Floating-Point Homomorphic Encryption

Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song

Department of Mathematical Sciences, Seoul National University,
Republic of Korea

{jhcheon, kimandrik, alfks500, lucius05}@snu.ac.kr

Abstract. Our paper suggests a general method to construct a Floating-Point Homomorphic Encryption (FPHE) scheme that allows the floating-point arithmetics of ciphertexts, thus computing encryptions of most significant bits of $m_1 + m_2$ and $m_1m_2$, given encryptions of floating-point numbers $m_1$ and $m_2$. Our concrete construction of leveled FPHE based on BGV scheme is almost optimal in the sense of noise growth and precision loss. More precisely, given encryptions of $d$ messages with $\eta$ bits of precision, our scheme of depth $\lceil \log d \rceil$ securely computes their product with $\eta - \lceil \log d \rceil$ bits of precision similarly to the case of unencrypted floating-point computation. The required bit size of the largest modulus grows linearly in the depth. We also describe algorithms for evaluating some floating-point arithmetic circuits containing polynomial, multiplicative inverse, and even exponential function, and analyze their complexities and output precisions. With the security parameter $\lambda = 80$, our rudimentary implementation takes 315ms and 168ms to compute a product of 16 ciphertexts and a multiplicative inverse of a ciphertext, respectively, when given ciphertexts have 20 bits of precision.

Keywords. Homomorphic Encryption, Floating-point Arithmetic, BGV Scheme, Somewhat Homomorphic Encryption

1 Introduction

The floating-point is the most common way to approximately represent real numbers in computers, and has been widely used in computer system and intensive scientific computations. A floating-point number is represented in terms of four integers as $x = \pm s \cdot b^{-k}$, where $b$ is the base, $e$ is the exponent, $k$ is the precision (the number of significant digits in significand), and $s$ is the significand satisfying $0 \leq s \leq b^k - 1$. The floating-point arithmetic consists of the usual arithmetic and rounding step to the significand. It allows us to represent real numbers using fixed number of bits, and supports a trade-off between range (size of representation) and precision.

Homomorphic encryption (HE) is a cryptographic scheme that allows homomorphic operations on encrypted data without decryption. Since Gentry discovered the first plausible construction of fully homomorphic encryption scheme [Gen09], many other HE schemes have been suggested following Gentry’s blueprint (e.g., [DGHV10, BV11a, BV11b, Bra12, BGV12, GSW13, CLT14, CS15, DM15]). Most of existing schemes encode the messages into a fixed modulus space, a product of several modulus spaces, or a polynomial ring over modulus space. However, HE schemes have not been used in many practical applications because of their inefficiency of real number computations, especially floating-point arithmetic.

Floating-Point Homomorphic Encryption. In this paper, we propose a general idea to construct a Floating-Point Homomorphic Encryption (FPHE) scheme supporting floating-point addition and multiplication of ciphertexts which output encryptions of some most significant bits (MSBs) of $m_1 + m_2$ and $m_1m_2$ respectively, upon input encryptions of two messages $m_1$ and $m_2$. Our concrete construction of a leveled FPHE scheme comes from the classical LWE-based scheme [BGV12] by modifying some operations such as modulus-switching procedure.
First of all, we place an encryption noise in the rightmost position together with a plaintext as in Figure 1, i.e., an encryption $c$ of $m$ encrypted by $s$ satisfies $\langle c, s \rangle = m + e \pmod{q}$ for some small $e$. We cannot recover the exact value $m$ from the ciphertext $c$, but the inserted noise $e$ can be considered as an error occurred during floating-point computations. This approximate value $m + e$ may substitute the original message in floating-point arithmetics with almost the same precision when $|e|$ is sufficiently small. If the input data is too small or large, then one may encode and regulate the magnitude of message by multiplying a scaling factor, and store the scaling factor (or exponent) independently. The homomorphic operations of our scheme output the ciphertexts $c_{\text{add}}$ and $c_{\text{mult}}$ satisfying $\langle c_{\text{add}}, s \rangle \pmod{q} \approx m_1 + m_2$ and $\langle c_{\text{mult}}, s \rangle \pmod{q} \approx m_1 m_2$. We use the key-switching technique [BV11a,BGV12] for multiplication procedure.

The modulus-switching technique was suggested in [BV11a,BGV12] to reduce the magnitude of inserted error, but it has a completely different role in our scheme. It rounds up some least significant bits (LSBs) of message to maintain the size of significand after homomorphic operations. The composition of this modulus-switching procedure and homomorphic operation (as in Figure 1) mimics the ordinary (unencrypted) floating-point computations. Our scheme is almost optimal in the sense of precision: precision of resulting message is lost by about one bit similar to the case of unencrypted floating-point arithmetic.

**Fig. 1. Ciphertext after a homomorphic multiplication**

**FPHE Arithmetics.** Based on our construction of FPHE, we homomorphically evaluate some typical circuits containing (real) constant addition/multiplication, product, polynomial, and multiplicative inverse. We describe algorithms to minimize the circuit depth and complexity, and analyze their noise and precision loss during homomorphic evaluation.

In our scheme, every integer is a trivial encryption of itself without error. Hence, the constant addition procedure almost does not change the absolute error of the ciphertext (fraction part can be added to noise). In addition, the constant multiplication process with suitable choice of scaling almost does not change the relative error of the input ciphertext. Given encryptions of messages $m_1, \ldots, m_d$ with $\eta$ bits of precision, our leveled FPHE scheme of depth $\lceil \log d \rceil$ computes their product with $(\eta - \lceil \log d \rceil)$ bits of precision in $(d - 1)$ homomorphic multiplications of ciphertexts. The magnitude of significands is invariably maintained, so the required bit size of the largest modulus is linear in depth $\lceil \log d \rceil$. For example, given 16 ciphertexts with 20 bits of precision, we can compute their product with 15 bits of precision in about 315ms. We get a similar result for the evaluation of polynomials, and extend the case to analytic functions such as exponential and logistic functions using Taylor decomposition.
Table 1. Summary of FPHE arithmetics

<table>
<thead>
<tr>
<th>Function</th>
<th>Bit precision of output</th>
<th>Depth</th>
<th>#((HM))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant addition</td>
<td>$a + x$</td>
<td>$\geq \eta$</td>
<td>0</td>
</tr>
<tr>
<td>Constant multiplication</td>
<td>$a \cdot x$</td>
<td>$\eta$</td>
<td>0</td>
</tr>
<tr>
<td>Product</td>
<td>$\prod_{i=1}^{d} x_i$</td>
<td>$\eta - \lceil \log d \rceil$</td>
<td>$\lceil \log d \rceil$</td>
</tr>
<tr>
<td>Power function</td>
<td>$x^d$</td>
<td>$\eta - \lceil \log d \rceil$</td>
<td>$\lceil \log d \rceil$</td>
</tr>
<tr>
<td>Polynomial</td>
<td>$\sum_{i=0}^{d} a_i x^i$</td>
<td>$\eta - \lceil \log d \rceil$</td>
<td>$\lceil \log d \rceil$</td>
</tr>
<tr>
<td>Multiplicative inverse</td>
<td>$x^{-1}$</td>
<td>$\eta - 1$</td>
<td>$\lceil \log \eta \rceil$</td>
</tr>
<tr>
<td>Exponential</td>
<td>$e^x$</td>
<td>$\eta - \lceil \log \eta \rceil$</td>
<td>$\lceil \log \eta \rceil$</td>
</tr>
</tbody>
</table>

$\eta$: bit precision of inputs
HM: homomorphic multiplications of ciphertexts

Let $\bar{x} = 1 - x$ for a given real number $x$ with $1/2 \leq x \leq 1$. From the equality $x(1 + \bar{x})(1 + \bar{x}^2) \cdots (1 + \bar{x}^{2^k - 1}) = (1 - \bar{x}^2)(1 + \bar{x}^2) \cdots (1 + \bar{x}^{2^k - 1})$ with a precision of $2^k$ bits. We transform this equality to securely compute the multiplicative inverse. More precisely, given a ciphertext with $\eta$ bits of precision, our FPHE scheme of depth $\log \eta$ can compute its multiplicative inverse with $(\eta - 1)$ bits of precision in $2 \lceil \log \eta \rceil$ multiplications of ciphertexts. For example, upon inputs the 20 bits of precision, one can compute an encryption of its multiplicative inverse with 19 bits of precision with only 8 multiplications of FPHE scheme of depth 5, and our implementation with the security parameter $\lambda = 80$ takes about 168ms.

**Related Works.** There have been a substantial number of studies concerned with processing real numbers over encryption. The obvious method is to scale them to integers, but a plaintext modulus is exponential in the length of message. For example, Bos et al. [BLN14] described how to privately predict the likelihood of having a heart attack by evaluating the logistic regression function with scaled input data. On the other hand, Dowlin et al. [DGL+15] presented an efficient method to represent fixed-point numbers – it is encoded as a polynomial with coefficients in the range $[-(B - 1)/2, (B - 1)/2]$ using its base-$B$ representation for an odd integer $B \geq 3$. However, when the number of significant bits is large, it results in large degree of the polynomial ring.

