Fast, Compact and Hardware-Friendly Bootstrapping in less than 3ms Using Multiple Instruction Multiple Ciphertext

Seunghwan Lee, Dohyuk Kim, and Dong-Joon Shin

Department of Electronic Engineering, Hanyang University,
Seoul, Korea
{kr3951,dohyuk1000,djshin}@hanyang.ac.kr

Abstract. This paper proposes a fast, compact key-size, and hardware-friendly bootstrapping using only 16-bit integer arithmetic and fully homomorphic encryption FHE16, which enables gate operations on ciphertexts using only 16-bit integer arithmetic. The proposed bootstrapping consists of unit operations on ciphertexts, such as (incomplete) number theoretic transform (NTT), inverse NTT, polynomial multiplication, gadget decomposition, and automorphism, under a composite modulus constructed from 16-bit primes. Since our bootstrapping does not use any floating-point operations, extra floating-point errors do not occur so that FHE16 can pack more message bits into a single ciphertext than TFHE-rs which utilizes floating-point operations. Furthermore, we propose multiple instruction multiple ciphertext(MIMC) method to accelerate the simultaneous execution of different homomorphic operations across multiple ciphertexts. Finally, experimental results show that the bootstrapping operation completes in 2.89 milliseconds for ciphertext dimension of 512.

Keywords: Lattice-based cryptography, fully homomorphic encryption, gate bootstrapping, 16-bit integer arithmetic, multiple instruction multiple ciphertext.

1 Introduction

A cryptosystem capable of performing arithemetic operations or gate operations on ciphertexts is referred to as fully homomorphic encryption (FHE) [20], and it has established itself as a cryptographic primitive in modern cryptography. In particular, FHE schemes that perform gate operations, such as TFHE/FHEW [17, 12], have made remarkable advancements in homomorphic operation speed. However, all current FHE schemes share a common problem: after performing operations, the error within the ciphertexts accumulates, necessitating bootstrapping to reduce the error before further computations are performed.

Gate-based FHE schemes are known to have significantly smaller key sizes and faster bootstrapping time, compared to FHE schemes that approximate real numbers or perform modular arithmetic. This makes them particularly suited for

applications such as oblivious pseudo-random functions [1] and multiparty FHE [37]. Currently, several FHE libraries supporting these applications are available, including OpenFHE [5], developed with support from the U.S. DARPA, and TFHE-rs [40], released by ZAMA, which supports fully homomorphic Ethereum Virtual Machine (fhEVM) applications.

Furthermore, it is noteworthy that current implementation of TFHE-rs is reported to be approximately twice as fast as that of OpenFHE. This performance difference is mainly attributed to the use of distinct polynomial multiplication techniques: OpenFHE utilizes number theoretic transform (NTT), whereas TFHE-rs employs fast Fourier transform (FFT).

However, two major issues have recently arisen due to the floating-point-based FFT in TFHE-rs. The first issue pertains to the design of algorithms for fast homomorphic integer arithmetic, which requires encryption up to 256 bits for fhEVM implementations [8]. When FFT is performed, significant errors are inevitably introduced, preventing use of a certain number of message bits in a single ciphertext for homomorphic operations. This limitation becomes a drawback when designing homomorphic arithmetic for large-bit integers. Additionally, controlling such FFT error requires a very large integer modulus, leading to an increase in communication overhead.

The second issue involves the use of TFHE-rs in threshold FHE (tFHE) [37], which extends FHE capabilities to multi-party computation (MPC) for evaluating a function f. For example, n servers encrypt their respective input data $x_1, ..., x_n$, individually compute the ciphertext of $f(x_1, ..., x_n)$, and decrypt the ciphertext by threshold decryption protocol [37] in tFHE. In order to decrypt the ciphertext, an identical ciphertext for all parties should be shared among during the homomorphic operation. However, since floating-point operations, which induce FFT errors, do not satisfy the associative law, the nature of the error varies depending on the implementation, compilation environment, and machine, breaking the consensus among ciphertexts.

Meanwhile, with the introduction of FHE schemes based on the NTRU assumption, more compact key sizes and faster homomorphic computations have become feasible [10, 27], but these fast computations are only possible when the message bits are small. When the integer modulus q of the ciphertext is increased to pack many message bits into a ciphertext, the polynomial degree d must also be increased. From the perspective of the NTRU assumption, d must inevitably be increased beyond the "fatigue point" at $q \approx 0.004d^{2.484}$, which requires $d = \Omega(\lambda \log(q)^2)$ to grow to ensure safety against sublattice attacks. In contrast, for LWE-based FHE, $d = \Omega(\lambda \log(q))$ suffices to be secure against Block Korkin-Zolotarev (BKZ)-based attacks [18], hence LWE-based FHE is more favorable to pack a lot of message bits compared to NTRU-based FHE.

"Can we design bootstrapping that supports fast gate homomorphic computation, encrypts and computes over multiple bits, and remains hardware-friendly while maintaining a compact key size?"

