $(c_0, \ldots, c_{n-1})$ to the right by one position with respect to n-dimensional vector c_i $(i = 0, \ldots, n-2)$ and moving c_{n-1} to the beginning of the vector is called cyclic shift. That is, for any n dimensional vector $(c_0, \ldots, c_{n-1})$, it is a mapping $\sigma : (c_0, c_1, \ldots, c_{n-1}) \mapsto (c_{n-1}, c_0, \ldots, c_{n-2})$.

Definition 3.8 (*Quasi-cyclic code*) Let $\boldsymbol{c} = (\boldsymbol{c}_0, \ldots, \boldsymbol{c}_{s-1}) \in (\mathbb{F}_2^n)^s$ be an arbitrary codeword of code $\mathbb{C}$ and let σ be a cyclic shift operation. If $(\sigma(\boldsymbol{c}_0), \ldots, \sigma(\boldsymbol{c}_{s-1}) \in \mathbb{C}$, $\mathbb{C}$ is called the s-quasi-cyclic code. In particular, when $s = 1$, $\mathbb{C}$ is called a cyclic code.

Definition 3.9 (*Systematic quasi-cyclic code*) An s-quasi-cyclic $[sn, n]$ code is called a systematic quasi-cyclic code if it has a parity check matrix of the form.

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{I}_n & 0 & \cdots & 0 & \boldsymbol{A}_1 \\ 0 & \boldsymbol{I}_n & & & \boldsymbol{A}_2 \\ & & \ddots & & \vdots \\ 0 & & \cdots & \boldsymbol{I}_n & \boldsymbol{A}_{s-1} \end{bmatrix} \tag{3.5}$$

Here, $\boldsymbol{A}_1, \ldots, \boldsymbol{A}_{s-1}$ is an $n \times n$ circulant matrix.

3.2.2 Security Assumptions

As mentioned above, the security of the public-key cryptosystem HQC is based on the computational difficulty of the quasi cyclic syndrome decoding problem. More specifically, its security is proved under the following quasi cyclic syndrome decoding decision assumptions.

Definition 3.10 (*quasi-cyclic syndrome decoding assumption*) The quasi-cyclic syndrome decoding decision problem of a s-quasi-cyclic code in which n and w are integers and the number of blocks is $s \geq 2$ is $(\mathbf{H}, \boldsymbol{y}^\top)$ when the parity check matrix $\mathbf{H} \xleftarrow{\$} \mathbb{F}^{(sn-n)\times sn}$ and the matrix $\boldsymbol{y} \xleftarrow{\$} \mathbb{F}^{sn-n}$ of random systematic quasi-cyclic code are given, every efficient algorithm distinguish only with negligible probability whether it is quasi-cyclic syndrome decoding distribution or the uniform distribution over $\mathbb{F}^{(sn-n)\times sn} \times \mathbb{F}^{(sn-n)}$.

As will be described later, since the security of the secure computation protocol proposed in this section is reduced to the security of HQC, the secure computation protocol of this section is proved to be secure under this assumption as well as under HQC.

3.2.3 Security Requirements for 2PC

Secure two-party computation is a subproblem of multi-party secure computation. The studies have been conducted by many researchers since it is closely related to many cryptographic protocols. The purpose of 2PC is to construct a general-purpose protocol so that arbitrary functions can be jointly computed without sharing the input values of the two parties with the other. One of the best-known examples of 2PCs is the millionaire problem [22] in Yao, where Alice and Bob do not reveal their money and decide who is richer. Specifically, suppose that Alice has a yen, and Bob has b yen. The problem is to decide whether $a \geq b$ or not while keeping each other secret. Generally speaking, the security requirement of 2PC is that the computation of any function is performed using a protocol without leaking the two inputs to the other, and only the computation result is known.

A two-party linear function evaluation is a kind of 2PC that satisfies the 2PC security requirements. In other words, the participants perform the evaluation without notifying the other party of their input. In addition, the function of the protocol is the evaluation of linear functions. Specifically, linear function secure computation protocol computes $f(m) = a \cdot m + b$. The participants in the protocol are called Alice and Bob. Alice's input is m, and Bob's input is linear function parameters a, b. Alice gets only the result of $f(m) = a \cdot m + b$ through the protocol, and Bob gets nothing.

Below we define the security requirements for two-party linear function secure computation.

Definition 3.11 (*Security against semi-honest adversaries*) Let $f = (f_A, f_B)$ be the function that maps the input x of Alice(A) and the input y of Bob(B) to $f_A(x, y), f_B(x, y)$. A aims to obtain $f_A(x, y)$ and B aims to obtain $f_B(x, y)$.

Let $f = (f_A, f_B)$ be a function of probabilistic polynomial time, and π be a two-way protocol for computing function f. Let the view of A with (x, y) execution $\pi(x, y)$ and the security parameter n be $\text{view}_A^\pi(x, y, n)$ and the view of B be $\text{view}_B^\pi(x, y, n)$. The output of A is $\text{output}_A^\pi(x, y, n)$ and the output of B is $\text{output}_B^\pi(x, y, n)$. In addition, the joint output of the two is denoted as $\text{output}^\pi(x, y, n) = (\text{output}_A^\pi(x, y, n), \text{output}_B^\pi(x, y, n))$.

For semi-honest adversaries, we say that the protocol $\pi(x, y)$ can securely compute the function f if there are probabilistic polynomial-time algorithms S_A and S_B that satisfy the following equations. For any x, y that satisfy $|x| = |y| = n, n \in \mathbb{N}$, the following holds:

$$
\begin{aligned}
&\{(S_A(1^n, x, f_A(x, y)), f(x, y))\}_{x,y,n} \\
\overset{c}{\equiv}&\{(\text{view}_A^\pi(x, y, n), \text{output}^\pi(x, y, n))\}_{x,y,n}, \\
&\{(S_B(1^n, x, f_B(x, y)), f(x, y))\}_{x,y,n} \\
\overset{c}{\equiv}&\{(\text{view}_B^\pi(x, y, n), \text{output}^\pi(x, y, n))\}_{x,y,n}.
\end{aligned}
$$

3.2.4 HQC Encryption Scheme

The protocols proposed in this section are based on the Hamming Quasi-Cyclic cryptosystem of Gaborit et al. First, we introduce the cryptosystem proposed by Gaborit et al. [20], which is a public key cryptosystem based on the quasi-cyclic syndrome decoding problem. In this cryptosystem, two kinds of codes quasi-cyclic code and error-correcting code $\mathbb{C}$ are used. The error-correcting code $\mathbb{C}$ is an arbitrary linear code (such as a BCH code) used for message encoding and decoding and with sufficient error correction capability. A quasi-cyclic code is used for a security requirement of this public key cryptosystem to generate noise that an adversary cannot decrypt.

The participants of the HQC cryptosystem are Alice (A) and Bob (B), and B aims to send the input message $\boldsymbol{m}$ securely to A. The cryptosystem is performed as follows:

1. Global parameter settings:
   Parameters param = $(n, k, \delta, w_x, w_r, w_e)$ and the sign $\mathbb{C}$ generation matrix $\mathbb{G} \in \mathbb{F}^{k \times n}$.
2. Key generation:
   A generates random $\boldsymbol{h} \xleftarrow{\$} \mathbb{R}$.
   Furthermore, $(\boldsymbol{x}, \boldsymbol{y}) \xleftarrow{\$} \mathbb{R}^2$ is generated, and the Hamming weight of $\boldsymbol{x}, \boldsymbol{y}$ is w_x. Secret information sk = $(\boldsymbol{x}, \boldsymbol{y})$ Public information pk = $(\boldsymbol{h}, \boldsymbol{s} = \boldsymbol{x} + \boldsymbol{h} \cdot \boldsymbol{y})$. A sends public information pk to B.
3. Encryption:
   B generates a random $\boldsymbol{e} \xleftarrow{\$} \mathbb{R}$, $(\boldsymbol{r_1}, \boldsymbol{r_2}) \xleftarrow{\$} \mathbb{R}^2$.
   The Hamming weight of $\boldsymbol{e}$ is w_e, and the Hamming weight of $\boldsymbol{r_1}$ and $\boldsymbol{r_2}$ is w_r.
   Then, we compute $\boldsymbol{u} = \boldsymbol{r_1} + \boldsymbol{h} \cdot \boldsymbol{r_2}$ and $\boldsymbol{v} = \boldsymbol{m} \cdot \mathbb{G} + \boldsymbol{s} \cdot \boldsymbol{r_2} + \boldsymbol{e}$ on input $\boldsymbol{m}$. B sends the ciphertext $\boldsymbol{u}, \boldsymbol{v}$ back to A.
4. Decryption:
   A uses the decoding function $\mathbb{C}$.Decode$(\boldsymbol{v} - \boldsymbol{u} \cdot \boldsymbol{y})$ of the error-correcting code $\mathbb{C}$ to recover the message $\boldsymbol{m}$ of B.