On the other hand, division is an essential operation for real-world applications such as the medical, financial, advertising domains, and machine learning algorithms. Algesheimer et al. [ACS02] introduced a problem of solving a secure integer division such that given two encryptions of $a$ and $b$, determine an encryption of $\lfloor a/b \rfloor$ without leaking any information about inputs. Their solution was based on Newton iteration using secure multiparty computation protocol. The protocols were later implemented by Jakobsen in the passively secure three-party setting [Jak06]. In [DNT12], they proposed secure computation protocols to solve the problem in a two-party setting. Their computation cost is tractable, but the protocols still require a lot of interactions. Recently, Cetin et al. [cDSM15] described new ideas for comparison and homomorphic division given word-based encrypted data. They found approaches by using approximation techniques, but there was no accurate explanation of the precision for the approximation.

**Road-map.** Section 2 briefly introduces notations and some preliminaries about floating-point arithmetics and the LWE problem. Then Section 3 presents a floating-point homomorphic encryption scheme and analyze the noise growth during basic homomorphic operations. In Section 4, we suggest some algorithms to evaluate typical circuits homomorphically, and compute the precision of outputs. Finally, in Section 5 we provide an implementation and performance of our scheme.
2 Preliminaries

Notation. All logarithms are base 2 unless otherwise indicated. We denote vectors in bold, e.g., \( \mathbf{a} \), and matrices in upper-case bold, e.g., \( \mathbf{A} \). Every vector in this paper is a column vector. We denote by \( \langle \cdot, \cdot \rangle \) the usual dot product of two vectors. For a real number \( r \), \( \lfloor r \rfloor \) denotes the nearest integer to \( r \), rounding upwards in case of a tie.

For an integer \( q \), we use \( \mathbb{Z} \cap (-q/2, q/2] \) as a representative of \( \mathbb{Z}_q \). We use \( x \leftarrow D \) to denote the sampling \( x \) according to distribution \( D \). It denotes the uniform sampling when \( D \) is a finite set. For a set \( S \), \( U(S) \) denotes the uniform distribution on \( S \).

Throughout the paper, we let \( \lambda \) denote the security parameter: all known valid attacks against the cryptographic scheme under scope should take \( \Omega(2^\lambda) \) bit operations.

2.1 Floating-Point Arithmetic

Floating-point is the formulaic representation that approximates a real number so as to support a trade-off between range and precision. A number is, in general, represented approximately to a fixed number of significant digits (significand), so real numbers can be approximated with the finite word length in a computer. The term floating refers to the fact that a number’s radix point can float: it can be placed anywhere relative to the significant digits of the number. More precisely, a floating-point number \( x \) can be represented in terms of four integers as \( x = \pm s \cdot b^{\epsilon-k} \), where \( b \) is the base or radix, \( \epsilon \) is the exponent, \( k \) is the precision (the number of significant digits in significand), and \( s \) is the significand satisfying \( 0 \leq s \leq b^k-1 \).

Let \( x^* \) be a real number. We use the notation \( x = \text{fl}(x^*) \) to represent the floating-point value of \( x^* \) (nearest number in the floating-point system). The most useful measures of the accuracy of \( x \) are its absolute error \( E_{\text{abs}}(x) = |x - x^*| \) and its relative error \( E_{\text{rel}}(x) = |x - x^*|/|x^*| \). To determine accuracy, we define the quantity \( u = \frac{1}{2} \cdot b^{1-k} \), called the unit round-off. It is the furthest distance relative to unity between a real number and the nearest floating-point number. It bounds the relative error in representing as \( x = x^*(1 + \epsilon) \) such that \( E_{\text{rel}}(x) = |\epsilon| \leq u \) \( [\text{Hig02}] \).

A simple method to add (resp. subtract) floating-point numbers is to first proceed with the usual addition (resp. subtraction). Then it is rounded to a precision of the same bits, which makes round-off error. To multiply, the significands are multiplied while the exponents are added, and then the result is rounded. For example, if we are given two floating-point numbers \( x = 1101_{(2)} \cdot 2^2 \) and \( y = 1001_{(2)} \cdot 2^{-3} \) with 4 bits of precision, we first get the true product value \( x \cdot y = 1110101_{(2)} \cdot 2^{-1} \) and then it is rounded to \( \text{fl}(x \cdot y) = 1111_{(2)} \cdot 2^{0} \) with 4 bits of precision.

Small errors may accumulate as floating-point operations are performed in succession. Dealing with these errors incurred in the floating-point operations is one of the main topics in error analysis. In particular, we focus on the forward errors which are concerned with how close the computed result is to the exact value of an algorithm. For more explicit overview, we recommend to refer to \([\text{Gol91, Hig02, Ein05}]\). Here is the standard model of floating-point arithmetic, introduced by Wilkinson \([\text{Wil61}]\): if \( x \) and \( y \) are floating-point numbers, then \( \text{fl}(x \cdot y) = (x \cdot y)(1 + \epsilon) \) for some \( |\epsilon| \leq u \), where \( \ast \equiv +, -, \cdot, / \).

2.2 Learning With Errors (LWE)

The LWE problem was introduced by Regev as a generalization of learning parity with noise \([\text{Reg05}]\). For positive integers \( n \), and \( q \geq 2 \), \( t \in \mathbb{Z}_q^n \) and a distribution \( \chi \) over \( \mathbb{Z} \), we define \( A_{q,\chi}(t) \) as the distribution obtained by sampling \( a \leftarrow \mathbb{Z}_q^n \) and \( e \leftarrow \chi \), and returning \( c = (\langle a, t \rangle + e, a) \in \mathbb{Z}_q \times \mathbb{Z}_q^n \).

Definition 1 (LWE). Let \( D \) be a distribution on \( \mathbb{Z}_q^n \). The (decision) learning with errors problem, denoted by \( \text{LWE}_{n,q,\chi}(D) \), is to distinguish arbitrarily many independent samples chosen according to \( A_{q,\chi}(t) \) for a fixed \( t \leftarrow D \), from the uniform distribution over \( \mathbb{Z}_q \times \mathbb{Z}_q^n \).
The \text{LWE} problem is self-reducible, that is, \text{LWE}_{n,q,\chi}(D) can be reduced to \text{LWE}_{n,q,\chi}(U \langle Z_q^n \rangle)$ for any distribution $D$. Moreover, one can efficiently reduce $\text{LWE}_{n,q,\chi}(U \langle Z_q^n \rangle)$ to $\text{LWE}_{n,q,\chi}(\chi)$ when $\chi$ is a Gaussian distribution with an appropriate parameter [ACPS09].

It was shown that the hardness of LWE can be established by a quantum reduction to approximate short vector problems in ideal lattices [Reg05]. Peikert also showed there is a classical reduction between LWE and worst-case lattice problems [Pei09]. We refer the readers to [Reg05,Pei09] for these results in more details.

\section{Floating-Point Homomorphic Encryption}

In this section, we describe a new leveled homomorphic encryption scheme based on the hardness of LWE problem which allows to carry out floating-point computations of ciphertexts. Given encryptions of $m_1$ and $m_2$, the floating-point addition (resp. multiplication) outputs an encryption of some MSBs of $m_1 + m_2$ (resp. $m_1 m_2$) upon input encryptions of two integers $m_1$ and $m_2$. It can be applied to compute an approximate value of product of integers, multiplicative inverse, and power series. Our concrete construction in this section is based on the BGV scheme [BGV12], but this idea can be adapted to other (Ring) LWE-based HE schemes such as [GHS12b].

\subsection{Basic Encryption Scheme}

We first describe a basic public-key homomorphic encryption scheme in this subsection. This scheme is IND-CPA secure under the assumption that $(n, q, \chi)$-LWE problem is hard. The scheme consists of the following algorithms.

- \textit{E.Setup}(1^\lambda). Take a modulus $q = q(\lambda)$. Choose the parameters $n = n(\lambda, q)$ and a $B_{\text{err}}$-bounded error distribution $\chi = \chi(\lambda, q)$ appropriately for \text{LWE}_{n,q,\chi} that achieves at least $2^\lambda$ security. Take $\tau = 2(n + 1)[\log q]$. Output the parameters $\text{params} = (n, q, \chi, \tau)$.
- \textit{E.SecretKeyGen}(\text{params}). Sample a vector $t \leftarrow \chi^n$ and set the secret key $s \leftarrow (1, -t) \in \mathbb{Z}_q \times \mathbb{Z}_q^n$.
- \textit{E.PublicKeyGen}(\text{params}, sk). Generate and output a $(n + 1) \times \tau$ matrix $pk = A \leftarrow A^{\text{LWE}}_{q,\chi}(t)^\tau$
  (Each column of $A$ is sampled independently from the distribution $A^{\text{LWE}}_{q,\chi}(t)$).
- \textit{E.Enc}_{pk}(m)$. Given a message $m \in \mathbb{Z}$, sample a vector $r \leftarrow \{0, 1\}^\tau$ and let $m' \leftarrow (m, 0, \ldots, 0)$. Output $c \leftarrow m + A \cdot r \mod q$.
- \textit{E.Dec}_{sk}(c). Output $m' \leftarrow (c, s) \mod q$.

