mathematics

MDPI

Article

A High Throughput BFV-Encryption-Based Secure Comparison Protocol

Tzu-Hsiang Kuo [1] and Ja-Ling Wu [1,2,*]

[1] Department of Computer Science & Information Engineering, National Taiwan University, Taipei 106, Taiwan
[2] Graduate Institute of Networking and Multimedia, National Taiwan University, Taipei 106, Taiwan
* Correspondence: wjl@cmlab.csie.ntu.edu.tw

Abstract: Secure comparison is a fundamental problem in multiparty computation. There are two different parties, each holding an l-bit integer, denoted by a and b, respectively. The goal of secure comparison is to compute the order relationship between a and b, say $(a > b) \in \{0, 1\}$, without revealing their inputs to any others. Since previous solutions based on homomorphic encryption need at least $\Omega(l)$ encryptions for each l-bit comparison, the total encryption time leads to a computational bottleneck for these protocols. This work presents a fast, semi-honest, secure comparison protocol based on the BFV encryption scheme. With its vector-like plaintext space, the number of required encryptions can be significantly reduced; actually, only six encryptions are needed for each comparison in our protocol. In other words, the proposed protocol can achieve the time complexity $\widetilde{O}(\lambda + l)$ for a given security parameter λ. As a result, 4096-bit integers can be securely compared within 12.08 ms, which is 280 times faster than the state-of-the-art homomorphic encryption-based secure comparison protocol. Furthermore, we can compare k pairs of $l \cdot k^{-1}$-bit integers with almost the same execution time as comparing l-bit integers and achieve higher throughput regardless of the compared integer size.

Keywords: secure comparison; multi-party computation; fully homomorphic encryption; Secure Auction; ring learning with error

MSC: 94A60

Citation: Kuo, T.-H.; Wu, J.-L. A High Throughput BFV-Encryption-Based Secure Comparison Protocol. *Mathematics* **2023**, *11*, 1227. https://doi.org/10.3390/math11051227

Academic Editor: Raúl M. Falcón

Received: 8 January 2023
Revised: 19 February 2023
Accepted: 24 February 2023
Published: 2 March 2023

1. Introduction

1.1. Homomorphic Encryption-Based Secure Comparison

The goal of multi-party computation (MPC) is to create an algorithm that enables different parties to compute a function jointly without revealing their inputs. In areas such as secret voting or e-commerce, MPC provides a solution for protecting each party's privacy without the involvement of a trusted third party (TTP). However, the high computational costs of MPC make these solutions practically infeasible. Secure comparison is one of the fundamental problems in MPC, with the goal of computing $(a > b) \in \{0, 1\}$ securely for two different parties with private inputs a and b. Since special-purpose secure comparison protocols (SP-SCPs) often outperform those protocols based on general-purpose MPC constructions, SP-SCPs can improve the performance of most privacy-preserved applications. This fact motivates us to develop high-speed, secure comparison protocols.

Homomorphic Encryption (HE) and Garbled Circuits (GC) are two of the most famous cryptographic techniques for constructing secure comparison protocols. Based on the study presented in [1], HE-based protocols can achieve faster speed and lower round complexity, whereas GC-based protocols provide a lower communication rate and higher flexibility. In this work, we focus on the computational cost and compete for the speed of our protocol to state-of-the-art HE-based comparison protocols. In general, the performance of an HE-based protocol is strongly related to which HE scheme is used. For example, the

protocol proposed in [2] needs only two encryptions for each bit, but the final performance is 60.4 milliseconds per bit (ms/bit) (Runs on i5-9400f@4.0 GHz, single core, and set security parameter $\lambda = 128$.) since each Paillier encryption [3] needs 23 ms. The authors in [4] replaced Paillier with subgroup-RSA [5], which needs only 0.692 ms/bit for each encryption and provides a much better performance of 1.74 ms/bit. Noticeably, the protocols proposed in [6,7] compare 8 bits simultaneously by the threshold property in the plaintext space $\mathbb{Z}_{2^{28}}$, which needs only two encryptions and one private-equality-test (Private-equality-test is to compute $(a == b) \in \{0, 1\}$ securely for two different parties. Notice. two encryptions are needed in the protocol.) for every 8-bit. In their works, the performance reaches 0.8 ms/bit. To the best of our knowledge, these two are the fastest 2-round secure comparison protocol [2,4,8–10] and the fastest secure comparison protocol [6,7]. We will refer to them as DGK protocol and CEK protocol, respectively. The Paillier and the subgroup-RSA mentioned above are HE schemes that can execute additions in the encryption domain. We refer to them as additive homomorphic encryption (AHE) schemes.

Fully homomorphic encryption (FHE) specifically addresses those HE schemes that can realize arbitrary arithmetic circuits in the encryption domain. The first construction of FHE was proposed by Gentry [11,12] in 2009. In his construction, Gentry first proposed a somewhat homomorphic encryption (SWE) scheme, which can implement a limited number of additions and multiplications in the encryption domain, and then used bootstrapping mechanism (Bootstrapping technique is to implement a decryption circuit in the encryption domain to create a new ciphertext associated with the same message but with fewer evaluations in the encryption domain.) "refresh" the ciphertexts. Inspired by Gentry's blueprint, different FHE schemes [13–19] have been proposed in the following years.

Our work was inspired by the HE scheme proposed by Fan and Vercauteren [17], inherited from [14–16,20–26], and is one of the widely used FHE schemes up to now. To avoid confusion, we always use the "BFV scheme" to represent SHE schemes proposed in [17]. Since the BFV is one of the SHE schemes, the corresponding ciphertext size is determined by the number of evaluations in the encryption domain. While the execution speed of each operation in the BFV scheme would decrease as the ciphertext-size increase; therefore, it is hard to realize complicated functions in a practical use case. On the other hand, if the target function is simple enough, then the ciphertexts can be small, and the performance of the BFV scheme can outperform subgroup-RSA. We observed this fact in our experiments; that is, the encryption of the BFV scheme can be realized within 0.21 ms in our protocol, whereas the encryption of subgroup-RSA needs 0.7 ms. The remarks about and comparisons with recently published related work will be presented in Section 5.3.

1.2. Secure Decryption

Let p be the public key and s be the secret key. We say an encryption scheme can implement an operation $\star$: $\mathcal{P} \times \mathcal{P} \to \mathcal{P}$ in the encryption domain if there exists a computable function T such that

$$\mathrm{Dec}\,(T\,(c_1, c_2, p), s) = \mathrm{Dec}(c_1, s) \star \mathrm{Dec}(c_2, s) \tag{1}$$

for arbitrary ciphertexts $c_1, c_2 \in \mathcal{C}$. Those encryption schemes that satisfy the property mentioned in Equation (1) are the HE schemes. Since HE enables us to compute over the ciphertexts, we may let the party that does not have a secret key compute the function in the encryption domain; and the other party decrypts the ciphertext(s) associated with the result. The problem is, if the latter party may learn other information while decrypting, then the security of the former party could be compromised. On the other hand, if the two parties can securely decrypt the result, we can achieve the security requirement of two-party computation (2PC), as shown in Figure 1.

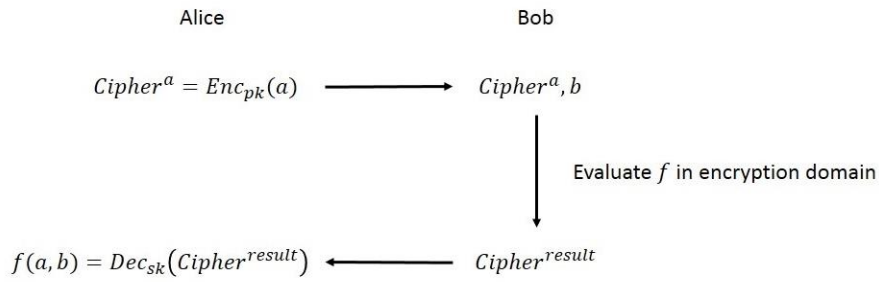

Figure 1. The Information Flow of Realizing 2PC via the BFV Scheme and the Secure Decryption Process. Party 1(say Alice) holds the secret key *sk* and public key *pk*; Party 2 (say Bob) holds the ciphertext Cipher and the public key *pk*. The goal of secure decryption is to securely compute Dec (Cipher, *sk*) for Party 1.

2. The BFV Encryption Scheme

BFV scheme is an attractive encryption scheme that is equipped with fast encryption and decryption operations, vector-like plaintext space (optional), and robust computations in the encryption domain. We review some functionalities of the BFV scheme in this section. Other BFV-related operations, such as bootstrapping, can be found in [11,15,17,27]. Moreover, for self-completeness, the algebraic structure and commonly used notations of the BFV scheme are presented in Appendix A.

This section addresses the assumptions and some characteristics related to the BFV scheme. As anonymous reviewers suggest, all the corresponding proof will be presented in Appendix B.

2.1. Security Assumption

Definition 1. *(Ring Learning with Error Distribution) Let $\mathcal{R}_q$ be the m-th cyclotomic ring with coefficients in $\mathbb{Z}_q$. Let $\mathbf{s} \in \mathcal{R}_q$ and χ be a small distribution with $Prob\{|\chi| > B\} = 0$. Then, the ring learning with error distribution $RLWE_{s,\chi}^{q,m}$ is the distribution of $(Y,\ Y \cdot \mathbf{s} + \chi^n)$, where Y is a uniform distribution in $\mathcal{R}_q$ and n is the degree of $\mathcal{R}_q$.*

Assumption 1. (*Infeasibility of decision-RLWE) The decision problem of Ring Learning with Error (RLWE), denoted by decision-*$\mathrm{RLWE}_{s,\chi}^{q,m}$*, is to distinguish uniform distribution and* $\mathrm{RLWE}_{s,\chi}^{q,m}$*. The decision-RLWE assumption is to assume decision-*$\mathrm{RLWE}_{s,\chi}^{q,m}$ *is infeasible.*

The decision-RLWE assumption establishes the security assumption of the BFV encryption scheme. Moreover, we can reduce it to the well-known shortest vector problem (SVP) if χ is a discrete Gaussian distribution with sufficient large standard error (e.g., $\sigma = \Omega(\sqrt{n})$) and the secret key is sampled from χ^n [28]. However, we sometimes use small standard error (e.g., $\sigma = 3.2$ in [29]) for better performance.

2.2. Ciphertext and Algebra

Given $m = 2^d$, $n = m/2$, and let $\Phi_m(X)$ be the m-th cyclotomic polynomial. where $\Phi_m(X) = (X^n + 1)$. Let $\mathcal{R} = \mathbb{Z}[X]/(\Phi_m(X))$ be the cyclotomic ring with degree n. Plaintexts are in $\mathcal{R}_p = \mathcal{R}/p\mathcal{R}$, and ciphertexts are in $\mathcal{R}_q^2 = (\mathcal{R}/q\mathcal{R})^2$, where $q \gg p$ and the ratio $\rho = q/p$ can be treated as the "tolerance" of error.

Let χ be a small distribution, say $Prob\{|\chi| > B\} = 0$.

2.3. Encryption and Decryption

2.3.1. Encryption and Decryption on Polynomial Algebra

Multiplication in $\mathcal{R}_q$ (denoted as $\mathrm{PolyMul}_n$, cf. Algorithm 1 for detailed definition) can be implemented in $\Theta(n \cdot \log n)$ operations in $\mathbb{Z}_q$ by Fast Fourier Transformation (FFT), and the time complexity would be $\Theta(n \cdot \log n \cdot \log q)$. For convenience, we use a bolder

case to denote a polynomial, which can be stored as n coefficients in $[0, q-1)$ in a computer, and it would be the requirement of all of the following algorithms. Similarly, each plaintext could be stored as n coefficients in $[0, p-1)$.

Algorithm 1 PolyMul_n

Input: $\mathbf{a}, \mathbf{b}, q, \alpha$
Output: $\mathbf{c}$

Ensure:
1. $\mathbf{c} \equiv \mathbf{a} \cdot \mathbf{b} \bmod (\Phi_n(X), q)$
2. $0 \leq c_i < q$

$\mathbf{A} = (A_0, A_1, \ldots, A_{n-1}) \leftarrow \text{FFT}_{n,q,\alpha}(\mathbf{a})$
$\mathbf{B} = (B_0, B_1, \ldots, B_{n-1}) \leftarrow \text{FFT}_{n,q,\alpha}(\mathbf{b})$
for $i = 0, 1, \ldots, n-1$ **do**
    $C_i \leftarrow A_i \cdot B_i \bmod q$
end for
$\mathbf{C} \leftarrow (C_0, C_1, \ldots, C_{n-1})$
$\mathbf{c} = (c_0, c_1, \ldots, c_{n-1}) \leftarrow \text{InvFFT}_{n,q,\alpha}(\mathbf{C})$

Once again, we assume the secret key, $\mathbf{s}$, is generated from a small distribution χ (cf. Algorithm 2 for details).

Algorithm 2 KeyGen

Input: $\perp$
Output: $\mathbf{s}$

for $i = 0, 1, \ldots, n-1$ **do**
    $s_i \leftarrow \chi$
end for
$\mathbf{s} \leftarrow (s_0, s_1, \ldots, s_{n-1})$

Algorithm 3 EncSym
Input: $\mathbf{m}$, $\mathbf{s}$

Output: $(\mathbf{c}_0, \mathbf{c}_1)$

Require:
1. $0 \leq m_i < p$
2. $\text{Prob}\{\chi > \rho\} = 0$

Ensure:
1. $\mathbf{m} = \left| \rho^{-1} \cdot \left(\mathbf{c}_0 + \text{PolyMul}_{n,q,\alpha}(\mathbf{c}_1, \mathbf{s}) \right) \right|$
2. Each element of $(\mathbf{c}_0, \mathbf{c}_1)$ is in the range $\{0, 1, \ldots, q-1\}$

for $i = 0, 1, \ldots, n-1$ **do**
    $c_i \leftarrow \{0, 1, \ldots, q-1\}$
end for
$\mathbf{c}_1 \leftarrow (c_0, c_1, \ldots, c_{n-1})$
for $i = 0, 1, \ldots, n-1$ **do**
    $e_i \leftarrow \chi$
end for
$\mathbf{e} \leftarrow (e_0, e_1, \ldots, e_{n-1})$
$\mathbf{c}_0 \leftarrow \mathbf{e} - \text{PolyMul}_n(\mathbf{c}_1, \mathbf{s}) + \mathbf{m} \cdot \rho$
$\mathbf{c}_0 \leftarrow \mathbf{c}_0 \bmod q$

EncSym (Algorithm 3) is to encrypt a message $\mathbf{m}$ with secret key $\mathbf{s}$. That is,

$$\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s} = \mathbf{m} \cdot \rho + \mathbf{e}. \tag{2}$$

We may use **Algorithm 4** to define the error with the corresponding message and ciphertext. Moreover, the ciphertext can be correctly decrypted if and only if the error is less than $\rho/2$ (see **Algorithm 5**). We can summarize this subsection's discussions into the following Lemma.