In the HQC cryptosystem, public information $\boldsymbol{s}$ is added to the message $\boldsymbol{m}$ encoded by the error-correcting code when it is encrypted. Since $\boldsymbol{s}$ is noise with a large Hamming weight generated by the quasi-cyclic code, security is guaranteed by the quasi-cyclic syndrome decoding decision assumption introduced above. In addition, A can use the secret key for the encrypted error-protected ciphertext in the decryption stage, and can remove a large amount of noise from $\boldsymbol{s}$. However, some noise of $\boldsymbol{x} \cdot \boldsymbol{r_2} - \boldsymbol{r_1} \cdot \boldsymbol{y} + \boldsymbol{e}$ remains. If the weight of this noise is smaller than the maximum number of correctable errors δ of the error-correcting code, correct decoding is possible. Hamming weights $w, w_r, w_e = O(\sqrt{n})$ are assumed and analyzed. Moreover, the conclusion that the probability of becoming $\omega(\boldsymbol{x} \cdot \boldsymbol{r_2} + \boldsymbol{e} - \boldsymbol{y} \cdot \boldsymbol{r_1}) \leq \delta$ increases as the code space n becomes larger is shown in the paper of Gaborit et al. In addition, the HQC cryptosystem is IND-CPA secure under the quasi-cyclic syndrome decoding decision assumption.

3.2.5 Proposed Protocol

3.2.5.1 Linear Function Evaluation

We introduce the secure evaluation protocol of the linear functions between two parties.

We use two codes, quasi-cyclic code and arbitrary error-correcting code $\mathbb{C}$, based on Gaborit's HQC cryptosystem. The participants in the protocol are Alice (A) and Bob (B). A's input is $m \in \mathbb{F}_2$, B's input is $a, b \in \mathbb{F}_2$, B's output is nothing, and A's output is $a \cdot m + b$. The protocol is given in Protocol 3.2.5.1.

Protocol Linear function evaluation protocol

input	A: $m \in \mathbb{F}_2$
	B: $a, b \in \mathbb{F}_2$
output	A: $a \cdot m + b$
	B: $\perp$

1. Global parameter param = $(n, k, \delta, w_x, w_r, w_e)$ and the sign $\mathbb{C}$ generation matrix $\mathbb{G} \in \mathbb{F}^{k \times n}$ are chosen.
2. A generates the random $\boldsymbol{h} \xleftarrow{\$} \mathbb{R}$. Furthermore, $(\boldsymbol{x}, \boldsymbol{y} \xleftarrow{\$} \mathbb{R}^2)$ is generated, and the Hamming weight of $\boldsymbol{x}$ and $\boldsymbol{y}$ is w. Secret information sk = $(\boldsymbol{x}, \boldsymbol{y})$, Public information pk = $(\boldsymbol{h}, \boldsymbol{s} = \boldsymbol{x} + \boldsymbol{h} \cdot \boldsymbol{y})$.
3. By padding the input m with 0, A makes $\boldsymbol{m} = (m, 0, \ldots, 0)$ of dimension k. A generates a random $\boldsymbol{r_A}, \boldsymbol{r_u}, \boldsymbol{r_v} \xleftarrow{\$} \mathbb{R}$. Here, the Hamming weight of $\boldsymbol{r_A}, \boldsymbol{r_u}, \boldsymbol{r_v}$ is w_r. Then, we compute $(\boldsymbol{u} = \boldsymbol{h} \cdot \boldsymbol{r_A} + \boldsymbol{r_u}, \boldsymbol{v} = \boldsymbol{m} \cdot \mathbb{G} + \boldsymbol{s} \cdot \boldsymbol{r_A} + \boldsymbol{r_v})$. A sends public information $\boldsymbol{h}, \boldsymbol{s}$ and ciphertext pair $\boldsymbol{u}, \boldsymbol{v}$ to B.
4. Let B be $\boldsymbol{b} = (b, 0, \ldots, 0)$. Generate $\boldsymbol{r_B} \xleftarrow{\$} \mathbb{R}$ and $(\boldsymbol{e_u}, \boldsymbol{e_v}) \xleftarrow{\$} \mathbb{R}^2$. Here, the Hamming weight of $\boldsymbol{r_B}$ is w_r, and the Hamming weight of $\boldsymbol{e_u}$ and $\boldsymbol{e_v}$ is w_e. B computes $\boldsymbol{u'} = a \cdot \boldsymbol{u} + \boldsymbol{h} \cdot \boldsymbol{r_B} + \boldsymbol{e_u}$ and $\boldsymbol{v'} = a \cdot \boldsymbol{v} + \boldsymbol{b} \cdot \mathbb{G} + \boldsymbol{s} \cdot \boldsymbol{r_B} + \boldsymbol{e_v}$. B sends $\boldsymbol{u'}, \boldsymbol{v'}$ back to A.
5. A uses $\mathbb{C}$. Decode$(\boldsymbol{v'} - \boldsymbol{u'} \cdot \boldsymbol{y})$ to decode the error-correcting code $\mathbb{C}$, and recovers $a \cdot m + b$ by taking the first bit of the result.

First, we set global parameters. n is the code length of the code, k is the number of information bits, δ is the maximum number of correctable errors in the error-correcting code, and w_x, w_r, w_e are Hamming weights set in advance. For example, it is half the weight of $O(\sqrt{n})$ assumed by Gaborit et al. The public parameter $\mathbb{G}$ is a generator matrix of error-correcting code $\mathbb{C}$, which maps messages and codewords as $\mathbb{F}_2^k \to \mathbb{F}_2^n$.

A generates random $\boldsymbol{h} \xleftarrow{\$} \mathbb{R}$ and $(\boldsymbol{x}, \boldsymbol{y}) \xleftarrow{\$} \mathbb{R}^2$ and computes $\mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y}$. Here,

$$\begin{aligned} \boldsymbol{s} &= \boldsymbol{x} + \boldsymbol{h} \cdot \boldsymbol{y} \\ &= \boldsymbol{x} + \boldsymbol{y} \cdot \mathbf{rot}(\boldsymbol{h})^{\top} \\ &= (\boldsymbol{x}\ \boldsymbol{y})(\boldsymbol{I}_n\ \mathbf{rot}(\boldsymbol{h}))^{\top}. \end{aligned} \tag{3.6}$$

It can be converted to and can be reduced to the quasi cyclic syndrome decoding problem. Then, A sets secret information sk as $(\boldsymbol{x}, \boldsymbol{y})$ and public information pk as $(\boldsymbol{h}, \boldsymbol{s})$.

A pads the input m with 0, making $\boldsymbol{m} = (m, 0, \ldots, 0)$ with dimension k. A generates $\boldsymbol{r_A}, \boldsymbol{r_u}, \boldsymbol{r_v} \xleftarrow{\$} \mathbb{R}$, encodes the value of $\boldsymbol{m}$ with an error-correcting code, and re-randomizes it. A generates a ciphertext pair of $(\boldsymbol{u} = \boldsymbol{h} \cdot \boldsymbol{r_A} + \boldsymbol{r_u}, \boldsymbol{v} = \boldsymbol{m} \cdot \mathbb{G} + \boldsymbol{s} \cdot \boldsymbol{r_A} + \boldsymbol{r_v})$ and send it to B. As for B, $\boldsymbol{v}$ has a noise $\boldsymbol{s}$ that cannot be decoded, and has no secret information that can be removed, so B cannot learn $\boldsymbol{m}$.