Apart from the most of existing schemes (\textit{e.g.}, [Bra12,GHS12b,BLLN13]), our scheme does not have a fixed or separate plaintext space from an inserted error. This floating-point encryption scheme may seem strange, because the output $m' = m + e$ of its decryption circuit is slightly different from the original message $m$. However, they can be considered to be \textit{actually the same} in floating-point computation sense if $e$ is small enough. More precisely, if the input message $m$ has a precision of $\eta$ bits to its true value $m^*$ and $|e| \leq 2^{-\eta}|m^*|$, then $m'$ is also an approximation to $m^*$ with $(\eta - 1)$ bits of precision.

When the input data is too small or large, we should encode the message and regulate its magnitude before encryption procedure to maintain the precision. For example, in order to encrypt the floating-point number $x = 1001\cdot2^{-3}$ with 4 bits of precision, one may move the radix point (multiplied by the constant $2^3$) and use the scaled value $m = 100100_2$ as an input message of encryption procedure. If the encryption error $e$ bounded by $|e| \leq 2^2$, then $m' = m + e$ is an approximation of $x \cdot 2^5$ with 9 bits of precision. The scaling factors of input data should be stored and managed independently for correct understanding of computing results.
This notion of approximate encryption has been partially used in previous work, for example, additional information for multiplication in [BV11a,Bra12,BGV12,CS15] and intermediate values in squashed decryption circuit in [DGHV10,CLT14] are encrypted in the same way. Note that these ciphertexts were used in approximate computations allowing small errors.

For homomorphic operations, one should take a larger parameter depending on the circuit complexity and also the size of input messages. We analyze the suitable parameters for various circuits in Section 4.

**Security.** The security of basic encryption scheme is directly obtained from the security proof of circuits in Section 4.

### 3.2 Leveled Floating-Point Homomorphic Encryption Scheme

We start by adapting some notations from [BGV12] to our context and recalling the key switching leveled floating-point HE scheme that will be described in next subsection.

Let $q$ be a positive integer. Given a vector $x \in \mathbb{Z}_q^n$, its bit decomposition and power of two are defined by $\text{BD}(x) = (u_0, \ldots, u_{\log q}) \in \{0,1\}^{n(1+\log q)}$ with $x = \sum_{i=0}^{\log q} 2^i u_i$, and $P_2(x) = (x, \ldots, 2^{\log q} x)$. Then we can see that $(\text{BD}(x), P_2(y)) = (x, y)$. We also recall the definition of tensor product $u \otimes v = (u_1 v_1, u_1 v_2, \ldots, u_m v_1, \ldots, u_m v_m)$ on the vector space $\mathbb{R}^n \times \mathbb{R}^m$, and its relation with the inner product $\langle u \otimes v, u' \otimes v' \rangle = \langle u, u' \rangle \cdot \langle v, v' \rangle$.

For $i = 1, 2$, let $c_i$ be ciphertexts of $m_i$ encrypted by the secret $s \in \mathbb{Z}_q \times \mathbb{Z}_q^n$. They satisfy $\langle c_i, s \rangle = m_i + e_i \mod q$ for some small $e_i$’s. Consider the quadratic equation $Q_{c_1,c_2}(x) := \langle c_1, x \rangle \cdot \langle c_2, x \rangle = \langle c_1 \otimes c_2, x \otimes x \rangle$. If $m_i' = m_i + e_i$’s are small enough, we obtain $Q_{c_1,c_2}(s) \mod q = m_1 m_2' = m_1 m_2 + e$ for some small $e$, as desired, and $c \leftarrow c_1 \otimes c_2$ is an encryption of $m_1 m_2$ with the secret $s \otimes s$. However, this comes at a cost of increasing the dimension of ciphertexts and the complexity of homomorphic multiplication. Brakerski and Vaikuntanathan [BV11a] use a key-switching technique to convert a LWE ciphertext under the long secret key $s \otimes s$ into another LWE ciphertext of the same plaintext under a different secret key $\tilde{s}$ of smaller dimension. Roughly speaking, this process publishes auxiliary information $s \otimes s = (s_i')_i$ encrypted by a new secret $\tilde{s}$, and converts a ciphertext $c = (c_i)_i$ into a new ciphertext $c' \leftarrow \sum_i c_i \cdot \text{Enc}(s_i')$. Hence, we need a chain of $L$ secret keys in order to evaluate up to $L$ multiplicative levels while keeping the same ciphertext size. In this paper, we assume the circular security for convenience, so it is safe to encrypt the leveled HE secret key under its own public key. This allows us to use the same secret key for all the levels.

Brakerski, Gentry and Vaikuntanathan [BV11a,BGV12] developed the modulus-switching technique to manage the size of errors in LWE-based HE schemes. It converts a ciphertext $c$ modulo $q$ with an error $e$ into a ciphertext $c'$ of the same message in a new modulus $q'$ and an error $e' \approx \frac{2}{q} e$. The ratio $e/q$ of the noise to the modulus size maintains almost the same, but it reduces the magnitude of the noise and the error growth in the next operation. As a result, the bit-size of largest modulus grows linearly with the multiplicative depth instead of exponentially. We apply the modulus-switching technique to floating-point HE but for a completely different purpose. Roughly speaking, we scale $c$ by a factor $q'/q$, and then round appropriately to get back an integer ciphertext to discard the LSBs (erroneous part) of messages and control the magnitude of significands.

- **FP.Setup**(1$^\lambda$,1$L$). Take a base modulus $p = p(\lambda, L)$. Let $q_\ell = p_\ell$ for $\ell = 1, \ldots, L$. Choose the parameters $n = n(\lambda, q_\ell)$ and a $B_{err}$-bounded error distribution $\chi = \chi(\lambda, q_\ell)$ appropriately for $(n, q_\ell, \chi)$-LWE problem that achieves at least $2^\lambda$ security. Let $\tau = 2(n+1)\lceil \log q_\ell \rceil$. Output the parameters params = $(n, q_\ell, \chi, \tau)$.
- **FP.SecretKeyGen**(params). Run $sk = s \leftarrow \text{E.SecretKeyGen}(\text{params})$.
- **FP.PublicKeyGen**(params, sk). Run $pk \leftarrow \text{E.PublicKeyGen}(\text{params}, sk)$. 

• FP.Enc_{pk}(m). Run c ← E.Enc_{pk}(m).
• FP.Dec_{sk}(c). Run m’ ← E.Dec_{sk}(c).
• FP.SwitchKeyGen(params, sk). Generate a matrix A’_{L} ← A_{L}^{LWE_{QL,\chi}(t)} and set the switching key swk = A_{L} ← P_2(s \otimes s)^T + A’_{L} (Add the vector P_2(s \otimes s)^T to the first row of A’_{L}). For 0 < \ell < L, let A_{\ell} be the \((n+1)^2\log q_{\ell}\) matrix consisting of the first \((n+1)^2\log q_{\ell}\) columns of A’_{L}.
• FP.Add(c_1, c_2). For two ciphertexts c_1, c_2 in the same level \ell, output c_{add} ← c_1 + c_2 mod q_{\ell}.
• FP.Mult_{sk}(c_1, c_2). For two ciphertexts c_1, c_2 in the same level \ell, output c_{mult} ← A_{\ell} \cdot BD(c_1 \otimes c_2) mod q_{\ell}.
• FP.ModSwitch_{l \rightarrow \ell’}(c). For a ciphertext c in level \ell, output c’ ← \frac{q_{\ell’}}{q_{\ell}} \cdot c.
• FP.ModEmbed_{l \rightarrow \ell’}(c). For a ciphertext c in level \ell, output c’ ← c mod q_{\ell’}.

Let LWE_{s,\ell}(m, B) be the set of \ell-level encryptions of m under s whose error is bounded by B, i.e., a vector c ∈ Z_q \times Z_{n\ell} is contained in LWE_{s,\ell}(m, B) if \langle c, s \rangle = m + e \pmod{q_{\ell}} for some |e| < B. We omit s and shortly denote LWE_{\ell}(m, B) when it is clear from the context.

There is a natural embedding FP.ModEmbed : c (mod q_{\ell}) ↦ c (mod q_{\ell’}) from LWE_{\ell}(m, B) to LWE_{\ell’}(m, B) for any \ell > \ell’. The converted ciphertext c’ ← FP.ModEmbed_{l \rightarrow \ell’}(c) has exactly the same message and error as c, so this procedure does not change the scale of message. This embedding can be used to carry out operations between ciphertexts in different levels.

**Security.** Let params = (n, q_{L}, \chi, \tau) ← FP.Setup(1^3, 1^L) so that the (n, q_{L}, \chi)-LWE problem achieves at least 2^L security and \tau = 2(n+1)^2\log q_{\ell}. If we generate sk ← E.SecretKeyGen(params), pk ← E.PublicKeyGen(params, sk) and swk ← FP.SwitchKeyGen(params, sk), then \(\langle pk, swk \rangle\) is computationally indistinguishable from the uniform distribution over \(\mathbb{Z}_{q_{\ell}}^{(n+1)^2 \times (n+1)^2 \log q_{\ell}}\) from the hardness of \(\langle n, q_{\ell}, \chi \rangle\) and the assumption of circular security. If pk is chosen in \(\mathbb{Z}_{q_{\ell}}^{(n+1)^2 \times \tau}\) uniformly, then the statistical distance of \(\langle pk, FP.Enc_{pk}(0) \rangle\) from the uniform distribution on \(\mathbb{Z}_{q_{\ell}}^{(n+1)^2 \times \tau} \times \mathbb{Z}_{q_{\ell}^{n+1}}\) is negligible from the leftover hash lemma [DGHV10]. Thus, our floating-point HE scheme described above is IND-CPA secure.