Lemma 1. *(Sufficient Condition for Correct Decryption). Given ciphertext* $\mathbf{c} = (\mathbf{c}_0,\ \mathbf{c}_1) \in \mathcal{R}_q^2$*, and message* $\mathbf{m} \in \mathcal{R}_p$*, and* $\mathbf{e} = \left[\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s} - [\mathbf{m}]_p \cdot \rho\right]_q$ *is the corresponding error, then* $\|\boldsymbol{e}\|_\infty < \rho \cdot 2^{-1}$ *implies* $[\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s}]_q \cdot \rho^{-1} \equiv [\mathbf{m}]_p \ mod\ p$*. Thus, we can decrypt correctly if the error is small enough.*

Proof. Please refer to Appendix B for detailed proof. □

Algorithm 4 CalculateError

Input: $(\mathbf{c}_0, \mathbf{c}_1),\ \mathbf{m},\ \mathbf{s}$
Output: $\mathbf{e}$

1: $\mathbf{e} \leftarrow \mathbf{c}_0 + \mathrm{PolyMul}_{n,q,\alpha}(\mathbf{c}_1, \mathbf{s}) - \mathbf{m} \cdot \rho$
2: $\mathbf{e} \leftarrow \mathbf{e} \bmod q$

Algorithm 5 Dec

Input: $(\mathbf{c}_0, \mathbf{c}_1),\ \mathbf{s}$
Output: $\mathbf{m}$

Require:
1. Each element of CalculateError$(\mathbf{c}_0, \mathbf{c}_1, \mathbf{m})$ is in the range $\{0, 1, \ldots, \rho/2 - 1\} \cup \{q - \rho/2, \ldots, q - 1\}$

1: $\mathbf{m} = (m_0, m_1, \ldots, m_{n-1}) \leftarrow \mathbf{c}_0 + \mathrm{PolyMul}_{n,q,\alpha}(\mathbf{c}_1, \mathbf{s}) \mod q$
2: **for** $i = 0, 1, \ldots, n-1$ **do**
3: $\quad m_i \leftarrow \lfloor (m_i + \rho/2) \cdot \rho^{-1} \rfloor$ ▷*Divide ρ and round*
4: **end for**
5: $\mathbf{m} \leftarrow (m_0, \ldots, m_{n-1})$

2.3.2. The Correctness of Homomorphic Operations

In this subsection, we introduce the addition between two ciphertexts (denoted as Add), the addition between a ciphertext and a plaintext (denoted as AddPlain, cf. Algorithm 6), and the multiplication between a ciphertext and a plaintext (denoted as MulPlain, cf. Algorithm 7).

Given $\mathbf{c}, \mathbf{c}' \in \mathcal{R}_q^2$, suppose the corresponding messages are $\mathbf{m},\ \mathbf{m}' \in \mathcal{R}_p$, and the corresponding errors are $\mathbf{e}, \mathbf{e}' \in \mathcal{R}$. The above-mentioned operations can be defined as follows:

$$\text{Add } (\mathbf{c}, \mathbf{c}'): \text{Return } (\mathbf{c}_0 + \mathbf{c}'_0, \mathbf{c}_1 + \mathbf{c}'_1)$$

$$\text{AddPlain } (\mathbf{c}, \mathbf{m}'): \text{Return } (\mathbf{c}_0 + [\mathbf{m}']_p \cdot \rho\ , \mathbf{c}_1)$$

$$\text{MulPlain } (\mathbf{c}, \mathbf{m}'): \text{Return } (\mathbf{c}_0 \cdot [\mathbf{m}']_p, \mathbf{c}_1 \cdot [\mathbf{m}']_p)$$

Claim 1. *(Correctness of Addition). Suppose* $\|\mathbf{e} + \mathbf{e}'\|_\infty$*, then*

$$\mathrm{Dec}\left(\mathrm{Add}\left(\mathbf{c}, \mathbf{c}'\right),\ s\right) \ =\ m + m'. \tag{3}$$

We illustrate the definition and the operation process of the homomorphic addition (denoted as Add, cf. Algorithm 8 for details).

Proof. Please refer to Appendix B.2.a for details. □

Claim 2. (*Correctness of Addition between a Plaintext and a Ciphertext). Suppose* $\|\mathbf{e}\|_\infty < 2^{-1} \cdot \rho$, *then*

$$\text{Dec}\ (\text{AddPlain}\ (\mathbf{c}, \mathbf{m'}), \mathbf{s}) = \mathbf{m} + \mathbf{m'}. \tag{4}$$

Proof. Please refer to Appendix B.2.b for details. □

Claim 3. *(Correctness of Multiplication between a Plaintext and a Ciphertext). Suppose* $\|\mathbf{e} \cdot \mathbf{m}'\|_\infty < 2^{-1} \cdot \rho$, *then*

$$\text{Dec}\ (\text{MulPlain}\ (\mathbf{c}, \mathbf{m'}), \mathbf{s}) = \mathbf{m} \cdot \mathbf{m'}. \tag{5}$$

Proof. Please refer to Appendix B.2.c for details. □

Notice that all the correctness mentioned above is associated with the magnitude of the norm of errors. Claims 1, 2, and 3 are essential for estimating how large q should be.

Algorithm 6 AddPlain	
Input:	$(\mathbf{c}_0, \mathbf{c}_1),\ \mathbf{m}'$
Output:	$\left(\mathbf{c}_0^{'}, \mathbf{c}_1^{'}\right)$
Require:	$\text{Dec}(\mathbf{c}_0, \mathbf{c}_1, \mathbf{s}) = \mathbf{m}$
Ensure:	$\text{Dec}\left(\mathbf{c}_0^{'}, \mathbf{c}_1^{'}, \mathbf{s}\right) = \mathbf{m} + \mathbf{m}'$
1:	$\mathbf{c}_0^{'} \leftarrow \mathbf{c}_0 + \mathbf{m}' \cdot \rho \mod q$
2:	$\mathbf{c}_1^{'} \leftarrow \mathbf{c}_1$

Algorithm 7 MulPlain	
Input:	$(\mathbf{c}_0, \mathbf{c}_1),\ \mathbf{m}'$
Output:	$\left(\mathbf{c}_0^{'}, \mathbf{c}_1^{'}\right)$
Require:	Let $\mathbf{e} = \text{CalculateError}(\mathbf{c}_0, \mathbf{c}_1, \mathbf{m}, \mathbf{s})$ $\mathbf{e}' = \text{PolyMul}_n(\mathbf{e}, \mathbf{m}') \bmod q$ Require $\|\mathbf{e}'\|_\infty < \rho/2$
Ensure:	$\text{Dec}\left(\mathbf{c}_0^{'}, \mathbf{c}_1^{'}, \mathbf{s}\right) \equiv \text{PolyMul}_n(\mathbf{m}, \mathbf{m}') \bmod q$
1:	$\mathbf{c}_0^{'} \leftarrow \text{PolyMul}_n(\mathbf{c}_0, \mathbf{m}') \mod q$
2:	$\mathbf{c}_1^{'} \leftarrow \text{PolyMul}_n(\mathbf{c}_1, \mathbf{m}') \mod q$

Algorithm 8 Add	
Input:	$(\mathbf{c}_0, \mathbf{c}_1),\ \left(\mathbf{c}_0^{'}, \mathbf{c}_1^{'}\right)$
Output:	$\left(\mathbf{c}_0^{add}, \mathbf{c}_1^{add}\right)$
Require:	Let $\mathbf{e} = \text{CalculateError}(\mathbf{c}_0, \mathbf{c}_1, \mathbf{m}, \mathbf{s})$, $\mathbf{e}' = \text{CalculateError}\left(\mathbf{c}_0^{'}, \mathbf{c}_1^{'}, \mathbf{m}', \mathbf{s}\right)$ Require $\|\mathbf{e} + \mathbf{e}'\|_\infty < \rho/2$
Ensure:	$\text{Dec}\left(\mathbf{c}_0^{'}, \mathbf{c}_1^{'}, \mathbf{s}\right) = \mathbf{m} + \mathbf{m}'$
1:	$\mathbf{c}_0^{add} \leftarrow \mathbf{c}_0 + \mathbf{c}_0^{'} \mod q$
2:	$\mathbf{c}_1^{add} \leftarrow \mathbf{c}_1 + \mathbf{c}_1^{'} \mod q$

2.4. Public Key Generation and Encryption

The public key generation (denoted as PubKeyGen) involved in our protocol can be illustrated in Algorithm 9. Moreover, the asymmetric encryption function (denoted as EncAsym) concerning our protocol can be depicted in Algorithm 10.

Algorithm 9 PubKeyGen

Input: $\mathbf{s}$
Output: $(\mathbf{pk}_0, \mathbf{pk}_1)$

1: $\mathbf{pk}_0, \mathbf{pk}_1 \leftarrow \text{EncSym}(0, \mathbf{s})$

Algorithm 10 EncAsym

Input: $\mathbf{m}, (\mathbf{pk}_0, \mathbf{pk}_1)$
Output: $(\mathbf{c}_0, \mathbf{c}_1)$

1: **for** $i = 0, 1, \ldots, n-1$ **do**
2: $\quad r_i \leftarrow \chi$
3: $\quad e_i \leftarrow \chi$
4: $\quad e_i^{'} \leftarrow \chi$
5: **end for**
6: $\mathbf{r} \leftarrow (r_0, r_1, \ldots, r_{n-1})$, $\mathbf{e} \leftarrow (e_0, e_1, \ldots, e_{n-1})$, $\mathbf{e}' \leftarrow \left(e_0^{'}, e_1^{'}, \ldots, e_{n-1}^{'}\right)$
7: $\mathbf{c}_0 \leftarrow \text{PolyMul}_n(\mathbf{pk}_0, \mathbf{r}) + \mathbf{e} + \mathbf{m} \cdot \rho \bmod q$
8: $\mathbf{c}_1 \leftarrow \text{PolyMul}_n(\mathbf{pk}_1, \mathbf{r}) + \mathbf{e}' \bmod q$

To claim the correctness of EncAsym, we should evaluate the norm of the error:

$$\text{CalculateError}\left(\text{EncAsym}_{\mathbf{pk}}(\mathbf{m}), \mathbf{s}\right) = \mathbf{r} \cdot (\mathbf{pk}_0 + \mathbf{pk}_1 \cdot \mathbf{s}) + \mathbf{e} + \mathbf{e}' \cdot \mathbf{s}$$

Notice $\mathbf{pk}_0 + \mathbf{pk}_1 \cdot \mathbf{s}$, $\mathbf{r}$, $\mathbf{e}$, $\mathbf{e'}$, and $\mathbf{s}$ are sampling from χ^n and bounded by B; thus, we have

$$\|\mathbf{r} \cdot (\mathbf{pk}_0 + \mathbf{pk}_1 \cdot \mathbf{s}) + \mathbf{e} + \mathbf{e}' \cdot \mathbf{s}\|_\infty \leq n \cdot B^2 + B + n \cdot B^2.$$

Recall Algorithm 5—the correctness of decryption is ensured if ρ bounds the error. Therefore, we can choose q large enough, so the correctness follows. That is,

$$\rho = q \cdot p^{-1} > 2 \cdot n \cdot B^2 + B \geq \left\|\text{CalculateError}\left(\text{EncAsym}_{\mathbf{pk}}(\mathbf{m}), \mathbf{s}\right)\right\|_\infty \tag{6}$$

2.5. Time Complexity

Additions in $\mathcal{R}_q$ needs n additions in $\mathbb{Z}_q$, and the time complexity would be $\Theta(n \cdot \log q)$. Multiplications can be implemented in $\Theta(n \cdot \log n \cdot \log q)$ by FFT implementation of the polynomial ring. Let λ be the security parameter, suppose the m-bit pseudo-random number can be generated in $\Theta(m \cdot \lambda)$, then the time complexity associated with each operation can be listed in Table 1.

Since $\rho = p^{-1} \cdot q$ can be regarded as the affordability of error, q should be large enough for the evaluations. However, the security level is inversely proportional to $\log q$, so n should increase proportional to preserve the same security level. Therefore, we should use the number of evaluations to represent the time complexity.

As seen in Table 1, the performance is dominated by n, q. Recall that the error should be less than $\rho \cdot 2^{-1}$. Each multiplication would increase the upper bound of the error by a factor of $n \cdot p \cdot 2^{-1}$, and the error of fresh ciphertext is less than B, so the error after d multiplications is at most $\left(n \cdot p \cdot 2^{-1}\right)^d \cdot B$, and we choose ρ, q large enough to keep the correctness:

$$\log_2 \rho - 1 > d \cdot \log\left(n \cdot p \cdot 2^{-1}\right) + \log B \tag{7a}$$

$$\log_2 q > d \cdot (\log_2 p + \log_2 n - 1) + \log_2 B + 1. \tag{7b}$$

Let $\lambda = \Theta(n/\log_2 q)$ be the security level, then $n = \Omega(\lambda \cdot \log_2 q) = \Omega(\lambda \cdot d \cdot \log_2 p)$. We conclude that the computational cost has a quadratic growth with respect to d.

Table 1. Time Complexity of BFV Operations.

Operation	Time Complexity
PolyMul	$n \cdot \log n \cdot \log q$
KeyGen	$n \cdot \log q \cdot \lambda$
EncSym	$n \cdot \log n \cdot \log q + n \cdot \log q \cdot \lambda$
PubKeyGen	$n \cdot \log n \cdot \log q + n \cdot \log q \cdot \lambda$
EncAsym	$n \cdot \log n \cdot \log q + n \cdot \log q \cdot \lambda$
Dec	$n \cdot \log n \cdot \log q$
AddPlain	$n \cdot \log q$
MulPlain	$n \cdot \log n \cdot \log q$
Add	$n \cdot \log q$

2.6. Batching

Batching is a commonly used technique to speed up cryptographic operations; however, its relation to mathematical derivations has seldom been presented and discussed. Since batching plays a crucial role in our proposed protocol, we detail its definition and derivation in this subsection. Readers who are familiar with batching can jump to the next section directly.

Suppose $\Phi_m(X) = f_1(X) \cdot f_2(X) \cdot \cdots \cdot f_k(X)$ where $f_1, \ldots, f_k$ are co-prime, then we can apply Chinese Remainder Theorem to derive the following isomorphic relation:

$$\mathbb{Z}_p[X]/(\Phi_m(X)) \cong \mathbb{Z}_p[X]/(f_1(X)) \times \cdots \times \mathbb{Z}_p[X]/(f_1(X)) \tag{8}$$

If we further assume $f_i(X)$ is irreducible, and $\deg(f_i) = d$ for each i, then

$$\mathbb{Z}_p[X]/(f_i(X)) \cong GF\left(p^d\right) \tag{9a}$$

$$\mathcal{R}_p \cong GF\left(p^d\right)^{n/d} \tag{9b}$$

Generally, the isomorphic relation given in (Equation (9)) always holds. In the following paragraphs, we will introduce how to choose p for some given d and n.

