HeeWon Chung and Myungsun Kim*

Encoding Rational Numbers for HE-based Applications

Abstract: This work addresses a basic problem of security systems that operate on very sensitive information, such as healthcare data. Specifically, we are interested in the problem of privately handling medical data represented by rational numbers. Considering the complicated computations on encrypted medical data, one of the natural and powerful tools for ensuring privacy of the data is homomorphic encryption (HE). However, because the plaintext domain of HE schemes is restricted to a set of quite small integers, it is not easy to obtain efficient algorithms for encrypted real numbers in terms of space and computation costs. Our observation shows that this inefficiency results from the representation of rational numbers using decimal expansion. To be precise, the naïve decimal representation considerably restricts the choice of parameters in employing an FHE scheme, particularly the plaintext size.
The starting point of our technique in this work is to encode rational numbers using *continued fractions*. Because continued fractions enable us to represent rational numbers as a sequence of integers, we can use a plaintext space with a small size while preserving the same quality of precision. However, this encoding technique requires performing very complex arithmetic operations, such as division and modular reduction. Theoretically, HE allows the evaluation of any function, including modular reduction at encrypted data, but it requires a Boolean circuit of very high degree to be constructed. Hence, we primarily focus on developing an approach to solve this efficiency problem using homomorphic operations with *small degrees*.

Keywords: continued fractions, encoding rational numbers, homomorphic encryption, Gosper algorithm

***Corresponding Author: Myungsun Kim:** The University of Suwon, E-mail: msumkim@suwon.ac.kr
HeeWon Chung: Seoul National University, E-mail: runghyun@snu.ac.kr

1 Introduction

Consider the following scenario. There is a server that stores patients' medical data, and it has considerable computing power such that it can compute a predictive model for each patient and inform patients on whether they are in the danger range.

For example, a Cox model is a statistical technique for exploring the relationship between the survival of a patient and several explanatory variables, and it provides an estimate of the effect of treatment on survival after adjusting for other explanatory variables. In addition, this technique allows us to estimate the hazard (or risk) of death for an individual given their prognostic variables. The prognostic variables are age, diabetes, smoking, systolic blood pressure, cholesterol, and HDL cholesterol. For instance, in reference [2], the predictive model for females is given by the function

$$\Pr_{\text{Female}}(X_i) = 1 - 0.95012^{\exp(\sum_i \beta_i X_i - 26.1931)}$$

where X_i is the input for each risk factor, $\exp(\cdot)$ is the exponential function, and β_i is the regression coefficient. Specifically, the regression coefficients for the model for females are given by

$$\begin{aligned}\textstyle\sum_i \beta_i X_i \;=\; & 2.32888 \cdot \log A + 1.20904 \cdot \log C - \\ & 0.70833 \cdot \log H + 2.76157 \cdot \log B + \\ & 0.52873 \cdot S + 0.69154 \cdot D\end{aligned} \qquad (1)$$

where A denotes the age, C denotes the cholesterol level, H denotes the HDL cholesterol level, B denotes the systolic blood pressure, and $D = 1$ if diabetes exists and $S = 1$ if an individual is a smoker; otherwise, $D = 0$ and $S = 0$.

To make this practical scenario work in the real world, we need to satisfy the following two basic requirements before other technical ones.

Privacy. For privacy reasons, one appealing approach is to have patients keep all medical data encrypted if possible. However, because decryption on the server side could cause a loss of privacy, we need to apply an encryption scheme that allows homomorphism to such sensitive data such that computations on its encryptions do not require decryption.

Inter-Domain Conflict. Again, let us examine Eq. (1). Due to the privacy requirement, we now need to compute the equation on the ciphertext domain of an underlying encryption scheme rather than its plaintext domain. The key point here is that the underlying encryption scheme should be able to take as input real numbers such as HDL cholesterol values, but in general, encryption schemes are restricted to encrypt group or ring elements. Consequently, even encryption schemes with multiplicative homomorphism only support multiplication between integers.

To resolve this conflict between the domain of a target function and the domain of an encryption scheme, message encoding before encryption appears to be the best choice. One widely accepted approach to encode a rational plaintext r is to use decimal expansion. Let $r \in \mathbb{R}$ be a rational number that has ℓ-digit below a decimal point. Given a real number $r = r_0.r_1r_2\cdots r_\ell$, an encryption scheme with decimal expansion first converts each digit r_i into $x_i = r_i \cdot 10^{i-1} \in \mathbb{Z}, 1 \leq i \leq \ell$ and then encrypts each x_i. This implies that the plaintext space of the encryption scheme must be large to represent an integer $x_\ell = r_\ell \cdot 10^{\ell-1}$. For example, suppose that a predictive algorithm uses a rational number $r = 0.1357908642$. Then, in this case, $x_{10} = 2\cdot 10^9 \approx 2^{31}$, and thus, a proper plaintext space superficially seems to be $\mathbb{Z}_{2^{31}}$.

However, if we think a little bit more about our choice of the plaintext space, we see that $\mathbb{Z}_{2^{31}}$ is not a correct one under decimal expansion. To explain why this is so, suppose that a homomorphic encryption (HE) scheme and a target function f for evaluation are given. One may simply think that he only has to fix t (e.g., $t = 2^{31}$) to the FHE's plaintext space. Apparently it is not the case. Because the encryption scheme should perform modular reduction when an evaluation result of f over HE ciphertexts carries a plaintext larger than t, the plaintext space can be determined only after estimating the target evaluation function. For example, when we consider a proper plaintext space for encoding $r = 0.1357908642$, $\mathbb{Z}_{2^{31}}$ may not be a good choice because we may have to encode $8 \cdot r$. Specifically, $1357908642 \times 8 = 10863269136 \approx 2^{34}$ and thus $10863269136 \bmod 2^{31}$ should be performed. Hence, we need to set $t \geq 2^{34}$. The more are required the number of multiplications, the bigger plaintext space should be chosen. In conclusion, a proper plaintext space depends on a target evaluation function in order to avoid modular reduction.

Furthermore, since the size of the noise after homomorphic operations is proportional to t, it may lead to an efficiency problem. For example, consider the Brakerski-Gentry-Vaikuntanathan (BGV) scheme [4]. If $\mathbb{Z}_2$ is taken as the plaintext space, one homomorphic multiplication consumes only one multiplicative depth. However, in the case of $\mathbb{Z}_{2^6}$, the same operation consumes two multiplicative depths.

Group homomorphic encryption (e.g., El Gamal [10] and Paillier [19]) can also be considered. However, this type of encryption scheme is not suitable for bitwise operations because it severely wastes their plaintext space.

Taking two requirements and the two together, we need to provide a better way of representing rational numbers to fit in homomorphic evaluations in the predictive model such that multiplications by rational numbers are needed. Thus, in this work, our technical goal is to represent rational numbers while using a small-sized plaintext space, but without losing its original precision.

Overview of our idea. Our key idea for achieving this goal is to use *continued fractions* to represent ratioanl numbers. Roughly, continued fractions can represent the same rational numbers using only relatively quite small integers. For example, consider the same $r = 0.1357908642$ as above. Then, we can represent the r as a sequence of integers

$$[0; 7, 2, 1, 2, 12, 2, 2, 7, 7, 2, 2, 1, 11, 2]$$

and indeed, it is a sequence of only 4-bit integers. That is, we can take $\mathbb{Z}_{2^4}$ rather than $\mathbb{Z}_{2^{31}}$ as the plaintext domain. We will discuss how to obtain each integer of the sequence in later sections.

By applying continued fraction representations of rational numbers to the prediction model, such as Eq. (1), we obtain the following observations.

1. In practice, all coefficients and input values are not given in real numbers but rather in rational numbers after truncating their lower digits.
2. Regression formulas of our interest perform multiplication between the encrypted medical data and some known constant values (e.g., coefficients 2.32888 and 1.20904 of Eq. (1)). This means that we do not need to perform multiplication of two encrypted medical data.