We analyze the noise growth of our basic scheme at encryption and addition, and provide a sufficient condition for decryption correctness.

**Lemma 1 (Encryption Noise).** Let c ← FP.Enc(m) for a message m. Then the initial ciphertext c is in LWE_{\ell}(m, B_{clean}) for B_{clean} = \tau \cdot B_{err} with overwhelming probability.

**Proof.** By definition, we have \langle c, s \rangle = \langle m + A \cdot r, s \rangle = m + r^T \cdot (A^T s). Since each column of A is a LWE sample, the infinite norm of vector e := A^T \cdot s is less than B_{err} with overwhelming probability. This lemma follows from |\langle r, e \rangle| \leq \tau \cdot B_{err}.

**Lemma 2 (Addition/Multiplication Noise).** Let c_i ∈ LWE_{\ell}(m_i, B_i) be encryptions of m_i satisfying |m_i| ≤ M_i for i = 1, 2. Let c_{add} ← FP.Add(c_1, c_2) and c_{mult} ← FP.Mult(c_1, c_2). Then, we have c_{add} ∈ LWE_{\ell}(m_1 + m_2, B_1 + B_2) and c_{mult} ∈ LWE_{\ell}(m_1m_2, M_1B_1 + M_2B_1 + B_1B_2 + B_{Ks,\ell}) for B_{Ks,\ell} = (n+1)^2\log q_{\ell} \cdot B_{err} with overwhelming probability.

**Proof.** The bound of addition noise is obvious. The definition of FP.Mult circuit implies \langle c_{mult}, s \rangle = \langle P_2(s \otimes s), BD(c_1 \otimes c_2) \rangle + \langle A^T \cdot BD(c_1 \otimes c_2), s \rangle. From a property of tensor product, we get \langle P_2(s \otimes s), BD(c_1 \otimes c_2) \rangle = \langle c_1, s \rangle \cdot \langle c_2, s \rangle = (m_1 + e_1)(m_2 + e_2) for some |e_1| ≤ B_1 and |e_2| ≤ B_2. We note that the infinite norm of vector e := A^T \cdot s is less than B_{err} with overwhelming probability. Therefore, \langle c_{mult}, s \rangle = m_1m_2 + e_{mult}, for e_{mult} = m_1e_2 + m_2e_1 + e_1e_2 + \langle e, BD(c_1 \otimes c_2) \rangle, which satisfies |e_{mult}| ≤ M_1B_2 + M_2B_1 + B_1B_2 + (n+1)^2\log q_{\ell} \cdot B_{err}.

\[\Box\]
Lemma 3 (Modulus-Switching Noise). Let $c \in \text{LWE}_\ell(m, B)$ be an encryption of $m$ and let $c' \leftarrow \text{FP.ModSwitch}_{\ell \to \ell'}(c)$. Then $c' \in \text{LWE}_\ell(q \ell, q \ell B + B_{\text{MS}})$ for $B_{\text{MS}} = \frac{w+1}{2} B_{\text{err}}$.

Proof. Let $e$ be the error of ciphertext $c$. Then $(c', s) = \frac{w}{q \ell}(c, s)^{q \ell} + (u, s) = \frac{w}{q \ell}(m + e) + (u, s)$ (mod $q \ell$) for a vector $u := c' - \frac{w}{q \ell}c$ with $\|u\|_\infty \leq 1/2$. The new error $e' = \frac{q \ell}{q \ell}e + (u, s)$ satisfies $|e'| \leq \frac{w}{q \ell}B + \frac{1}{2} |s|_1 \leq \frac{w}{q \ell}B + B_{\text{MS}}$.

Effect of Modulus-Switching. As noted before, our modulus-switching technique is similar to one of Brakerski, Gentry and Vaikuntanathan [BV11a,BGV12]. However, in our paper, the procedure $\text{FP.ModSwitch}$ changes the message from $m$ to a different and smaller message $\frac{w}{q \ell}m = p^{\ell'-\ell}m$. It does not mean that the message is ruined, but we just and change the representation (scale) of message. More precisely, if we express numbers in the floating-point system with base $p$, the modulus-switching procedure rounds up the least $(\ell - \ell')$-digits of message and shifts the radix point to the left while the exponent part is increased by $(\ell - \ell')$. As a result, most significant digits of message are invariant over this modulus-switching procedure, but only magnitude of significand is reduced.

Ciphertext Format with Auxiliary Information. For a practical usage of our scheme, it is recommended to keep additional information in a full ciphertext of the form $(c, \ell, M, B, \exp)$ in order to remember the position of radix point and compute the bounds of message and error during homomorphic evaluation. Here, $\ell, M, B$ and $\exp$ denote the level, the bounds of message and error, and exponent part of message of the ciphertext $c$, respectively. Then our scheme can be described as follows:

- $\text{FP.Enc} : (m, M) \mapsto (c, L, M, B_{\text{clean}}, 0)$,
- $\text{FP.Dec} : (c, \ell, M, B, \exp) \mapsto (m, B, \exp)$,
- $\text{FP.Add} : ((c_1, \ell, M_1, B_1, \exp_1), (c_2, \ell, M_2, B_2, \exp_2)) \mapsto (c_{\text{add}}, \ell, M_1 + M_2, B_1 + B_2, \exp)$,
- $\text{FP.Mult} : ((c_1, \ell, M_1, B_1, \exp_1), (c_2, \ell, M_2, B_2, \exp_2)) \mapsto (c_{\text{mult}}, \ell, M_1 M_2, M_1 B_2 + M_2 B_1 + B_{\text{KS}}, \ell, \exp_1 + \exp_2)$,
- $\text{FP.ModSwitch} \ell \to \ell' : (c, \ell, M, B, \exp) \mapsto (c, \ell', p^{\ell'-\ell} M, p^{\ell'-\ell} B + B_{\text{MS}}, \exp + (\ell - \ell'))$,
- $\text{FP.ModEmbed} \ell \to \ell' : (c \mod q \ell, \ell, M, B, \exp) \mapsto (c \mod q \ell, \ell', M, B, \exp)$.

Homomorphic Operations of Ciphertexts in Different Levels. To multiply a ciphertext $c$ in $\text{LWE}_\ell(m, B)$ with a ciphertext in lower level $\ell'$, we first should convert $c$ into a ciphertext in level $\ell'$. The modulus-switching procedure $\text{FP.ModSwitch}_{\ell \to \ell'}$ changes the message to $p^{\ell'-\ell}m$ with additional noise, while the modulus-embedding procedure $\text{FP.ModEmbed}_{\ell \to \ell'}$ does not. Therefore, the modulus-switching procedure storages smaller message $p^{\ell'-\ell}m$ which makes it more space efficient, and the modulus-embedding procedure does not create additional noises to initial message which makes calculations more careful and precise. See Figure 2 for an illustration.

In the rest of our paper, if the two ciphertexts $c_1, c_2$ do not belong to the same level, $\text{FP.Add}(c_1, c_2)$ and $\text{FP.Mult}(c_1, c_2)$ denote the homomorphic operations after performing the modulus-embedding procedure to the lower level.
Let $c$ be a ciphertext in $\text{LWE}_i(m, B)$ of a message $|m| \leq M$ and $m' \leftarrow \text{FP.Dec}(c)$. The absolute error $e = m' - m$ is bounded by $B$, but sometimes it is convenient to consider the relative error of message in floating-point arithmetics as we mentioned in Subsection 2.1. However, the definition of relative error in floating-point arithmetics cannot be used directly in homomorphic evaluation of circuits since one cannot see the intermediate values during secure evaluation of circuits. Instead, we suggest a slightly different definition of relative error $\alpha := B/M$ in this paper using the bounds of message and error. The following theorem gives an upper bound of relative error after the evaluation of circuit corresponding to the ordinary floating-point multiplication: rounding after multiplication of significands.

**Theorem 1.** Let $c_i \in \text{LWE}_i(m_i, B_i)$ be encryptions of $m_i$ such that $|m_i| \leq M_i$ and $B_i = \alpha_i M_i$ for $i = 1, 2$. Let $\epsilon = [\log q_L]^{-1}$ and $\Delta = 1 + \epsilon$. If (1) $p \geq 2(n + 1)[\log q_L]$, (2) $\alpha_i \leq \epsilon$, and (3) $p^k \cdot B_{\text{clean}} \leq (\alpha_1 + \alpha_2)M_1 M_2$, then

$$c \leftarrow \text{FP.ModSwitch}_{\epsilon \to \epsilon - k}(\text{FP.Mult}(c_1, c_2))$$

is contained in $\text{LWE}_{\epsilon - k}(\frac{1}{p^k}m_1 m_2, \alpha M)$ for $\alpha = \Delta(\alpha_1 + \alpha_2)$ and $M = \frac{1}{p^k}M_1 M_2$.

**Proof.** It follows from Lemmas 1 and 2 that the vector $c' \leftarrow \text{FP.Mult}(c_1, c_2)$ is in $\text{LWE}_{\epsilon - k}(m_1 m_2, B')$ for $B' = M_1 M_2 + B_1 B_2 + B_{\text{KS}}, B$. From Lemma 3, the output $c \leftarrow \text{FP.ModSwitch}_{\epsilon \to \epsilon - k}(c')$ of modulus switching procedure belongs to $\text{LWE}_{\epsilon - k}(\frac{1}{p^k}m_1 m_2, B)$ for $B = \frac{1}{p^k}B' + B_{\text{MS}}$.