For convenience, we always choose $m = 2^{t+1}$, so $n = \varphi(m) = 2^t$, and $\Phi_m(X) = X^n + 1$. We claim that $\mathcal{R}_p \cong \mathbb{Z}_p^n$ if p is a prime with $p = 2kn + 1$:

We can find a generator $g \in \mathbb{Z}_p^*$, $\alpha = g^k$ has order $2n$, and $\alpha^n = -1$ is a root of Φ_m. Similarly, let $\alpha_i = \alpha^{2i+1}$, then α_i has order $2n$ and is also a root of $X^n + 1$. Since $\{\alpha_i\}_{0 \le i < n}$ are n different roots of Φ_m, we may write

$$X^n + 1 = (X - \alpha_0) \cdot (X - \alpha_1) \cdot \cdots \cdot (X - \alpha_{n-1}) \tag{10}$$

and

$$\mathbb{Z}_p[X]/(\Phi_m(X)) \cong \mathbb{Z}_p[X]/(X - \alpha_0) \times \cdots \times \mathbb{Z}_p[X]/(X - \alpha_{n-1}) \cong \mathbb{Z}_p^n \tag{11}$$

Notice that $f \mapsto (f \bmod (f(X) - \alpha_i))_{0 \le i < n}$ is an isomorphism from $\mathbb{Z}_p[X]/(\Phi_m(X))$ to $\mathbb{Z}_p^n$, and $f \bmod (f(X) - \alpha_i)$ can be determined by $f(\alpha_i)$. We may apply the Number Theoretic Transform (NTT) to send f to $(f(\alpha_i))_{0 \le i < n}$ with $O(n \cdot \log n)$ operations in $\mathbb{Z}_p$. We may also let $Y = X^d$, so

$$Y^{n/d} + 1 = (Y - \beta_0) \cdot (Y - \beta_1) \cdot \cdots \cdot (Y - \beta_{n/d-1})$$

$$X^n + 1 = \left(X^d - \beta_0\right) \cdot \left(X^d - \beta_1\right) \cdot \cdots \cdot \left(X^d - \beta_{n/d-1}\right), \tag{12}$$

where $\beta_i = \beta^{2i+1}$, and $\beta = \alpha^d$. We derive another isomorphic relation

$$\mathbb{Z}_p[X]/(\Phi_m(X)) \cong \mathbb{Z}_p[X]/\left(X^d - \beta_0\right) \times \cdots \times \mathbb{Z}_p[X]/\left(X^d - \beta_{n/d-1}\right) \tag{13}$$

$$\cong \left(\mathbb{Z}_p[X]/\left(X^d + 1\right)\right)^d \tag{14}$$

We may write $f = \sum_{j=0}^{d-1} f_j(Y) \cdot X^j$ with $Y = X^d$, then the isomorphism (cf. Equation (13)) can be defined as

$$f \mapsto \sum_{j=0}^{d-1} \left(f_j(Y) \; mod \; (Y - \beta_i)\right)_{0 \le i < n/d} \cdot X^j \tag{15}$$

Similarly, $f_j \; mod \; (Y - \alpha_i)$ can be determined by $f_j(\beta_i)$, and the values of $\left(f_j(\beta_i)\right)_{0 \le i < n/d}$ can be calculated by n/d-dimension NTT, which needs $\boldsymbol{O}$ $((\mathrm{nd}^{-1}) \cdot \log (\mathrm{nd}^{-1}))$ operations in $\mathbb{Z}$p or the time complexity is $\boldsymbol{O}$ $((\mathrm{nd}^{-1}) \cdot \log (\mathrm{nd}^{-1})) \cdot \boldsymbol{O}$ (log q). Consequently, the time complexity of the whole map becomes the summation of d different NTTs plus the time needed for combining them, which equals

$$\boldsymbol{O}\ (\mathrm{d} \cdot (\mathrm{nd}^{-1}) \cdot \log (\mathrm{nd}^{-1}) \cdot \log \mathrm{q}) + \boldsymbol{O}\ (\mathrm{n} \cdot \log \mathrm{q}) = \boldsymbol{O}\ (\mathrm{n} \cdot \log \mathrm{q} \cdot (\log \mathrm{n} - \log \mathrm{d})). \tag{16}$$

As illustrated in Figure 2, the function that maps $(\mathrm{Zp}\ [\mathrm{X}\ ]/(\Phi \mathrm{d}\ (\mathrm{X})))^{\mathrm{n/d}}$ to $\mathrm{R_{m,p}}$ can be regarded as an "encoder", and its inverse is the "decoder". Notice we always suppose $2 \cdot (\mathrm{n/d})$ divides (p—1), and n is a power of 2. In the following, we will use $\mathrm{Encode}_{n/d,\alpha^d}$ to denote the function from $\mathbb{Z}_p[X]/\left(X^d - \beta_0\right) \times \cdots \times \mathbb{Z}_p[X]/\left(X^d - \beta_{n/d-1}\right)$ to $\mathbb{Z}_p[X]/(X^n + 1)$, and $\mathrm{Decode}_{n/d,\alpha^d}$ will be the inverse of Encode.

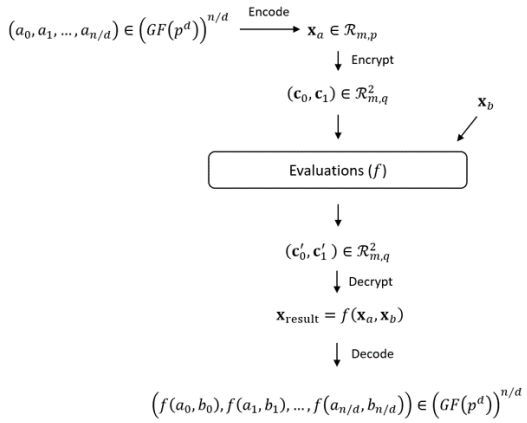

Figure 2. The Information Flow and the Implementation of the Proposed HE-Domain's Encoding and Decoding Functions.

3. Two-Party Computation and BFV Encryption

In the following contexts, we name the two players (or parties) Alice and Bob. With the BFV scheme, Alice can delegate the computation to Bob without revealing her private input. Let **c**result denote the ciphertext computed by Bob. Suppose there is a protocol that can derive the message in **c**result without leaking any information which may infringe the privacy of Bob. In that case, we say the protocol satisfies the security requirement of two-party computation via the BFV scheme, or the protocol can derive the message securely through "secure decryption".

3.1. The Secure Decryption Protocol

Suppose Bob holds some ciphertext $\mathbf{c}^{\text{result}}$, Alice holds the secure key $\mathbf{s}$, and the goal of secure decryption is to compute Dec ($\mathbf{c}^{\text{result}}$, s) securely, which can be formulated as in Table 2 notice if the secure decryption exists, and Bob can compute the function f in the encryption domain, then they can securely compute f by

Table 2. Formulas Associated with the Secure Decryption.

Party	*A*	*B*
Input	$\mathbf{s}$	$\mathbf{c}^{result}, \mathbf{pk}$
Output	$\mathrm{Dec}\left(\mathbf{c}^{result}, \mathbf{s}\right)$	$\perp$

(i) Alice encrypting her private input(s) and sending the results to Bob.

(ii) Bob computing the function f in the encryption domain and obtaining the ciphertext $\mathbf{c}^{\text{result}}$ associated with the desired result, called $\mathbf{m}^{\text{result}}$.

(iii) Performing the above "secure decryption" protocol; then, Alice learns the $\mathbf{m}^{\text{result}}$ without knowing any other information.

Notice that the first term of the public key, say p_1, is sampled from a uniform distribution. If Bob randomly sampled $\mathbf{r} \leftarrow \chi^n, \mathbf{e} \leftarrow \chi^n$, then the pair $(\mathbf{pk}_1, \mathbf{pk}_1 \cdot \mathbf{r} + \mathbf{e})$ "looks like" a pair of uniformly sampled samples in the view of Alice, by the RLWE assumption. Based on this idea, we will construct our first secure decryption protocol in the remaining context of this section.

Suppose the error of $\mathbf{c}^{result}$ is $\mathbf{e}_{result}$, the error of $\mathbf{c}^{zero}$ is $\mathbf{e}_{zero}$. Let $\mathbf{c}' = \mathbf{c}^{result} + \mathbf{c}^{zero} + (\mathbf{e}, 0)$, then the error in $\mathbf{c}'$ is $\mathbf{e}_{result} + \mathbf{e}_{zero} + \mathbf{e}$. By Lemma 2, $\|\mathbf{e}_{result} + \mathbf{e}_{zero} + \mathbf{e}\|_\infty$ implies Dec ($\mathbf{c}'$, $\mathbf{s}$) = Dec ($\mathbf{c}^{\text{result}}$, $\mathbf{s}$). Before proving the security of Protocol 3.1, we first analyze the distinguishability between $\mathbf{e}_{\text{result}} + \mathbf{e}_{\text{zero}} + \mathbf{e}$ and a uniform sample from $\{-T, \ldots, T\}^n$.

Lemma 2. (*Distinguishability of a uniform sample with little bias). For some* $\mathrm{e} \in \mathbb{Z}$, *distinguishing two samples, one is sampled uniformly from* $\{-T, -T+1, \ldots, T\}$, *the other is from* $\{-T+e, -T+1, \ldots, T+e\}$, *the advantage of the adversary is no more than* $|e| \cdot (2T)^{-1}$.

Proof. Please refer to Appendix C for details. □

Corollary 1. *(Distinguishability of a uniform vector with little bias). For some* $\boldsymbol{e} \in \mathbb{Z}^n$, distinguishing two samples, one is sampled from $\{-T, -T+1, \ldots, T\}^n$, the other is from $\{-T, -T+1, \ldots, T\}^n + \boldsymbol{e}$, *the advantage of the adversary is no more than* $\Sigma_i |e_i| \cdot (2T)^{-1}$.

Proof. Since each entry of the vector is independent to each other, the adversary of i-th entry is no more than $|e_i|$ by Lemma 2. Adversary of the whole vector is no more than summing up the adversary of each entry, which is $\Sigma_i |e_i| \cdot (2T)^{-1}$. □

Notice that her security of Alice is guaranteed based on the security of the BFV scheme, so we need only to deal with the security of Bob. We will prove this by simulating the view of Alice from Dec ($\mathbf{c}^{\text{result}}$, $\mathbf{s}$) and $\mathbf{s}$.

Claim 4. *(The Security of Bob in Protocol 3.1). Suppose* $\|\boldsymbol{e}result + \boldsymbol{e}zero\|_1 < Esum, T > Esum \cdot (2\varepsilon)^{-1}$, *and RLWE assumption holds, then the view of Alice, which is* $\boldsymbol{c}result \in (R_{m,q})^2$, *can be simulated by Dec* ($\boldsymbol{c}_b$, $\boldsymbol{s}$) *and* $\boldsymbol{s}$.

Proof. Please refer to Appendix C for details. □

Protocol 3.1. The Proposed Secure Decryption Protocol.

Protocol 3.1: Decryption Protocol		
Party:	*A*	*B*
Input:	$\mathbf{s}$	$\mathbf{c}^{result}, \mathbf{pk}$
Output:	$\mathrm{Dec}\left(\mathbf{c}^{result}, \mathbf{s}\right)$	$\perp$
	Round 1 (*B*'s turn)	
1:	$\mathbf{c}^0 \leftarrow \mathrm{EncAsym}_{\mathbf{pk}}(0)$	
2:	$\mathbf{e} \leftarrow \{-T, \ldots, T\}^n$	
3:	$\mathbf{c}' = \left(\mathbf{c}_0^{'}, \mathbf{c}_1^{'}\right) \leftarrow \mathbf{c}^{result} + \mathbf{c}^0$	
4:	$\mathbf{c}_0^{'} \leftarrow \mathbf{c}_0^{'} + \mathbf{e} \bmod q$	
5:	*B* sends $\left(\mathbf{c}_0^{'}, \mathbf{c}_1^{'}\right)$ to *A*	
	Round 2 (A's turn)	
6:	$\mathbf{m} \leftarrow \mathrm{Dec}(\mathbf{c}', \mathbf{s})$	
7:	Establish $\mathbf{m}$	

In a real case, we do not know the exact value of E_{sum}, but we can estimate an upper bound of $\|\mathbf{e}_b + \mathbf{e}_{zero}\|_\infty$, say $E > \|\mathbf{e}_b + \mathbf{e}_{zero}\|_\infty$, by Claims 1, 2, 3. Therefore, $E_{sum} < n \cdot E$ (we sometimes estimate E_{sum} as $n^{1/2} \cdot E$ since EV[Esum] $\approx n^{1/2} \cdot E$) (notice $\varepsilon < n \cdot E \cdot q^{-1}$ can be smaller than $p(n)^{-1}$ for any polynomial p and sufficient large n since q is exponential to n for some fixed security parameter and $n \cdot E = \boldsymbol{O}(n^2)$ for a fixed task in the encryption domain). In the next section, we will introduce "separate decryption" to improve performance.

3.2. Separate Decryption

Let $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1) \in \mathcal{R}_q^2$; the result of decryption can be computed from $\mathbf{c}_0$ and $(\mathbf{c}_1 \cdot \mathbf{s})$. The idea of separate decryption is: Bob holds $\mathbf{c}_0$, Alice holds $(\mathbf{c}_1 \cdot \mathbf{s})$, and they compute the decryption result jointly.

Let $\mathbf{m}$ be the message, $\mathbf{e}$ be the correspond error. Further suppose

$$[\mathbf{c}_0]_q = [\mathbf{m}_0]_p \cdot \rho + \mathbf{e}_0 \text{ and } [\mathbf{c}_1 \cdot \mathbf{s}]_q = [\mathbf{m}_1]_p \cdot \rho + \mathbf{e}_1$$

where the coefficients of $\mathbf{e}_0, \mathbf{e}_1$ are in $\left(-\rho \cdot 2^{-1}, \rho \cdot 2^{-1}\right]$. By the definition of error, we also have $[\mathbf{c}_0]_q + [\mathbf{c}_1 \cdot \mathbf{s}]_q \equiv [\mathbf{m}]_p \cdot \rho + \mathbf{e} \bmod q$, so $\mathbf{e}_0 + \mathbf{e}_1 \equiv \mathbf{e} \bmod \rho$. By discussing the magnitude of $\mathbf{e}_0, \mathbf{e}_1, \mathbf{e}$, there exists $\mathbf{v}$ s.t.

$$(\mathbf{e}_0)_i + (\mathbf{e}_1)_i = (\mathbf{e})_i + \rho \cdot (\mathbf{v})_i \tag{17}$$

where $(\mathbf{v})_i \in \{0, 1\}$, and the message can be computed by $[\mathbf{m}]_p \equiv [\mathbf{m}_0]_p + [\mathbf{m}_1]_p + \mathbf{v}. \bmod p$.

For convenience, we will write $e_{0,i} = (\mathbf{e}_0)_i$, $e_{1,i} = (\mathbf{e}_1)_i$, $e_i = (\mathbf{e})_i$, and $v_i = (\mathbf{v})_i$. Let t ∈ {0,1}, we have

(i) If $e_{t,i} \geq 0$, then $e_{0,i} + e_{1,i} \geq e_{1-t,i} > -\rho \cdot 2^{-1}$, so v_i is either 0 or 1.

(ii) If $e_{t,i} < 0$, then $e_{0,i} + e_{1,i} < e_{1-t,i} \leq \rho \cdot 2^{-1}$, so v_i is either 0 or -1.

Suppose $|e_i| < E < \rho \cdot 4^{-1}$, and define

$$u_{t,i} = \begin{cases} 1 & \text{if } e_{t,i} \geq E \\ -1 & \text{if } e_{t,i} \leq -E \\ 0 & \text{else} \end{cases} \tag{18}$$

$$s_{t,i} = \begin{cases} 1 & \text{if } e_{t,i} > 0 \\ -1 & \text{if } e_{t,i} < 0 \\ 0 & \text{else} \end{cases} \tag{19}$$

So v_i can be evaluated from $u_{t,i}, s_{t,i}$:

(i) If $e_{t,i} = 1$, $s_{1-t,i} = 1$, then $v_i = 1$.

(ii) If $e_{t,i} = -1$, $s_{1-t,i} = 1$, then $v_i = -1$.
(iii) Else, $v_i = 0$

The procedures of separate decryption can now be listed as in Table 3. From Equation (17), we have $e_{0,i} = e_i - e_{1,i} + \rho \cdot v_i$. So, $v_i = 0$ if $-2^{-1} \cdot \rho < e_i - e_{1,i} < 2^{-1} \cdot \rho$, further suppose $e_{1,i}$ sampled uniformly from $\left(-\rho \cdot 2^{-1}, \rho \cdot 2^{-1}\right]$. Then, Prob$\{v_i = 0\} \geq 1 - |1 + e_i| \cdot \rho^{-1}$.

Table 3. Procedures of the Separable Secure Decryption (also known as **Protocol 3.2** in this write-up).