In a nutshell, when we homomorphically evaluate a linear multivariate polynomial f on encryptions, we first encode all inputs into a sequence of small integers and encrypt each small integer using HE scheme. This leads us to establish the HE scheme with a small-sized plaintext space. Note that the second observation states that the coefficients of f need not be encrypted, unlike the input values.

Because a continued fraction (CF) representation can express a rational number as a sequence of small integers, it can provide significant efficiency gains, even in the case where elements of the set should be encrypted, with respect to the size of ciphertexts and later homomorphic arithmetic operations.

When encoding a rational number into CF, we use the so-called Gosper's algorithm [13]. However, some nontrivial problems arise when one has to perform computations on encrypted rational numbers that were encoded by the algorithm.

The essential technical obstacle among these problems is that the algorithm requires modular arithmetic and division. Theoretically, FHE can compute arbitrary functions on encryptions, but in practice, both operations are fairly difficult to efficiently implement when their operands are ciphertexts. Throughout the remainder of the paper, we thus seek a solution to replace modular arithmetic and division with other operations with a *small degree*, i.e., efficiently realized. Indeed, we develop a variant of Gosper's algorithm that works on ciphertexts of rational numbers encoded using the CF technique. Throughout the paper, by degree, we mean the multiplication depth required to perform a function. [1]

One may think that an HE scheme is overkill for this application and that Paillier's cryptosystem is the best solution while using decimal expansion of rational numbers. [2] If an encryption scheme were chosen only for this class of target functions, this may be a correct choice. However, if we should evaluate various functions at two encrypted medical data, this choice may be incorrect. For example, consider an equality check between two encrypted hematocrit values. In general, an equality comparison cannot be efficiently performed over Paillier ciphertexts. Moreover, it is not clear whether we can require patients to provide multiple ciphertexts of the same plaintext that are encrypted by distinct encryption schemes.

Summary of Our Results. In summary, our contributions are as follows:

1. We show that we can enable to divide an integer by an encrypted integer with a special case of Gosper algorithm. Further, we can compute a linear polynomial evaluation whose variables are rational numbers represented by continued fractions and its computational complexity is almost the same cost regardless of polynomial's coefficient.
2. We address some efficiency challenges posed by using the Gosper algorithm during the encoding of rational numbers. Specifically, modular reduction over ciphertexts is very expensive in terms of multiplicative depth and thus we develop an approach to substitute a combination of other operations requiring a quite *less* multiplicative degree for modular reduction. We apply this technique to the original Gosper algorithm.
3. On the practical side, we provide an implementation of our version of the Gosper algorithm running on encrypted rational numbers. We provide experimental results to demonstrate that CF representations of rational numbers can have a potentially wide range of HE-based applications.

The Structure of the Paper. Section 2 provides a series of materials for better understanding our work and it consists of two parts: . We describe our main result in Section 3 and Section 4, along with analysis of correctness and computational costs. In Section 5, we describe our proof-of-concept implementation and its experimental results. We provide a survey of closely related works in Section 6. We end with some concluding remarks in Section 7.

2 Model and Toolkits

In this section, we provide some basic materials to better understand the remainder of the paper. Background of the two different fields is required: one is mathematics, and the other is cryptography.

In the following, we begin by describing the simple model that we are considering.

2.1 System Model

We demonstrate the models that we focus on in this paper. For this purpose, we first identify all participants of the system model. Then, we discuss our definition of security.

Participants and System Model. We consider a simple system model that consists of two participants:

1 Of course, the function should first be transformed into a binary Boolean circuit or an arithmetic circuit.

2 Regarding the Boneh-Goh-Nissim encryption scheme [1], we can also apply the same argument as above.

server and user. Specifically, a server is a logical entity that has both powerful computing and storage capabilities for evaluating known linear polynomials whose coefficients over rational numbers and a user wants to accomplish some computation with his private input data.

Security Model. As with any cryptographic protocol, we require completeness and soundness to hold. Thus, a securely real-valued function evaluating scheme should meet the following security requirements:

Correctness. The secure evaluation should be performed correctly, i.e., the computation indeed returns the same result as the evaluation of the function in the clear.

Security. The server should not learn any information about the data they store in its storage and information about the data contained in the results. However, we allow the server to learn the specifications of a target function.

In the cryptography literature, two standard security models are considered: semi-honest and malicious adversarial models. Semi-honest participants are allowed to attempt to learn some information from the data they receive, including intermediate values, but not allowed to deviate from the protocol instructions. Malicious participants, on the other hand, are allowed to arbitrarily deviate from the computation specified in a protocol.

In this work, we only consider security against semi-honest servers. This is a natural assumption in practice because the server would run a high risk of penalty for compromising a user's medical information with coalition.

We formally define security using the standard notion in secure multiparty computation for semi-honest adversaries. Because a target function is fixed by a server, its coefficients should be interpreted as no private input to the function that the server is evaluating. For the purpose of defining security, all data the server receives before or during the evaluation will be considered to be a part of the function evaluation and therefore must not leak information.

Definition 1. *[[]] Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a function, and let $i \in \{1,2\}$. Let two participants P_1 and P_2 engage in a protocol π that evaluates $f(x_1,x_2) = (y_1,y_2)$, where x_i and y_i denote the input and output, respectively, of P_i. Then, the view of P_i during an execution of π on (x_1,x_2), denoted by $\text{VIEW}_{\pi,P_i}(x_1,x_2)$, is $(x_i, r_i, m_1, \ldots, m_t)$, where r_i is the outcome of P_i's internal coin tosses and m_j is the j-th message that it has received. The output of each participant following the execution of π on (x_1,x_2), denoted by $\text{OUT}_{\pi,P_i}(x_1,x_2)$, is implicit in the participant's own view of the execution, $\text{OUT}_{\pi}(x_1,x_2) = (\text{OUT}_{\pi,P_1}(x_1,x_2), \text{OUT}_{\pi,P_2}(x_1,x_2))$. We say that π securely computes f if there exist probabilistic polynomial-time simulators, denoted by $\mathcal{S}_1$ and $\mathcal{S}_2$, such that for $x_1, x_2 \in \{0,1\}^*$*

$$\{\mathcal{S}_i(x_i, f(x_1,x_2)\} \approx \{\text{VIEW}_{\pi,P_i}(x_1,x_2), \text{OUT}_{\pi}(x_1,x_2)\}$$

where $\approx$ denotes computational indistinguishability.

2.2 Mathematical Tools

A set of integers is closed under addition and multiplication, but not division. However, there are some methods representing rational numbers to integers, e.g., continued fractions and decimal representations. Continued fractions are more *mathematically natural* representations of rational numbers than decimal representations. First of all, the continued fraction representation for a rational number is finite, but decimal representation for a rational number may be infinite. Moreover, every rational number has an unique continued fraction representation with some restrictions. The successive approximations generated in finding the continued fraction representation of a number, i.e. by truncating the continued fraction representation, are in a certain sense (described below) the best possible. Therefore, it has occasionally been considered to be a great tool in mathematics and it has been researched for a long time in a variety of topics. Here, we rephrase only those related to our works.

Continued Fractions. A continued fraction can be obtained through an iterative process of representing a number as the sum of its integer part and the reciprocal of remaining part, and then writing the remaining part as the sum of its integer part and remaining part, and so on. In other words, given a real number x and $r_i > 1$ for all i, we have

$$x = x_0 + \frac{1}{r_0} = x_0 + \frac{1}{x_1 + \frac{1}{r_1}} = x_0 + \frac{1}{x_1 + \frac{1}{x_2 + \frac{1}{r_2}}} = \cdots$$

and use $x = [x_0; x_1, x_2, \ldots]$ to denote this. Note that x_0 can be any integer, but for $i \in \mathbb{N}, x_i$ must be positive and with this restriction, the continued fraction of x is unique. We can define further terminologies related to continued fractions.