This paper proposes a solution to the above question in the form of a 16-bit arithmetic-based bootstrapping.

1.1 Our Contributions

This paper proposes 16-bit arithmetic-based bootstrapping that performs homomorphic operations over ciphertexts using only 16-bit arithmetic and we call the FHE scheme FHE16 which utilizes 16-bit arithmetic-based bootstrapping. This approach offers several advantages in terms of bootstrapping speed, key size, and hardware-level implementation:

Fast. FHE16 outperforms OpenFHE by a factor of 4.3x and TFHE-rs by a factor of 1.4x for the same parameter settings. Furthermore, if the parameters are optimized as proposed in this paper, FHE16 becomes 6.2x faster than OpenFHE, 4x faster than TFHE-rs, and even 1.1x faster than NTRU-based FHE schemes [27].

Compact. With the proposed parameter optimization, we can make the blind rotation key size of FHE16 (See Section 3.4) comparable to that of OpenFHE, around 12 to 14 MiB, and 3.8x smaller than that of TFHE-rs.

Hardware Friendly. The study of improving the speed of FHE schemes has evolved towards the use of GPUs or hardware accelerators designed by FPGA and ASIC technologies. The 16-bit arithmetic-based bootstrapping offers two main advantages in terms of hardware compatibility: First, all operations in bootstrapping rely solely on multiple 16-bit additions and multiplications using arithmetic logic units (ALU), and this makes it suitable for designing low-power, high-density circuit acceleration chips. Second, while gate-based FHE schemes are known to be difficult to parallelize, 16-bit arithmetic-based bootstrapping offers several opportunities for parallelization within its algorithm. This allows for more efficient use of streaming processors (SMs) when accelerated with multi-core devices like GPUs (See Section 5.1).

1.2 Technical Overview

The key technical components of 16-bit arithmetic-based bootstrapping are as follows.

16-bit-based polynomial unit operations: NTT_w, INTT_w, AUT_w, and Decom. Since FHE16 uses a product of n 16-bit primes $q_1, ..., q_n$ as the integer modulus q, traditional NTT operations cannot be applied. Therefore, we employ incomplete NTT, denoted by $\mathsf{NTT}_{w>1}$ and its inverse, denoted by $\mathsf{INTT}_{w>1}$, along with multiplication in the $\mathsf{NTT}_{w>1}$ domain based on Karatsuba multiplication [15]. Also, we propose a 16-bit-based gadget decomposition algorithm and a method for performing automorphism in the $\mathsf{NTT}_{w>1}$ domain.

Refining Bootstrapping Error Model with Two Types of Decomposition. We integrate the bootstrapping error model in $\mathbb{Z}_q$ [17, 26] with a bootstrapping technique that improves speed by using two decomposition parameters in approximate torus $\mathbb{T}_q$ cryptosystems like TFHE-rs [7] (See Section 5.2). This combined model allows for further parameter minimization, reducing key size and improving bootstrapping speed.

Multiple Instruction on Multiple Ciphertext (MIMC). We propose a fast bootstrapping for performing multiple instruction on multiple ciphertexts in GINX bootstrapping [12] via rearranging the operation order (See Section 5.1).

1.3 Related Works

Some previous works related to our works are introduced below.

Lightweight Bootstrapping Key. Kim et al. [24] propose a method to reduce the bootstrapping key size to within 1MiB by using Galois automorphism, although it is twice as slow as OpenFHE. Since our bootstrapping leverages 16-bit Galois automorphism and is much faster than OpenFHE, by applying the technique in [24], the bootstrapping key can also be made smaller than 1MiB.

SIMC for Gate Bootstrapping. It is an open question whether gate-based FHE schemes like TFHE/FHEW can improve speed by simultaneously bootstrapping multiple ciphertexts. Liu and Wang [28] propose a method to perform functional bootstrapping within 7ms using BGV, and Bae et al. [6] introduce a method to bootstrap at least 243 ciphertexts simultaneously using CKKS for speed improvement. However, both methods require conversion to other FHE ciphertexts, which significantly increases the bootstrapping key size. Moreover, these methods are limited to applying the same operation to all ciphertexts.

Hardware Optimization. Seiler [36] shows that a 16-bit arithmetic-based encryption system can be efficiently optimized on Intel/AMD CPUs by utilizing SIMD instructions provided by the CPU. Furthermore, lattice-based zero-knowledge proofs benefit from using composite 16-bit primes as a modulus, that improves both proof generation time and verification time [32].

2 Preliminaries

In this section, we present the terminology and the mathematical operations necessary for this paper.