Protocol 3.2: Decryption Protocol			
Party:	A	B	
Input:	$\mathbf{s}$	$\mathbf{c}^{result}, \mathbf{pk}$	
Output:	$\mathrm{Dec}\left(\mathbf{c}^{result}, \mathbf{s}\right)$	$\perp$	
	Round 1 (A's turn)		
1:	$\mathbf{c}^0 \leftarrow \mathrm{EncAsym}_{\mathbf{pk}}(0)$		
2:	$\mathbf{c}' = \left(\mathbf{c}_0', \mathbf{c}_1'\right) \leftarrow \mathbf{c}^{result} + \mathbf{c}^0 \bmod q$		
3:	B sends $\mathbf{c}_1'$ to A		
	Round 2 (A's turn)		
4:	$\mathbf{m} \leftarrow \mathrm{PolyMul}_n\left(\mathbf{c}_1', \mathbf{s}\right)$		
5:	**for** $i = 0, 1, \ldots, n-1$ **do**		
6:	$t \leftarrow m_i + \rho \cdot 2^{-1} \bmod q$		
7:	$e \leftarrow t \bmod p$		
8:	$u_{0,i} \leftarrow 0$		▷*Compute* $u_{t,i}$ *in Equation* (18)
9:	**if** $e \geq E$ **and** $e \leq \rho/2$ **then**		
10:	$u_{0,i} \leftarrow 1$		
11:	**end if**		
12:	**if** $e \leq \rho - E$ **and** $e > \rho/2$ **then**		
13:	$u_{0,i} \leftarrow -1$		
14:	**end if**		
15:	$s_{0,i} \leftarrow 0$		▷*Compute* $s_{t,i}$ *in Equation* (19)
16:	**if** $e > \rho/2$ **then**		
17:	$s_{0,i} \leftarrow 1$		
18:	**end if**		
19:	**if** $e < \rho/2$ **then**		
20:	$s_{0,i} \leftarrow -1$		
21:	**end if**		
22:	$m_i \leftarrow t \cdot \rho^{-1}$		
23:	**end for**		
24:	$\mathbf{m} \leftarrow (m_0, m_1, \ldots, m_{n-1})$		
25:	A send $u_{0,i}$, $s_{0,i}$ to B.		
	Round 2 (B's turn)		
26:	$\left(c_{0,0}', c_{0,1}', \ldots, c_{0,n-1}'\right) \leftarrow \mathbf{c}_0'$		
27:	**for** $i = 0, 1, \ldots, n-1$ **do**		
28:	$t \leftarrow c_{0,i}' + \rho \cdot 2^{-1} \bmod q$		
29:	$e \leftarrow t \bmod p$		
30:	$u_{1,i} \leftarrow 0$		▷*Compute* $u_{t,i}$ *in Equation* (18)
31:	**if** $e \geq E$ **and** $e \leq \rho/2$ **then**		
32:	$u_{1,i} \leftarrow 1$		
33:	**end if**		
34:	**if** $e \leq \rho - E$ **and** $e > \rho/2$ **then**		
35:	$u_{1,i} \leftarrow -1$		
36:	**end if**		
37:	$s_{1,i} \leftarrow 0$		▷*Compute* $s_{t,i}$ *in Equation* (19)
38:	**if** $e > \rho/2$ **then**		
39:	$s_{1,i} \leftarrow 1$		
40:	**end if**		
41:	**if** $e < \rho/2$ **then**		
42:	$s_{1,i} \leftarrow -1$		
43:	**end if**		
44:	$v_i \leftarrow u_{0,i} + u_{1,i}$		

Table 3. *Cont.*

45:	**if** $sign(v_i) \neq sign(s_{0,i} + s_{1,i})$ **then**
46:	$v_i \leftarrow 0$
47:	**end if**
48:	$m_i^{'} \leftarrow t \cdot \rho^{-1} + v_i \bmod p$
49:	**end for**
50:	$\mathbf{m}' \leftarrow \left(m_0^{'}, m_1^{'}, \ldots, m_{n-1}^{'}\right)$
52:	B sends $\mathbf{m}'$ to A
	Round 3 (A's turn)
53:	$\mathbf{m} \leftarrow \mathbf{m} + \mathbf{m}' \bmod p$ $\triangleright$ *which is* $\mathrm{Dec}\left(\mathbf{c}^{result}, \mathbf{s}\right)$

Suppose $\sum_i \mathrm{Prob}\{\mathrm{ui} \neq 0\} < \varepsilon$ for some sufficient small $\varepsilon > 0$, then $\mathbf{m} = \mathbf{m}_0 + \mathbf{m}_1$ with probability greater than $1 - \varepsilon$. Therefore, the view of Alice, which is $(\mathbf{c}', \mathbf{m}_0)$, is equal to $(\mathbf{c}', \mathbf{m} - \mathbf{m}_1)$ with a probability Prob $\{\mathbf{u} = \mathbf{0}\} > 1 - \varepsilon$. Since $\mathbf{m}_1 = \mathrm{F}(\mathbf{c}', \mathbf{s})$ for some computable function F, $(\mathbf{c}', \mathbf{m} - \mathbf{m}_1) = (\mathbf{c}', \mathbf{m} - \mathrm{F}(\mathbf{c}', \mathbf{s}))$ is not distinguishable from $(\mathbf{r}, \mathbf{m} - \mathrm{F}(\mathbf{r}))$ (to Alice) by RLWE assumption.

An advantage of separate decryption is that we do not need to sample e from a large uniform distribution. Its other advantage is the lower communication rate because the size of $\mathbf{m}'$ is smaller than that of $\mathbf{c}_0^{'}$. However, we still need to choose a relatively large q for the indistinguishability in Lemma 2.

3.3. Two Party Computation via BFV Scheme

Since BFV encryption can implement arbitrary computable function f in the encryption domain, we can construct the two-party computation (2PC) protocol based on the BFV encryption scheme. That is, a 2PC protocol can be constructed in the following steps (more details are depicted in Table 4):

Table 4. The 2PC Computations via the BFV Scheme.

Protocol 3.3 2PC Protocol		
Party:	A	B
Input:	$\mathbf{a}^0, \mathbf{a}^1, \ldots, \mathbf{a}^{n_a}, \mathbf{s}$	$\mathbf{b}^0, \mathbf{b}^1, \ldots, \mathbf{b}^{n_b}, \mathbf{pk}$
Output:	$f\left(\mathbf{a}^0, \mathbf{a}^1, \ldots, \mathbf{a}^{n_a}, \mathbf{b}^0, \mathbf{b}^1, \ldots, \mathbf{b}^{n_b}\right)$	$\perp$
	Round 1 (A's turn)	
1:	**for** $i = 0, 1, \ldots, n_a$ **do**	
2:	$\mathbf{c}^i \leftarrow \mathrm{EncSym}\left(\mathbf{a}^i, \mathbf{s}\right)$	
3:	**end for**	
4:	A sends $\mathbf{c}^0, \ldots, \mathbf{c}^{n_a}$ to B	
5:	Round 2 (B's turn)	
6:	$\mathbf{c}^{result} \leftarrow f\left(\mathbf{c}^0, \ldots, \mathbf{c}^{n_a}, \mathbf{b}^0, \mathbf{b}^1, \ldots, \mathbf{b}^{n_b}\right)$	
	Round 3	
7:	A and B perform secure decryption to decrypt $\mathbf{c}^{result}$, and A derived $\mathrm{Dec}\left(\mathbf{c}^{result}, \mathbf{s}\right) = f\left(\mathbf{a}^0, \mathbf{a}^1, \ldots, \mathbf{a}^{n_a}, \mathbf{b}^0, \mathbf{b}^1, \ldots, \mathbf{b}^{n_b}\right)$	

Step 1. Alice encrypts her input and sends the encrypted result to Bob.

Step 2. Bob directly computes the ciphertext(s) corresponding to the result in the encryption domain and performs the secure decryption protocol to send the message to Alice.

4. A High Throughput, Semi-Honest, Secure Comparison Protocol

4.1. Improving DGK Secure Comparison Protocol via the BFV Scheme

As mentioned in Section I, the BFV scheme performs well if the target function has shallow multiplication depth. Since the multiplication depth of the function in DGK protocol is 1, we can accelerate DGK protocol via BFV encryption. We briefly describe the DGK protocol in the following:

DGK Protocol (Setup, KeyGen, Enc (Round 1, Round2), Dec)

Setup. Suppose both a and b are l-bit integers; say $0 \leq a, b < 2^l$, further assume $a = \sum_{i=0}^{l-1} a_i \cdot 2^i$, $b = \sum_{i=0}^{l-1} b_i \cdot 2^i$ with $a_i, b_i \in \{0,1\}$. Our goal is to compute whether $(a \geq b) \in \{0, 1\} \triangleq \{False, Truth\}$ securely. We use the plaintext space $\mathbb{Z}_p$, where $p > l+1$ is a prime number. Choose q large enough for evaluation and n large enough for desired security level. Before the protocol starts, Alice generates the secret key $\mathbf{s} \leftarrow \text{KeyGen}(\,)$ and the public key $\mathbf{pk} \leftarrow \text{PubKeyGen}(\mathbf{s})$, and sends the public key to Bob.

Enc. (Encryption)

Round 1. Alice computes $\mathbf{c}_i^a \leftarrow \text{EncSym}_{\mathbf{s}}(a_i)$, sends $\mathbf{c}_i^a$ to Bob.

Round 2. Bob generates the samples $r_i \leftarrow \mathbb{Z}_p^*$, and computes

$d_i = r_i \cdot \left(a_i - b_i + 1 + \sum_{j>i}(a_j - b_j) \cdot 2^{j+2}\right)$ in the encryption domain and permutes the ciphertexts randomly, so the corresponding message of $\mathbf{c}_i^{\text{result}}$ is $d_{\sigma(i)}$.

Let $\mathbf{c}^{\mathbf{result}} \leftarrow \sum_{0 \leq i<l} \mathbf{c}_i^{result} \cdot X^{\sigma(i)}$.

Dec. (Decryption)

Alice and Bob perform secure decryption, so Alice derives $d_{\sigma(i)}$, and establishes "$a < b$" if, and only if, $d_{\sigma(i)} = 0$ for some i.

Correctness of DGK.

Notice that $d_i = 0$ if and only if $a_i = 0, b_i = 1$, and $a_j = b_j$ for all $j > i$.

Security of DGK. Since ri is uniformly sampled from $\mathbb{Z}_p^*$, $d_i = r_i \cdot (a_i - b_i + 1 + \sum_{j>i}(a_j - b_j) \cdot 2^{j+2})$ is uniform in $\mathbb{Z}_p^*$ whenever $d_i \neq 0$. Therefore, the vector $\left(d_{\sigma(i)}\right) \sim \left(\mathbb{Z}_p^*\right)^n$, if $a \geq b$. In the case $b > a$, there is exactly one i such that $d_{\sigma(i)} = 0$. Since the permutation is random, the distribution is uniform in

$$\left\{(v_0, \ldots, v_{n-1}) \in \mathbb{Z}_p^n \,\middle|\, \text{Exactly one } v_i = 0\right\}$$

Each secure decryption protocol needs one encryption operation, so the Basic Secure Comparison Protocol (we named it Protocol 4.1 in this write-up) presented in Table 5 needs $l+1$ encryptions for each l-bit comparison. We will present and analyze various proposed high-speed comparison protocols in the following section.

4.2. High Throughput Comparison Protocol

To enhance the performance, we will use the batching technique presented in Section 2.6 to improve the utility of the plaintext space. Recall that

$\mathcal{R}_p \cong \left(\mathbb{Z}_p[X]/\left(X^d + 1\right)\right)^{n/d}$.

Given $(X_0, \ldots, X_{n/d-1}), (Y_0, \ldots, Y_{n/d-1}) \in \left(\mathbb{Z}_p[X]/\left(X^d + 1\right)\right)^{n/d}$, the arithmetic operations of $GF(pd)^{n/d}$ can be defined as (cf. see the following Elementwise Mul-Add Protocol for details):

$$(X_0, \ldots, X_{n/d-1}) + (Y_0, \ldots, Y_{n/d-1}) = (X_0 + Y_0, \ldots, X_{n/d-1} + Y_{n/d-1}) \tag{20a}$$

$$(X_0, \ldots, X_{n/d-1}) \cdot (Y_0, \ldots, Y_{n/d-1}) = (X_0 \cdot Y_0, \ldots, X_{n/d-1} \cdot Y_{n/d-1}) \tag{20b}$$

That is, we can implement the vector-like operation in $\left(\mathbb{Z}_p[X]/\left(X^d + 1\right)\right)^{n/d}$.

Further, the function evaluated in the encryption domain should be as simple as possible. So, the only function computed in the encryption domain would be

$$f(\mathbf{a}, \mathbf{b}, \mathbf{b}') = \mathbf{a} \cdot \mathbf{b} + \mathbf{b}' \tag{21}$$

Any other operation should be evaluated in the plaintext domain by the one who holds the secret key.

Table 5. The Basic Secure Comparison Protocol (or **Protocol 4.1**).

Protocol 4.1 Basic Secure Comparison Protocol			
Party:		A	B
Input:		$a_0, \ldots, a_{l-1}, \mathbf{s}$	$b_0, \ldots, b_{l-1}, \mathbf{pk}$
Output:		$(a \geq b)$	$\perp$
		Round 1 (A's turn)	
	1:	**for** $i = 0, 1, \ldots, l-1$ **do**	
	2:	$\quad \mathbf{c}_i^a \leftarrow \text{EncSym}(\mathbf{a}^i, \mathbf{s})$	
	3:	**end for**	
	4:	A sends $\mathbf{c}_0^a, \ldots, \mathbf{c}_{l-1}^a$ to B	
		Round 2 (B's turn)	
	5:	$\mathbf{c}^{result} \leftarrow (0, 0)$	
	6:	$\mathbf{u} \leftarrow (0, 0)$	
	7:	$s \leftarrow \{0, 1, \ldots, l-1\}$	
	8:	**for** $i = l-1, \ldots, 0$ **do**	
	9:	$\quad \mathbf{t} \leftarrow \text{AddPlain}(\mathbf{c}_i^a, 0 - b_i)$	
	10:	$\quad \mathbf{c} \leftarrow \text{Add}(\mathbf{t}, \mathbf{u})$	
	11:	$\quad \mathbf{c} \leftarrow \text{AddPlain}(\mathbf{c}, 1)$	
	12:	$\quad r \leftarrow \mathbb{Z}_p^*$	
	13:	$\quad \mathbf{c} \leftarrow \text{MulPlain}(\mathbf{c}, r)$	$\triangleright\ r \cdot (a_i - b_i + 1 + \Sigma_{j>i}(a_j - b_j) \cdot 2^{j+2})$
	14:		$\triangleright\ \Sigma_{j>i}(a_j - b_j) \cdot 2^{j+2}$
		$\mathbf{u} \leftarrow \text{Add}(\mathbf{u}, \text{MulPlain}(\mathbf{t}, 2^{i+2}))$	
	15:	$\quad tmp \leftarrow i + s \bmod l$	
	16:	$\quad \mathbf{c}^{result} \leftarrow \text{Add}(\mathbf{c}^{result}, MulPlain(\mathbf{c}, X^{tmp}))$	
	17:	**end for**	
		Round 3 Secure Decrypt c^{result} to A, where the associated message would be $\mathbf{m}^{result} = \Sigma_{i=0}^{l-1} d_i \cdot X^{i+s \bmod l}$	
		Round 4 (A's Turn)	
	18:	$(m_0^{result}, m_1^{result}, \ldots, m_{l-1}^{result}, 0, \ldots, 0) \leftarrow \mathbf{m}^{result}$	
	19:	$result \leftarrow 1$	
	20:	**for** $i = 0, \ldots, l-1$ **do**	
	21:	$\quad$ **if** $m_i^{result} = 0$ **then**	
	22:	$\quad\quad result \leftarrow 0$	
	23:	Establish $result$	

4.2.1. The Single Pair Integer Comparison Protocol

Parameters. Choose a prime p such that $p = 2k \cdot (n/d) + 1$ and $n/d \geq l$ for some integers k and d. Notice the plaintext space $\mathcal{R}_p$ is isomorphic to $(\mathbb{Z}_p)^n$, and $\text{Encode}_{n/d}$ and $\text{Decode}_{n/d}$ are the isomorphisms between them. The following protocol can compare a pair of l-bit integers securely, and we named it Protocol 4.2 in this write-up for ease of discussion.