B sets $\boldsymbol{b} = (b, 0, \ldots, 0)$ and generates $\boldsymbol{r_B} \xleftarrow{\$} \mathbb{R}$ and $(\boldsymbol{e_u}, \boldsymbol{e_v}) \xleftarrow{\$} \mathbb{R}^2$. B produces $\boldsymbol{u'} = a \cdot \boldsymbol{u} + \boldsymbol{h} \cdot \boldsymbol{r_B} + \boldsymbol{e_u}, \boldsymbol{v'} = a \cdot \boldsymbol{v} + \boldsymbol{b} \cdot \mathbb{G} + \boldsymbol{s} \cdot \boldsymbol{r_B} + \boldsymbol{e_v}$ and re-randomize $\boldsymbol{u}$ and $\boldsymbol{v}$ after updating. Since the error-correcting code is a linear code, $\boldsymbol{u'}$ and $\boldsymbol{v'}$ after update are

$$\boldsymbol{u'} = \begin{cases} \boldsymbol{h} \cdot \boldsymbol{r_B} + \boldsymbol{e_u} & \text{(In the case of a = 0)} \\ \boldsymbol{u} + \boldsymbol{h} \cdot \boldsymbol{r_B} + \boldsymbol{e_u} & \text{(In the case of a = 1).} \end{cases} \tag{3.7}$$

$$\boldsymbol{v'} = \begin{cases} \boldsymbol{b} \cdot \mathbb{G} + \boldsymbol{s} \cdot \boldsymbol{r_B} + \boldsymbol{e_v} & \text{(In the case of a = 0)} \\ \boldsymbol{v} + \boldsymbol{b} \cdot \mathbb{G} + \boldsymbol{s} \cdot \boldsymbol{r_B} + \boldsymbol{e_v} & \text{(In the case of a = 1).} \end{cases} \tag{3.8}$$

Finally, A uses his secret information to decrypt $\boldsymbol{v'} - \boldsymbol{u'} \cdot \boldsymbol{y}$. The result is

$$\begin{aligned} &\boldsymbol{v'} - \boldsymbol{u'} \cdot \boldsymbol{y} \\ =&(a\boldsymbol{m} + \boldsymbol{b})\mathbb{G} + \boldsymbol{x}(a\boldsymbol{r_A} + \boldsymbol{r_B}) - \boldsymbol{y}(a\boldsymbol{r_u} + \boldsymbol{e_u}) + (a\boldsymbol{r_v} + \boldsymbol{e_v}) \\ =&\begin{cases} \boldsymbol{b}\mathbb{G} + \boldsymbol{x}\boldsymbol{r_B} - \boldsymbol{y}\boldsymbol{e_u} + \boldsymbol{e_v} & \text{(in the case of a = 0)} \\ (\boldsymbol{m} + \boldsymbol{b})\mathbb{G} + \boldsymbol{x}(\boldsymbol{r_A} + \boldsymbol{r_B}) - \boldsymbol{y}(\boldsymbol{r_u} + \boldsymbol{e_u}) + (\boldsymbol{r_v} + \boldsymbol{e_v}) & \\ \qquad \text{(in the case of a = 1).} & \end{cases} \end{aligned} \tag{3.9}$$

As shown by the Eq. (3.9), the result of $\boldsymbol{v'} - \boldsymbol{u'} \cdot \boldsymbol{y}$ is the result of removing $\boldsymbol{h}$ and $\boldsymbol{s}$. Taking the first bit makes $a \cdot m + b$ available to A.

3.2.5.2 Correctness and Security of the Proposed Protocol

The correctness of the two-way linear function evaluation protocol proposed in this study obviously depends on the decoding ability of the code $\mathbb{C}$. Specifically, assuming that $\mathbb{C}$. Decode decodes $\boldsymbol{v} - \boldsymbol{u} \cdot \boldsymbol{y}$ correctly, the following equation is satisfied:

$$\mathrm{Decrypt}(sk, \mathrm{Encrypt}(pk, a \cdot \boldsymbol{m} + \boldsymbol{b})) = a \cdot \boldsymbol{m} + \boldsymbol{b}. \tag{3.10}$$

Also, let ϵ be the error of $\boldsymbol{v} - \boldsymbol{u} \cdot \boldsymbol{y}$. The error is

$$\epsilon = \begin{cases} \boldsymbol{x}\boldsymbol{r}_B - \boldsymbol{y}\boldsymbol{e}_u + \boldsymbol{e}_v & \text{(In the case of a = 0)} \\ \boldsymbol{x}(\boldsymbol{r}_A + \boldsymbol{r}_B) - \boldsymbol{y}(\boldsymbol{r}_u + \boldsymbol{e}_u) + (\boldsymbol{r}_v + \boldsymbol{e}_v) & \\ & \text{(In the case of a = 1)} \end{cases} \tag{3.11}$$

for the error correction capability of the code $\mathbb{C}$. In the paper of Gaborit et al., $\mathbb{C}$.Decode can work correctly when $\omega(\boldsymbol{x} \cdot \boldsymbol{r_2} + \boldsymbol{e} - \boldsymbol{y} \cdot \boldsymbol{r_1}) \leq \delta$ is satisfied, and w_r and w_e have the same value when actually evaluated. If the Hamming weight of $\boldsymbol{r_0}, \boldsymbol{r_1}, \boldsymbol{r_u}, \boldsymbol{r_v}, \boldsymbol{r_B}$ of the protocol proposed in this section is set to $1/2$ of w_r of Gaborit et al., then, the Hamming weight of $\boldsymbol{e_u}, \boldsymbol{e_v}$ is set to $1/2$ of w_e of Gaborit et al. The Hamming weight of the error Eq. (3.11) is less than or equal to the Hamming weight of errors in Gaborit et al.'s setting. Therefore, the conclusion of the paper of Gaborit et al. also holds for the proposed protocol. As the code length n increases, the decoding failure rate of the error-correcting code decreases. If the appropriate code space size n and noise Hamming weights w_r and w_e are set, the decoding failure rate approaches 0.

The security requirements of the proposed protocol are described above. In this section, we prove the security against semi-honest adversaries.

Theorem 3.2 *Under the quasi-cyclic syndrome decoding assumption, the 2PC protocol securely computes linear functions for semi-honest adversaries.*

Proof First, consider the semi-honest adversary A. With the global parameter omitted, the view of A is $\text{view}_A = (\boldsymbol{m}; \boldsymbol{h}, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{r_0}, \boldsymbol{r_1}, \boldsymbol{r_u}, \boldsymbol{r_v}; \boldsymbol{u'}, \boldsymbol{v'})$. We construct a simulator $S_A(\boldsymbol{m}, \boldsymbol{x}, \boldsymbol{y})$ as follows:

1. Generate $\widetilde{\boldsymbol{h}}, \widetilde{\boldsymbol{r_0}}, \widetilde{\boldsymbol{r_A}}, \widetilde{\boldsymbol{r_u}}, \widetilde{\boldsymbol{r_v}}, \widetilde{\boldsymbol{u'}}, \widetilde{\boldsymbol{v'}} \xleftarrow{\$} \mathbb{R}$ randomly.
   Here, the Hamming weight of $\widetilde{\boldsymbol{r_A}}, \widetilde{\boldsymbol{r_u}}, \widetilde{\boldsymbol{r_v}}$ is w_r.
2. Output $(\boldsymbol{m}, \boldsymbol{x}, \boldsymbol{y}; \widetilde{\boldsymbol{h}}, \widetilde{\boldsymbol{r_A}}, \widetilde{\boldsymbol{r_u}}, \widetilde{\boldsymbol{r_v}}; \widetilde{\boldsymbol{u'}}, \widetilde{\boldsymbol{v'}})$.