2.1 Notations

We define the torus as $\mathbb{T} \triangleq \mathbb{R}/\mathbb{Z}$, and the scaled torus as $\mathbb{T}_q \triangleq \mathbb{T}/q^{-1}\mathbb{T} \cong q^{-1}\mathbb{Z}/\mathbb{Z}$. We define $\mathbb{Z}_m^*$ be the multiplicative group whose elements are units in $\mathbb{Z}_m$. The m-th cyclotomic polynomial $\Phi_m(X) \in \mathbb{Q}[X]$ is defined as the unique irreducible polynomial with integer coefficients whose roots are the primitive m-th roots of unity. The degree of $\Phi_m(X)$ is denoted as d and we restrict m to powers of 2, for which $\Phi_m(X) = X^{m/2} + 1$, implying $m = 2d$. A vector over $\mathbb{Z}_q$ or $\mathbb{T}_q$ is denoted as $\vec{a}$, and an element of $R_{q,d}$ is denoted as $\boldsymbol{a}$. A vector whose components are polynomials is represented as $\vec{\boldsymbol{a}}$. Given $a \in \mathbb{Z}_q$, we say a is in canonical form if $-q/2 < a \leq \lfloor q/2 \rfloor$. Similarly, a vector $\vec{a}$ or $\vec{\boldsymbol{a}}$ is in canonical form if all of its components or coefficients a_i are in canonical form. For a natural number n and a unit $\zeta \in \mathbb{Z}_n$, $\mathsf{ord}_n(\zeta)$ denotes the multiplicative order of ζ, meaning that ζ is a primitive $\mathsf{ord}_n(\zeta)$-th root of unity in $\mathbb{Z}_n$. Let $\zeta_m \in \mathbb{Z}_p$ be a primitive m-th root of unity in the $\mathbb{Z}_p$, if it exists. The notations used in this paper are summarized in Table 1.

Notation	Description
KS, BL	Key-switching key, blind rotation key
$R_{q,d}$	Quotient ring $\mathbb{Z}[X]/\langle X^d+1, q\rangle$, where d is a power of two
sk_1, sk_2	sk_1: secret key for KS and ciphertext, sk_2: secret key for BL
k_1	Dimension of KS and LWE ciphertext
k_2	Dimension of MLWE ciphertext in BL
d	Degree of MLWE ciphertext in BL
B_2^A, B_2^B	Gadget decomposition bases for the a and b parts of BL
l_2^A, l_2^B	Number of gadget decompositions for the a and b parts of BL
B_{ks}	Gadget decomposition basis for the a part of KS
l_{ks}	Number of gadget decompositions for the a part of KS
q_1, q_2	q_1: modulus for KS, q_2: modulus for BL
σ_{ks}, σ_{bl}, σ_{Aut}	Standard deviations for key KS, BL, and Aut respectively
w	Degree of modulus polynomial after performing NTT_w algorithm
U_t	Support of secret keys where $U_t = (-t/2, \lfloor t/2 \rfloor] \subset \mathbb{Z}$ and $\lvert U_t \rvert = t$
v	Window size of AP^+ blind rotation
Δ	Scaling factor for MLWE ciphertext
β	Machine word size, e.g., $2^{16}, 2^{32}, 2^{64}$
p_i	Special primes listed in Table 9 which size is less than 2^{16}.

Table 1: Parameters for the 16-bit arithmetic-based bootstapping

2.2 Signed Montgomery Reduction: SM

In this section, we describe the integer moduli q_1 and q_2 with the machine's integer word size $\beta = 2^{16}$ used in this paper. The modulus q_1 is chosen as a power of 2, and the modulus q_2 is a composite number, determined by multiplying small primes less than 16 bits in size, as listed in Table 9 of Supplementary Material A. Our 16-bit arithmetic-based bootstrapping always computes modular q_2 computation by performing the Montgomery multiplication algorithm SM for signed integers [36], as in Algorithm 1. Note that Line 5 in SM can be omitted and then, the result r will satisfy $-q \leq r < q$.

Algorithm 1 Signed Montgomery multiplication $\mathsf{SM}(a, b)$ with word size β [36]

Input: Odd q with $0 < q < \frac{\beta}{2}$, $-\beta/4 < a \leq \lfloor \beta/2 \rfloor$, $-q/4 < b \leq \lfloor q/2 \rfloor$
Output: $r = \beta^{-1}(ab) \bmod q$, $0 \leq r < q$
1: Calculate a_0 and a_1 such that $-\frac{\beta}{2} \leq ab = a_1\beta + a_0 < \frac{\beta}{2}q$, where $0 \leq a_0 < \beta$
2: $m \leftarrow a_0 q^{-1} \bmod \beta$ ▷ m is pre-calculated as in canonical form
3: $t_1 \leftarrow \lfloor mq/\beta \rfloor$
4: $r \leftarrow a_1 - t_1$
5: $r \leftarrow r < 0 \ ? \ r + q : r$ ▷ Optional
6: **return** r

To use the output r of SM for modular multiplication over $\mathbb{Z}_q$, r must be multiplied by β again. However, instead of the original input b, if we put $b' = b\beta \mod q$ into SM, we obtain the right result $ab \mod q$ without the need to multiply the output by β. Let us call the elements $b\beta \mod q$ transformed from b as elements in the Montgomery domain. Since all modular multiplications occurring during the ciphertext operations in the proposed bootstrapping are the multiplication of elements in the Montgomery domain, we can implement all modular multiplications by using only SM.