Protocol 4.2. (The Single Pair Integer Comparison Protocol)

Setup. Alice generates the secret key **s** and the public key **pk** with the parameters above. Bob generates the samples u ← {0, 1,..., n/d − 1}, $\mathbf{r}_1$, $\mathbf{r}_2 \leftarrow \mathcal{R}_p$ and $\mathbf{r}' \leftarrow \left(\mathbb{Z}_p^*\right)^n$. We can describe the protocol as three simple 2PC sub-protocols.

2PC-1st. Alice computes $\mathbf{x}_{a1} = \text{Encode}_{n/d}((a_i)_{0 \leq i < n/d})$. Bob computes $\mathbf{x}b_1 = \mathbf{Encode}_{n/d}((b_i)_{0 \leq i < n/d})$ and $\mathbf{x}_{r1} = \text{Encode}_{n/d}((\mathbf{r}_1)_i)_{0 \leq i < n/d})$.

They securely compute $\mathbf{x}_{mask} = \mathbf{x}_{a1} + \mathbf{x}_{b1} - 2 \cdot \mathbf{x}_{a1} \cdot \mathbf{x}_{b1} + \mathbf{x}_{r1}$ for Alice. (Notice that $a_i = b_i = 0$ when $i \geq l$).

2PC-2nd. Alice computes $\text{Decode}_{n/d}(\mathbf{x}_{mask})$ and derives $((a_i - b_i)^2 + (\mathbf{r}_1)_i)_{0 \leq i < n/d}$. Let $\mathbf{x}_{a2} = \text{Encode}_{n/d}((\sum_{j>i}(a_j - b_j)^2 + (\mathbf{r}_1)_i)_{0 \leq i < n/d})$. Bob computes $\mathbf{x}_{b2} = \text{Encode}_{n/d}((\sum_{j>i}(\mathbf{r}_1)_i)_{0 \leq i < n/d})$, $\mathbf{x}_{r2} = \text{Encode}_{n/d}((\mathbf{r}_2)_i)_{0 \leq i < n/d})$, and $\mathbf{x}' = \text{Encode}_{n/d}((\mathbf{r}')_i)_{0 \leq i < n/d})$. They securely compute $\mathbf{d}\text{mask} = (\mathbf{x}_{a1} + \mathbf{x}_{a2} - \mathbf{x}_{b1} - \mathbf{x}_{b2} + \text{Encode}_{n/d}((1)_{0 \leq i < n/d})) \cdot \mathbf{x}' + \mathbf{x}_{r2}$ for Alice (notice that $(a_i - b_i)^2 = a_i \oplus b_i$).

2PC-3rd. Similarly, Alice decodes $\mathbf{d}_{mask}$ and derives $(a_i - b_i + 1 + \sum_{j>i} a_i \oplus b_i) \cdot (\mathbf{r}')_i + (\mathbf{r}_2)_i$ for each $0 \leq i < n/d$. Let $\mathbf{x}_{a3} = \sum_{0 \leq i < n/d}\left(\left(a_i - b_i + 1 + \sum_{j>i} a_i \oplus b_i\right) \cdot (\mathbf{r}')i + (\mathbf{r}_2)_i\right) \cdot X^{i \cdot d}$. They securely compute $\mathbf{d}' = (\mathbf{x}_{a3} - \mathbf{r}_2) \cdot X^{u \cdot d}$ for Alice.

Finally, Alice outputs 1 if the number of i such that $(\mathbf{d}')_i$ is exactly n/d; Alice outputs 0 in other cases.

Correctness of Protocol 4.2. Notice that $\mathbf{Decode}_{n/d}\ (\mathbf{d}_{mask} - \mathbf{r}_2) = ((a_i - b_i + 1 + \sum_{j>i} a_i \oplus b_i) \cdot (\mathbf{r}')i)_{0\le i<n/d}$ which shares the same value with the d_i in Table 5, so the correctness holds for the same reason discussed in the last section.

Security of Protocol 4.2. We first notice that there are three 2PCs in our protocol; Alice derives $\mathbf{x}_{mask}$, $\mathbf{d}_{mask}$, and $\mathbf{d}'$ from them, respectively. Since $\mathbf{x}_{mask}$ and $\mathbf{d}_{mask}$ are masked by $\mathbf{r}_1$ and $\mathbf{r}_2$, they behave like samples taken from a uniform distribution in the view of Alice. $\mathbf{d}'$ can be simulated by examining whether $(a \ge b) \in \{0, 1\} \triangleq$ {False, Truth} and $\mathbf{s}$ by the same arguments presented in Section 4.1, which proves the security of Bob. On the other hand, Bob's view consists of ciphertexts, so Alice would be secure as long as the BFV scheme is secure. We depict the details of Protocol 4.2 in Table 6.

Table 6. Secure Comparison Protocol for a Pair of Integers (also known as **Protocol 4.2**).

Protocol 4.2: Secure Comparison Protocol			
Party:	A	B	
Input:	$a_0, \ldots, a_{n/d-1}, \mathbf{s}$	$b_0, \ldots, b_{n/d-1}, \mathbf{pk}$	
Output:	$(a \ge b)$	$\perp$	
	Round 1 (A's turn)		
1:	$\mathbf{a} \leftarrow \mathrm{Encode}_{n/d,\alpha^d}((a_0, a_1, \ldots, a_{n/d-1}))$		
2:	$\mathbf{c}_a \leftarrow \mathrm{EncSym}(\mathbf{a}, \mathbf{s})$		
3:	Sends $\mathbf{c}_a$ to B.		
	Round 2 (B's turn, computes xor)		
4:	$\mathbf{b} \leftarrow \mathrm{Encode}_{n/d,\alpha^d}((b_0, b_1, \ldots, b_{n/d-1}))$		
5:	$\mathbf{c}^{round2} \leftarrow \mathrm{MulPlain}(\mathbf{c}_a, -2 \cdot \mathbf{b})$		$\triangleright(-2 \cdot a_i \cdot b_i)$
6:	$\mathbf{c}^{round2} \leftarrow \mathrm{Add}\left(\mathbf{c}^{round2}, \mathbf{c}_a\right)$		$\triangleright(a_i - 2 \cdot a_i \cdot b_i)$
7:	$\mathbf{c}^{round2} \leftarrow \mathrm{AddPlain}\left(\mathbf{c}^{round2}, \mathbf{b}\right)$		$\triangleright(a_i - 2 \cdot a_i \cdot b_i + b_i)$
8:	**for** $i = 0, 1, \ldots, l-1$ **do**		
9:	$\quad mask_i \leftarrow \{0, 1, \ldots, p-1\}$		
10:	**end for**		
11:	$\mathbf{mask} \leftarrow \mathrm{Encode}_{n/d,\alpha^d}((mask_0, \ldots, mask_{n/d-1}))$		
12:	$\mathbf{c}^{round2} \leftarrow \mathrm{AddPlain}\left(\mathbf{c}^{round2}, \mathbf{mask}\right)$		
13:	Secure Decrypt $\mathbf{c}^{round2}$ for A		
14:	Round 3 (A's turn, computes $\sum_{j>i}\left(a_j - b_j\right)^2 \cdot j^{t+2}$		
15:	Derived $a_i - 2 \cdot a_i \cdot b_i + b_i + mask_i$ from secure decryption		
16:	$u \leftarrow 0$		
17:	**for** $i = n/d - 1, \ldots, 0$ **do**		
18:	$\quad x_i \leftarrow u + a_i + 1$		
19:	$\quad u \leftarrow u + (a_i - 2 \cdot a_i \cdot b_i + b_i + mask_i) \cdot 2^{i+2}$		
20:	**end for**		
21:	$\mathbf{x} \leftarrow \mathrm{Encode}_{n/d,\alpha^d}((x_0, \ldots, x_{n/d-1}))$		
22:	$\mathbf{c}_x \leftarrow \mathrm{EncSym}(\mathbf{x}, \mathbf{s})$		
23:	Sends $\mathbf{c}_x$ to B.		
	Round 4 (B's turn, compute $r_i \cdot \left(a_i - b_i + 1 + \sum_{j>i}\left(a_j - b_j\right)^2 \cdot 2^{j+2}\right)$		
24:	$u \leftarrow 0$		
25:	**for** $i = n/d - 1, \ldots, 0$ **do**		
26:	$\quad t_i \leftarrow u$		
27:	$\quad u \leftarrow u + mask_i \cdot 2^{i+2}$		
28:	**end for**		

Table 6. *Cont.*

29:	$\mathbf{t} \leftarrow \text{Encode}_{n/d,\alpha^d}((t_0, t_1, \ldots, t_{n/d-1}))$
30:	$\mathbf{c}_x \leftarrow \text{AddPlain}(\mathbf{c}_x, -\mathbf{t})$ ▷ remove the mask
31:	$\mathbf{c}^{round4} \leftarrow \text{AddPlain}(\mathbf{c}_x, -\mathbf{b})$
32:	**for** $i = n/d - 1, \ldots, 0$ **do**
33:	$r_i \leftarrow \mathbb{Z}_p^*$
34:	$mask_i \leftarrow \{0, 1, \ldots, p-1\}$
35:	**end for**
36:	$\mathbf{r} \leftarrow \text{Encode}_{n/d,\alpha^d}((r_0, \ldots, r_{n/d-1}))$
37:	$\mathbf{mask} \leftarrow \text{Encode}_{n/d,\alpha^d}((mask_0, \ldots, mask_{n/d-1}))$
38:	$\mathbf{c}^{round4} \leftarrow \text{MulPlain}\left(\mathbf{c}^{round4}, \mathbf{r}\right)$
39:	▷ $d_i = r_i \cdot \left(a_i - b_i + 1 + \sum_{j>i}\left(a_j - b_j\right)^2 \cdot 2^{j+2}\right)$
40:	$\mathbf{c}^{round4} \leftarrow \text{AddPlain}\left(\mathbf{c}^{round4}, \mathbf{mask}\right)$
	Secure Decrypt c^{round4} for A
41:	Round 5. (A's turn, convert message to polynomial mod for rotation)
42:	Derived d_i from secure decryption
43:	$\mathbf{c}_d \leftarrow \text{EncSym}((d_0, 0, .., 0, d_1, 0, \ldots, 0, d_{n/d-1}, 0, ..0), \mathbf{s})$
44:	▷ non-batch format, 1 message followed by d-1 zeros
	Sends $\mathbf{c}_d$ to B.
45:	Round 6 (B's turn, randomly rotate)
46:	$\mathbf{mask} \leftarrow (mask_0, 0, \ldots, 0, mask_i, 0, \ldots, 0, mask_{n/d-1}, 0, \ldots, 0)$
47:	▷ put $mask_i$ as the $i \cdot d$-th coefficient
48:	$\mathbf{c}_d \leftarrow \text{AddPlain}(\mathbf{c}_d, -\mathbf{mask})$ ▷ remove the mask
49:	$r \leftarrow \{0, \ldots, n/d - 1\}$
	$r \leftarrow r \cdot d$
50:	$\mathbf{c}^{round6} \leftarrow \text{MulPlain}(\mathbf{c}_d, X^r)$
51:	Secure Decrypt $\mathbf{c}^{round6}$ for A
52:	Round Final. (A's turn, establish result), suppose $\mathbf{m}^{round6} = \text{Dec}\left(\mathbf{c}^{round6}, \mathbf{s}\right)$
53:	$\left(m_0^{round6}, \ldots, m_n^{round6}\right) \leftarrow \mathbf{m}^{round6}$
54:	$\delta \leftarrow 1$
55:	**for** $i = n/d - 1, \ldots, 0$ **do**
56:	**if** $m_{i \cdot d}^{round6} = 0$ **then**
57:	$\delta \leftarrow 0$
58:	**end if**
59:	**end for**
60:	Establish δ

4.2.2. The Protocol for Comparing Pairs of Integers Simultaneously

The plaintext space has the best utilization when $n = l$, while $n \geq 1024$ in most practical use cases; however, we rarely need to compare such a large integer. We now modify Protocol 4.2 to compare k pairs of n/k-bit integers simultaneously. For making a differentiation, we called this modified version **Protocol 4.3** in this write-up.

Assume Alice's inputs are $a_{i,j} \in \{0,1\}$ and Bob's inputs are $b_{i,j} \in \{0,1\}$ where $0 \leq i < k, 0 \leq j < n/k$—our goal is to find the order relationships between $a_i = \sum_j a_{i,j} \cdot 2^j$ and $b_i = \sum_j b_{i,j} \cdot 2^j$, denoted by $\delta_i = (a_i \geq b_i) \in \{0,1\}$. We can complete this task by following almost the same steps in Table 6.

Protocol 4.3. (The Protocol for Comparing Pairs of Integers Simultaneously)

Setup. The same as Protocol 4.2.

2PC-1st. Alice computes $\mathbf{x}_{a1}$ = **Encode**$_n$ $((a_{ij})_{0\leq i<k,\ 0\leq j<n/k})$. Bob computes $\mathbf{x}_{b1}$ = **Encode**$_n((b_{ij})_{0\leq i<k,0\leq j<n/k})$ and $\mathbf{x}_{r1}$ = **Encode**$_n(\mathbf{r}_1)$. They securely compute $\mathbf{x}_{mask} = \mathbf{x}_{a1} + \mathbf{x}_{b1} - 2 \cdot \mathbf{x}_{a1} \cdot \mathbf{x}_{b1} + \mathbf{x}_{r1}$ for Alice.

2PC-2nd. Alice computes **Decode**$n(\mathbf{x}_{mask})$ to derive $((a_{ij} - b_{ij})^2 + (\mathbf{r}_1)_{ij})_{0\leq i<k,0\leq j<n/k}$. Let $\mathbf{x}_{a2}$ = **Encode**$_n$ $((\sum_{t>j}(a_{it} - b_{it})^2 + (\mathbf{r}_1)_{it})_{0\leq i<k,0\leq j<n/k})$. Bob computes $\mathbf{x}_{b2}$ = **Encode**$_n$ $((\sum_{t>j}(\mathbf{r}_1)_{it})_{0\leq i<k,0\leq j<n/k})$, $\mathbf{x}_{r2}$ = **Encode**$_n(\mathbf{r}_2)$, and

$\mathbf{x}' = \mathbf{Encode}_n(\mathbf{r}')$. They securely compute
$\mathbf{d}\text{mask} = (\mathbf{x}_{a1} + \mathbf{x}_{a2} - \mathbf{x}_{b1} - \mathbf{x}_{b2} + \mathbf{Encode}_n((1)_{0\le i<k, 0\le j<n/k})) \cdot \mathbf{x}' + \mathbf{x}_{r2}$ for Alice.
(Notice that $(a_i - b_i)^2 = ai \oplus b_i$)

2PC-3rd. Similarly, Alice decodes dmask and derives $(a_{ij} - b_{ij} + 1 + \sum_{t>j}(a_{it} \oplus b_{it})) \cdot (\mathbf{r}')_{ij} + (\mathbf{r}_2)_{ij}$, for each $0 \le i < n/d$. Let $\mathbf{x}_{a3} = \mathbf{Encode}_k((\sum_{0 \le j < n/k}(\mathbf{d}_{mask})_{ij} \cdot X^j)_{0\le i<k})$.

Bob computes $\mathbf{x}_{b3} = \mathbf{Encode}_k((\sum_{0\le j<n/k}(\mathbf{r}_2)_j \cdot X^j)_{0\le i<k})$, $\mathbf{x}u = \mathbf{Encode}_k((X^{ui})_{0\le i<k})$ They securely compute $\mathbf{d}' = (\mathbf{x}_{a3} - \mathbf{x}_{b3}) \cdot \mathbf{x}_u$ for Alice.