Definition 2 ([15]). *For a continued fraction $x = [x_0; x_1, \ldots]$,*

- *x_i is called a partial quotient of x for all i.*
- *A continued fraction x is finite if the number of partial quotients of some x is finite.*

Keeping this definition in mind, the following theorem states that the correspondence between rational numbers and a finite continued fraction $[a_0; a_1, \ldots, a_n]$ with an integer a_0 and positive integer a_i for $i > 0$ and $a_n > 1$ is one-to-one. We consider the situation that an integer is divided by another integer, so only rational numbers are considered in this paper.

Theorem 1 ([15]). *Any rational number can be represented as a finite continued fraction and the continued fraction representation is unique when the last partial quotient is larger than 1.*

Theorem 2 ([15]). *Let $\alpha = [a_0; a_1, a_2, \cdots]$ and $p_i/q_i = [a_0; a_1, \cdots, a_i]$.r For any rational number $\frac{a}{b}$ with $a \in \mathbb{Z}$ and $b \in \mathbb{N}$, and $1 \le b \le q_i$,*

$$\left|\frac{p_i}{q_i} - \alpha\right| \le \left|\frac{a}{b} - \alpha\right|,$$

with equality if and only if $a/b = p_i/q_i$.

Theorem 2 indicates that p_i/q_i is the best possible approximation to α among all rational numbers with the same or smaller denominator. In other words, a continued fraction is a best approximation tool of real numbers.

Lastly, we give a assertion that the partial quotients have the small in terms of bits. With some restriction in Theorem 1, continued fractions and rational numbers have one-to-one correspondence, however, one can find two different continued fractions for a rational number when restrictions are weaken. For some $k \in \mathbb{N}$, we can easily derive that $[\ldots, a_k + a_{k+1}, \ldots] = [\ldots, a_k, 0, a_{k+1}, \ldots]$ because

$$\frac{1}{a_k + a_{k+1} + \frac{1}{a_{k+2}}} = \frac{1}{a_k + \frac{1}{0 + \frac{1}{a_{k+1} + \frac{1}{a_{k+2}}}}}$$

By the reason, we insist that one can find a continued fraction with bounded partial quotient for an arbitrary rational number and it causes the length of partial quotients is longer. Thus, the maximum of partial quotients can be determined by user's environment.

2.2.1 Arithmetics over CF Representations

Given two rational numbers x and y represented by decimal expansion, it is easy to compute $z = x + y$, as we have learned. However, we have not learned an arithmetic algorithm if one of the rational numbers is given by continued fraction form. In 1972, Bill Gosper [13] proposed the general arithmetic algorithm on continued fractions. This algorithm enables arithmetics between two continued fractions as well as a continued fraction and a rational number. We recall a brief description of Gosper's algorithm.

Gosper Algorithm. The goal of Gosper's algorithm is, for a real number x, computing $\frac{a+bx}{c+dx}$ for $a, b, c, d \in \mathbb{Z}$ because every elementary single operation of two rational numbers, such as addition, subtraction, multiplication and division, can be replaced by a linear fractional transformation. Of course, it can also represent various equations in addition to the elementary single operations, and thus, we can calculate more complex arithmetic than elementary operations.

Suppose that x is a rational number with a continued fraction form and that $z := \frac{a+bx}{c+dx}$ for $a, b, c, d \in \mathbb{Z}$. The main idea is that z has a value between $\frac{a}{c}$ and $\frac{b}{d}$ for almost everywhere, and thus, if $\frac{a}{c}$ and $\frac{b}{d}$ have the same integer part, the integer part of z is $\lfloor \frac{a}{c} \rfloor = \lfloor \frac{b}{d} \rfloor$; otherwise, z requires more information about x. Because z behaves differently for each case, it requires two sub-algorithms, say InTake and OutTake. For convenience, we denote z by $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$.

The following is the Gosper algorithm for an arithmetic algorithm on continued fractions using the above idea. For each iteration, call either OutTake or InTake according to the condition of whether two truncations are the same.

Algorithm 1 *Gosper's algorithm*
Input: $x = [x_0; x_1, x_2, \ldots], f(X) = \frac{a+bX}{c+dX}$
Output: $f(x) = [y_0; y_1, y_2, \ldots]$

```
1.  $\begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix} \leftarrow \begin{pmatrix} a & b \\ c & d \end{pmatrix}$
2.  $i \leftarrow 0, j \leftarrow 0$
3.  while ($z_2$ and $z_3$ are not all zero)
4.        $q_1 \leftarrow \lfloor z_0/z_2 \rfloor, q_2 \leftarrow \lfloor z_1/z_3 \rfloor$
5.        if ($q_1 = q_2$)
6.          then $y_i \leftarrow q_1$
7.                 $\begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix} \leftarrow \begin{pmatrix} z_2 & z_3 \\ z_0 - z_2 y_i & z_1 - z_3 y_i \end{pmatrix}$
8.                 $i \leftarrow i + 1$
9.          else $\begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix} \leftarrow \begin{pmatrix} z_1 & z_0 + z_1 x_j \\ z_3 & z_2 + z_3 x_j \end{pmatrix}$
```

```
10.            $j \leftarrow j+1$
11.  return $[y_0; y_1, y_2, \ldots]$
```

The Details of OutTake Algorithm. It occurs when z exactly knows its own integer part, as we stated above. The integer part of z should be $q := \lfloor \frac{z_0}{z_2} \rfloor$ if the value of the flooring two ratios is the same. Thus, z can determine its own integer part by comparing the two values, and when they have the same value, z must emit q and becomes the reciprocal of $z-q$.

Let $z' = \frac{1}{z-q}$. Then,

$$
\begin{aligned}
z' &= \frac{1}{z-q} = \left(\frac{a+bx}{c+dx} - q\right)^{-1} = \left(\frac{a-cq+bx-dqx}{c+dx}\right)^{-1} \\
&= \frac{c+dx}{a-cq+(b-dq)x} \\
&= \begin{pmatrix} c & d \\ a-cq & b-dq \end{pmatrix}
\end{aligned}
$$

Because $a = cq + a \bmod c$, $a \bmod c = a - cq$; thus, OutTake actually performs the modular arithmetic. In summary, if $z = \begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix}$ knows its own integer part, then it becomes

$$
\begin{pmatrix} z_2 & z_3 \\ z_0 \bmod z_2 & z_1 \bmod z_3 \end{pmatrix} \qquad (2)
$$

The Details of InTake Algorithm. It occurs when z does not exactly know its own integer part. If the value of the two ratios is not the same, then z cannot determine which integer is correct, and thus, z requires more information of x and thus requests a term from x. In this case, additional information of x is its partial quotient.

Let $x = p + \frac{1}{x'}$ for some x'. Then,

$$
\begin{aligned}
z &= \frac{a+bx}{c+dx} \\
&= \frac{a+b(p+1/x')}{c+d(p+1/x')} = \frac{b+(a+bp)x'}{d+(c+dp)x'} \\
&= \begin{pmatrix} b & a+bp \\ d & c+dp \end{pmatrix}
\end{aligned}
$$

In summary, if $z = \begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix}$ requests more information from x and obtains p, then it becomes $\begin{pmatrix} z_1 & z_0+z_1p \\ z_3 & z_2+z_3p \end{pmatrix}$. Additionally, InTake takes a partial quotient of x one at a time. Because

$$
\cdots + \frac{1}{x_n} = \cdots + \frac{1}{x_n + \frac{1}{\infty}},
$$

it is a fact that $[\ , \ldots, x_n] = [\ , \ldots, x_n, \infty]$. Note that ∞ is just a symbolic of the last partial quotient of any rational number and so we do not need to store and encrypt ∞. When InTake takes an input ∞, $z = \begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix}$ becomes $\begin{pmatrix} z_1 & z_1 \\ z_3 & z_3 \end{pmatrix}$ because $b \cdot \infty$ and $d \cdot \infty$ dominate any a and c, respectively; thus, two columns of z always behave in the same way after taking ∞. This implies that z always has the same integer part, and thus, $x = \infty$ is the last input of InTake.