Benefit of Implementing Algorithms 1 with $\beta = 2^{16}$ on the Intel CPU. As observed in Lines 1 and Line 3 of Algorithm 1, these operations involve multiplying two word-size numbers and then extracting the higher bits from the result. If β is set to 2^{32} or 2^{64}, there is no single instruction in Intel CPUs to directly support the above operations, requiring multiple instructions and registers for implementation. However, since β is set to 2^{16} in this paper, this operation can be obtained using the Intel-provided instruction $\mathsf{vpmulhw}$, allowing for implementation with a single instruction.

2.3 Fast Polynomial Multiplication Methods: NTT_w, FFT, and MUL_w

In this section, we introduce some fast polynomial multiplication methods: NTT_w, FFT, and MUL_w over the rings $R_{q,d}$ and $\mathbb{T}_q[X]/\langle X^d + 1\rangle$.

Lemma 1 (Theorem 2.3 in [29]) *Let $m = \prod p_i^{e_i}$ with $e_i \geq 1$ and $z = \prod p_i^{f_i}$ with $1 \leq f_i \leq e_i$ where p_i's are distinct primes. If q is a prime such that $q = 1 \mod z$ and $\mathsf{ord}_m(q) = m/z$, then the m-th cyclotomic polynomial $\Phi_m(X)$ factors as*

$$\Phi_m(X) = \prod_{j \in \mathbb{Z}_z^*} (X^{m/z} - r_j) \mod q \tag{1}$$

for distinct $r_j \in \mathbb{Z}_q^*$ where $X^{m/z} - r_j$ are irreducible over $\mathbb{Z}_q[X]$.

Now, consider Lemma 1 with $m = 2d$, where d is a power of 2, and $z = 2d/w$ such that $q = 1 \mod 2d/w$ and $\mathsf{ord}_{2d/w}(q) = 1$. By Lemma 1, $\Phi_{2d}(X)$ factors into d/w irreducible polynomials of degree w. Since all the roots of $\Phi_{2d}(X)$ are d primitive $2d$-th roots of unity, all r_j in Eq. (1) can be expressed as powers of a primitive $2d/w$-th root of unity $\zeta_{2d/w} \in \mathbb{Z}_q$, i.e., $r_j = \zeta_{2d/w}^j$. Thus, by the Chinese Remainder Theorem (CRT), the following isomorphic structure holds:

$$\mathbb{Z}_q[X]/\langle X^d + 1\rangle \cong \bigoplus_{i \in \mathbb{Z}_{2d/w}^*} \mathbb{Z}_q[X]/\langle X^w - \zeta_{2d/w}^i\rangle. \tag{2}$$

The algorithm that transforms from the left-hand structure to the right-hand structure in Eq. (2) is called NTT_w, and the inverse transformation algorithm is called INTT_w [1]. NTT_w can be implemented using the Cooley-Tukey

[1] In [15], NTT and INTT are used when $w = 1$, and the term "Incomplete NTT" is used when $w > 1$. In this paper, we characterize these transformations with w.

approach, while INTT_w can use the Gentleman-Sande approach, both of which require $O(d \log d)$ multiplications in $\mathbb{Z}_q$ [36]. When $w = 1$ and $\Phi_{2d}(X) \in \mathbb{Z}_q[X]$ splits, the polynomial multiplication on the right-hand structure in Eq. (2) is the component-wise multiplication in $\mathbb{Z}_q$, requiring only $O(d)$ multiplications. The space for the right-hand structure is referred to as the NTT_w space, and the algorithm that performs polynomial multiplication algorithm is denoted as MUL_w.

If the modulus $q = q_1 \cdots q_n$ is a composite number of primes q_i, the following isomorphic structure in Eq. (3) holds:

$$\mathbb{Z}_q[X]/\langle X^d + 1\rangle \cong \bigoplus_{i=1}^{n} \mathbb{Z}_{q_i}[X]/\langle X^d + 1\rangle. \tag{3}$$

Thus, NTT_w algorithm can be performed using n parallel paths.

Algebraic Description for Using FFT [33] in $\mathbb{T}_q[X]/\langle X^d + 1\rangle$. If w is large or if the polynomial coefficients are in $\mathbb{T}_{2^{32}}$ or $\mathbb{T}_{2^{64}}$, it is impossible to perform polynomial multiplication efficiently, and also the NTT_w operation becomes inefficient or infeasible. In such cases, the modulus q is ignored for the moment, and the canonical-form coefficients are mapped to $\mathbb{C}[X]/\langle X^d + 1\rangle$, allowing for polynomial multiplication using FFT and IFFT. In fact, FFT and IFFT are performed using a primitive $2d$-th root of unity in $\mathbb{C}$ numerically [33]. Although the exact results can be obtained with infinite precision, in practice, floating-point arithmetic introduces approximation errors, as discussed in Section 5.2.