Finally, Alice derives d'_{ij} from the **2PC-3rd**. Let $\delta_i = 0$ if there is some j, such that $d'_{ij} = 0$, and $\delta_i = 1$ in other cases. Output $(\delta_i)_{0\le i<k}$.

Notice that the major difference between Protocols 4.2 and 4.3 is in the **2PC-3rd** step. We need to rotate $(\mathbf{d}_{ij})_{0\le j<n/k}$ by u_i for each $0 \le i < k$ simultaneously, so we compute the encoded version of $(\sum_{0\le j<n/k}(\mathbf{d})_{ij} \cdot X^j)_{0\le i<k}$, and rotate it by multiplying the encoded version of $(X_{ui})_{0\le i<k}$. We summarize the detailed processing steps of **Protocol 4.3** in Table 7.

Table 7. The Information Flow and Processing Steps of **Protocol 4.3**.

Protocol 4.3: Secure Comparison Protocol for k-pairs			
Party:	A	B	
Input:	$\left(a_{ij}\right)_{0\le i<k, 0\le j<n/k}$, $\mathbf{s}$	$\left(b_{ij}\right)_{0\le i<k, 0\le j<n/k}$, $\mathbf{pk}$	
Output:	$(a_i \ge b_i)$	$\perp$	
	Round 1 (A's turn)		
1:	$\mathbf{a} \leftarrow \text{Encode}_{n,\alpha}\left(\left(a_{0,0}, a_{0,1}, \ldots, a_{k-1,n/k-1}\right)\right)$		
2:	$\mathbf{c}_a \leftarrow \text{EncSym}(\mathbf{a}, \mathbf{s})$		
3:	Sends $\mathbf{c}_a$ to B.		
	Round 2 (B's turn, computes XoR)		
4:	$\mathbf{b} \leftarrow \text{Encode}_{n,\alpha}\left(\left(b_{0,0}, b_{0,1}, \ldots, b_{k-1,n/k-1}\right)\right)$		
5:	$\mathbf{c}^{round2} \leftarrow \text{MulPlain}(\mathbf{c}_a, -2 \cdot \mathbf{b})$		$\triangleright\left(-2 \cdot a_{ij} \cdot b_{ij}\right)$
6:	$\mathbf{c}^{round2} \leftarrow \text{Add}\left(\mathbf{c}^{round2}, \mathbf{c}_a\right)$		$\triangleright\left(a_{ij} - 2 \cdot a_{ij} \cdot b_{ij}\right)$
7:	$\mathbf{c}^{round2} \leftarrow \text{AddPlain}\left(\mathbf{c}^{round2}, \mathbf{b}\right)$		$\triangleright\left(a_{ij} - 2 \cdot a_{ij} \cdot b_{ij} + b_{ij}\right)$
8:	**for** $i = 0, 1, \ldots, k-1$ **do**		
9:	**for** $j = 0, 1, \ldots, n/k - 1$ **do**		
10:	$mask_{ij} \leftarrow \{0, 1, \ldots, p-1\}$		
11:	**end for**		
12:	**end for**		
13:	$\mathbf{mask} \leftarrow \text{Encode}_{n,\alpha}\left(\left(mask_{0,0}, \ldots, mask_{k-1,n/k-1}\right)\right)$		
14:	$\mathbf{c}^{round2} \leftarrow \text{AddPlain}\left(\mathbf{c}^{round2}, \mathbf{mask}\right)$		
15:	Secure Decrypt c^{round2} for A		
	Round 3 (A's turn, computes $\sum_{t>j}(a_{it} - b_{it})^2 \cdot 2^{t+2}$		
16:	Derived $a_{ij} - 2 \cdot a_{ij} \cdot b_{ij} + b_{ij} + mask_{ij}$ from secure decryption		
17:	**for** $i = 0, 1, \ldots, k-1$ **do**		
18:	$u \leftarrow 0$		
19:	**for** $t = n/k - 1, \ldots, 0$ **do**		
20:	$x_{it} \leftarrow u + a_{it} + 1$		
21:	$u \leftarrow u + (a_{it} - 2 \cdot a_{it} \cdot b_{it} + b_{it} + mask_{it}) \cdot 2^{t+2}$		
22:	**end for**		
23:	**end for**		
24:	$\mathbf{x} \leftarrow \text{Encode}_{n,\alpha}\left(\left(x_{0,0}, \ldots, x_{k-1,n/k-1}\right)\right)$		
25:	$\mathbf{c}_x \leftarrow \text{EncSym}(\mathbf{x}, \mathbf{s})$		
26:	Sends $\mathbf{c}_x$ to B.		

Table 7. *Cont*.

27:	Round 4 (B's turn, compute $r_{ij} \cdot \left(a_{ij} - b_{ij} + 1 + \sum_{t>j}(a_{it} - b_{it})^2 \cdot 2^{t+2}\right)$
28:	$u \leftarrow 0$
29:	**for** $i = 0, 1, \ldots, k-1$ **do**
30:	$u \leftarrow 0$
31:	**for** $j = 0, 1, \ldots, n/k-1$ **do**
32:	$t_{ij} \leftarrow u$
33:	$u \leftarrow u + mask_{ij} \cdot 2^{j+2}$
34:	**end for**
35:	**end for**
36:	$\mathbf{t} \leftarrow \text{Encode}_{n/d,\alpha^d}\left(\left(t_{0,0}, \ldots, t_{k-1,n/k-1}\right)\right)$
37:	$\mathbf{c}_x \leftarrow \text{AddPlain}(\mathbf{c}_x, -\mathbf{t})$ ▷ remove the mask
38:	$\mathbf{c}^{round4} \leftarrow \text{AddPlain}(\mathbf{c}_x, -\mathbf{b})$
39:	**for** $i = 0, 1, \ldots, k-1$ **do**
40:	**for** $j = 0, 1, \ldots, n/k-1$ **do**
41:	$mask_{ij} \leftarrow \{0, 1, \ldots, p-1\}$
42:	$r_{ij} \leftarrow \mathbb{Z}_p^*$
43:	**end for**
44:	**end for**
45:	$\mathbf{r} \leftarrow \text{Encode}_{n,\alpha}\left(\left(r_{0,0}, \ldots, r_{k-1,n/k-1}\right)\right)$
46:	$\mathbf{mask} \leftarrow \text{Encode}_{n,\alpha}\left(\left(mask_{0,0}, \ldots, mask_{k-1,n/k-1}\right)\right)$
47:	$\mathbf{c}^{round4} \leftarrow \text{MulPlain}\left(\mathbf{c}^{round4}, \mathbf{r}\right)$
48:	▷ $d_{ij} = r_{ij} \cdot \left(a_{ij} - b_{ij} + 1 + \sum_{t>j}(a_{it} - b_{it})^2 \cdot 2^{t+2}\right)$
49:	$\mathbf{c}^{round4} \leftarrow \text{AddPlain}\left(\mathbf{c}^{round4}, \mathbf{mask}\right)$
50:	Secure Decrypt c^{round4} for A
51:	Round 5. (A's turn, convert message to polynomial mod for rotation)
52:	Derived d_i from secure decryption
53:	$\mathbf{d} \leftarrow \text{Encode}_{k,\alpha^{n/k}}\left(\left(d_{0,0}, \ldots, d_{k-1,n/k-1}\right)\right)$
54:	▷ k-batch of degree $n/k-1$ polynomials
55:	$\mathbf{c}_d \leftarrow \text{EncSym}(\mathbf{d}, \mathbf{s})$
	Sends $\mathbf{c}_d$ to B.
	Round 6 (B's turn, randomly rotate)
57:	$\mathbf{mask} \leftarrow \text{Encode}_{k,\alpha^{n/k}}\left(\left(mask_{0,0}, \ldots, mask_{k-1,n/k-1}\right)\right)$
58:	$\mathbf{c}_d \leftarrow \text{AddPlain}(\mathbf{c}_d, -\mathbf{mask})$ ▷ remove the mask
59:	**for** $i = 0, 1, \ldots, k-1$ **do**
60:	$r_i \leftarrow \{0, 1, \ldots, n/k-1\}$
61:	$\mathbf{e}_i \leftarrow (0, \ldots, 0, 1, 0, \ldots, 0)$
62:	▷ length is n/k, only r_i-th entry is 1
63:	**end for**
64:	$\mathbf{e} \leftarrow \text{Encode}_{k,\alpha^{n/k}}((\mathbf{e}_0, \mathbf{e}_1 \ldots, \mathbf{e}_{k-1}))$
65:	$\mathbf{c}^{round6} \leftarrow \text{MulPlain}(\mathbf{c}_d, \mathbf{e})$
66:	Secure Decrypt $\mathbf{c}^{round6}$ for A
	Round Final. (A's turn, establish result), suppose $\mathbf{m}^{round6} = \text{Dec}\left(\mathbf{c}^{round6}, \mathbf{s}\right)$
67:	$\left(m_{0,0}^{round6}, \ldots, m_{k-1,n/k-1}^{round6}\right) \leftarrow \mathbf{m}^{round6}$
68:	**for** $i = 0, 1, \ldots, k-1$ **do**
69:	$\delta_i \leftarrow 1$
70:	**for** $j = 0, 1, \ldots, n/k-1$ **do**
71:	**if** $m_{ij}^{round6} = 0$ **then**
72:	$\delta_i \leftarrow 0$
73:	**end if**
74:	**end for**
75:	**end for**
76:	Establish δ_i

5. Experiment Results

5.1. Encryption Schemes

Since the performance of an HE-based secure comparison protocol is strongly related to which homomorphic encryption scheme it used, we examine the performances of Paillier, standard-RSA, subgroup-RSA in [5], and subgroup-RSA in [7] in Table 8. Notice that Paillier is used in the earlier version of DGK protocol [2], subgroup-RSA is used in the DGK [4] and the CEK protocols. BFV scheme is used to build our comparison protocols. Notice that the decryption of subgroup-RSA needs to implement a discrete logarithm on the plaintext space, so the plaintext space needs to be very small (for a ciphertext c in subgroup-RSA, we may decrypt c by computing m = (logg cS) · S^{-1} for some secret key S and public key g, namely, cS = gmS. Since DGK protocol needs only to determine whether the message is zero, Alice does not need to perform discrete logarithms to derive the direct value of m).

Table 8. Execution Time of Each One of the Benchmarked Encryption Schemes. (Runs ib i5-9400f@4.0gHz with security level 128, unit ms).

Schemes	Plaintext Space	Enc	Dec	KeyGen	Add	MulPlain
Paillier	$\mathbb{Z}_p$	22.98	7.756	1871	0.021	-
RSA	$\mathbb{Z}_p$	0.079	2.216	1792	-	0.008
RSA [5]	$\mathbb{Z}_{37}$	0.692	0.209	1038	0.002	-
RSA [7]	$\mathbb{Z}_{2^{256}}$	0.685	55.49	1325	0.002	-
BFV	$\mathbb{Z}_{193}/(X^{1024}+1)$	0.459	0.055	0.360	0.002	0.065
BFV-opt	$\mathbb{Z}_{193}/(X^{1024}+1)$	0.251	0.060	0.198	0.002	0.058

In [7], the authors claimed that if the plaintext space is bd for some small prime b, then they can implement discrete logarithms effectively. However, the speed of decryption is still much slower than that of the encryption without the aid of a sizeable pre-computed table. In the experiment of CEK protocol, we exclude the decryption time since the discrete logarithm is not necessary in their protocol.

We use the GMP library to implement the operations of big numbers. Although the codes run on Golang (Reference code: https://github.com/howard-kuo/cek_protocol/blob/master/rsaCek.go accessed on 18 February 2023), the performance would not be very different from the codes written on C directly. Our claim comes from the fact that the major computation cost lies in operations for evaluating big numbers. Furthermore, those operations are implemented by GMP, which is an efficient c-base library. Moreover, we use the Microsoft SEAL library in C++ to implement the BFV encryption scheme. Since the main computational bottleneck of our protocols is it to draw the samples from a discrete Gaussian distribution χ, we do some optimization on the sampler, as follows:

Since Prob $\{|\chi| > 19\} = 0$ in the implementation of SEAL, we can store the commutative density function at each point (i.e., CDF(x) = Prob $\{\chi > x\}$ for $-19 \leq x \leq 19$). We may draw a sample from χ by the following function:

DiscreteGaussianSampler():
    $u \leftarrow \{0, 1, ..., 2^{32} - 1\} \cdot 2^{-32}$
    $x = \arg\max\{y \in \{-19, ..., 19\} \mid CDF(y) > u\}$
Return x

We denote the optimized BFV implementation as "BFV-opt" in Table 8.

5.2. Performance Analyses

5.2.1. Computational Cost

The execution time of the DGK protocol, CEK protocol, and our protocols are shown in Table 9. Recall that DGK protocol uses RSA proposed in [5], CEK protocol uses RSA proposed in [7], and our protocols use an optimized BFV scheme. We use 3072-bit ciphertexts in DGK and CEK protocols (we use the same parameters and algorithms described in [6], and the parameters of our protocol depend on the size of compared integers and

the secure decryption protocol we used. As discussed in Chapter 3, Alice may learn other information from the ciphertext generated by Bob. We suggest the reader uses two times the bit length for the security level they claimed (i.e., log2 n = 6144)). As discussed in Section 3.2, in the Secure Decryption portion, **Protocol 3.1** needs to use a larger q to achieve indistinguishability; nevertheless, **Protocol 3.2** needs to assume further that Bob cannot break the encryption system with some hint of the errors. As a result, **Protocol 4.1** uses n = 1024, log2q = 27 and n = 2048, log2q = 55, respectively. **Protocol 4.3** uses n = 1024, log2 q = 27 for 32-bit comparison; n = 2048, log2 q = 55 for 2048-bit comparison; n = 4096, log2 q = 109 for 4096-bit comparison, respectively.

Table 9. The Execution Time of Each One of the Benchmarked Secure Comparison Protocols. (Runs ib i5-9400f@4.0gHz with security level 128, unit ms, notice $\beta = O(\log \log n)$, $p(\lambda)$ is the time complexity of subgroup-RSA).

Schemes	Secure Decryption	Round	Complexity	32-Bit	2048-Bit	4096-Bit
Protocol 4.1	Protocol 3.2	4	$\widetilde{O}(\lambda \cdot l)$	7.496	479.7	966.6
Protocol 4.1	Protocol 3.1	2	$\widetilde{O}(\lambda \cdot l)$	15.45	1000	1996.6
Protocol 4.3	Protocol 3.2	8	$\widetilde{O}(\lambda + l)$	1.686	4.246	-
Protocol 4.3	Protocol 3.1	6	$\widetilde{O}(\lambda + l)$	-	-	12.06
DGK [10]	Add Encryption Zero	2	$\widetilde{O}(p(\lambda) \cdot l)$	55.71	3488	6766
CEK [7]	Add Encryption Zero	3~5	$\widetilde{O}(p(\lambda) \cdot l \cdot \beta^{-1})$	28.16	1696	3471

It might sound weird when the ciphertext size, which is $n \cdot \log_2 q$, grows quadratic to n in the specific case in Table 9, but the execution time is not (cf. Figure 3). This is because the main computational cost is spent on sampling pseudo-random numbers, and the number of random numbers is linear to n.