Dividing $B$ by $M = \frac{1}{p^k}M_1 M_2$, we obtain

$$\frac{B}{M} = \alpha_1 + \alpha_2 + \frac{B_{\text{KS}} + p^k \cdot B_{\text{MS}}}{M_1 M_2} \leq \alpha_1 + \alpha_2 + \alpha_1 \alpha_2 + \left(\frac{n + 1}{2} + \frac{p^k}{4[\log q_L]} \right) B_{\text{clean}} M_1 M_2.$$

By making use of the inequalities $\alpha_1 \alpha_2 \leq \frac{\epsilon}{2}(\alpha_1 + \alpha_2)$ from (2), and $(\frac{n + 1}{2} + \frac{p^k}{4[\log q_L]} \frac{B_{\text{clean}}}{M_1 M_2}) \leq \frac{\epsilon}{2}(\alpha_1 + \alpha_2)$ from (1) and (3), we get the desired bound $B/M \leq \Delta(\alpha_1 + \alpha_2)$. □

For convenience, we use the notations $\epsilon = [\log q_L]^{-1}$ and $\Delta = 1 + \epsilon$ in the rest of our paper. Note that $\epsilon \approx (L \log p)^{-1}$ and $\Delta \approx (1 + \frac{1}{\log p}) \approx e^{1/\log p}$ is very close to 1. After a homomorphic multiplication with modulus switching procedure, the relative error becomes $\alpha = \Delta(\alpha_1 + \alpha_2)$. In other words, the result ciphertext has a significand with $\log \alpha^{-1} \approx -1 + \log \alpha_1^{-1}$ bits of precision. Hence, this process decreases the precision of message about one bit similar to the case of unencrypted floating-point computation.

The condition (1) only depends on the scheme parameters $p, n$ and $L$ from $\text{FP.Setup}$, and most of parameters used in leveled HE schemes satisfy this condition. Hence, we assume that
the condition (1) always hold in this paper. We have \((\alpha_1 + \alpha_2)M_1M_2 = M_1B_2 + M_2B_1\) in the condition (3). Under the assumption that \(B_i \geq B_{\text{clean}}\) for \(i = 1, 2\), this condition can be relaxed to \(\max\{M_1, M_2\} \geq p^k\). In addition, the error bound of ciphertext \(c\) in Theorem 1 satisfies \(B = \alpha M \geq (\alpha_1 + \alpha_2)\ell_M \geq B_{\text{clean}}\). Therefore, we only need to consider the relaxed condition \(\max\{M_1, M_2\} \geq p^k\) when the error bound of ciphertexts are not smaller than \(B_{\text{clean}}\).

4 Homomorphic Evaluation of Floating-Point Arithmetics

We now show that noise growth in our FPHE is almost same as in unencrypted floating-point computations. We start with evaluating some typical floating-point circuits homomorphically—addition and multiplication by constants, monomial and power polynomial. We also extend these facts to general polynomial, inverse, exponent, and analytic function. We deduce bounds on the noise growth that occurs during evaluation and analyze the precision of results.

4.1 Addition and Multiplication by Constants

We start with presenting simple operations to evaluate the functions \(f(x) = x + a\) and \(f(x) = ax\) for a real number \(a \in \mathbb{R}\). Briefly speaking, we add \(a \in \mathbb{R}\) to the first component of a ciphertext for constant addition, and multiply a ciphertext by \(a \in \mathbb{R}\) for constant multiplication.

**Lemma 4 (Addition by Constant).** Let \(c \in \text{LWE}_\ell(m, B)\) be an encryption of \(m\) and \(a \in \mathbb{R}\) be a real number. Then, \(c' \leftarrow c + ([a], 0, \ldots, 0) \pmod{q}\) is contained in \(\text{LWE}_\ell(m + [a], B) \subseteq \text{LWE}_\ell(m + a, B + 1/2)\).

**Proof.** It is obvious from \(\langle c', s \rangle = [a] + \langle c, s \rangle = ([a] + m) + e \pmod{q}\) for some \(|e| \leq B\). \(\quad\square\)

In the case of an integer \(a \in \mathbb{Z}\), the constant addition procedure does not change the error bound \(B\) but change the message bound to \(M + a\). As a result, the relative error also changes.

**Lemma 5 (Multiplication by Constant).** Let \(c \in \text{LWE}_\ell(m, B)\) be an encryption of \(m\) such that \(|m| \leq M\), and \(a \in \mathbb{R}\) be a real number. Then, the vector \(c' \leftarrow [p^u \cdot a]c\) is contained in \(\text{LWE}_\ell(p^u \cdot am, B')\) for \(B' = (p^u \cdot |a| + \frac{1}{2})B + \frac{1}{2}M\).

**Proof.** From the fact that \(\langle c, s \rangle = m + e \pmod{q}\) for some \(|e| \leq B\), we have \(\langle c', s \rangle = [p^u \cdot a] (m + e) = p^u \cdot am + e'\) such that \(|e'| \leq (p^u \cdot |a| + \frac{1}{2})B + \frac{1}{2}M\). \(\quad\square\)

In the previous lemma, let \(\alpha = B/M\) be the relative error of given ciphertext \(c\). The vector \(c' \leftarrow [p^u \cdot a]c\) has the message bound \(M' = p^u \cdot |a|M\), and the relative error \(\alpha' = \frac{B'}{M'} = \alpha + \frac{B + M}{2p^u \cdot |a|M} \leq \alpha + \frac{1}{p^{u} \cdot |a|}\). We may choose an integer \(u\) large enough so that the relative error does not increase too much. In particular, if the scalar \(a \in \mathbb{R}\) is of the form \(p^{-u} \cdot \hat{a}\) for some integers \(u\) and \(\hat{a}\), then the relative error remains the same after constant multiplication.

4.2 Monomial and Power Polynomial

We introduce some algorithms to evaluate the power function \(x^d\) and the monomial \(\prod_{i=1}^{d} x_i\) in this subsection. We start from the simplest case \(f(x) = x^2\), the power polynomial of a power of two degree. For simplicity, we assume that the bound \(M\) of message \(m\) is equal to the base \(p\).

Algorithm 1 performs the modulus switching procedure repeatedly after each squaring step to maintain the size of significand. The output of Algorithm 1 is thus not an encryption of \(f(m) = m^2\), but an encryption of the scaled value \(p \cdot f(m/p) = m^{2k}/p^{2k-1}\). The following lemma checks the correctness of Algorithm 1 and computes the relative error of output ciphertext.
Algorithm 1: Power polynomial $f(x) = x^d$ of power-of-two degree

1: procedure Power($c \in \mathbb{Z}_q \times \mathbb{Z}_q^n, d = 2^k$)
2:   $c_0 \leftarrow c$
3:   for $i = 1$ to $k$ do
4:     $c_i \leftarrow \text{FP.ModSwitch}_{(i-1)\rightarrow(i)}(\text{FP.Mult}(c_{i-1}, c_{i-1}))$
5:   end for
6: return $c_k$
7: end procedure

Lemma 6 (Power Polynomial of Power-of-Two Degree). Let $c \in \text{LWE}_k(m, B)$ be an encryption of $m$ such that $|m| \leq p$ and $B = \alpha_0p$. If $\alpha_0 \leq (2\Delta)^{-k}(k-1) \cdot \epsilon$ and $B \geq B_{\text{clean}}$, then the output $c_k \leftarrow \text{Power}(c, 2^k)$ of Algorithm 1 is contained in $\text{LWE}_k(m^2/p^{2^k-1}, \alpha_kp)$ where $\alpha_k = (2\Delta)^k \cdot \alpha_0$.

Proof. Let $\alpha_i = 2\Delta \cdot \alpha_{i-1}$ for $i = 1, \ldots, k$. From Theorem 1, we can inductively show that $c_i \leftarrow \text{FP.ModSwitch}_{(i-1)\rightarrow(i)}(\text{FP.Mult}(c_{i-1}, c_{i-1}))$ belongs to $\text{LWE}_k(m^2/p^{2^k-1}, \alpha_kp)$ for all $i$. Thus, after $k$ iterations, $c_k$ is contained in $\text{LWE}_k(m^2/p^{2^k-1}, \alpha_kp)$ where $\alpha_k = (2\Delta)^k \cdot \alpha_0$. □

Algorithm 1 is used as a subroutine of the following extended algorithm which evaluates a power polynomial $f(x) = x^d$ for any degree $d$.

Algorithm 2: Power polynomial $f(x) = x^d$

1: procedure Power($c \in \mathbb{Z}_q \times \mathbb{Z}_q^n, d$)
2:   Let $d = 2^k_1 + \cdots + 2^k_r$ where $0 \leq k_1 < \cdots < k_r = [\log d]$
3:   $c_1 \leftarrow \text{Power}(c, 2^{k_1})$
4:   $v_1 \leftarrow c_1$
5:   for $i = 2$ to $r$ do
6:     $c_i \leftarrow \text{Power}(c_{i-1}, 2^{k_i-k_{i-1}})$
7:     $v_i \leftarrow \text{FP.ModSwitch}_{(i-k_i)\rightarrow(i-k_{i-1})}(\text{FP.Mult}(c_i, v_{i-1}))$
8:   end for
9: return $v_r$
10: end procedure

Similar to the previous case, Algorithm 2 outputs an encryption of the scaled value $p \cdot f(m/p) = m^d/p^{d-1}$. We show the correctness and analyze the relative error in the following theorem.