2.4 Automorphisms on $R_{q,d}$ and AUT_w

An isomorphism from a field K to itself is called an automorphism, and the set of such functions is denoted by $\mathsf{Aut}(K)$. Given towers of fields K over F, the Galois group can be defined as follows:

$$\mathsf{Gal}(K/F) \triangleq \{\sigma : K \to K \mid \sigma \in \mathsf{Aut}(K), \quad \sigma(x) = x \text{ for all } x \in F\}. \tag{4}$$

Let $\omega_m \in \mathbb{C}$ denote a primitive m-th root of unity in $\mathbb{C}$. The number field $K_m \triangleq \mathbb{Q}(\omega_m)$ is referred to as cyclotomic field. Furthermore, $\mathsf{Gal}(K_m/\mathbb{Q}) = \mathsf{Aut}(K_m)$ since $\mathbb{Q}$ is a prime field (i.e., it has no nontrivial subfields). In this paper, we only consider the Galois group $\mathsf{Gal}(K_m/\mathbb{Q})$ which is $\mathsf{Aut}(K_m)$. Moreover, for any $\sigma \in \mathsf{Aut}(K_m)$, we denote it as σ_i by using the index $i \in \mathbb{Z}_m^*$. In particular, when m is a power of 2, it is known that $\mathbb{Z}_m^*$ is isomorphic to the additive group $\mathbb{Z}_2 \times \mathbb{Z}_{m/4}$ [16], which is used to describe the AP^+ blind rotation in Section 3.4.

All polynomials in $\mathsf{FHE16}$ are stored as the elements in the NTT_w space. Thus, we propose an algorithm of performing automorphisms in the NTT_w space, called AUT_w, as discussed in Section 4.1.

2.5 Residue Number System, Mixed-Radix System, and Conversion Algorithm Decom

For co-prime natural numbers $q_1, \ldots, q_n$, the following isomorphic structure is well known:

$$\mathbb{Z}_{q=q_1 \cdots q_n} \cong \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_n}. \tag{5}$$

Here, the Residue Number System (RNS) is a numerical system where an element $a \in \mathbb{Z}_q$ on the left-hand side of Eq. (5) is represented as a tuple $(a_1^*, \ldots, a_n^*)$ on the right-hand side, and component-wise arithmetic operations are performed. To implement the isomorphism from right to left in Eq. (5), we define a few auxiliary variables:

$$q_i^* \triangleq \prod_{j \neq i}^{n} q_j \in \mathbb{Z}_{q_i}, \quad \hat{q}_i \triangleq (q_i^*)^{-1} \in \mathbb{Z}_{q_i}, \quad \tilde{q}_i \triangleq \prod_{j=1}^{i} q_j \in \mathbb{Z}. \tag{6}$$

If we can accurately compute the value $v = \left\lceil \sum_{i=1}^{n} \left(a_i^* \hat{q}_i \bmod q_i\right)/q_i \right\rfloor$, then it is known that we can compute the inverse map in Eq. (7), and also express the element in its canonical form in $\mathbb{Z}_q$ [22]:

$$(a_1^*, \ldots, a_n^*) \mapsto \left(\sum_{i=1}^{n} [a_i^* \hat{q}_i \bmod q_i] \cdot q_i^* \right) - vq, \text{ for some } v \in \mathbb{Z}. \tag{7}$$

On the other hand, for canonical-form $a \in \mathbb{Z}_q$ and natural numbers $q_1, \ldots, q_n$ (which are not necessarily coprime), there exists a canonical-form $\tilde{a} = (\tilde{a}_1, \ldots, \tilde{a}_n)$ that satisfies the following expression via signed division algorithm:

$$a = \tilde{a}_1 \tilde{q}_1 + \tilde{a}_2 \tilde{q}_2 + \cdots + \tilde{a}_n \tilde{q}_n, \quad -q_i/2 < \tilde{a}_i \leq \lfloor q_i/2 \rfloor. \tag{8}$$

We refer to $\vec{\tilde{q}} = (\tilde{q}_1, \ldots, \tilde{q}_n)$ as a gadget, and the canonical-form $(\tilde{a}_1, \ldots, \tilde{a}_n)$ is called the $\vec{\tilde{q}}$-mixed radix system (MRS) number [19]. The gadgets used in this paper are always of the form $(D, DB, \ldots, DB^l)$, where D and B are powers of 2, and $\tilde{q}_i \leq DB^i$. We denote this by (D, B, l)-gadget. In particular, when $D = 2^{\lceil \log_2(q) \rceil - \log_2(B) l}$, meaning the bit-length of q minus the top $l \log_2(B)$-bits, we call it as the (B, l)-gadget. If we obtain the (D, B, l)-MRS number $(a_0, a_1, \ldots, a_l)$ for $a \in \mathbb{Z}_q$, the following inequality holds:

$$\left| a - \sum_{i=1}^{l} a_i B^i \right| \leq \frac{D}{2}, \quad |a_i| \leq \frac{B}{2}. \tag{9}$$

In this paper, an algorithm Decom is proposed in Section 4.2, which outputs $(a_1, \ldots, a_l)$ satisfying Eq. (9) for a given (D, B, l)-gadget, where a may not be saved as canonical form. It is particularly important to obtain the MRS number with respect to the (B, l)-gadget for $a = (a_1^*, \ldots, a_n^*)$ in RNS, which is discussed in detail in Section 4.2.