Since, $\boldsymbol{h}, \boldsymbol{r_A}, \boldsymbol{r_u}, \boldsymbol{r_v}$ and $\widetilde{\boldsymbol{h}}, \widetilde{\boldsymbol{r_A}}, \widetilde{\boldsymbol{r_u}}, \widetilde{\boldsymbol{r_v}}$ follow the same distribution, the following equation holds:

$$\begin{aligned} & (\boldsymbol{m}, \boldsymbol{x}, \boldsymbol{y}; \widetilde{\boldsymbol{h}}, \widetilde{\boldsymbol{r_A}}, \widetilde{\boldsymbol{r_u}}, \widetilde{\boldsymbol{r_v}}; \widetilde{\boldsymbol{u'}}, \widetilde{\boldsymbol{v'}}) \\ \equiv_s \ & (\boldsymbol{m}, \boldsymbol{x}, \boldsymbol{y}; \boldsymbol{h}, \boldsymbol{r_A}, \boldsymbol{r_u}, \boldsymbol{r_v}; \widetilde{\boldsymbol{u'}}, \widetilde{\boldsymbol{v'}}). \end{aligned} \tag{3.12}$$

At view_A, $\boldsymbol{u'} = a \cdot \boldsymbol{u} + \boldsymbol{h} \cdot \boldsymbol{r_B} + \boldsymbol{e_u}$, $\boldsymbol{v'} = a \cdot \boldsymbol{v} + \boldsymbol{b} \cdot \mathbb{G} + \boldsymbol{s} \cdot \boldsymbol{r_B} + \boldsymbol{e_v}$, and it holds

$$\begin{bmatrix} \boldsymbol{h} \cdot \boldsymbol{r_B} + \boldsymbol{e_u} \\ \boldsymbol{s} \cdot \boldsymbol{r_B} + \boldsymbol{e_v} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_n & 0 & \text{rot}(\boldsymbol{h}) \\ 0 & \boldsymbol{I}_n & \text{rot}(\boldsymbol{s}) \end{bmatrix} \begin{bmatrix} \boldsymbol{e_u} \\ \boldsymbol{e_v} \\ \boldsymbol{r_B} \end{bmatrix}. \tag{3.13}$$

Therefore, the adversary of probabilistic polynomial time cannot distinguish between $(\boldsymbol{h} \cdot \boldsymbol{r_B} + \boldsymbol{e_u}, \boldsymbol{s} \cdot \boldsymbol{r_B} + \boldsymbol{e_v})$ and uniform random numbers under the assumption of 3-quasi-cyclic syndrome decoding of quasi-cyclic code. Since $\boldsymbol{u}$ and $\boldsymbol{v}$ are also under the 3-quasicyclic syndrome decoding decision assumption, they cannot distinguish between $\boldsymbol{u}$ and $\boldsymbol{v}$ and uniform random numbers. Thus, the distribution

of $\boldsymbol{u}'$ and $\boldsymbol{v}'$ also approaches uniform random numbers and satisfies the following equation:

$$\begin{aligned} &(\boldsymbol{m}, \boldsymbol{x}, \boldsymbol{y}; \boldsymbol{h}, \boldsymbol{r}_A, \boldsymbol{r}_u, \boldsymbol{r}_v, \widetilde{\boldsymbol{u}'}, \widetilde{\boldsymbol{v}'}) \\ \equiv_c\ &(\boldsymbol{m}, \boldsymbol{x}, \boldsymbol{y}; \boldsymbol{h}, \boldsymbol{r}_A, \boldsymbol{r}_u, \boldsymbol{r}_v, \boldsymbol{u}', \boldsymbol{v}'). \end{aligned} \tag{3.14}$$

Thus, the distributions of the view view_A of A and the simulator S_A are indistinguishable against polynomial-time adversaries:

$$\begin{aligned} &S_A(\boldsymbol{m}, \boldsymbol{x}, \boldsymbol{y}) \\ \equiv_c\ &\mathrm{view}_A(\boldsymbol{m}, \boldsymbol{x}, \boldsymbol{y}; \boldsymbol{h}, \boldsymbol{r}_A, \boldsymbol{r}_u, \boldsymbol{r}_v; \boldsymbol{u}', \boldsymbol{v}'). \end{aligned} \tag{3.15}$$

Next, consider the semi-honest adversary B. With the global parameter omitted, the view of B is $\mathrm{view}_B = (a, b; \boldsymbol{h}, \boldsymbol{s}, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{r}_B, \boldsymbol{e}_u, \boldsymbol{e}_v)$. Configure the simulator $S_B(a, b)$ as follows:

1. Randomly generate $\widetilde{\boldsymbol{h}}, \widetilde{\boldsymbol{s}}, \widetilde{\boldsymbol{u}}, \widetilde{\boldsymbol{v}}, \widetilde{\boldsymbol{r}_B}, \widetilde{\boldsymbol{e}_u}, \widetilde{\boldsymbol{e}_v} \xleftarrow{\$} \mathbb{R}$. Here, the Hamming weight of $\widetilde{\boldsymbol{r}_B}$ is w_r, and the Hamming weight of $\widetilde{\boldsymbol{e}_u}$ and $\widetilde{\boldsymbol{e}_v}$ is w_e
2. Output $(a, b; \widetilde{\boldsymbol{h}}, \widetilde{\boldsymbol{s}}, \widetilde{\boldsymbol{u}}, \widetilde{\boldsymbol{v}}, \widetilde{\boldsymbol{r}_B}, \widetilde{\boldsymbol{e}_u}, \widetilde{\boldsymbol{e}_v})$.

Since, $\boldsymbol{h}, \boldsymbol{r}_B, \boldsymbol{r}_u, \boldsymbol{r}_v$ and $\widetilde{\boldsymbol{h}}, \widetilde{\boldsymbol{r}_B}, \widetilde{\boldsymbol{r}_u}, \widetilde{\boldsymbol{r}_v}$ follow the same distribution, the following equation holds:

$$\begin{aligned} &(a, b; \widetilde{\boldsymbol{h}}, \widetilde{\boldsymbol{s}}, \widetilde{\boldsymbol{u}}, \widetilde{\boldsymbol{v}}, \widetilde{\boldsymbol{r}_B}, \widetilde{\boldsymbol{e}_u}, \widetilde{\boldsymbol{e}_v}) \\ \equiv_s\ &(a, b; \boldsymbol{h}, \widetilde{\boldsymbol{s}}, \widetilde{\boldsymbol{u}}, \widetilde{\boldsymbol{v}}, \boldsymbol{r}_B, \boldsymbol{e}_u, \boldsymbol{e}_v). \end{aligned} \tag{3.16}$$

Note that $\boldsymbol{s}$ can be reduced to 2-cyclic syndrome decoding decision, and the distribution cannot be distinguished from uniform random numbers for the adversary in polynomial time. Therefore, the following equation is satisfied.

$$\begin{aligned} &(a, b; \boldsymbol{h}, \widetilde{\boldsymbol{s}}, \widetilde{\boldsymbol{u}}, \widetilde{\boldsymbol{v}}, \boldsymbol{r}_B, \boldsymbol{e}_u, \boldsymbol{e}_v) \\ \equiv_c\ &(a, b; \boldsymbol{h}, \boldsymbol{s}, \widetilde{\boldsymbol{u}}, \widetilde{\boldsymbol{v}}, \boldsymbol{r}_B, \boldsymbol{e}_u, \boldsymbol{e}_v). \end{aligned} \tag{3.17}$$

Moreover, since $\boldsymbol{u}$ and $\boldsymbol{v}$ are indistinguishable between $(\boldsymbol{h} \cdot \boldsymbol{r}_B + \boldsymbol{e}_u, \boldsymbol{s} \cdot \boldsymbol{r}_B + \boldsymbol{e}_v)$ and uniform random numbers based on the assumption of quasi-cyclic syndrome decoding and the adversary of probabilistic polynomial time cannot be distinguished, the following holds:

$$\begin{aligned} &(a, b; \boldsymbol{h}, \boldsymbol{s}, \widetilde{\boldsymbol{u}}, \widetilde{\boldsymbol{v}}, \boldsymbol{r}_B, \boldsymbol{e}_u, \boldsymbol{e}_v) \\ \equiv_c\ &(a, b; \boldsymbol{h}, \boldsymbol{s}, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{r}_B, \boldsymbol{e}_u, \boldsymbol{e}_v). \end{aligned} \tag{3.18}$$

Therefore, the distributions of the view view_B of B and the simulator S_B cannot be distinguished against the adversary of polynomial time:

$$\begin{aligned} &S_B(a, b) \\ \equiv_c\ &\mathrm{view}_B(a, b; \boldsymbol{h}, \boldsymbol{s}, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{r}_B, \boldsymbol{e}_u, \boldsymbol{e}_v). \end{aligned} \tag{3.19}$$

□

The above protocol works over $\mathbb{F}_2$, but one can see that this can be easily extended to a larger field $\mathbb{F}_q$ by using appropriate error-correcting linear codes over $\mathbb{F}_q$.