A Toy Example. We will provide a brief example of how this algorithm actually works. Let $x = \frac{13}{11} = [1; 5, 2]$ and $f(X) = X + \frac{1}{2} = \frac{1+2X}{2}$.

$$
\begin{tikzcd}
\begin{pmatrix} 1 & 2 \\ 2 & 0 \end{pmatrix} \arrow[r, "\mathsf{InTake}", "x_0=1"'] & \begin{pmatrix} 2 & 3 \\ 0 & 2 \end{pmatrix} \arrow[r, "\mathsf{InTake}", "x_1=5"'] & \begin{pmatrix} 3 & 17 \\ 2 & 10 \end{pmatrix} \arrow[d, "\mathsf{OutTake}", "y_0=1"'] \\
\begin{pmatrix} 7 & 15 \\ 3 & 7 \end{pmatrix} \arrow[d, "\mathsf{OutTake}"', "y_2=2"] & \begin{pmatrix} 10 & 22 \\ 7 & 15 \end{pmatrix} \arrow[l, "\mathsf{OutTake}"', "y_1=1"] & \begin{pmatrix} 2 & 10 \\ 1 & 7 \end{pmatrix} \arrow[l, "\mathsf{InTake}"', "x_2=2"] \\
\begin{pmatrix} 3 & 7 \\ 1 & 1 \end{pmatrix} \arrow[r, "\mathsf{InTake}", "x_3=\infty"'] & \begin{pmatrix} 7 & 7 \\ 1 & 1 \end{pmatrix} \arrow[r, "\mathsf{OutTake}", "y_3=7"'] & \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}
\end{tikzcd}
$$

Fig. 1. A toy example of Gosper's algorithm

Because OutTake yields $z_0 = 1, z_1 = 1, z_2 = 2$ and $z_3 = 7$, $y = f(x) = [y_0; y_1, y_2, y_3] = [1; 1, 2, 7] = 37/22$. It is trivial that $13/11 + 1/2 = 37/22$ and thus we can verify this algorithm works correctly in this case.

2.3 Cryptographic Tools

In the model in which we are considering described in Section 2.1, the server has powerful computing power and

Private information such as medical data and location information should not be leaked to any others. To prevent leakage of information by hacking or carelessness, all private information should be stored as encrypted. One may consider a trivial approach to store only hashed private information in the server through a one-way hash function such as SHA3 [17]. However, a hash function can only judge the equality, but does not have the homomorphic property. By the reason, a service range the server provided will be strictly restricted.

A strong candidate for fixing this problem is homomorphic encryptions (HE) which enable to evaluate without decryption. In the following, we proceed to defining the concept of HE.

Homomorphic Encryptions. In 2009, Gentry constructed the first FHE scheme [11] based on ideal lattices, and subsequently, many cryptosystems achieved the homomorphic property such as [7, 9, 20] based on integers and [3–6, 12] based on learning with errors.

An HE scheme, denoted by $\mathsf{FHE} = (\mathsf{Kg}, \mathsf{E}, \mathsf{D}, \mathsf{Ev})$, is a quadruple of probabilistic polynomial-time algorithms, as follows.

Key generation. The algorithm takes the security parameter λ and outputs a public encryption key pk, a public evaluation key ek and a secret decryption key sk. We write the algorithm as $(pk, ek, sk) \leftarrow \mathsf{Kg}(1^\lambda)$ and assume that the public key specifies the plaintext space P and the ciphertext space C.

Encryption. The algorithm $\bar{x} \leftarrow \mathsf{E}_{pk}(x)$ takes the public key pk and a message $x \in \mathsf{P}$ and outputs a ciphertext $\bar{x} \in \mathsf{C}$.

Decryption. The algorithm $x^* \leftarrow \mathsf{D}_{sk}(\bar{x})$ takes the secret key sk and a ciphertext c and outputs a message $x^* \in \mathsf{P}$.

Homomorphic evaluation. The algorithm takes the evaluation key ek, a function $f : (\{0,1\}^*)^n \rightarrow \{0,1\}^*$, and a set of n ciphertexts $\bar{\alpha}_1, \ldots, \bar{\alpha}_n$ and outputs a ciphertext $\bar{\alpha}_f$, denoted by $\bar{\alpha}_f \leftarrow \mathsf{Ev}_{ek}(f, \bar{\alpha}_1, \ldots, \bar{\alpha}_n)$.

We defer the security definition of HE to the Appendix. We do not consider an HE scheme satisfying circuit privacy simply because of our efficiency requirement. However, there have been known techniques to achieve circuit privacy (e.g., [11, 18, 20]).

A Concrete Instantiation. For the security parameter λ, we choose an $N \in \mathbb{Z}$ that defines the N-th cyclotomic polynomial $\Phi_N(X)$. For a polynomial ring $R = \mathbb{Z}[X]/\langle \Phi_N(X) \rangle$, we set the message space to $R_t := R/tR$ for some fixed $t \geq 2$ and the ciphertext space to $R_q := R/qR$ for an integer q. We choose a chain of moduli $q_0 < q_1 < \cdots < q_L = q$ whereby the leveled FHE scheme can evaluate a depth-L arithmetic circuit. The RLWE-based BGV FHE scheme is as follows:

Key generation. The algorithm Kg chooses a Hamming-weight h secret key s and generates an RLWE instance (δ_0, δ_1) relative to that secret key. We set the secret key $sk = \mathsf{s}$ and the public key $pk = (\delta_0, \delta_1)$.

Encryption. To encrypt a message $x \in R_t$, the algorithm chooses a small polynomial v and two Gaussian polynomials e_0, e_1 (with variance σ^2). It outputs the ciphertext $\mathbf{c} = (c_0, c_1)$ by computing

$$(c_0, c_1) = (x, 0) + (\delta_1 v + te_0, \delta_0 v + te_1) \bmod q_L.$$

Decryption. Given a ciphertext $\mathbf{c} = (c_0, c_1)$ at level l, the algorithm outputs $x = c_0 - \mathsf{s} \cdot c_1) \bmod q_l \bmod t$.

Homomorphic evaluation. If the function f is an addition over ciphertexts, the algorithm outputs the ciphertext performed by simple component-wise addition of the two ciphertexts. If f is a multiplication over ciphertexts, it outputs the one performed using a tensor product.

In fact, our description may offer a big picture of the BGV cryptosystem because we intentionally omitted many details of the encryption scheme. We thus advise the readers to refer to the reference [4].

Definition 3. Multiplicative depth *of the circuit under homomorphic encryption is the total number of reduced levels in a circuit that is being evaluated homomorphically.*

Multiplicative depth has a significant effect on the performance, and thus, it plays an important role in terms of complexity.

3 Divided by Encrypted Integer

In this section, we demonstrate a method for dividing an integer by an encrypted integer. Since a set of integers is not closed under division, a rational number comes out with high probability. The message space of HE is a set of integers, so one may think that HE is not suitable for this setting. However, a rational number can be represented in various ways. We usually use a decimal representation to represent a rational number, but continued fraction representation can also represent a rational number and it consists of a set of integers. Thus, dividing an integer by encrypted integer is possible when a outcome comes out in a continued fraction form. Fortunately, we can achieve this by a special case of Gosper algorithm.

Notation. For readability, we define some notations and terminologies for this section.

– A bar over some integer means that the integer is encrypted by an underlying FHE scheme, that is, for $x \in \mathbb{Z}$,

$$\bar{x} = \mathsf{E}(x)$$

where $\mathsf{E}(\cdot)$ is an FHE encryption algorithm [4].