3 (M)LWE Ciphertext and Bootstrapping

3.1 Introduction to (M)LWE Ciphertext and its Variants

In this section, we introduce the lattice-based cryptosystems considered in this paper. Given a dimension k, the LWE ciphertext is defined as follows [34]:

$$\mathsf{LWE}^i[\Delta m] = (\vec{a}, b) = \Big(\vec{a}, \sum_{j=1}^{k} a_j s_j + e\Big)^t + \Big(\underbrace{0, \ldots, 0}_{i-1}, \underbrace{\Delta m}_{\text{index } i}, \underbrace{0, \ldots, 0}_{k-i+1}\Big)^t \in \mathbb{Z}_q^{(k+1)\times 1},$$

where Δ is a scaling factor used to separate the message from the error. Each element of $\vec{a} = (a_1, \ldots, a_k)$ is chosen uniformly at random from $\mathbb{Z}_q$, while the secret key $\vec{s}$ and the error e are sampled from specific distributions φ_s and φ_e, respectively. The distribution φ_s is a uniform distribution over $U_t \triangleq \{s \in \mathbb{Z} \mid s \in (-t/2, t/2]\}$, and φ_e is a discrete Gaussian distribution with variance σ^2. If $i = k + 1$, the index i is omitted.

Similarly, for dimension k, the MLWE ciphertext is defined as follows:

$$\mathsf{MLWE}^i[\Delta \mathbf{m}] = (\vec{\mathbf{a}}, \mathbf{b}) = \Big(\vec{\mathbf{a}}, \sum_{j=1}^{k} \mathbf{a}_j \mathbf{s}_j + \mathbf{e}\Big)^t + \Big(\underbrace{0, \ldots, 0}_{i-1}, \underbrace{\Delta \mathbf{m}}_{\text{index } i}, \underbrace{0, \ldots, 0}_{k-i+1}\Big)^t \in R_{q,n}^{(k+1)\times 1},$$

where $\vec{\mathbf{a}}$, $\mathbf{s}_i$, $\mathbf{e}$, and Δ are similarly defined as the LWE ciphertext, and the index i is omitted if $i = k + 1$. Lastly, we define the $\mathsf{preMGSW}$ ciphertext and the MGSW ciphertext [21] necessary for defining the TFHE/FHEW. The $\mathsf{preMGSW}$ ciphertext is defined in Eq. (10) using the (B, l)-gadget parameters and MLWE:

$$\mathsf{preMGSW}^i_{B,l}[m] = \Big[\mathsf{MLWE}^i[Bm] \Big| \mathsf{MLWE}^i[B^2 m] \Big| \ldots \Big| \mathsf{MLWE}^i[B^l m]\Big]. \tag{10}$$

Then, given two distinct sets of gadget parameters (B_a, l_a) and (B_b, l_b), the MGSW ciphertext is defined as follows:

$$\mathsf{MGSW}[m] = \Big[\mathsf{preMGSW}^1_{B_a,l_a}[m] \Big| \ldots \Big| \mathsf{preMGSW}^k_{B_a,l_a}[m] \Big| \mathsf{preMLWE}^{k+1}_{B_b,l_b}[m]\Big], \tag{11}$$

where $\mathsf{MGSW}[m]$ is an $R_{q,n}$-matrix of the size $(k + 1) \times (kl_a + l_b)$ [2].

External Product. The external product between an MLWE ciphertext and an MGSW ciphertext, denoted as $\mathsf{MLWE} \boxdot \mathsf{MGSW}$, is defined as in Fig. 1 [3]. The external product is a critical unit operation for determining the bootstrapping speed of the TFHE/FHEW. Therefore, reducing the number of external products or improving its computation speed directly enhance the bootstrapping performance. Furthermore, the size of errors generated during the external product

[2] This definition adopts an optimization using two types of gadget parameters as proposed in [7] to improve bootstrapping speed.

[3] In this paper, all MGSW ciphertexts are assumed to be in the NTT_w space.