3.2.5.3 Secure Comparison

Two-party secure comparison protocol proposed in this section is based on the size comparison method used in the secure decision tree classification protocol of Wu et al. [23]. In this section, we used the following criteria given in Proposition 3.1 for comparison.

Proposition 3.1 *For a t-bit x, y, if there is an $i \in [t]$ such that the following expression holds, then $x < y$.*

$$x_i - y_i + 1 + 3\sum_{j<i}(x_j \oplus y_j) = 0.$$

In this section, we introduce the proposed protocol for two-party secret comparison protocol. The proposed protocol for two-party secret comparison protocol uses a quasi-cyclic code and an arbitrary error-correcting code (For example, Reed-Solomon code) on $\mathbb{F}_q$. The participants in the protocol are Alice (A) and Bob (B). The input of A is $c \in \mathbb{N}$, and the input of B is $d \in \mathbb{N}$. The output of A is the result of the comparison between c and d, and the output of B is none.

The flow of two-party secret comparison is shown as follows:

Protocol Two-party secret comparison protocol

Input	A : $c \in \mathbb{N}$
	B : $d \in \mathbb{N}$
Output	A : Comparison result of c and d
	B : $\perp$

1. A and B perform binary expansion of c and d for each input so that $\boldsymbol{c} = c_1 c_2 \ldots c_l, \boldsymbol{d} = d_1 d_2 \ldots d_l$. Then, each bit c_i, d_i is padded to make $\boldsymbol{c_i}, \boldsymbol{d_i}, i \in [l]$ of k bits. In addition, they set the global parameter param = (n, k, δ, w_x, w_r) and the generator matrix $\mathbb{G} \in \mathbb{F}_q^{k \times n}$ of code $\mathbb{C}$.
2. A generates random $\boldsymbol{h} \xleftarrow{\$} \mathbb{R}$. Furthermore, $(\boldsymbol{x}, \boldsymbol{y} \xleftarrow{\$} \mathbb{R}^2)$ with Hamming weight w_x is generated. Private key $sk = (\boldsymbol{x}, \boldsymbol{y})$, and public key $pk = (\boldsymbol{h}, \boldsymbol{s} = \boldsymbol{x} + \boldsymbol{h} \cdot \boldsymbol{y})$.
3. A generates a random $\boldsymbol{r_{Ai}}, \boldsymbol{r_{ui}}, \boldsymbol{r_{vi}} \xleftarrow{\$} \mathbb{R}, i \in [l]$ with Hamming weight w_r. Then, A computes $\boldsymbol{u_i} = \boldsymbol{h} \cdot \boldsymbol{r_{Ai}} + \boldsymbol{r_{ui}}$ and $\boldsymbol{v_i} = \boldsymbol{c_i} \cdot \boldsymbol{G} + \boldsymbol{s} \cdot \boldsymbol{r_{Ai}} + \boldsymbol{r_{vi}}$ for l pairs and sends l pairs of ciphertext $\boldsymbol{u_i}, \boldsymbol{v_i}$ to B.
4. B generates $(\boldsymbol{r_{Bi}}, \boldsymbol{e_{ui}}, \boldsymbol{e_{vi}}) \xleftarrow{\$} \mathbb{R}^3$ with Hamming weight w_r^* and computes the expression $c_i - d_i + 1 + 3\sum_{w<i}(c_w \oplus d_w)$ for c_i. Specifically, B substitutes plaintext d_i for $i \in [l]$ in the above formula and sets appropriate $a_{1i}, a_{2i}, \ldots, a_{li}$, $\boldsymbol{b_i}$. B computes $\boldsymbol{u_i}' = a_{1i} \cdot \boldsymbol{u_1} + \cdots + \boldsymbol{h} \cdot \boldsymbol{r_{Bi}} + \boldsymbol{e_{ui}}$ and $\boldsymbol{v_i}' = a_{1i} \cdot \boldsymbol{v_1} + \cdots + \boldsymbol{b_i} \cdot \mathbb{G} + \boldsymbol{s} \cdot \boldsymbol{r_{Bi}} + \boldsymbol{e_{vi}}$ for l pairs. Then, the order of $(\boldsymbol{u_i}', \boldsymbol{v_i}')$ of l pairs is randomly replaced and sent to A in a random order.

5. A computes $\boldsymbol{v_i}' - \boldsymbol{u_i}' \cdot \boldsymbol{y}$ for each $i \in [l]$ and decrypts the result. If there is 0 in the first bit of the decoded results, $c < d$ is output. Conversely, if there is no 0, $c \geq d$ is output.

Protocol Description

1. In step 1, A and B expand c and d of each input to l-bit binary input, so that $\boldsymbol{c} = c_1 c_2 \ldots c_l$ and $\boldsymbol{d} = d_1 d_2 \ldots d_l$. Where $c_i, d_i, i \in [l]$ is the ith digit of $\boldsymbol{c}, \boldsymbol{d}$, and l is the bit length. To encode, pad each input to $\boldsymbol{c_i}, \boldsymbol{d_i}, i \in [l]$ with bit length k.
   In addition, set global parameters. n is the code length, k is the number of information bits, δ is the maximum number of errors that can be corrected by the error-correcting code, and w_x and w_r are the Hamming weights set in advance. The public parameter $\mathbb{G}$ is the generator matrix(For example, the Reed-Solomon code generator matrix) of the error-correcting code $\mathbb{C}$, which maps the message and code length as $\mathbb{F}_q^k \to \mathbb{F}_q^n$.
2. In step 2, A generates a private key and public key for HQC encryption scheme.
3. In step 3, A uses the public key and encrypts each of the $\boldsymbol{c_i}$ pieces. Send $(\boldsymbol{u_i}, \boldsymbol{v_i}), i \in [l]$ of the encrypted result to B.
4. Step 4 uses Proposition 3.1 for the evaluation of $c_i - d_i + 1 + 3\sum_{w<i}(c_w \oplus d_w)$. In other words, $c < d$ if $i \in [l]$ exists such that

$$c_i - d_i + 1 + 3\sum_{w<i}(c_w \oplus d_w) = 0. \tag{3.20}$$

   In particular, since B has plaintext d_i and encrypted c_i, Eq. (3.20) can be regarded as an equation with c_i as an unknown and can be computed. In addition, for XOR operations, B can transform $x_i \oplus y_i$ into

$$x_i \oplus y_i = \begin{cases} x_i & (y_i = 0) \\ 1 - x_i & (y_i = 1). \end{cases} \tag{3.21}$$

   Therefore, the XOR operation requires only the additive homomorphism of HQC encryption scheme.
   That is, B substitutes plaintext $d_i, i \in [l]$ into the above equation, sets the appropriate $a_{1i}, a_{2i}, \ldots, a_{li}, \boldsymbol{b_i}$, and computes as follows:

$$\boldsymbol{u_i}' = a_{1i} \cdot \boldsymbol{u_1} + \cdots + a_{li} \cdot \boldsymbol{u_l} + \boldsymbol{h} \cdot \boldsymbol{r_{Bi}} + \boldsymbol{e_{ui}}. \tag{3.22}$$

$$\boldsymbol{v_i}' = a_{1i} \cdot \boldsymbol{v_1} + \cdots + a_{li} \cdot \boldsymbol{v_l} + \boldsymbol{b_i} \cdot \boldsymbol{G} + \boldsymbol{s} \cdot \boldsymbol{r_{Bi}} + \boldsymbol{e_{vi}}. \tag{3.23}$$

   Here, the Hamming weight of $\boldsymbol{r_{Bi}}, \boldsymbol{e_{ui}}, \boldsymbol{e_{vi}}, i \in [l]$ is w_r^*.
   Furthermore, to not leak the information about which bits are different to A, B needs to replace the order of each $(\boldsymbol{u_i}', \boldsymbol{v_i}')$ computed at random.