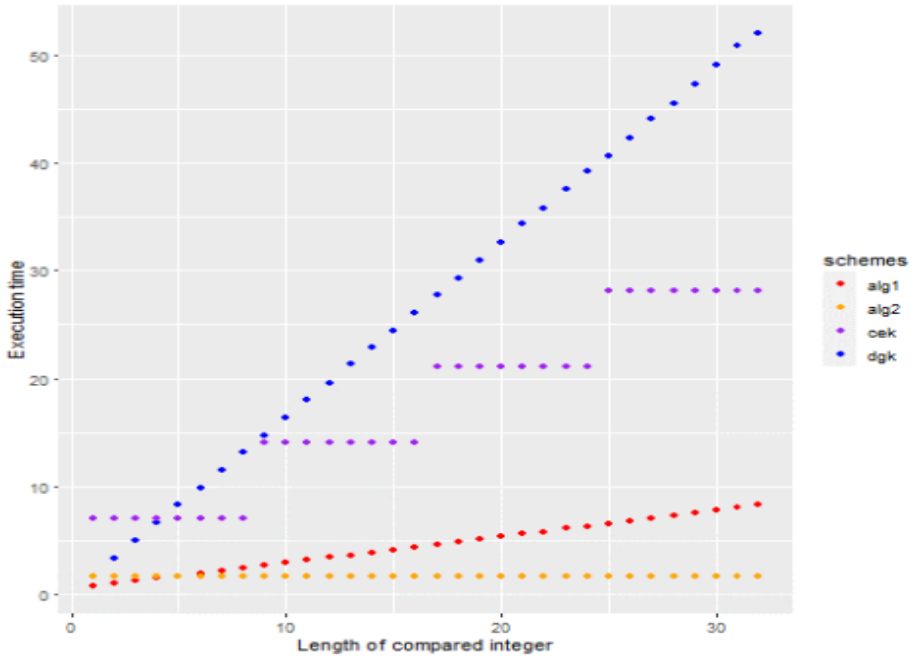

Figure 3. The Tendencies in Execution Time of the Two Proposed Protocols (**Alg-1** in the Figure is the Secure Comparison **Protocol 4.1** with Secure Decryption **Protocol 3.4**, and **Alg-2** is Secure Comparison Protocol 4.2 with Secure Decryption **Protocol 3.4**).

5.2.2. Communication Cost

Ciphertexts in the BFV scheme have size $2 \cdot n \cdot \log_2 q$. In **Protocol 4.1**, we need to send $l + 1$ ciphertext in total, and in **Protocol 4.2**, we need to send 6 ciphertexts. The

DGK protocol needs to send $2 \cdot l$ ciphertexts, and CEK protocols need to send about $2 \cdot l \cdot 8^{-1}$ ciphertexts ($2 \cdot l \cdot 8^{-1}$ RSA ciphertexts and 2 ECC ciphertexts).

We examine their required communication costs and report the results in Tables 10 and 11, respectively. Since the ciphertext size of the BFV scheme is much larger than that of the RSA, RSA's communication rate is more extensive than the CEK or DGK protocol when l is small. On the other hand, Protocol 4.2 needs only constant-size ciphertexts, and it would have a better communication rate than the others when l is large. For ease of comparison, we also show the tendency in required communication rates of the two proposed protocols in Figure 4.

Table 10. Communication Rate of Each One of the Secure Comparison Protocols with Security Level 128 (for short l, i.e., $0 \leq l < 32$).

Schemes	Parameters	Ciphertext Size (KB)	Communication Cost (KB)
Protocol 4.1	n = 1024, $\log_2$ q = 27	6.75	6.75·(l+1)
Protocol 4.2	n = 1024, $\log_2$ q = 27	6.75	40.5
DGK [10]	$\log_2$ N = 3072	0.375	$0.65 \cdot l$
CEK [7]	$\log_2$ N = 3072	0.375	$0.08125 \cdot 1$

Table 11. Communication Rates for Larger l (i.e., for l = 32, 2048, and 4096).

Schemes	l = 32 (KB)	l = 2048 (KB)	l = 4096 (KB)
Protocol 4.1	222.75	13,830.75	27654.75
Protocol 4.2	40.5	165	652.4
DGK [10]	24	1536	3072
CEK [7]	3	196	384

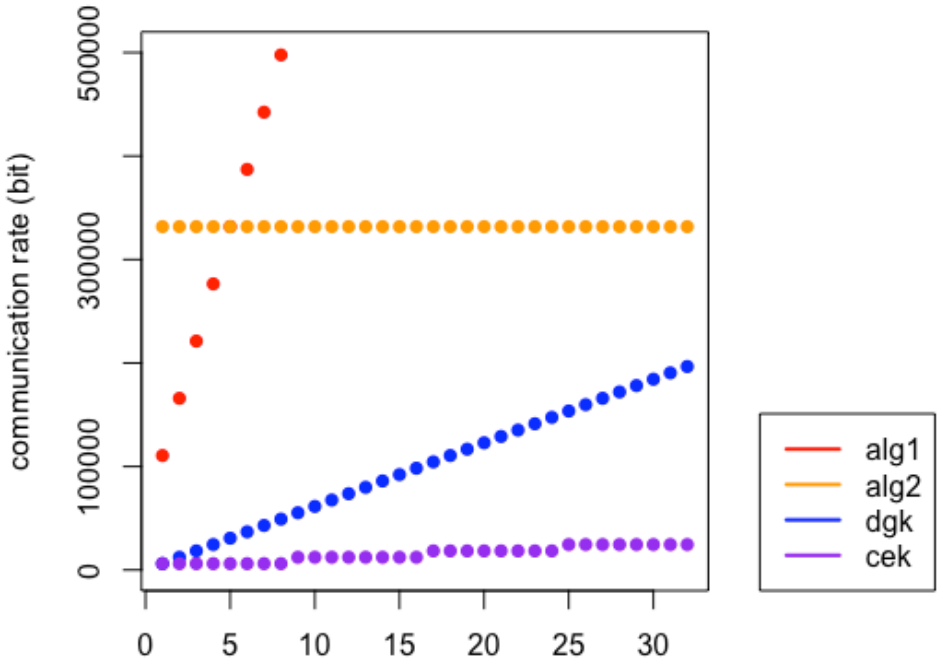

Figure 4. The Tendencies in Communication Rates of the Two Proposed Protocols (Alg-1 in the Figure is the Secure Comparison Protocol 4.1 with Secure Decryption Protocol 3.4, and Alg-2 is the Secure Comparison Protocol 4.2 with Secure Decryption Protocol 3.4).

5.3. Remarks on Recently Published Related Works [22–25]

Badawi et al. [22] applied their vector operation-based multi-GPU Levelled-FHE to implement the inference circuit of two CNNs to perform homomorphically image classification on encrypted images from the MNIST and CIFAR 10 datasets. Their implementation gained 1 to 3 orders of magnitude speed-up compared with the CPU implementation on ordinary vector/polynomial operations. Further, Ref. [22] presented data parallelism approaches and applied workload partitioning methods to implement a variant of the BFV scheme on the NVIDIA K80 and P100 GPU clusters. Likewise, Ref. [23] worked toward speeding up the NTT, INTT, and NTT-based polynomial multiplication operations on various GPU platforms and reported that the encryption and the decryption operations of the BFV scheme on Microsoft's SEAL library could be more than 100 times faster than on its Intel i9-7900X CPU counterpart. More positively, a single NTT operation for polynomials of degree 32768 with 61-bit coefficients can reach the ten-microsecond range on Nvidia GTX and Tesla series GPUs. These encouraging results give us confidence that with the aid of GPU's computing power, the applicability of our protocol will be primarily enhanced (notice that our computing platform is Intel i5 CPU families only).

In our experiment, the execution time of a single BFV encryption of degree 32,768 is longer than that of our secure comparison protocol (with degree 1024, security level 128). Reducing the complexity of the target function and the ciphertext size is the core idea of our protocol. In more detail, if n = 8192, q has 109 bits, experimentally, a single encryption time cost of SEAL exceeds 10ms in our experimental settings. Under this circumstance, it would be hard to construct a comparison protocol that needs less than 12 ms execution time. On the other hand, 12 ms is the whole time needed for our 4096-bit protocol. This result comes from the fact that, for the same given degree n, a matrix-based LWE encryption would be slower than its polynomial counterpart because the respective time complexities are n^2 and n log n.

Specific to the encryption domain comparison, Ref. [24] designed an integer-and-FHE-based private database query (PDQ) protocol supporting compound conditions with equality and order comparisons, which scales efficiently for the length of input integers by applying techniques from the finite field theory. According to [24], the comparison algorithm needs about 25.259 s to compare 697 pairs of 64-bit integers using the BGV-level FHE scheme with SIMD techniques at more than 138 bits of security. This yields an amortized rate of just 36 ms per comparison. In contrast, Ref. [25] showed that FHE schemes suitable for arithmetic circuits (e.g., BGV or BFV) have a similar performance as FHE schemes for non-arithmetic circuits (such as the TFHE) in basic comparison tasks such as less-than, maximum, and minimum operations. Ref. [25] reported that the execution of the less-than function in the HElib library is up to 3 times faster than the prior work [24] based on BGV/BFV schemes. In [21], comparing a pair of 64-bit integers, sorting 64 32-bit integers, and finding the minimum of 64 32-bit integers can be completed in 11 milliseconds, in 19 s, and in 9.5 s, respectively, on an average laptop without multi-threading. In contrast, our comparison protocol can compare 64 pairs of 64-bit integers in 12.06 ms, which yields an amortized cost of 0.188ms. An intuitive reason for better performance is that the degree of our ciphertext is much smaller (n = 4096). On the other hand, we can only implement an elementary target function in the encryption domain since the ciphertext is tiny, forcing our protocol into a higher round of complexity (say, six rounds).

5.4. Summary

The computational cost of the proposed algorithms can be lower than that of the other protocols in most cases, while the communicational cost is much higher. With the rapid growth of internet speed, our protocol can be more practical than the others. Suppose the users attempt to compare many pairs of integers securely. In that case, the method introduced in Section 3.1 provides a solution to compare them securely with almost the same execution time examined in Section 5.2, which has lower communication costs and a faster execution speed than DGK and CEK protocols. Concrete execution times and

speed-up ratios of various FHE operators have been reported above; however, those values depend on specific parameters, target functions, hardware configurations, and software implementation techniques adopted at the experiments.

Moreover, working with FHE also introduces significant engineering challenges in practice. Different schemes offer varying performance tradeoffs, and optimal choices are heavily application-dependent. In other words, judging the competing approaches' superiority is only possible and fair with a standard testing environment and a common target task.

6. Conclusions

Homomorphic encryption enables end-to-end data security by performing computations directly on ciphertexts without needing in-advance decryption. However, due to the unavoidable ciphertext expansion and complicated error-controlling mechanisms, richness in computational and storage resources play dominating roles in the success of applying a homomorphic cryptosystem in real-world applications.

Comparison is a standard function required in many applications concerning orders of involved items; consequently, its homomorphic evaluation has been the object of many FHE-related works. Since inputs are encrypted, an HE-based comparison algorithm cannot terminate, like its plaintext domain counterpart, whenever it finds the first difference between the most significant bits. Comparison is a standard function required in many applications concerning orders of involved items; consequently, its homomorphic evaluation has been the object of many FHE-related works. Since inputs are encrypted, an HE-based comparison algorithm cannot terminate, like its plaintext domain counterpart, whenever it finds the first difference between the most significant bits. As a result, HE-based comparisons correspond to the worst-case complexity in the plain domain.

This work presents a fast, semi-honest, secure comparison protocol based on the BFV encryption scheme. With its vector-like plaintext space and the aid of batching technique, the number of required encryptions can be significantly reduced; each comparison needs only six encryptions in our protocol. In other words, the proposed protocol can achieve the time complexity for a given security parameter λ. As a result, 4096-bit integers can be securely compared within 12.08 ms, which is much faster than the state-of-the-art homomorphic encryption-based secure comparison protocol. Furthermore, we can compare k pairs of bit integers with almost the same execution time as comparing bit integers and achieve higher throughput regardless of the compared integer size.

As for our future work, shortly, we will conduct and examine the following research topics to better the overall BFV scheme's performance:

(i). For every 2PC protocol with some task f, we may decompose f into simple subtasks, as we have addressed in Section IV, implying that those protocols can be sped up further via BFV encryption.

(ii). The computational bottleneck of our protocols comes mainly from the execution of random number sampling because the cryptographic hash function is much slower than generating RLWE samples in the implementation of SEAL. We can generate pseudo-random numbers by generating RLWE samples, accelerating the proposed protocol furthermore.

Notably, in machine learning, FHE has been used for tasks ranging from linear and logistic regression to Encrypted Neural Network inference. For example, FHE may be critical in running privacy-preserving ML-as-a-Service (MLasS) applications. Consequently, there has been increasing interest in FHE-based secure computation solutions. This tendency explains why Gartner [29] projected that "by 2025, at least 20% of companies will have a budget for projects that include FHE." Therefore, how to embed our fast, secure comparison protocol to enhance appropriate MLasS applications will undoubtedly be one of our future research topics.

Author Contributions: Formal analysis, T.-H.K.; Funding acquisition, J.-L.W.; Investigation, T.-H.K. and J.-L.W.; Methodology, T.-H.K.; Project administration, J.-L.W.; Resources, J.-L.W.; Software, T.-H.K.; Supervision, J.-L.W.; Writing—original draft, T.-H.K.; Writing—review & editing, T.-H.K. and J.-L.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Minister of Science and Technology, Taiwan MOST 109-2218-E-002-015 and MOST 111-2221-E-002-134-MY3.

Data Availability Statement: https://github.com/howard-kuo/bfv_dgk (accessed on 18 February 2023).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Algebraic Structures and Notations of the BFV Scheme

Appendix A.1. Quotient Ring

Basic objects in the BFV scheme are elements in the polynomial ring $\mathcal{R} = \mathbb{Z}[X]/(\Phi_{\mathrm{m}}(X))$, where $\Phi_m(X)$ is the m -th cyclotomic polynomial, and denotes $\mathcal{R}_t = \mathcal{R}/t\mathcal{R}$ for $t \in \mathbb{N}$ and "$/$" as the ring quotient operator. We always treat an element in $\mathcal{R}$ or $\mathcal{R}_t$ as an integer polynomial with a degree less than n. For example, given $\mathbf{a} \in \mathcal{R}$ or $\mathcal{R}_t$ we can assume

$$\mathbf{a} = a_0 + a_1 \cdot X + \ldots + a_{n-1} \cdot X^{n-1} + (\Phi_m(X)) \tag{A1}$$

and we will say a_i the "i-th coefficient" of $\mathbf{a}$.

The definitions in Table A1 can be extended to elements in $\mathcal{R}\mathrm{m}$ or $\mathcal{R}\mathrm{m}$, t by operating on each coefficient of them. For example, if $\mathbf{a} \in \mathcal{R}\mathrm{m}$, then

Table A1. Notations and Definitions of Mappings between Different Algebraic Structures.

π_t	$: \mathbb{Z} \to \mathbb{Z}_t$	defined by sending an integer to its residual class, say $z \mapsto z + t\mathbb{Z}$
$[\cdot]_t$	$: \mathbb{Z}_t \to \mathbb{Z}$	defined by sending $z + t\mathbb{Z}$ to the unique element in $(z + t\mathbb{Z}) \cap (-t/2, t/2]$.
$\cdot$	$: \mathbb{R} \to \mathbb{Z}$	defined by sending x to $\max\{ z \in \mathbb{Z} \vert z \leq x\}$
$\cdot$	$: \mathbb{R} \to \mathbb{Z}$	defined by sending x to $x + 1/2$
$\Vert\cdot\Vert_\infty$	$: \mathbb{Z}^n \to \mathbb{Z}$	is the infinity norm

$$\Vert\mathbf{a}\Vert_\infty = \max_{0 \leq i < \mathrm{n}} a_i \tag{A2}$$

$$[\mathbf{a} + t \cdot \mathcal{R}]_t = \sum_{0 \leq i < n} [a_i]_t \cdot X^i \tag{A3}$$

modulus operation can be defined as

$$a \bmod p = \left[\pi_p(a)\right]_p - p \cdot 2^{-1} \tag{A4}$$

Appendix A.2. Distribution and Sampling

For a distribution D, we will use $x \leftarrow D$ to denote that "x is sampled from a distribution D". For a finite set S, we will use $x \leftarrow S$ to denote that "x is sampled uniformly from a finite set S". Notice that the symbol $\leftarrow$ is also used as an assignment operator, which can be regarded as sampled from some degenerate distributions.