1. Given $\mathsf{MLWE}[\Delta \boldsymbol{m}_1] = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k, \boldsymbol{b})$, apply the Decom in Section 4.2 to all coefficients to obtain the MRS number $\vec{\boldsymbol{c}} \in R_{q,d}^{(l_a k + l_b)\times 1}$

2. Apply the NTT_w in Section 4.3 to the all elements of $\vec{\boldsymbol{c}}$

3. Using the MUL_w in Section 4.4, multiply the $\mathsf{MGSW}[\boldsymbol{m}_2] \in R_{q,n}^{(k+1)\times(l_a k + l_b)}$ by $\vec{\boldsymbol{c}} \in R_{q,d}^{(l_a k + l_b)\times 1}$ to obtain $\vec{\boldsymbol{d}} \in R_{q,d}^{(k+1)\times 1}$

4. Apply the INTT_w in Section 4.3 to the $k+1$ elements of $\vec{\boldsymbol{d}}$ to obtain $\mathsf{MLWE}[\Delta \boldsymbol{m}_1 \boldsymbol{m}_2]$

Fig. 1: External product $\mathsf{MLWE}[\Delta \boldsymbol{m}_1 \boldsymbol{m}_2] = \mathsf{MLWE}[\Delta \boldsymbol{m}_1] \boxdot \mathsf{MGSW}[\boldsymbol{m}_2]$.

affects the overall computation speed and the size of the BL. A detailed error analysis is presented in Section 5.2.

How to Interprete Torus-based Ciphertexts. Even if the coefficients of ciphertext are defined on $\mathbb{T}_q$ as in $\mathsf{TFHE\text{-}rs}$, these coefficients can be viewed as the elements of $\mathbb{Z}_q$ without any transformation process. See Supplementary Material D.

3.2 A Bootstrapping Example: Homomorphic NAND Operation

In this section, we examine an example of performing homomorphic NAND gate operation using the previously introduced $\mathsf{KeySwitching}$ and $\mathsf{blind\ rotation}$ techniques [12, 17, 26]. Note that bootstrapping consists of two main algorithms, $\mathsf{KeySwitching}$ and $\mathsf{blind\ rotation}$, which are explained in next Sections 3.3 and 3.4. The message m of every LWE ciphertext is either 0 or 1, and the scaling factor is $\Delta = \lfloor q_2/4 \rfloor$.

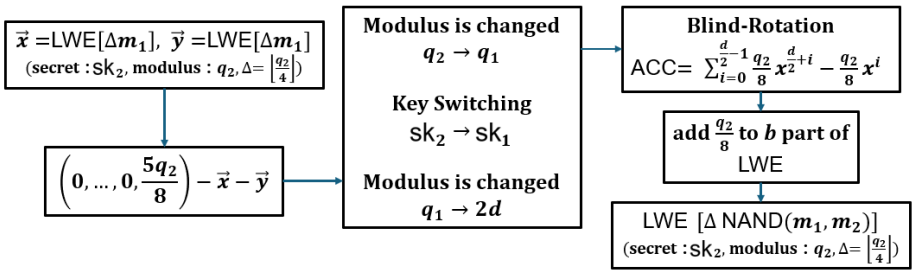

Fig. 2: Bootstrapping for evaluating NAND operation.

As in Fig. 2, after subtracting sum of two ciphertexts $\vec{x}+\vec{y}$ from the $(0, \ldots, 0, -5q_2/8)$, the integer modulus is changed from q_2 to q_1, and the secret key sk_2 is switched to sk_1 using $\mathsf{KeySwitching}$. The integer modulus is then converted from q_1 to $2d$. Next, through $\mathsf{blind\ rotation}$, the message of the MLWE ciphertext

becomes $\mathsf{acc}X^{-m-e}$. Finally, using only the a part of the ciphertext and the constant part of b, the ciphertext $\mathsf{LWE}[\Delta\mathsf{NAND}(m_1, m_2) - q_2/8]$ is constructed and the NAND operation ends by adding $q_2/8$ to the b part.

3.3 Key Switching and Automorphism

Key switching is the first step of bootstrapping, which changes the secret key of the ciphertext while keeping the message [4]. Given a ciphertext encrypted under the secret key $\mathsf{sk} = (\mathbf{s}_1, \ldots, \mathbf{s}_k)$, in order to switch sk to a new secret key sk', the key-switching key that encrypts sk under sk' is required:

$$\mathsf{KS}_i = \mathsf{preMGSW}[\mathbf{s}_i], \tag{12}$$

and the $\mathsf{KeySwitching}$ algorithm is performed (see Section 3 of [17]). Specifically, for a automorphism $\sigma_i \in \mathsf{Aut}(K_m)$, the automorphism key switching key from $\sigma_i(\mathsf{sk})$ to sk in Eq. (12) is referred to as AK_i. This allows the message $\mathbf{m}$ of ciphertext $\mathsf{MLWE}[\Delta\mathbf{m}]$ encrypted under sk is changed to $\sigma_i(\mathbf{m})$ as follows: (i) The ciphertext is converted to $\mathsf{MLWE}[\Delta\sigma_i(\mathbf{m})]$ encrypted under $\sigma_i(\mathsf{sk})$ by performing AUT_w to each polynomial in the ciphertext; (ii) the key $\sigma_i(\mathsf{sk})$ is changed to sk again via automorphism key-switching key AK_i. Such operation is denoted as $\mathsf{Aut}(\mathsf{MLWE}[\Delta m], \mathsf{AK}_i)$ [26]. In Section 4.1, we propose an efficient method for performing automorphisms in the $\mathsf{NTT}_{w>1}$ space.