– A *maximum of a continued fraction* signifies that the largest integer among its partial quotients, namely, for a continued fraction $x = [x_0; x_1, x_2, \ldots]$,

$$\max x := \max_i |x_i|$$

- A *continued fraction is encrypted* means that each partial quotient of the continued fraction is encrypted, and homomorphic encryption is used to allow arithmetic on partial quotients with each other. In other words, for a continued fraction $x = [x_0; x_1, x_2, \ldots]$,
$$\mathsf{E}(x) := [\mathsf{E}(x_0); \mathsf{E}(x_1), \mathsf{E}(x_2), \ldots]$$
and it can sometimes be abbreviated as $\bar{x} := [\bar{x}_0; \bar{x}_1, \bar{x}_2, \ldots]$.
- For a rational number x, denote its fractional linear transformation $\frac{a+bx}{c+dx}$ for $a, b, c, d \in \mathbb{Z}$ by $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$.
- For two algorithms A and B, $\mathsf{A}(z)$ means that A takes as an input z and $\mathsf{B} \circ \mathsf{A}(z)$ means call A and B in order as an input z.

A representation of continued fraction for an integer is the same as the decimal representation and it means that the number of partial quotient of the integer is the only one. Consequently, InTake occurs the only once and then a couple of OutTake are followed. Originally, Gosper algorithm requires that determining which sub-algorithm should be invoked and it is big problem when adopting HE to Gosper algorithm. But, in this case, the sequence of sub-algorithm is already determined and thus it does not matter anymore.

However, there are still some obstacles to directly applying HE to Gosper algorithm. Although HE theoretically supports both addition and multiplication, it is very difficult to divide two ciphertexts and to compute its remainder because it requires many ciphertext multiplications. The bigger problem is that Gosper's algorithm requires divisions due to the condition that determines which sub-algorithm should be invoked. In addition, Eq. (2) in OutTake also incurs modular-reduction operations. In the following, we show how to substitute modular arithmetic with low degree homomorphic operations and our construction of arithmetic algorithm based on Gosper's algorithm.

3.1 Efficient Modular Arithmetic on FHE Ciphertexts

To apply HE, some additional theorems and tools are required. In general, however, over FHE ciphertexts, there are no efficient division and modular arithmetic algorithms. Unfortunately, the OutTake sub-module to run the Gosper algorithm requires modular arithmetic and hence, we need to *replace modular arithmetic* with some other operations of *low degrees.*

Our idea for an alternative to modular arithmetic is to use a *greater-than comparison* circuit. According to reference [8], a circuit exists that enables the comparison of two encryptions, say a GT circuit. Furthermore, the authors show that we can construct a GT circuit of depth $\lceil \log n \rceil + 1$ when a plaintext x represented by a binary string $(x_0 x_1 \cdots x_{n-1})$ has been encrypted in a bit-by-bit manner. By writing $\bar{x}$ for an integer x, recall that we mean $\bar{x} := (\bar{x}_0, \bar{x}_1, \ldots, \bar{x}_{n-1})$.

A GT circuit takes as inputs two encryptions $\bar{m}_1$ and $\bar{m}_2$ and outputs an encryption of 1 if $m_1 \geq m_2$ and an encryption of 0 otherwise. Namely,
$$\mathsf{GT}(\bar{m}_1, \bar{m}_2) = \begin{cases} \bar{1} & \text{if } m_1 \geq m_2 \\ \bar{0} & \text{if } m_1 < m_2 \end{cases}$$
The output of the GT circuit is an encryption of 0 or 1, and thus, any information about two input integers is not revealed. We provide a way to construct the GT circuit of $\lceil \log n \rceil + 1$ in Appendix A.1.

Then, the following theorem indicates that we can find the upper bound of partial quotients of resulting value in terms of maximum partial quotient and a coefficient of a fractional linear transform. Thus, we can perform modular reduction by iteratively running only the greater-than comparison test at most the number of upper bound partial quotient according to Theorem 3.

Theorem 3 ([16])**.** *Let $x = [x_0; x_1, x_2, \ldots, x_n]$ be a continued fraction. Then,*
$$\max \frac{a+bx}{c+dx} \leq |ad - bc| \cdot \max x.$$

As stated in the example in Section 2.2.1, $x = \frac{13}{11} = [1; 5, 2]$ and $\max x = 5$. By Theorem 3, we may assume that the maximum partial quotient of $\frac{1+2x}{2+0x}$ is
$$|1 \times 0 - 2 \times 2| \cdot \max x = 20$$
and it can easily be verified that $\max \frac{1+2x}{2} = 7 \leq 20$ because $\frac{1+2x}{2} = [1; 1, 2, 7]$.

In summary, it is a considerable burden to divide two ciphertexts even if efficient FHE schemes are used because its multiplicative depth is too much large. However, by applying Theorem 3, modular arithmetic can be replaced by GT circuits, which require significantly smaller multiplicative depth. This results from the fact that every partial quotient is bounded.

3.2 Our Suggestion

An integer has the same form regardless of a decimal representation and a continued fraction and it has the only one partial quotient. Hence, InTake invokes only once and then OutTake invokes several times depending on the desired accuracy. To perform OutTake, a modular arithmetic is required and Section 3.1 is the solution. In Algorithm 2, we provide a modified algorithm that enables to divide an integer by an encrypted integer.

Algorithm 2 *Our variant of Gosper Algorithm*
Input: $\bar{x}, f(X) = \frac{a}{X}, \mathsf{MAX}$
Output: $f(\bar{x}) = [\bar{y}_0; \bar{y}_1, \ldots, \bar{y}_{n-1}]$

```
1.  $\begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix} \leftarrow \begin{pmatrix} a & a \\ \bar{x} & \bar{x} \end{pmatrix}$  /* InTake */
2.  for $i = 0$ to $n-1$ /* OutTake */
3.      $\bar{y}_i \leftarrow 0$
4.      do $tmp \leftarrow z_1$
5.          for $j = 0$ to $\mathsf{MAX}$
6.              do $\bar{t} \leftarrow \mathsf{GT}(tmp, z_3)$
7.                  $\bar{y}_i \leftarrow \bar{y}_i + \bar{t}$
8.                  $tmp \leftarrow tmp - \bar{t} \cdot z_3$
9.          $\begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix} \leftarrow \begin{pmatrix} z_3 & z_3 \\ tmp & tmp \end{pmatrix}$
10. return $[\bar{y}_0; \bar{y}_1, \ldots, \bar{y}_{n-1}]$
```

3.3 Analysis

In this section, we analyze the performance of Algorithm 2 in terms of the number of multiplications. Note that multiplication over HE ciphertext is the most expensive operation.

Correctness. We first need to argue the correctness of our suggestion, which is a special case ($n = 1$) of Gosper algorithm. Algorithm 2 is fundamentally based on Gosper algorithm and thus we compare Gosper algorithm and our suggestion.

Without loss of generality, we may assume that two truncate ratios are different and so InTake always incurs at first. This is the same procedure that Gosper algorithm, but only differs whether or not encryption. After executing InTake, the remaining step is to run OutTake until the algorithm is over. OutTake requires modular reductions, but we do not perform modular reductions because we know the upper bound of partial quotients by Theorem 3. Modular reductions between a and b ($a > b$) in the clear can be viewed as continually subtracting b from a until b is larger and the number of subtract is quotient of a and b. In this sense, this should be the same value compared with the output of OutTake, which corresponds to line 7 to line 12.

To sum up, every line of Algorithm 2 corresponds to the original algorithm and every step does not influence the result. Therefore, the resulting value of our suggestion is the same as the output of Gosper algorithm which implies that Algorithm 2 works correctly. This completes the proof of correctness.

Security. Next, we show that our variant of Gosper's algorithm is secure against a semi-honest server. Although security is straightforward, for completeness we provide the security analysis. A server's computations consist of repetitive applications of our variant to each term of a prediction model and thus it suffices to prove the security of our variant in the semi-honest security model. Thus, we formally prove in the next theorem the security of Algorithm 2 run by a semi-honest server.

Theorem 4. *Assuming that the underlying FHE scheme* $\mathsf{FHE} = (\mathsf{Kg}, \mathsf{E}, \mathsf{D}, \mathsf{Ev})$ *provides semantic security, Algorithm 2 is secure in the presence of a semi-honest server.*

Proof. To show the security, we construct a polynomial-time algorithm $\mathcal{S}$ that, given the server's input and output, produces the server's view, which is indistinguishable from the algorithm execution. The server's input consists of the specification of a target function $f(X)$ and an encryption $\bar{x}$, and at the conclusion of the execution, the server learns an evaluation result $f(\bar{x})$, denoted by $\bar{f}_x$.