Appendix B. Detailed Proofs Associated with Section 2

Appendix B.1. Proof of Lemma 1 (Sufficient Condition for Correct Decryption)

Given ciphertext $\mathbf{c} = (\mathbf{c}0,\ \mathbf{c}1) \in \mathcal{R}_q^2$, message $\mathbf{m} \in \mathcal{R}_p$, and $\mathbf{e} = \left[\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s} - [\mathbf{m}]_p \cdot \rho\right]_q$ as the corresponding error, then implies $[\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s}]_q \cdot \rho^{-1} \equiv [\mathbf{m}]_p \bmod p$.

Proof. By the definition of error, we may write

$$\mathbf{e} + \mathbf{v} \cdot q = [\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s}]_q - [\mathbf{m}]_p \cdot \rho \tag{A5}$$

for some $\mathbf{v} \in \mathbb{Z}^n$. Multiply both side with ρ^{-1}, we obtain

$$\mathbf{e} \cdot \rho^{-1} + \mathbf{v} \cdot p = [\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s}]_q \cdot \rho^{-1} - [\mathbf{m}]_p \tag{A6}$$

Since $\|[\mathbf{e}]_q\|_\infty < \rho \cdot 2^{-1}$, elements of $\mathbf{e} \cdot \rho^{-1}$ are less than $1/2$, so

$$\mathbf{v} \cdot p = [\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s}]_q \cdot \rho^{-1} - [\mathbf{m}]_p$$

and the result follows:

$$\left|[\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s}]_q \cdot \rho^{-1}\right| \equiv [\mathbf{m}]_p \pmod{p} \tag{A7}$$

□

Appendix B.2. *Proofs of Claims 1 to 3*

a. Poof of Claim 1 (Correctness of Additions)

By the definition of error, we assume $\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s} = \mathbf{e} + [\mathbf{m}]_p \cdot \rho$ and $\mathbf{c}_0^{'} + \mathbf{c}_1^{'} \cdot \mathbf{s} = \mathbf{e}' + [\mathbf{m}']_p \cdot \rho$, so

$$\begin{aligned}
[\mathbf{c}_0 + \quad & \mathbf{c}_0^{'} + \left(\mathbf{c}_1 + \mathbf{c}_1^{'}\right) \cdot \mathbf{s}]_q \equiv \ \mathbf{e} + \mathbf{e}' + \left([\mathbf{m}]_p + [\mathbf{m}']_p\right) \cdot \rho \pmod{q} \\
& \equiv \mathbf{e} + \mathbf{e}' + \left([\mathbf{m} + \mathbf{m}']_p + p \cdot \mathbf{v}\right) \cdot \rho \text{ for some } \mathbf{v} \in \mathcal{R} \\
& \equiv \mathbf{e} + \mathbf{e}' + \left([\mathbf{m} + \mathbf{m}']_p\right) \cdot \rho + q \cdot \mathbf{v} \\
& \equiv \mathbf{e} + \mathbf{e}' + \left([\mathbf{m} + \mathbf{m}']_p\right) \cdot \rho
\end{aligned} \tag{A8}$$

Thus, the error of $\mathrm{Add}(\mathbf{c}, \mathbf{c}')$ is $\mathbf{e} + \mathbf{e}' \bmod q$, and $\|e + e'\|_\infty < \rho \cdot 2^{-1}$ implies.
Dec (Add ($\mathbf{c}$, $\mathbf{c}'$), $\mathbf{s}$) = $\mathbf{m}$ + $\mathbf{m}'$ by Lemma 1

b. Proof of Claim 2 (Correctness of Addition between a Plaintext and a Ciphertext)**.**

Since $[\mathbf{c}_0 + m' \cdot \rho + \mathbf{c}_1 \cdot \mathbf{s}]_q \equiv \ \mathbf{e} + \left([\mathbf{m}]_p + [\mathbf{m}']_p\right) \cdot \rho \equiv \ \mathbf{e} + \left([\mathbf{m} + \mathbf{m}']_p\right) \cdot \rho$, the error of new ciphertext would be $\mathbf{e}$, which can be correctly decrypted since original ciphertext $\mathbf{c}$ can.

c. Proof of Claim 3 (Correctness of Multiplication of between a Plaintext and a Ciphertext)

Since $[\mathbf{c}_0 + \mathbf{c}_1 \cdot \mathbf{s}]_q \equiv \ \mathbf{e} + [\mathbf{m}]_p \cdot \rho$, multiply both side with $[\mathbf{m}']_p$ and derive

$$\begin{aligned}
\left[\mathbf{c}_0 \cdot [\mathbf{m}']_p + \mathbf{c}_1 \cdot [\mathbf{m}']_p \cdot \mathbf{s}\right]_q \ & \equiv \ \mathbf{e} \cdot [\mathbf{m}']_p + [\mathbf{m}]_p \cdot [\mathbf{m}']_p \cdot \rho \\
& \equiv \ \mathbf{e} \cdot [\mathbf{m}']_p + \left([\mathbf{m} + \mathbf{m}']_p + p \cdot \mathbf{v}\right) \cdot \rho \pmod{q} \\
& \equiv \ \mathbf{e} \cdot [\mathbf{m}']_p + [\mathbf{m} + \mathbf{m}']_p \cdot \rho \pmod{q}
\end{aligned}$$

The error of new ciphertext is $\mathbf{e} \cdot [\mathbf{m}']_p$, so $\|\mathbf{e} \cdot [\mathbf{m}']_p\|_\infty < \rho \cdot \ 2^{-1}$ can ensure the correctness.

Appendix C. Detailed Proofs Associated with Section 3

Appendix C.1. *Proof of Lemma 2 (Distinguishability of a uniform sample with little bias)*

Suppose $X \sim \{-T, \ldots, T\}$. Any algorithm $A : \mathbb{Z} \to \{0, 1\}$ can be defined by $x \mapsto I_S(x)$ with

$$I_s(x) = \begin{cases} 1, & \text{for } x \in \mathrm{S} \\ 0, & \text{for } x \notin \mathrm{S}. \end{cases} \tag{A9}$$

We can compute

$$\text{Prob}\{A(X) = 1\} = \text{Prob}\{X \in S\} = (2T)^{-1} \cdot |S \bigcap \{-T,...,T\}|, \tag{A10}$$

and

$$\text{Prob}\{A(X + e) = 1\} = (2T) - 1 \cdot |S \bigcap \{-T + e,..., T + e\}|. \tag{A11}$$

Since $|S \cap \{-T,..., T\}| - |S \cap \{-T + e,..., T + e\}| \leq |e|$, we have

$$|\text{Prob}\{A(X) = 1\} - \text{Prob}\{A(X + e) = 1\}| \leq (2T) - 1 \cdot |e|. \tag{A12}$$

Therefore, the advantage of the adversary is no more than $(2T)^{-1} \cdot |e|$.

Appendix C.2. *Proof of Claim 4 (The Security of Bob in Protocol 3.1)*

We define our simulator by:

(i) Sample from proper distributions: $r \leftarrow \mathcal{R}_q$ and $\mathbf{u} \leftarrow \{-T,...,T\}^n$.

(ii) Output: $(\mathbf{r} \cdot \mathbf{s} + \mathbf{u} + \pi_q([\text{Dec}(\mathbf{c}^{\mathbf{result}}, \mathbf{s})]_p \cdot \rho, -\mathbf{r})$.

Notice the ciphertext $\mathbf{c}'$ in Protocol 3.1 can be computed by $\mathbf{c}'_1$, $\mathbf{e}'$, and Dec ($\mathbf{c}'$, $\mathbf{s}$), so it is sufficient to say that ($\mathbf{c}'_1$, $\mathbf{e}'$, Dec ($\mathbf{c}'$, $\mathbf{s}$)) is indistinguishable to ($\mathbf{r}$, $\mathbf{u}$, Dec ($\mathbf{c}^{\text{result}}$, $\mathbf{s}$)). By Corollary 1 and the correctness of Protocol 3.1, ($\mathbf{c}'_{b,1}$, $\mathbf{e}'_b$, Dec ($\mathbf{c}'_b$, $\mathbf{s}$)) is indistinguishable to ($\mathbf{c}'_{b,1}$, $\mathbf{u}$, Dec ($\mathbf{c}_b$, $\mathbf{s}$)) with an adversary less than ε. By the RLWE assumption, $\mathbf{r}$ is indistinguishable from $\mathbf{c}'_{b,1}$, and we complete our proof.

References

1. Veugen, T.; Blom, F.; de Hoogh, S.J.A.; Erkin, Z. Secure comparison protocols in the semi-honest model. *IEEE J. Sel. Top. Signal Process.* **2015**, *9*, 1217–1228. [CrossRef]
2. Damga, I.; Geisler, M.; Krøigaard, M. Homomorphic encryption and secure comparison. *IJACT* **2008**, *1*, 22–31. [CrossRef]
3. Paillier, P. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology—EUROCRYPT '99, Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques, Prague, Czech Republic, 2–6 May 1999*; Springer: Berlin/Heidelberg, Germany, 1999; pp. 223–238.
4. Damga, I.; Geisler, M.; Krøigaard, M. A correction to 'efficient and secure comparison for on-line auctions. *IJACT* **2009**, *1*, 323–324. [CrossRef]
5. Groth, J. Cryptography in subgroups of Z∗n. In *Theory of Cryptography, Proceedings of the Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, 10–12 February 2005*; Springer: Berlin/Heidelberg, Germany, 2005; pp. 50–65.
6. Carlton, R.A. Secure Integer Comparisons using the Homomorphic Properties of Prime Power Subgroups. Ph.D. Dissertation, The University of Western Ontario, London, ON, Canada, 2017.
7. Carlton, R.; Essex, A.; Kapulkin, K. Threshold properties of prime power subgroups with application to secure integer comparisons. In *Topics in Cryptology—CT-RSA 2018, Proceedings of the Cryptographers' Track at the RSA Conference, San Francisco, CA, USA, 16–20 April 2018*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 137–156.
8. Damga, I.; Geisler, M.; Krøigaard, M. Efficient and secure comparison for on-line auctions. In *Information Security and Privacy, Proceedings of the 12th Australasian Conference, ACISP 2007, Townsville, Australia, 2–4 July 2007, Lecture Notes in Computer Science*; Pieprzyk, J., Ghodosi, H., Dawson, E., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4586, pp. 416–430. [CrossRef]
9. Veugen, T. Improving the DGK comparison protocol. In Proceedings of the 2012 IEEE International Workshop on Information Forensics and Security (WIFS), Costa Adeje, Spain, 2–5 December 2012; pp. 49–54.
10. Veugen, T. Correction to" improving the DGK comparison protocol. *IACR Cryptol. Eprint Arch.* **2018**, *2018*, 1100.
11. Gentry, C.; Boneh, D. *A Fully Homomorphic Encryption Scheme*; Stanford University: Stanford, CA, USA, 2009; Volume 20.
12. Gentry, C. Fully homomorphic encryption using ideal lattices. In Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, Bethesda, MD, USA, 31 May 2009–2 June 2009; pp. 169–178.
13. Van Dijk, M.; Gentry, C.; Halevi, S.; Vaikuntanathan, V. Fully homomorphic encryption over the integers. In *Advances in Cryptology—EUROCRYPT 2010, Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, France, 30 May–3 June 2010*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 24–43.
14. Brakerski, Z.; Vaikuntanathan, V. Efficient fully homomorphic encryption from (standard) LWE. In Proceedings of the 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science, Palm Springs, CA, USA, 22–25 October 2011; FOCS '11. IEEE Computer Society: Washington, DC, USA, 2011; pp. 97–106. [CrossRef]
15. Brakerski, Z.; Gentry, C.; Vaikuntanathan, V. (leveled) fully homomorphic encryption without bootstrapping. In Proceedings of the 3rd Innovations in Theoretical Computer Science Conference, ITCS '12, Cambridge, MA, USA, 8–10 January 2012; Association for Computing Machinery: New York, NY, USA, 2012; pp. 309–325. [CrossRef]

16. Brakerski, Z. Fully homomorphic encryption without modulus switching from classical GAPSVP. In *Advances in Cryptology—CRYPTO 2011, Proceedings of the Annual Cryptology Conference, Santa Barbara, CA, USA, 14–18 August 2011*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 868–886.
17. Fan, J.; Vercauteren, F. Somewhat practical fully homomorphic encryption. *IACR Cryptol. Eprint Arch.* **2012**, *2012*, 144.
18. Gentry, C.; Sahai, A.; Waters, B. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology—CRYPTO 2013, Proceedings of the Annual Cryptology Conference, Santa Barbara, CA, USA, 18–22 August 2013*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 75–92.
19. Cheon, J.H.; Kim, A.; Kim, M.; Song, Y. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology—ASIACRYPT 2017, Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, Hong Kong, China, 3–7 December 2017*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 409–437.
20. Brakerski, Z.; Vaikuntanathan, V. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Advances in Cryptology—CRYPTO 2011, Proceedings of the Annual Cryptology Conference, Santa Barbara, CA, USA, 14–18 August 2011*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 505–524.
21. Albrecht, M.; Chase, M.; Chen, H.; Ding, J.; Goldwasser, S.; Gorbunov, S.; Halevi, S.; Hoffstein, J.; Laine, K.; Lauter, K.; et al. Homomorphic Encryption Security Standard. HomomorphicEncryption.org: Toronto, ON, Canada. 2018. Available online: http://homomorphicencryption.org/wp-content/uploads/2018/11/HomomorphicEncryptionStandardv1.1.pdf (accessed on 15 February 2023).
22. Al Badawi, A.; Veeravalli, B.; Lin, J.; Xiao, N.; Kazuaki, M.; Mi, A.K.M. Multi-gpu design and performance evaluation of homomorphic encryption on gpu clusters. *IEEE Trans. Parallel. Distrib. Syst.* **2021**, *32*, 379–391. [CrossRef]
23. Özerk, Ö.; Elgezen, C.; Mert, A.C.; Öztürk, E.; Savaş, E. Efficient number theoretic transform implementation on GPU for homomorphic encryption. *J. Supercomput.* **2022**, *78*, 2840–2872. [CrossRef]
24. Tan, B.H.M.; Lee, H.T.; Wang, H.; Ren, S.Q.; Khin, A.M.M. Efficient private comparison queries over encrypted databases using fully homomorphic encryption with finite fields. In *IEEE Transactions on Dependable and Secure Computing*; IEEE: Piscataway, NJ, USA, 2020; pp. 2861–2874.
25. Iliashenko, I.; Zucca, V. Faster homomorphic comparison operations for BGV and BFV. *Proc. Priv. Enhancing Technol.* **2021**, *2021*, 246–264. [CrossRef]
26. Driver, M. Emerging technologies: Homomorphic encryption for data sharing with privacy. *Gartner Inc. Tech. Rep.* Published: 23 April 2020. Available online: https://www.gartner.com/en/documents/3983970 (accessed on 12 February 2023).
27. Huo, M.; Wu, K.; Ye, Q. A note on lower digits extraction polynomial for bootstrapping. *arXiv* **2019**, arXiv:1906.02867.
28. Lyubashevsky, V.; Peikert, C.; Regev, O. On ideal lattices and learning with errors over rings. In Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, France, 30 May–3 June 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 1–23.
29. Microsoft SEAL (Release 3.4). Oct. 2019, Microsoft Research, Redmond, WA. Available online: https://github.com/Microsoft/SEAL (accessed on 11 January 2023).