5. In step 5, A computes $\boldsymbol{v_i}' - \boldsymbol{u_i}' \cdot \boldsymbol{y}, i \in [l]$. The result is

$$\begin{aligned} & \boldsymbol{v_i}' - \boldsymbol{u_i}' \cdot \boldsymbol{y} \\ = & (a_{1i} \cdot \boldsymbol{m_1} + \cdots + a_{li} \cdot \boldsymbol{m_l}) \cdot \mathbb{G} \\ & + \boldsymbol{x} \cdot (a_{1i} \cdot \boldsymbol{r_{A1}} + \cdots + a_{li} \cdot \boldsymbol{r_{Al}} + \boldsymbol{r_{Bi}}) \\ & - \boldsymbol{y} \cdot (a_{1i} \cdot \boldsymbol{r_{u1}} + \cdots + a_{li} \cdot \boldsymbol{r_{ul}} + \boldsymbol{e_{ui}}) \\ & + (a_{1i} \cdot \boldsymbol{r_{v1}} + \cdots + a_{li} \cdot \boldsymbol{r_{vl}} + \boldsymbol{e_{vi}}). \end{aligned} \tag{3.24}$$

Then, the evaluation result is decoded by the error-correcting code. A takes out the first 1 bit of each of l decoding results, and outputs $c < d$ if there is 0 in it. If there is no 0, $c \geq d$ is output.

3.2.5.4 Correctness and Security of the Proposed Protocol

Correctness

First, we explain step 4 w_r^*. The Hamming weight of the polynomial coefficient vector $\boldsymbol{x}, \boldsymbol{y}$ is w_x, and the Hamming weight of $\boldsymbol{r_{Ai}}, \boldsymbol{r_{ui}}, \boldsymbol{r_{vi}}, i \in [l]$ is w_r. Since each is selected uniformly and independently, the probability of each bit value of the vector is expressed as follows:

$$x_i = y_i = \begin{cases} 0 & \text{w.p. } 1 - p \\ 1 & \text{w.p. } p = \frac{w_x}{n}. \end{cases} \tag{3.25}$$

Similarly,

$$r_{Ai,j} = r_{ui,j} = r_{vi,j} = \begin{cases} 0 & \text{w.p. } 1 - p_r \\ 1 & \text{w.p. } p_r = \frac{w_r}{n}. \end{cases} \tag{3.26}$$

Let L be the set of $a_{1i}, a_{2i}, \ldots, a_{li} \neq 0$ in each $a_{1i} \cdot \boldsymbol{r_{A1}} + a_{2i} \cdot \boldsymbol{r_{A2}} + \cdots + a_{li} \cdot \boldsymbol{r_{Al}}$ for the expression $i \in [l]$.

$$L = \{a_{ki} | a_{ki} \neq 0\}$$

Let $|L|$ be the number of elements in set L. Set the Hamming weights w_r^* for $\boldsymbol{r_{Bi}}, \boldsymbol{e_{ui}}, \boldsymbol{e_{vi}}$ be as follows:

$$w_r^* = (n - |L| + 1) w_r.$$

Thus, the value of each w_r^* can be determined based on the nonzero numbers in a_i and $i \in [l]$.

Next, we analyze the validity of the proposed protocol.

The legitimacy of the proposed bilateral linear function secure computation protocol clearly depends on the decoding ability of $\mathbb{C}$. Set the $\boldsymbol{v'} - \boldsymbol{u'} \cdot \boldsymbol{y}$ error to ϵ. For the error correction capability of code $\mathbb{C}$, the error is

$$\begin{aligned} \epsilon = \ & \boldsymbol{x} \cdot (a_{1i} \cdot \boldsymbol{r_{A1}} + \cdots + a_{li} \cdot \boldsymbol{r_{Al}} + \boldsymbol{r_{Bi}}) \\ & - \boldsymbol{y} \cdot (a_{1i} \cdot \boldsymbol{r_{u1}} + \cdots + a_{li} \cdot \boldsymbol{r_{ul}} + \boldsymbol{e_{ui}}) \\ & + (a_{1i} \cdot \boldsymbol{r_{v1}} + \cdots + a_{li} \cdot \boldsymbol{r_{vl}} + \boldsymbol{e_{vi}}). \end{aligned} \tag{3.27}$$

In other words, if $\epsilon < \delta$, decoding is successful. Here, δ is the maximum number of errors that can be corrected by error-correcting code $\mathbb{C}$. In addition, in order to analyze the validity of the proposed protocol, we generalize the validity of the HQC encryption scheme proved by Gaborit et al. [20].

The following proposition holds for the Hamming weight of the error.

Proposition 3.2 *There are polynomial coefficient vectors $\boldsymbol{x} = (X_1, \ldots, X_n)$ and $\boldsymbol{r} = (R_1, \ldots, R_n)$, and $\boldsymbol{y} = \boldsymbol{x} \cdot \boldsymbol{r} = (Y_1, \ldots, Y_n)$. The probability that the sum of the random variables $Y_i, i \in [n]$ on $\mathbb{F}_q$ is 0 is*

$$\Pr[Y_1 + \cdots + Y_n = 0] = \frac{1}{q}\{1 + (1 - \frac{q}{q-1}p)^n \cdot (q-1)\}. \tag{3.28}$$

Where the probability distribution of the random variable Y_i is

$$Y_i = \begin{cases} 0 & \text{w.p. } p_0 = 1 - p \\ 1 & \text{w.p. } p_1 = \frac{p}{q-1} \\ 2 & \text{w.p. } p_1 = \frac{p}{q-1} \\ \vdots & \\ q-1 & \text{w.p. } p_1 = \frac{p}{q-1}. \end{cases} \tag{3.29}$$

Proof For Y_i, the following equation holds:

$$\begin{aligned} & \Pr[Y_1 + \cdots + Y_n = 0] \\ = & \sum_{\substack{i_0 + i_1 + \cdots + i_{q-1} = n \\ i_0 \cdot 0 + i_1 \cdot 1 + \cdots + i_{q-1} \cdot (q-1) = 0}} \left(\frac{n!}{i_0! \cdots i_{q-1}!} \right) p_0^{i_0} \cdots p_{q-1}^{i_{q-1}}, \end{aligned} \tag{3.30}$$

where $i_0, \ldots, i_{q-1}$ is the number of times the corresponding $0, \ldots, q-1$ appears. From the polynomial theorem, the following equation holds:

$$
\begin{aligned}
&\{p_0+p_1+\ldots+p_{q-1}\}^n+\{p_0+(\omega_q)p_1+\cdots+(\omega_q^{q-1})p_{q-1}\}^n\\
&+\cdots+\{p_0+(\omega_q)^{q-1}p_1+\cdots+(\omega_q^{q-1})^{q-1}p_{q-1}\}^n\\
=&\sum_{i_0+\cdots+i_{q-1}=n}\left(\frac{n!}{i_0!\cdots i_{q-1}!}\right)p_0^{i_0}\cdots p_{q-1}^{i_{q-1}}\\
&\{1+(\omega_q)^{i_1}(\omega_q^2)^{i_2}\cdots(\omega_q^{q-1})^{i_{q-1}}+\cdots\\
&+(\omega_q)^{(q-1)i_1}(\omega_q^2)^{(q-1)i_2}\cdots(\omega_q^{q-1})^{(q-1)i_{q-1}}\}\\
=&\sum_{i_0+\cdots+i_{q-1}=n}\left(\frac{n!}{i_0!\cdots i_{q-1}!}\right)p_0^{i_0}\cdots p_{q-1}^{i_{q-1}}\\
&\{1+\omega_q^{i_1+2i_2+\cdots+(q-1)i_{q-1}}+\cdots\\
&+\omega_q^{(q-1)\{i_1+2i_2+\cdots+(q-1)i_{q-1}\}}\}.
\end{aligned}
\tag{3.31}
$$

Where ω_q is the q root of 1 and has the following properties:

$$
1+\omega_q+\omega_q^2+\cdots+\omega_q^{q-1}=0 \tag{3.32}
$$

Substituting $i_0\cdot 0+i_1\cdot 1+\cdots+i_{q-1}\cdot(q-1)=0$ into Eq. 3.31 can be transformed as follows:

$$
\begin{aligned}
&\{p_0+p_1+\cdots+p_{q-1}\}^n\\
&+\{p_0+(\omega_q)p_1+\cdots+(\omega_q^{q-1})p_{q-1}\}^n+\cdots\\
&+\{p_0+(\omega_q)^{q-1}p_1+\cdots+(\omega_q^{q-1})^{q-1}p_{q-1}\}^n\\
=&\sum_{\substack{i_0+\cdots+i_{q-1}=n\\ i_0\cdot 0+\cdots+i_{q-1}\cdot(q-1)=0}}\left(\frac{n!}{i_0!\cdots i_{q-1}!}\right)p_0^{i_0}\cdots p_{q-1}^{i_{q-1}}\cdot q.
\end{aligned}
\tag{3.33}
$$