Theorem 2 (Power Polynomial). Let $c \in \text{LWE}_k(m, B)$ be an encryption of $m$ such that $|m| \leq p$ and $B = \alpha_0p$. Let $d$ be a positive integer with binary representation $d = 2^k_1 + \cdots + 2^k_r$ for an integer $r$ and some integers $0 \leq k_1 < \cdots < k_r$. Let $k = [\log d]$. If $\alpha_0 \leq (2\Delta)^{-k}(k-1) \cdot \epsilon$ and $B \geq B_{\text{clean}}$, then the output $v \leftarrow \text{Power}(c, d)$ of Algorithm 2 is in $\text{LWE}_k(m^d/p^{d-1}, \alpha_kp)$ for $\alpha_k = (2\Delta)^k \cdot \alpha_0$.

Proof. The case $r = 1$ is obvious by Lemma 6, so assume that $r \geq 2$. Let $\alpha_i = 2\Delta \cdot \alpha_{i-1}$ for $i = 1, \ldots, k$. From the proof of Lemma 6, the vector $c_1$ is contained in $\text{LWE}_k(m^{2^{k_1}}/p^{2^{k_1}-1}, \alpha_{k_1}p)$.

Using the induction on $i$, we may show that $c_i \in \text{LWE}_k(m^{2^{k_i}}/p^{2^{k_i}-1}, \alpha_{k_i}p)$ and $v_i \in \text{LWE}_k(m^{2^{k_i}}/p^{2^{k_i}-1}, \alpha_{k_i+1}p)$ for all $i = 1, \ldots, r$. Therefore, the output of Algorithm 2 is contained in $\text{LWE}_k(m^d/p^{d-1}, \alpha_kp)$ for $k = k_r + 1 = [\log d]$. □
One may homomorphically evaluate a monomial \( f(x_1, \ldots, x_d) = \prod_i x_i \) using the binary tree structure, and its relative error can be computed similarly to Theorem 2. Let \( k = \lceil \log d \rceil \). For given level-\( k \) ciphertexts \( \{c_i \in \text{LWE}_k(m_i, \alpha_i \beta) : 1 \leq i \leq d \} \) such that \( |m_i| \leq p \) and \( \alpha_i \leq (2\Delta)^{-(k-1)} \cdot \epsilon \), one may compute their product by \( (d-1) \)-number of multiplications in depth \( k \). The resulting ciphertext is contained in \( \text{LWE}_{k-\ell}((\prod_{i=1}^d m_i/p^{d-1}) \alpha \beta) \) for \( \alpha = \Delta^k \cdot \sum_{i=1}^d \alpha_i \). Comparing this result to the unencrypted case in Section 2.1, our scheme achieves almost optimal precision of floating-point computation.

4.3 Polynomial and Approximating Power Series

The goal of this subsection is to evaluate polynomial and analytic functions and analyze the relative error of output ciphertext during homomorphic evaluation. We start by presenting an algorithm for evaluating polynomial functions.

**Algorithm 3** Polynomial function \( f(x) = \sum_{i=0}^d a_ix^i \)

1: procedure POLYNOMIAL\((c \in \mathbb{Z}_{q^e} \times \mathbb{Z}_{q^e}^n, f(x) = \sum_{i=0}^d a_ix^i \in \mathbb{R}[x], u \in \mathbb{Z})\)
2: \( \mathbf{v} \leftarrow ([p^{u+1} \cdot a_0], 0, \ldots, 0) \)
3: for \( i = 1 \) to \( d \) do
4: \( \mathbf{v} \leftarrow \text{FP.Add}(\mathbf{v}, [p^u \cdot a_i] \cdot \text{POWER}(c, i)) \)
5: end for
6: return \( \mathbf{v} \)
7: end procedure

**Theorem 3 (Polynomial Evaluation).** Let \( f(x) = \sum_{i=0}^d a_ix^i \in \mathbb{R}[x] \) be a real polynomial of degree \( d \) with \( k = \lceil \log d \rceil \), and \( c \in \text{LWE}_k(m, B) \) be an encryption of \( m \) such that \( |m| \leq p \) and \( B = \alpha_0 \beta \). If \( \alpha_0 \leq (2\Delta)^{(k-1)} \cdot \epsilon \), then, the output \( \mathbf{v} \leftarrow \text{POLYNOMIAL}(c, f(x), u) \) of Algorithm 3 is in \( \text{LWE}_{k-\ell}(p^{u+1} \cdot f(m/p), \alpha_0 p^{u+1}) \) where \( \alpha_f = \left( \sum_{i=1}^d |a_i| \right) \cdot \beta \cdot \Delta^k \alpha_0 + (d+1)\beta^{-u} \).

**Proof.** From Theorem 2 and Lemma 5, for all \( i = 1, \ldots, d \), the ciphertexts \( \text{POWER}(c, i) \) and \( [p^u \cdot a_i] \cdot \text{POWER}(c, i) \) are contained in \( \text{LWE}_{k-\lceil \log i \rceil}(m^i/p^{i-1}, B_i) \) and \( \text{LWE}_{k-\lceil \log i \rceil}(p^{u+1} \cdot a_im_i/p^i, B'_i) \) for \( B_i = \Delta^{\ell} \alpha_0 \beta \) and \( B'_i = p^u \cdot a_iB_i + p \), respectively.

Therefore, the output \( \mathbf{v} \) of Algorithm 3 is an encryption of \( p^{u+1} \cdot f(m/p) \) with the error bound \( 1/2 + \sum_{i=1}^d B'_i = 1/2 + \left( \sum_{i=1}^d |a_i| \cdot (2\Delta)^i \right) \cdot \alpha_0 p^{u+1} + dp \), which is less than \( \alpha_0 p^{u+1} \) for \( \alpha_f = \left( \sum_{i=0}^d |a_i| \right) \cdot (2\Delta)^k \alpha_0 + (d+1)\beta^{-u} \).

The resulting ciphertext of above theorem is an encryption of \( p^{u+1} \cdot f(m/p) \), with the message bound \( p^{u+1} \cdot \left( \sum_{i=0}^d |a_i| \right) \) and the relative error \( \frac{\alpha_f}{\sum_{i=0}^d |a_i|} = (2\Delta)^k \alpha_0 + \frac{(d+1)}{p^{u+1} \sum_{i=0}^d |a_i|} \), which is almost equal to \( (2\Delta)^k \alpha_0 \) for a sufficiently large integer \( u \).

By storing some intermediate computation results, we can compute the encryptions of \( m, m^2/p, \ldots, m^d/p^{d-1} \) simultaneously in \( d \) homomorphic multiplications and so reduce the complexity of Algorithm 3 to \( d \). The evaluation point \( m/p \) is in the interval \([-1, 1]\) in Theorem 3, but it can be extended to general cases by taking an alternative polynomial (for example, \( g(x) = f(px) \)).

Now consider an analytic function \( f(x) \) and its Taylor decomposition \( f(x) = T_d(x) + R_d(x) \) for \( T_d(x) = \sum_{i=0}^d \frac{f^{(i)}(0)}{i!}x^i \) and \( R_d(x) = f(x) - T_d(x) \). If \( R_d(x) \) goes to zero as \( d \to \infty \), then \( T_d(x) \)
which is an approximate value of $p_\alpha$ error is bounded by $(\alpha_T + |R_d(m/p)|) \cdot p^{u+1}$ for $\alpha_T = \left( \sum_{i=0}^{d} \left| \frac{f^{(i)}(0)}{d!} \right| \right) \cdot (2\Delta)^{\lfloor \log d \rfloor} \alpha_0 + (d + 1)p^{-u}$ by Theorem 3.

Consider the exponential function $f(x) = e^x$ as a concrete example. It has the Taylor polynomial $T_d(x) = \sum_{i=0}^{d} \frac{x^i}{i!}$ and the remainder $R_d(x)$ is bounded by $|R_d(x)| \leq \frac{e^x}{(x+1)!}$ when $|x| \leq 1$. For an encryption of $m$ such that $|m| \leq p$ with relative error $\alpha_0$, take two integers $u$ and $d$ such that $p^u \geq 2\alpha_0^{-1}$ and $d(d + 1)! \geq 2\alpha_0^{-1}$. Note that the choice of $d = \lceil \log \alpha_0^{-1} \rceil$ holds the required condition. Then Algorithm 3 outputs an encryption of $p^{u+1} \cdot T_d(m/p)$ with an error bounded by $\alpha_T R_d^{u+1}$ for $\alpha_T \leq e \cdot (2\Delta)^{\lfloor \log d \rfloor} \alpha_0 + (d + 1)p^{-u}$, and it can be also viewed as an encryption of $p^{u+1} \cdot \beta^{\log \alpha_0^{-1}}$ with the message bound $e p^{u+1}$ and the relative error $2(2\Delta)^{\lfloor \log d \rfloor} \alpha_0$. Hence, the precision of ciphertext is decreased by $\lfloor \log d \rfloor + 1 = \lfloor \log \log \alpha_0^{-1} \rfloor + 1$ bits compared to the input ciphertext.