The simulator $\mathcal{S}$ first invokes $\mathsf{Kg}(1^\lambda)$ and sends (pk, ek) to the server while keeping the secret key sk. Following the specification of the function f, $\mathcal{S}$ chooses a set of random small integers in the plaintext space, i.e., $(r_0, r_1, \ldots, r_{n-1})$, and outputs the corresponding encryptions $\bar{r} := (\bar{r}_0, \bar{r}_1, \ldots, \bar{r}_{n-1})$. To simulate the evaluation of f, $\mathcal{S}$ sends $\bar{r}$ to the server. Then, the server runs the variant of Gosper's algorithm working on the encryptions, and $\mathcal{S}$ can be modified to produce the evaluation result $\bar{f}_r$ as its output.

It is quite clear that the simulator runs in polynomial time. Next, we examine whether the server can distinguish the above simulation and the real execution of the algorithm. The public keys (pk, ek) that the server receives during the simulation are distributed identically to the real algorithm execution, and thus, the server cannot tell the difference. Moreover, the server cannot distinguish the ciphertext $\bar{r}$ that it receives during the simulation from the ciphertext $\bar{x}$ in the real algorithm execution due to the semantic security of the underlying FHE scheme. Finally, the server also cannot distinguish the result $\bar{f}_r$ that it produces during the simulation from the result $\bar{f}_x$ that it outputs during the real execution due to the semantic security of the FHE scheme. Hence, we can conclude that no polynomial-time semi-honest adversary can learn anything about the private input and its evaluation result. □

Complexity. We provides an analysis of the computation complexity and space complexity without relying on any asymptotic notation. We first examine computational complexity for Algorithm 2. For readability, the remainder of this section does not use the bar notation, so instead of $\bar{x}, \bar{y}$, etc. we will use x, y, etc.

Computation Complexity. For any $z = \begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix}$, let us define $z' = \mathsf{InTake}(z) = \begin{pmatrix} z'_0 & z'_1 \\ z'_2 & z'_3 \end{pmatrix}$. Since $z'_1 = z_0 + z_1 x$ and $z'_3 = z_2 + z_3 x$, there are two multiplications required. After then, when taking ∞, there needs no multiplication and just substitute.

Similarly, during one instance of OutTake running, $z'_1 \bmod z'_3$ could support the number of MAX multiplications because we replace modular arithmetic by GT circuits. Moreover, the upper line, that is, ciphertexts z'_0 and z'_1, are not affected because there is no operation, only substitution; thus, the required depth of each component during k times OutTake is $\frac{k\mathsf{MAX}}{2}$, where k is determined by the desired accuracy.

In total, when performing Algorithm 2, the total number of multiplications for each component is $1 + \frac{n\mathsf{MAX}}{2}$, and thus HE scheme supports at least this number of multiplications.

Space Complexity. Let x be an integer and thus it has the only one partial quotient. In our suggested algorithm, the storage requires that a ciphertext corresponding to x and additional four ciphertexts corresponding to $\begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix}$. Let γ denote a bit-size of a ciphertext. Since the total number of ciphertexts are 5, the memory requires that 5γ-bit to perform our proposed algorithm.

4 Extending Our Proposal

Section 3 indicates that a method for dividing an integer by an encrypted integer. In this section, we extend our idea described in Section 3 to enable computation when rational numbers are encrypted in the form of continued fractions. We only examine a special case (the only one partial quotient) of Gosper algorithm in Section 3, but we can also apply the same idea for general case (several number of partial quotients) of Gosper algorithm.

4.1 Tuning the Gosper algorithm

A rational number has a couple of partial quotients and thus InTake should be occurred as many as the number of partial quotients. Then, according to Algorithm 1 described in Section 2.2.1, the order of OutTake and InTake is determined by the condition of whether two truncations are the same. However, it is a big burden to check the condition for each iteration, so prior to presenting the specific description of our construction, we slightly modify Gosper algorithm to reduce the complexity.

In this section, we will provide two theorems to reduce the complexity of Gosper algorithm by removing the condition of whether two rational numbers have the same integer part, and these might be very helpful for our construction.

Theorem 5. *If $\lfloor \frac{a}{c} \rfloor = \lfloor \frac{b}{d} \rfloor$, then $\lfloor \frac{a+bq}{c+dq} \rfloor = \lfloor \frac{a}{c} \rfloor = \lfloor \frac{b}{d} \rfloor$ for any $q \in \mathbb{Z}$.*

Proof. Denote $p = \lfloor \frac{a}{c} \rfloor = \lfloor \frac{b}{d} \rfloor$. Then,

$$a = cp + r_1 \text{ for } 0 \le r_1 < c$$
$$b = dp + r_2 \text{ for } 0 \le r_2 < d$$

For any $q \in \mathbb{Z}$,

$$a + bq = p(c + dq) + r_1 + r_2 q,$$

and it implies that

$$\frac{a + bq}{c + dq} = p + \frac{r_1 + r_2 q}{c + dq}$$

Because $0 \le r_1 + r_2 q < c + dq$, $\left\lfloor \frac{a+bq}{c+dq} \right\rfloor = p$. This completes the proof. □

Theorem 6. *For $\begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix}$ with $\lfloor \frac{z_0}{z_2} \rfloor = \lfloor \frac{z_1}{z_3} \rfloor$,*

$$\mathit{InTake} \circ \mathit{OutTake}(z) = \mathit{OutTake} \circ \mathit{InTake}(z).$$

Proof. Suppose that $z = \begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix}$, $q := \lfloor \frac{z_0}{z_2} \rfloor = \lfloor \frac{z_1}{z_3} \rfloor$ and the next input partial quotient is $p \in \mathbb{Z}$. First, we look at the left-hand side of the equality.

$$\mathsf{InTake} \circ \mathsf{OutTake}(z) = \mathsf{InTake}\left(\begin{pmatrix} z_2 & z_3 \\ z_0 - z_2 q & z_1 - z_3 q \end{pmatrix} \right)$$
$$= \begin{pmatrix} z_3 & z_2 + z_3 p \\ z_1 - z_3 q & (z_0 - z_3 q) + p(z_1 - z_3 q) \end{pmatrix}$$

By Theorem 5, $q = \lfloor z_1/z_3 \rfloor = \lfloor \frac{z_0+z_1p}{z_2+z_3p} \rfloor$. Thus, the right-hand side of the equality becomes

$$\mathsf{OutTake} \circ \mathsf{InTake}(z) = \mathsf{OutTake}\left(\begin{pmatrix} z_1 & z_0 + z_1 p \\ z_3 & z_2 + z_3 p \end{pmatrix} \right)$$
$$= \begin{pmatrix} z_3 & z_2 + z_3 p \\ z_1 - z_3 q & (z_0 + z_1 p) - q(z_2 + z_3 p) \end{pmatrix}.$$

Therefore, $\mathsf{InTake} \circ \mathsf{OutTake}(z) = \mathsf{OutTake} \circ \mathsf{InTake}(z)$, and thus, we may conclude the theorem. □

By Theorem 5 and Theorem 6, Gosper's algorithm can be slightly modified to performing InTake for all partial quotients of input including ∞ and then performing OutTake, not alternative. Thus, the condition of whether two rational numbers have the same integer part is no longer required.

4.2 Our Proposal

Although Gosper's algorithm supports any continued fractions despite an infinite continued fraction, we limit the input as a finite continued fraction and rational numbers. That is, for some $n \in \mathbb{N}$, a real number is approximated by the first n partial quotients of r whenever the partial quotients are produced more than n, and thus, InTake and OutTake perform each at most n times in our construction. In Algorithm **??**, we provide a modified algorithm that enables rational numbers and encrypted continued fractions to be computed.