Substituting Eq. (3.33) into Eq. (3.30), the proposition holds:

$$
\begin{aligned}
&\Pr[Y_1+\cdots+Y_n=0]\\
=&\frac{1}{q}\{(p_0+p_1+\cdots+p_{q-1})^n+\cdots\\
&+(p_0+(\omega_q)^{q-1}p_1+\cdots+(\omega_q^{q-1})^{q-1}p_{q-1})^n\}\\
=&\frac{1}{q}\{1^n+(1-p+\frac{p}{q-1}(\omega_q+\omega_q^2+\cdots+\omega_q^{q-1}))^n\cdot(q-1)\}\\
=&\frac{1}{q}\left\{1+\left(1-\frac{q}{q-1}p\right)^n\cdot(q-1)\right\}.
\end{aligned}
\tag{3.34}
$$

□

In addition, the following analysis is the same as the validity analysis in Gaborit et al. [20]. According to the analysis result of [20], in the case of $\mathbb{F}_2$, the decoding failure rate can be controlled by setting an appropriate code space size n and noise Hamming weights w_x and w_r. Therefore, in the case of $\mathbb{F}_q$, it can be expected that the decoding failure rate can be controlled by setting the appropriate parameters.

Security

This section describes the security of the proposed secret comparison protocol.

First, consider semi-honest adversaries A and $\text{output}_\text{A} = (\text{c} < \text{d})$. Omitting global parameters, A's view is $\text{view}_A = (c, \boldsymbol{x}, \boldsymbol{y}; \boldsymbol{h}, \{\boldsymbol{r}_{Ai}\}_{i=1}^{l}, \{\boldsymbol{r}_{ui}\}_{i=1}^{l}, \{\boldsymbol{r}_{vi}\}_{i=1}^{l}, \{\boldsymbol{u}_i{}'\}_{i=1}^{l}, \{\boldsymbol{v}_i{}'\}_{i=1}^{l})$. However, the first bit is 0 only for $\boldsymbol{u}_{i*}{}' - \boldsymbol{v}_{i*}{}' \cdot \boldsymbol{y}$ with index $i*$. The simulator $S_A(c, \boldsymbol{x}, \boldsymbol{y})$ is configured as follows:

1. Generates $\widetilde{\boldsymbol{h}}, \{\widetilde{\boldsymbol{r}_{Ai}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{r}_{ui}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{r}_{vi}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{u}_i{}'}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{v}_i{}'}\}_{i=1}^{l} \xleftarrow{\$} \mathbb{R}$ at random. Here, the Hamming weight of $\{\widetilde{\boldsymbol{r}_{Ai}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{r}_{ui}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{r}_{vi}}\}_{i=1}^{l}$ is w_r. It also selects random $i* \in [l]$, the first bit of $\boldsymbol{u}_{i*}{}' - \boldsymbol{v}_{i*}{}' \cdot \boldsymbol{y}$ is 0, and the first bit of other $\{\widetilde{\boldsymbol{u}_i{}'} - \widetilde{\boldsymbol{v}_i{}'} \cdot \boldsymbol{y}\}_{i=1, i\neq i*}^{l}$ is non-zero.
2. This replaces $\{\widetilde{\boldsymbol{u}_i{}'}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{v}_i{}'}\}_{i=1}^{l}$ at random to make $\{\widetilde{\boldsymbol{u}_j{}'}\}_{j=1}^{l}, \{\widetilde{\boldsymbol{v}_j{}'}\}_{j=1}^{l}$ in random order.
3. This outputs $(c, \boldsymbol{x}, \boldsymbol{y}; \widetilde{\boldsymbol{h}}, \{\widetilde{\boldsymbol{r}_{Ai}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{r}_{ui}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{r}_{vi}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{u}_j{}'}\}_{j=1}^{l}, \{\widetilde{\boldsymbol{v}_j{}'}\}_{j=1}^{l})$.

Since $\boldsymbol{h}, \{\boldsymbol{r}_{Ai}\}_{i=1}^{l}, \{\boldsymbol{r}_{ui}\}_{i=1}^{l}, \{\boldsymbol{r}_{vi}\}_{i=1}^{l}$ and $\widetilde{\boldsymbol{h}}, \{\widetilde{\boldsymbol{r}_{Ai}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{r}_{ui}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{r}_{vi}}\}_{i=1}^{l}$ follow the same distribution, the following equation holds:

$$\begin{aligned}&(\widetilde{\boldsymbol{h}}, \{\widetilde{\boldsymbol{r}_{Ai}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{r}_{ui}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{r}_{vi}}\}_{i=1}^{l})\\ \equiv_s &(\boldsymbol{h}, \{\boldsymbol{r}_{Ai}\}_{i=1}^{l}, \{\boldsymbol{r}_{ui}\}_{i=1}^{l}, \{\boldsymbol{r}_{vi}\}_{i=1}^{l}).\end{aligned} \tag{3.35}$$

From the assumption of quasi-cyclic syndrome decoding of quasi-cyclic codes, the probabilistic polynomial time adversary cannot distinguish between $\boldsymbol{u}_j{}', \boldsymbol{v}_j{}', j \in [l]$ and uniformly random ones. Furthermore, since $\{\widetilde{\boldsymbol{u}_i{}'}\}_{i=1}^{l}$ and $\{\widetilde{\boldsymbol{v}_i{}'}\}_{i=1}^{l}$ are replaced randomly, the first bit is 0, and the index of $\widetilde{\boldsymbol{u}_{i*}} - \widetilde{\boldsymbol{v}_{i*}} \cdot \boldsymbol{y}$ where the index $i*$ is a uniformly random one satisfying the following expression:

$$\left(\{\widetilde{\boldsymbol{u}_j{}'}\}_{j=1}^{l}, \{\widetilde{\boldsymbol{v}_j{}'}\}_{j=1}^{l}\right) \equiv_c \left(\{\boldsymbol{u}_i{}'\}_{i=1}^{l} \{\boldsymbol{v}_i{}'\}_{i=1}^{l}\right). \tag{3.36}$$

Therefore, the distribution of the view view_A and simulator S_A when A is $\text{output}_A = (c < d)$ is indistinguishable against polynomial time opponents.

Semi-honest adversary A and $\text{output}_A = (c \geq d)$ are the same as the security proof in the case of $\text{output}_A = (c < d)$, so details are omitted.

Next, we consider semi-honest adversary B. Omitting the global parameters, B's view is $\text{view}_B = (d; \boldsymbol{h}, \boldsymbol{s}, \{\boldsymbol{u}_i\}_{i=1}^{l}, \{\boldsymbol{v}_i\}_{i=1}^{l}, \{\boldsymbol{r}_{Bi}\}_{i=1}^{l}, \{\boldsymbol{e}_{ui}\}_{i=1}^{l}, \{\boldsymbol{e}_{vi}\}_{i=1}^{l})$. Configure simulator $S_B(d)$ as follows:

1. Generates $\widetilde{\boldsymbol{h}}, \widetilde{\boldsymbol{s}}, \{\widetilde{\boldsymbol{u}_i}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{v}_i}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{r}_{Bi}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{e}_{ui}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{e}_{vi}}\}_{i=1}^{l} \xleftarrow{\$} \mathbb{R}$ at random. Here, the Hamming weight of $\{\widetilde{\boldsymbol{r}_{Bi}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{e}_{ui}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{e}_{vi}}\}_{i=1}^{l}$ is w_r^*.

2. This outputs $(\boldsymbol{d}; \widetilde{\boldsymbol{h}}, \widetilde{\boldsymbol{s}}, \{\widetilde{\boldsymbol{u_i}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{v_i}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{r_{Bi}}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{e_{ui}}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{e_{vi}}}\}_{i=1}^{l})$.