Now consider another example $f(x) = \frac{e^x}{1 + x^2}$. The logistic function is widely used in medicine as a predictive equation in logistic regression. For example, logistic regression was used for prediction the likelihood to have a heart attack in an unspecified period for men in [BLN14]. It was also used as a predictive equation to screen for diabetes in [TH02]. One can evaluate the logistic function $e p^{u+1} \cdot \beta^{\log \alpha_0^{-1}}$ with relative error $2(2\Delta)^{\lfloor \log d \rfloor} \alpha_0$. Hence, the precision of ciphertext is decreased by $\lfloor \log d \rfloor + 1 = \lfloor \log \log \alpha_0^{-1} \rfloor + 1$ bits compared to the input ciphertext.

### 4.4 Multiplicative Inverse

We may refer the previous subsection to compute its Taylor decomposition and apply Theorem 3, to securely compute the multiplicative inverse. However, we will use another method, which is more easy to investigate, and almost preserves the precision of floating-point computation. We review the approximate method using a convergence algorithm [cDSM15]. Given $x \in [\frac{1}{2}, \frac{3}{2}]$, let $\bar{x} = 1 - x \in [0, \frac{1}{2}]$. Then we see that

$$x(1 + \bar{x})(1 + \bar{x}^2)(1 + \bar{x}^2) \cdots (1 + \bar{x}^{2^{k-1}}) = 1 - \bar{x}^{2^k}.$$  

(1)

Note that this product is in the interval $[1 - 2^{-2^k}, 1]$ and converges to one as $k$ goes to infinity. Hence, $\prod_{i=0}^{k-1} (1 + \bar{x}^i) = x^{-1}(1 - \bar{x}^{2^k})$ can be considered as an approximate multiplicative inverse of $x$ with an relative error bounded by $2^{-2^k}$.

We now return to our subject. Assume that an integer $m$ has a range of $[p/2, p]$, and denote $\bar{m} = p - m$. The standard approach starts by normalizing those numbers to be in the unit interval by setting $x = m/p \in \left[\frac{1}{2}, 1\right]$. Since we cannot multiply fractions over encrypted data, the precision point has to move to the left for each term of (1). We multiply both sides of the equation (1) by $p^{k}$, and then it yields

$$m(p + \bar{m})(p^2 + \bar{m}^2)(p^4 + \bar{m}^4) \cdots (p^{2^k-1} + \bar{m}^{2^k-1}) = p^{2^k} - \bar{m}^{2^k}.$$  

Therefore, $\prod_{i=0}^{k-1} (p^{2^i} + \bar{m}^{2^i})/p^{2^k}$ can be seen as approximate inverse of $m$ with an relative error bounded by $(\bar{m}/p)_{2^k} \leq 2^{-2^k}$. We first suggest an algorithm for evaluating this circuit, and analyze the resulting relative error and optimal number of iterations in the following theorem.

**Theorem 4 (Inverse Function).** Let $c \in \text{LWE}_c(m, B_0)$ be an encryption of $m$ such that $p/2 \leq m \leq p$ and $B_0 = \alpha_0 p/2$. If $\alpha_0 \leq (2\Delta)^{-(k-1)} \cdot \epsilon$ and $B_0 \geq 2^{2^{k-1}-(k-1)} \cdot B_{\text{clean}}$, then the output $v_k \leftarrow$
Algorithm 4 Inverse function $f(x) = x^{-1}$

1: procedure INVERSE($c \in \mathbb{Z}_q \times \mathbb{Z}_q^n$, $k$)
2: \hspace{1em} $p \leftarrow (p, 0, \ldots, 0)$
3: \hspace{1em} $c_0 \leftarrow p - c$
4: \hspace{1em} $v_1 \leftarrow \text{FP.ModEmbed}_{\ell \rightarrow (\ell - 1)}(p + c_0)$
5: \hspace{1em} for $i = 1$ to $k - 1$
6: \hspace{2em} $c_i \leftarrow \text{FP.ModSwitch}_{(\ell - i) \rightarrow (\ell - i - 1)}(\text{FP.Mult}(c_{i-1}, c_{i-1}))$
7: \hspace{1em} $v_{i+1} \leftarrow \text{FP.ModSwitch}_{(\ell - i) \rightarrow (\ell - i - 1)}(\text{FP.Mult}(v_i, p + c_i))$
8: \hspace{1em} end for
9: \hspace{1em} return $v_k$
10: end procedure

\textsc{Inverse}$(c, k)$ of Algorithm 4 is contained in $\text{LWE}_{\ell - k}(m', \alpha \cdot 2p)$ where $m' = (\ell^2 \alpha \cdot \beta) \left(1 - (1 - \frac{m}{p})^{2^k}\right)$ and $\alpha = \Delta^k \cdot \alpha_0$. In particular, with the input $k = \lceil \log \log \alpha_0^{-1} \rceil$, this algorithm outputs a ciphertext in $\text{LWE}_{\ell - k}(p^2/m, 2 \alpha \cdot 2p)$.

\textbf{Proof.} Let $\bar{m} = p - m$. From Lemma 4, the ciphertexts $c_0$ and $v_1$ are contained in $\text{LWE}_{\ell}(\bar{m}, B_0)$ and $\text{LWE}_{\ell - 1}(p + \bar{m}, B_0)$, respectively. Note that they have the same error bound $B_0 = \alpha_0(2^{-1}p)$, but different message bounds $2^{-1}p$ and $(1+2^{-1})p$, and different relative errors $\alpha_0$ and $\alpha'_0 = (2+1)^{-1}\alpha_0$.

Let $\alpha_i = (2\Delta^i) \cdot \alpha_0$ for $i = 1, \ldots, k - 1$. From Theorem 1, it is easy to check that $c_i$ belongs to $\text{LWE}_{\ell - i}(m^{2^i}/(p^{2^i} - 1), \alpha_0(2^{i-1}p))$ for all $i$. By Lemma 4, the vector $p + c_i$ is contained in $\text{LWE}_{\ell - i}(m^{2^i}/(p^{2^i} - 1), \alpha_0(2^{i-1}p))$, and it has the increased message bound $(1+2^{-i})p$ and the relative error $\alpha'_i = \frac{1}{(2\Delta^i)} \alpha_0$.

Let $\alpha''_i = \Delta \cdot \alpha'_0$ and $\alpha''_i = \Delta(\alpha'_{i-1} + \alpha''_{i-1})$ for $i = 2, \cdots, k$. Using the induction on $i$, we can show that $v_i$ is contained in

\[
\text{LWE}_{\ell - i} \left( \prod_{j=0}^{i-1} \frac{(p^{2^j} + \bar{m}^{2^j})}{p^{2^{j+2}} - 1}, \alpha''_i \cdot \prod_{j=0}^{i-1} (1 + 2^{-2^j}) \right),
\]

that is, $v_i$ can be seen as an encryption of $\prod_{j=0}^{i-1} \frac{(p^{2^j} + \bar{m}^{2^j})}{p^{2^{j+2}} - 1} = (p^2/m)(1 - (\bar{m}/p)^{2^i})$ with the message bound $\prod_{j=0}^{i-1} (1 + 2^{-2^j})p = (1 - 2^{-2^i})/(2p) \leq 2p$ and the relative error \[
\alpha''_i = \sum_{j=0}^{i-1} \Delta^{i-j} \cdot \alpha'_j = \Delta^i \cdot \left( \sum_{j=0}^{i-1} \frac{2^j}{2^{2j+1} + 1} \right) \alpha_0 \leq \Delta^i \cdot \alpha_0,
\]

from the fact that $\sum_{j=0}^{\infty} \frac{2^j}{2^{2j+1} + 1} = 1$. Therefore, the output of Algorithm 4 is contained in $\text{LWE}_{\ell - k}(m', \alpha \cdot 2p)$ for $m' = (p^2/m)(1 - (\bar{m}/p)^{2^k})$ and $\alpha = \Delta^k \cdot \alpha_0$.

In the case of $k = \lceil \log \log \alpha_0^{-1} \rceil$, the difference of $m'$ and $(p^2/m)$ is bounded by $2^{-2^k}(p^2/m) \leq \alpha \cdot 2p$ also, and this algorithm outputs an encryption of $(p^2/m)$ with the message bound $2p$ and the relative error $2\alpha = 2\Delta^k \cdot \alpha_0$.

This algorithm outputs a ciphertext whose relative error is $2\alpha = 2\Delta^k \cdot \alpha_0 \approx 2\alpha_0$, so it loses about one bit of precision. The optimal number of iterations can be changed when we know more/less information about the magnitude of $m$. Assume that the message $m$ is contained in the interval $((1 - \beta)p,p)$ for some value $0 < \beta < 1$. Let $\bar{m} = p - m \in [0, \beta p]$. Then we have...
\[\prod_{i=0}^{k-1}(p^{2^i} + m^2) = (p^{2^k}/m) \cdot (1 - (m/p)^2^k),\]
which is an approximation of \(p^{2^k}/m\) with \(2^k \cdot \log \beta^{-1}\) bits of precision. Therefore, the optimal number of iteration is \(k \approx \log \log \alpha_0^{-1} - \log \log \beta^{-1}\) in general case.