Algorithm 3 *Our variant of Gosper's algorithm*

```
Input: $\bar{x} = [\bar{x}_0; \bar{x}_1, \ldots, \bar{x}_{n-1}]$, $f(X) = \frac{a+bX}{c+dX}$, MAX
Output: $f(\bar{x}) = [\bar{y}_0; \bar{y}_1, \ldots, \bar{y}_{n-1}]$
1.  $\begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix} \leftarrow \begin{pmatrix} a & b \\ c & d \end{pmatrix}$
2.  for $i = 0$ to $n-1$
3.      do $\begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix} \leftarrow \begin{pmatrix} z_1 & z_0 + z_1\bar{x}_i \\ z_3 & z_2 + z_3\bar{x}_i \end{pmatrix}$
4.  $\begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix} \leftarrow \begin{pmatrix} z_1 & z_1 \\ z_3 & z_3 \end{pmatrix}$
5.  for $i = 0$ to $n-1$ /* OutTake */
6.          $y_i \leftarrow 0$
7.      do $tmp \leftarrow z_0$
8.          for $j = 0$ to MAX
9.              do $\bar{t} \leftarrow \mathsf{GT}(tmp, z_2)$
10.                 $\bar{y}_i \leftarrow \bar{y}_i + \bar{t}$
11.                 $tmp \leftarrow tmp - \bar{t} \cdot z_2$
12.         $\begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix} \leftarrow \begin{pmatrix} z_3 & z_3 \\ tmp & tmp \end{pmatrix}$
13. return $[\bar{y}_0; \bar{y}_1, \ldots, \bar{y}_{n-1}]$
```

4.3 Analysis

Similar to Section 3.3, we also analyze the performance of Algorithm 3. The correctness and security are almost the same as before, so we omit the proof in this section. Further, we compare our suggestion to additive homomorphic encryptions (AHE).

Complexity. We provides an analysis of the computation complexity and space complexity without relying on any asymptotic notation. We first examine computational complexity for Algorithm 2. For readability, the remainder of this section does not use the bar notation, so instead of $\bar{x}, \bar{y}$, etc. we will use x, y, etc.

Computation Complexity. For any z, let us define $z' = \mathsf{InTake}(z) = \begin{pmatrix} z'_0 & z'_1 \\ z'_2 & z'_3 \end{pmatrix}$. Because z'_1 and z'_3 are the output of multiplications and $z_0 + z_1 x$ and $z_2 + z_3 x$ are stored during the next InTake in z'_1 and z'_3 positions, respectively, no multiplications are required when z'_0 and z'_2 are stored. We would like to emphasize that no multiplications are required when an input of InTake is ∞. Hence, if z is the production after performing InTake $n+1$ times including input as ∞, the left side of z, that is, ciphertexts z'_0 and z'_2, should be supported by at least $n-1$ multiplications, and the right side of z, that is, ciphertexts z'_1 and z'_3, should be supported by n multiplications.

Since the computational complexity of OutTake is the same as before, the required total number of multiplications for each component is $n + \frac{n\mathsf{MAX}}{2}$, Hence, an FHE scheme supports at least this number of multiplications to run our construction of the algorithm.

Space Complexity. For a rational number x, suppose x has n partial quotients. Through Gosper algorithm, the storage requires that n partial quotients and additional four integers corresponding to $\begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix}$. In the same way, in our suggested algorithm, it requires n ciphertexts corresponding to n partial quotients and an additional four ciphertexts corresponding to $\begin{pmatrix} z_0 & z_1 \\ z_2 & z_3 \end{pmatrix}$. Let γ denote a bit-size of one ciphertext. Since $4+n$ ciphertexts are used, the memory requires that $(4+n)\gamma$-bit to perform Algorithm 3.

5 Experiments

We implemented Algorithm 3 in order to prove that it works well. Since the number of partial quotients are flexible depending on an input, we made experiments by varying the number of partial quotients by changing input rational numbers. Firstly, we give a specific implementation environment and concrete parameters for our underlying FHE scheme.

Test Settinvgs. Our implementation environment is that Intel Core i7 2.3 GHz and 16 GB RAM under Mac OS X 10.11. The main construction is to apply FHE to Gosper algorithm using our idea demonstrated in Section 4.1. To our knowledge, because HElib is the most efficient and credible FHE open source and it also supports scalar addition, we took HElib as a basic tool for instantiating the BGV-type FHE scheme.

FHE cryptosystem parameters. To initialize the BGV cryptosystem via HElib, we need to choose a security parameter λ, a multiplicative depth L, and a message space $\mathbb{Z}_p$. Then HElib determines the N-th cyclotomic polynomial $\Phi_N(X)$ where $N \in \mathbb{Z}$. In Table 1 we provide a full set of parameters for our implementation.

Table 1. System parameters ($\lambda = 80$)

Message space	L	N
$\mathbb{Z}_{2^{10}}$	20	18632

To use the GT circuit which homomorphically compares two encryptions, we need to first encode a rational number x into its CF representation $[x_0; x_1, \ldots, x_n]$ and then represent each integer x_i into a sequence of bits. For this reason, we take the plaintext modulus $p = 2$ and thus, the message space becomes $\mathbb{Z}_2$. In addition, considering a specific calculation as in the toy example of Section 2.2.1, we set a multiplicative depth $L = 20$. Then the value N is automatically determined by a HElib routine; we have $N = 18632$ as in Table 1.

5.1 Implementation Issues

Our implementation of Algorithm 3 mainly relies on two sub-modules. One is a greater-than comparison circuit and the other is an integer addition circuit of size $\tilde{n}$ for an integer $\tilde{n} > n$. The main reason for considering such an integer addition is to support line 10 and line 11 of Algorithm 3. We defer the details of their efficient constructions to Appendix A.1 and Appendix A.2, respectively.

Precisely, an implementation of line 11 of Algorithm 3 does not require an integer addition but requires an integer subtraction. To do this, we applied a simple technique to support subtraction over encryptions. Let denote by $\mathsf{FA}(\bar{x}, \bar{y})$ a full adder over two encryptions $\bar{x}, \bar{y}$. Our subtraction was implemented as follows:

1. Compute the two's complement of $\bar{y}$ by xoring all $\bar{y}_i$ and adding 1 to the result. Specifically, compute $\bar{y}_i = \bar{y}_i + 1$ for all $0 \le i \ne n-1$ and $\bar{y} = \mathsf{FA}(\bar{1}, \bar{y})$. Here $\bar{1}$ means an encryption of $(0 \cdots 01)$.
2. Compute $\bar{z} = \mathsf{FA}(\bar{x}, \bar{y})$.

The reason for our technique working correctly is that the BGV encryption supports add a constant to an encryption.

5.2 Results

We give an experiment result for our simple implementation without depending on optimization techniques to speed up and applying SIMD techniques for circuit constructions such as the GT and FA circuits.

Our construction consists of largely two parts; one is InTake and the other is OutTake. As mentioned above, we implemented $x + 1/2$ where $x = 13/11 = [1; 5, 2]$. In other words, three InTake steps occurs. (Indeed, four InTake occurs including $x_3 = \infty$, but the last InTake just substitutes.) Moreover, several times OutTake occurs, but nobody knows the exact number of OutTake because every partial quotients are encrypted. We assume that we need at most five partial quotients of outputs, which means that we ignore every partial quotients behind the sixth quotient. Hence, we invoked OutTake only five times in our implementation.

Table 2 shows the running times of our main submodules. We presents three types of the running times; the average time of InTake and OutTake and the total time depending on the number of input partial quotients, denoted by n.

Table 2. Experiment Results

n	InTake	OutTake	Total Time
3	315.75 s	656.51 s	4,229.8 s
5	289.13 s	641.35 s	4,651.82 s

We would like to note that the primary goal of our experiments is to provide a PoC implementation for proving correctness of our idea. Therefore, there is considerable room for improvement in performance through known optimization techniques including SIMD techniques and computations on GPU.