Since $\boldsymbol{h}, \{\boldsymbol{r_{Bi}}\}_{i=1}^{l}, \{\boldsymbol{e_{ui}}\}_{i=1}^{l}, \{\boldsymbol{e_{vi}}\}_{i=1}^{l}$ and $\widetilde{\boldsymbol{h}}, \{\widetilde{\boldsymbol{r_{Bi}}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{e_{ui}}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{e_{vi}}}\}_{i=1}^{l}$ follow the same distribution, the following equation holds:

$$\begin{aligned}&(\boldsymbol{h}, \{\boldsymbol{r_{Bi}}\}_{i=1}^{l}, \{\boldsymbol{e_{ui}}\}_{i=1}^{l}, \{\boldsymbol{e_{vi}}\}_{i=1}^{l})\\ \equiv_s &(\widetilde{\boldsymbol{h}}, \{\widetilde{\boldsymbol{r_{Bi}}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{e_{ui}}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{e_{vi}}}\}_{i=1}^{l}).\end{aligned} \tag{3.37}$$

$\boldsymbol{s}$ can be reduced to a 2-quasi-cyclic syndrome decoding decision assumption, and the distribution is indistinguishable from uniform random numbers for probabilistic polynomial-time adversaries. Thus, $\widetilde{\boldsymbol{s}} \equiv_c \boldsymbol{s}$ holds.

In addition, since $\boldsymbol{u_i}, \boldsymbol{v_i}, i \in [l]$ are based on the assumption of quasi-cyclic syndrome decoding, an adversary in probabilistic polynomial time cannot distinguish between $\boldsymbol{u_i}, \boldsymbol{v_i}, i \in [l]$ and uniform random numbers.

$$(\{\widetilde{\boldsymbol{u_i}}\}_{i=1}^{l}, \{\widetilde{\boldsymbol{v_i}}\}_{i=1}^{l}) \equiv_c (\{\boldsymbol{u_i}\}_{i=1}^{l}, \{\boldsymbol{v_i}\}_{i=1}^{l}). \tag{3.38}$$

Therefore, the distribution of B's view view_B and simulator S_B is indistinguishable against polynomial time adversaries.

3.2.6 *Support Vector Machine from Secure Linear Function Evaluation and Secure Comparison*

We can construct a code-based protocol for a support vector machine from the protocols for evaluation of linear functions and comparison described above. Note that the result of secure evaluation of linear function is in $\mathbb{F}_q$ while that of secure composition is a bit string. Therefore, we need to provide secure bit-decomposition protocol. The bit-decomposition protocols have been already studied well in the research area of secure computation, and indeed, we can use the bit-decomposition protocol given in [24] with secure computation protocol from a threshold homomorphic encryption [25]. (It is straightforward to construct a threshold version of HQC scheme by setting $sk_A = (\boldsymbol{x}_1, \boldsymbol{y}_1)$ and $sk_B = (\boldsymbol{x}_2, \boldsymbol{y}_2)$ as distributed decryption keys for A and B. Then, the encryption key is $(\boldsymbol{h}, (\boldsymbol{x}_1 + \boldsymbol{x}_2) + \boldsymbol{h} \cdot (\boldsymbol{y}_1 + \boldsymbol{y}_2))$.

We describe the overview of the protocol below. For simplification, we denote $[m]$ as the ciphertext for m under HQC encryption scheme over $\mathbb{F}_q$.

Protocol

Input	A : $m \in \mathbb{F}_q$
	B : $a, b, t \in \mathbb{F}_q$
Output	A : $a \cdot m + b > t$ or not
	B : $\perp$

1. A and B perform the secure linear evaluation protocol over $\mathbb{F}_q$. Then, B sends A $[a \cdot m + b]$ at step 4 in the original protocol.
2. A and B start the secure bit-decomposition protocol on $[a \cdot m + b]$.
3. From the result of the bit-decomposition protocol, B obtains the binary representation $[(a \cdot m + b)_1], \ldots, [(a \cdot m + b)_\ell]$.
4. A and B perform the secure comparison protocol from step 4.

References

1. L. Kissner, D. Song, Privacy-preserving set operations, in *CRYPTO 2005*. LNCS, vol. 3621 (Springer, Berlin, 2005), pp. 241–257
2. Y. Sang, H. Shen, Efficient and secure protocols for privacy-preserving set operations. ACM Trans. Inf. Syst. Secur. **13**(1), 9:1–9:35 (2009)
3. R. Egert, M. Fischlin, D. Gens, S. Jacob, M. Senker, J. Tillmanns, Privately computing set-union and set-intersection cardinality via bloom filters, in *ACISP 2015*. LNCS, vol. 9144 (Springer, Berlin, 2015), pp. 413–430
4. D. Many, M. Burkhart, X. Dimitropoulos, Fast private set operations with sepia. Technical Report, 345 (2012)
5. O. Goldreich, Secure multi-party computation. Manuscript, Preliminary version (1998)
6. B.H. Bloom, Space/time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7), 422–426 (1970)
7. A. Broder, M. Mitzenmacher, Network applications of bloom filters: a survey. Internet Math. **1**(4), 485–509 (2004)
8. P. Paillier, Public-key cryptosystems based on composite degree residuosity classes, in *EUROCRYPT 1999*. LNCS, vol. 1592 (Springer, Berlin, 1999), pp. 223–238
9. R. Cramer, R. Gennaro, B. Schoenmakers, A secure and optimally efficient multi-authority election scheme. Eur. Trans. Telecommun. **8**(5), 481–490 (1997)
10. Y. Desmedt, Y. Frankel, Threshold cryptosystems, in *CRYPTO 1989*. *LNCS*, vol. 1462 (Springer, Berlin, 1989), pp. 307–315
11. M.J. Freedman, K. Nissim, B. Pinkas, Efficient private matching and set intersection, in *EUROCRYPT 2004*. *LNCS*, vol. 3027 (Springer, Berlin, 2004), pp. 1–19
12. Y. Azar, A.Z. Broder, A.R. Karlin, E. Upfal, Balanced allocations. SIAM J. Comput. **29**(1), 180–200 (1999)
13. E. De Cristofaro, G. Tsudik, Practical private set intersection protocols with linear complexity, in *FC 2010*. *LNCS*, vol. 6052 (Springer, Berlin, 2010), pp. 143–159
14. E. De Cristofaro, J. Kim, G. Tsudik, Linear-complexity private set intersection protocols secure in malicious model, in *ASIACRYPT 2010*. LNCS, vol. 6477 (Springer, Berlin, 2010), pp. 213–231
15. F. Kerschbaum, Outsourced private set intersection using homomorphic encryption, in *ACMCCS 2012* (ACM, 2012), pp. 85–86
16. S. Goldwasser, S. Micali, Probabilistic encryption. J. Comput. Syst. Sci. **28**(2), 270–299 (1984)
17. Y. Ishai, J. Kilian, K. Nissim, E. Petrank, Extending oblivious transfers efficiently, in *CRYPTO 2003*. *LNCS*, vol. 2729 (Springer, Berlin, 2003), pp. 145–161
18. M.O. Rabin, How to exchange secrets with oblivious transfer. Technical Memo, TR-81 (1981)
19. C. Dong, L. Chen, Z. Wen, When private set intersection meets big data: an efficient and scalable protocol, in *ACMCCS 2013* (ACM, 2013), pp. 789–800
20. C. Aguilar, O. Blazy, J.-C. Deneuville, P. Gaborit, G. Zémor, Efficient encryption from random quasi-cyclic codes. IEEE Trans. Inf. Theory **64**(5), 3927–3943 (2018)

21. National Institute of Standards and Technology. Post-quantum cryptography, round 2 submissions (2019), https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions
22. A.C.-C. Yao, How to generate and exchange secrets, in *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science* (1986), pp. 162–167
23. D.J. Wu, T. Feng, M. Naehrig, K. Lauter, Privately evaluating decision trees and random forests, in *Proceeding on Privacy Enhancing Technologies*, vol. 4 (2016), pp. 1–21
24. I. Dangaard, M. Fitzi, E. Kiltz, J.B. Nielsen, T. Toft, Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation, in *TCC2006: Theory of Cryptography* (2006), pp. 285–304
25. R. Cramer, I. Damgaard, J.B. Nielsen, Multiparty computation from threshold encryption, in *Eurocrypt* (2001), pp. 280–299