The condition \(B_0 \geq 2^{2^k-1-(k-1)} \cdot B_{\text{clean}}\) in Theorem 4 is stronger than our general assumption \(B_0 \geq B_{\text{clean}}\). However, we may use a simple transformation ciphertexts to make them satisfy the required condition. For a given encryption \(c \in \text{LWE}_m(B_0)\) of \(m \in [p/2, p]\) such that \(B_0 \geq B_{\text{clean}}\) and \(B_0 = \alpha_0 p\), let \(c^* = p \cdot c \pmod{q_\ell}\). Then \(c^*\) is an encryption of \(m^* \in [p^*/2, p^*]\) with error bound \(B_0^*\) and relative error \(\alpha_0\). After this transformation, we can use Theorem 4 and compute the multiplicative inverse of \(m\) homomorphically since \(B_0^* = p \cdot B_0 \geq 2^{2^i-1-(k-1)} B_{\text{clean}}\).

5 Comparison and Implementation

In this section, we provide a theoretical comparison of our FPHE scheme and previous HE scheme for evaluating arithmetic circuits described in Section 4. We also adapted our method to BGV scheme with Shoup-Halevi’s HE library [HS13] (called HElib). All experiments reported in our paper were performed on a machine with an Intel Xeon 2.6 GHz processor running a Linux 3.16.0 operating system.

5.1 Comparison with previous HE schemes

Gentry, Halevi and Smart [GHS12b] constructed an efficient BGV-type somewhat HE scheme. The security of this scheme is based on the (decisonal) Ring Learning With Errors (RLWE) assumption, which was first introduced by Lyubashevsky, Peikert and Regev [LPR10]. This scheme has a chain of ciphertext moduli by a set of primes of roughly the same size, \(p_0, \ldots, p_L\), that is, the \(i\)-th modulus \(q_i\) is defined as \(q_i = \prod_{k=0}^{i} p_k\). For simplicity, assume that \(P\) is the approximate size of the \(p_i\)'s. We choose an integer \(m\) that defines the \(m\)-th cyclotomic polynomial \(\Phi_m(x)\). For a polynomial ring \(R = \mathbb{Z}[x]/(\Phi_m(x))\), set the plaintext space to \(R_t := R/tR\) for some fixed \(t \geq 2\) and the ciphertext space to \(R_q := R/qR\) for an integer \(q\).

Assume that we are given encryptions of messages \(m_1, \ldots, m_d\) with \(\eta\) bits of precision. To securely compute a product of \(m_i\)'s, we can consider three distinct message encoding possibilities with BGV-type schemes: the first two naive methods are called scalar encoding method, where each input (bit or large integer) is encoded as a constant polynomial. The last one is called balanced base-B encoding method for an odd integer \(B \geq 3\), which represents a fixed-point number with base \(B\) and encodes as a polynomial with coefficients in the range \([-B-1)/2, (B-1)/2]\).

- All the messages are encrypted bit-wise. We represent the entire functions as binary circuits, which correspond to arithmetic on bits in \(\mathbb{Z}_2\). Note that \(\eta\)-bit multiplication is computable by boolean circuits of depth \(O(\log \eta)\) with \(O(\eta^2)\) complexity [Vol13]. Now suppose that we only obtain the most significant \(\eta\) bits. Then the total depth amount to \(O(\log \eta \cdot \log d)\) and it requires \(O(d \cdot \eta^2)\) homomorphic multiplication in total. So the approximate bit size of the ciphertext modulus \(q_L\) is \((L+1) \cdot \log P = O(\log \eta \cdot \log d)\). We note that it is encrypted as a set of \(d \cdot \eta\) ciphertexts.
- The second method is to use a large integer ring as a message space instead of a binary field. It allows us to evaluate with only \((d-1)\) homomorphic multiplications. We first choose sufficiently large \(t\) so that no reductions modulo \(t\) occurs in the plaintext space, so we take \(t\) as the smallest integer which satisfies \(\log t > \eta \cdot d\). Next, given the Hamming weight \(h\) of the secret key, it follows from Section 5.1 in [KL15] that \(L = \log d \cdot \left\lceil \log (h \cdot \phi(m) \cdot t^x) \right\rceil + 1 = O(\eta \cdot d \cdot \log d)\). Thus the bit size of the ciphertext modulus \(q_L\) is approximately \((L+1) \cdot \log P = O(\eta \cdot d \cdot \log d)\) and we only
have \( d \) ciphertexts. The reader can consider another method which is based on decomposition modulus \( W \).

- Dowlin et al. [DGL+15] suggested a balanced base-B encoding method of fixed-point numbers for an odd integer \( B \geq 3 \). If a fixed-point number \( b \) is represented as \( b = \sum_{i=1}^{t} b_i B^i \) for some \( b_i \in \{-1(B-1)/2, (B-1)/2\} \), it is encoded as a polynomial \( b(X) = X^{t-1} + \sum_{i=1}^{t} b_i X^i \). It follows from [CSVW16] that when computing the encryption of product of \( m_i \)’s homomorphically, the size of \( t \) is needed to ensure correctness as follows: \( t > \lceil (B-1) \cdot (\log_B 2^n) \rceil d \). Thus the bit size of the largest modulus is approximately \( \mathcal{O}(\log d \cdot \log t) = \mathcal{O}(\log \eta \cdot d \cdot \log d) \), and we also only need \( d \) ciphertexts.

In case of our scheme, we may take \( p = 2^n \) and \( L = \log d \) for the correctness of decryption procedure after product of ciphertexts of \( m_i \)’s. Then total computation cost is \( (d-1) \) and the bit size of the ciphertext modulus is \( \log q_L = \log p \cdot (L + 1) = \eta \cdot (\log d + 1) \). Even though our scheme only computes an encryption of arithmetic solution with \( (\eta - \lceil \log d \rceil) \) bits of precision, the circuit depth is logarithm in \( d \), and the numbers of homomorphic multiplications and ciphertexts are linear in \( d \). We provide a better description of comparison in Table 2.

<table>
<thead>
<tr>
<th>Table 2. Product of ( d ) ciphertexts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Circuit Depth</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>Previous</td>
</tr>
<tr>
<td>Bit ptx</td>
</tr>
<tr>
<td>Large ptx</td>
</tr>
<tr>
<td>Balanced</td>
</tr>
<tr>
<td>Ours</td>
</tr>
</tbody>
</table>

5.2 Our Implementation with Concrete Parameters

In Table 3, we present the parameter setting and performance results for multiplying 16 integers of 20 bits and computing the multiplicative inverse of 20-bit integer with our scheme. All the parameters provide 80-bit security level. When we measured the average running times, we excluded computing times used in data encryption and decryption.

To evaluate the arithmetic circuit \( \prod_{i=1}^{16} x_i \) homomorphically, BGV-type scheme requires huge computation cost with bit-wise encryption or at least 320-bit plaintext space, so it is impossible to be evaluated in practice. Meanwhile, our scheme can compute the approximate value with 15 bits of precision in about 315ms. Moreover, upon input with 20 bits of precision of \( x \) for \( 1/2 \leq x \leq 1 \), we may compute its multiplicative inverse \( x^{-1} \) with only 8 homomorphic multiplications and depth 5, and its implementation takes about 168ms.

\( ^7 \) For an integer \( W \), we first decompose our messages modulo \( W \) and then evaluate the function over the integers. More precisely, an initial message can be represented as \( m_i = \sum_{j=0}^{r} m_{i,j} \cdot W^j \) for \( m_{i,j} \in [0, W) \) and \( r = \lceil \log_W 2^n \rceil \). Assume that it is encrypted as a set of \( (r + 1) \cdot d \) ciphertexts of \( m_{i,j} \), denote by \( \text{Enc}(m_{i,j}) \). One can perform \( \sum_{j_1 + j_2 = l} \text{Enc}(m_{i_1,j_1}) \cdot \text{Enc}(m_{i_2,j_2}) \) for \( 0 \leq l \leq 2r \), which corresponds to the multiplication of \( m_{i_1} \) and \( m_{i_2} \). After \( \log d \) steps, we obtain a set of ciphertexts of the form \( \{\text{ct}_i\}_{0 \leq i \leq d \cdot r} \) and then compute \( \sum_{i=0}^{d \cdot r} \text{Dec}(\text{ct}_i) \cdot W^i \) as desired. This method provides some parameter/complexity tradeoff, since obtaining a smaller value of \( t \) (i.e., \( \log t = d \cdot \log W \)) makes parameters small but requires more number of homomorphic multiplications.
Table 3. Implementation results for $\eta = 20$ with 80 bits security

<table>
<thead>
<tr>
<th>Function</th>
<th>Bit precision of output</th>
<th>Circuit depth</th>
<th>#(HM)</th>
<th>Size of modulus $(\log q_L)$</th>
<th>Timing (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\prod_{i=1}^{16} x_i$</td>
<td>15</td>
<td>4</td>
<td>15</td>
<td>76</td>
<td>315</td>
</tr>
<tr>
<td>$x^{-1}$</td>
<td>18</td>
<td>5</td>
<td>8</td>
<td>76</td>
<td>168</td>
</tr>
</tbody>
</table>

References