6 Related Work

As closely related works, we have Graepel et al.'s result [14] and Bos et al.'s work [2]. In both works, the authors first fix a desired precision, multiply through by a fixed denominator, and round to the nearest integer because any real number can be approximated by rational numbers to arbitrary numbers and subsequently encoded to ring elements. However, this approach to represent real numbers has some drawbacks. When two encoded rational numbers are multiplied, it should be performed without any modular reduction and thus the plaintext space of HE must be sufficiently large.

7 Concluding Remark

In this paper, we proposed an arithmetic algorithm that enables a constant divided by an encrypted integer using a special case of Gosper algorithm. Our algorithm outputs a rational number in form of continued fractions form. Continued fractions are a great tool for representing rational numbers to a sequence of small integers and they are a best approximation of rational numbers.

Further, we can extend our suggestion to an arithmetic with a rational number. With our extending algorithm, addition, subtraction, and multiplication for encrypted rational numbers are possible, and linear fractional transformation also can be evaluated and the complexity is almost the same because every step is the same for any arithmetic.

We leave it as future work to provide a new version of the current implementation to which we apply SIMD techniques to construct underlying circuits and further optimization techniques.

References

[1] D. Boneh, E.-J. Goh, and K. Nissim. Evaluating 2-DNF formulas on ciphertexts. In *TCC*, pages 325–341, 2005. 3

[2] J. Bos, K. Lauter, and M. Naehrig. Private predictive analysis on encrypted medical data. *Journal of Biomedical Informatics*, 50:234–243, 2014. 1, 13

[3] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Advances in Cryptology–Crypto*, pages 868–886, 2012. 7

[4] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012. 2, 7

[5] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, pages 97–106, 2011.

[6] Z. Brakerski and V. Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *Advances in Cryptology–Crypto*, pages 505–524, 2011. 7

[7] J. H. Cheon, J. Coron, J. Kim, M. S. Lee, T. Lepoint, M. Tibouchi, and A. Yun. Batch fully homomorphic encryption over the integers. In *Advances in Cryptology–Eurocrypt*, pages 315–335, 2013. 7

[8] J. H. Cheon, M. Kim, and M. Kim. Search-and-compute on encrypted data. In *WAHC*, LNCS 8976, pages 1–18, 2015. 8, 14

[9] J. H. Cheon and D. Stehlé. Fully homomophic encryption over the integers: Revisited. In *Advances in Cryptology–Eurocrypt*, pages 513–536, 2015. 7

[10] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and D. Chaum, editors, *Advances in Cryptology-Crypto*, LNCS 196, pages 10–18, 1984. 2

[11] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009. 6, 7

[12] C. Gentry, S. Halevi, and N. Smart. Homomorphic evaluation of the AES circuit. In *Advances in Cryptology–Crypto*, pages 850–867, 2012. 7

[13] R. Gosper. Continued fraction arithmetic. In *HAKMEM Item 101B, MIT Artificial Intelligence Memo 239*, 1972. 3, 5

[14] T. Graepel, K. Lauter, and M. Naehrig. ML confidential: Machine learning on encrypted data. In *ICISC*, volume 7839, pages 1–21, 2013. 13

[15] G. Hardy and E. Wright. *An Introduction to the Theory of Numbers*. Oxford science publications. Clarendon Press, 1979. 5

[16] J. Lagarias and J. Shallit. Linear fractional transformations of continued fractions with bounded partial quotients. *Journal de théorie des nombres de Bordeaux*, 9(2):267–279, 1997. 8

[17] NIST. SHA-3 standard: Permutation-based hash and extendable-output functions. In *FIPS 202*, 2015. 6

[18] R. Ostrovsky, A. Paskin-Cherniavsky, and B. Paskin-Cherniavsky. Maliciously circuit-private FHE. In *Advances in Cryptology–Crypto*, pages 536–553, 2014. 7

[19] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Advances in Cryptology-EuroCrypt*, LNCS 1592, pages 223–238, 1999. 2

[20] M. van Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan. Fully homomorphic encryption over the integers. In *Advances in Cryptology–Eurocrypt*, pages 24–43, 2010. 7

A Circuit Construction

We provide a brief description of each circuit construction for our construction. The below two circuits are used in replacing modular reduction with cheaper operations in terms of multiplicative depth.

A.1 Greater-than Comparison

We provide a brief description of an approach to construct an efficient greater-than circuit in depth.

For two n-bit integers, a greater-than circuit $\mathsf{GT}(\bar{x}, \bar{y})$ outputs $\bar{1}$ if $x \geq y$ and $\bar{0}$ otherwise. This operation can be recursively defined as follows:

$$\mathsf{GT}(\bar{x}, \bar{y}) = 1 - \bar{c}_{n-1},$$

where $\bar{c}_i = (1 + \bar{x}_i) \cdot \bar{y}_i + (1 + \bar{x}_i + \bar{y}_i) \cdot \bar{c}_{i-1}$ for $i \geq 1$ with an initial value $\bar{c}_0 = (1 + \bar{x}_0) \cdot \bar{y}_0$.

The naïve construction of this circuit incurs $O(n^2)$ homomorphic multiplications. However, using an FHE scheme such as the BGV scheme that supports the single-instruction multiple-data (SIMD) technique enables us to construct a circuit that requires only $2n - 2$ homomorphic multiplications. This means that the circuit has multiplicative depth $\lceil \log n \rceil + 1$. See reference [8] for the details of construction and its analysis.

A.2 Full Adder

We briefly describe an approach to construct an efficient full-adder circuit in depth. As above, let x and y be n-bit integers. We then obtain two $\tilde{n}$-bit integers by padding zeros on the left for an integer $\tilde{n} > n$. We define a full-adder of size $\tilde{n}$, denoted by FA, in a recursive manner as follows:

$$\mathsf{FA}(\bar{x}, \bar{y}) = (\bar{z}_0, \bar{z}_1, \ldots, \bar{z}_{\tilde{n}-1})$$

where a sum $\bar{z}_i = \bar{x}_i + \bar{y}_i + \bar{c}_{i-1}$ and a carry-out $\bar{c}_i = (\bar{x}_i \cdot \bar{y}_i) + ((\bar{x}_i + \bar{y}_i) \cdot \bar{c}_{i-1})$ for $1 \leq i \in \tilde{n} - 1$ with initial values $\bar{z}_0 = \bar{x}_0 + \bar{y}_0$ and $\bar{c}_0 = \bar{x}_0 \cdot \bar{y}_0$. The more important thing with respect to efficiency is that using the SIMD technique allows us to construct the integer addition circuit of depth $\lceil \log(\tilde{n} - 2) \rceil + 1$ [8].

B Semantic Security

We define the security game for FHE between a challenger C and an adversary A. In the security game the adversary A is allowed to adaptively choose the plaintext on which it will be challenged. Precisely, the security of FHE is defined as follows:

Definition 4 (Semantic Security Game)**.** *Consider the following game played by an adversary A and a challenger C:*

1. *The challenger C runs $\mathsf{Kg}(1^\lambda)$ and obtains a secret key sk for FHE.*
2. *The adversary determines the number of encryption queries adaptively. In each encryption query, the adversary A sends x_i and receives its encryption $\bar{x}_i$.*
3. *Once the adversary A decides that query is over, A outputs two equal length plaintext x_0^* and x_1^* on which it wishes to be challenged. The only restriction is that x_0^* and x_1^* has never been queried before.*
4. *The challenger C picks $b \in \{0, 1\}$ randomly and encrypts m_b under the secret key sk. It then sends the ciphertext $\bar{x}_b$ to the adversary A.*
5. *The adversary A outputs his guess $b' \in \{0, 1\}$ of which message he received and wins if $b = b'$.*

Now, we define the advantage of an adversary A playing the security game by $Adv_A(\lambda) := \left|\Pr[b = b'] - \frac{1}{2}\right|$.

Definition 5 (Semantic Security)**.** *We say that FHE is* semantically secure *if no polynomially-bounded adversary has a non-negligible advantage in the above game.*