Circular Security Assumption To perform bootstrapping in FHE, it is necessary to have ciphertexts that encrypt the secret keys $\mathbf{sk}_1$ and $\mathbf{sk}_2$ using each other, as shown below:

$$\left(\mathsf{Enc}_{\mathbf{sk}_1}(\mathbf{sk}_2), \mathsf{Enc}_{\mathbf{sk}_2}(\mathbf{sk}_1)\right), \tag{13}$$

where Enc refers to the encryption algorithm used in the cryptosystem. The circular security assumption adopted in this paper assumes that the ciphertexts in (13) are computationally indistinguishable from uniformly random sources in the lattice-based cryptosystem introduced in Section 3.1, which is a critical assumption of all FHE schemes [20].

3.4 Blind Rotation Implementation: GINX and AP$^+$

In this section, we introduce two blind rotation, a crucial component of bootstrapping, specifically the GINX method [12] and the improved AP$^+$ method [26], which is an enhancement of the AP method [17]. Consider the LWE ciphertext $\mathsf{LWE}[\Delta m] = (\vec{a}, b) = (a_1, \ldots, a_{k_1}, b)$ and the secret key for encryption, $\mathsf{sk}_1 = (s_1, \ldots, s_{k_1}) \in U_t^{k_1}$. The ultimate goal of blind rotation is to produce the following ciphertext:

$$\mathsf{MLWE}[\mathsf{acc}X^{\sum_{i=1}^{n} a_i s_i - b}] \tag{14}$$

[4] We follow the $\mathsf{KeySwitching}$ method in [17].

where the accumulating polynomial $\mathsf{acc} \in R_{q_2,d}$ includes a scaling factor Δ, and it is well known that the coefficients of acc can be determined to program the message after bootstrapping [17, 13].

GINX Blind Rotation. In order to define the GINX blind rotation, we introduce the blind rotation key:

$$\mathsf{BL}_{i,u} = \begin{cases} \mathsf{MGSW}[1] & \text{if } s_i = u \\ \mathsf{MGSW}[0] & \text{if } s_i \neq u \end{cases} \quad \text{for } 1 \leq i \leq k, \quad u \in U_t, \tag{15}$$

where the message in $\mathsf{BL}_{i,u}$ is 1 if the i-th secret value s_i equals u, and 0 otherwise. Using the blind rotation key in Eq. (15), the GINX blind rotation method for generating the ciphertext in Eq. (14) is outlined in Algorithm 10, Supplementary Material E [12].

AP$^+$ Blind Rotation. We first introduce the mathematical background before explaining AP$^+$ blind rotation. Given the m-th cyclotomic field K_m, from the property $\mathsf{Aut}(K_m) \cong \mathbb{Z}_{2d}^* \cong (\mathbb{Z}_2 \times \mathbb{Z}_{d/2}, +)$, we can always find $g \neq -1$ such that $\mathsf{ord}_{2d}(g) = d/2$. This means that for any $a \in \mathbb{Z}_{2d}^*$, we can uniquely express $a = (-1)^i g^j$ for $0 \leq i \leq 1$ and $0 \leq j < d/2$. Therefore, for $a = (-1)^i g^j$, we define the functions $\psi_1(a) = i$ and $\psi_2(a) = j$.

The AP$^+$ blind rotation key consists of k_1 MGSW ciphertexts BL and $(v+1)k_2$ automorphism key-switching keys AK, defined as follows:

$$\begin{aligned} \mathsf{BL}_i &\triangleq \mathsf{MGSW}[X^{s_i}], && \text{for } 1 \leq i \leq k_1, \\ \mathsf{AK}_{i,j} &\triangleq \mathsf{preMGSW}[\sigma_j(\mathbf{s}_i)], && \text{for } 1 \leq i \leq k_2,\ j \in \{g^{-1}, g^1, g^2, \ldots, g^v\}. \end{aligned} \tag{16}$$

In Supplementary Material F, Algorithm 11 is the AP$^+$ blind rotation process that generates the ciphertext in Eq. (14) using the keys defined in Eq. (16) [26].

It has been experimentally demonstrated that, among the KeySwitching and blind rotation which constitute bootstrapping Fig. 2, the primary speed bottleneck is the blind rotation. Moreover, since the execution speed of blind rotation is determined by the external product $\boxdot$ in Fig. 1 and in order to accelerating blind rotation, (i) accelerating the external product $\boxdot$; (ii) reducing the number of external product operations. In the Section 4 we will introduce five core operations NTT_w, INTT_w, MUL_w, AUT_w, and Decom for accelerating external product $\boxdot$. Also in the Section 5.1, we propose GINX blind rotation capable of performing MIMC operations and analysis bootstrapping error for reducing the number of performing external product in blind rotation.

4 Core Homomorphic Algorithms on $R_{q,n}$ When q is a Product of 16-bit Primes

In this Section, we introduce 16-bit based five core operations for implementing the external product $\boxdot$.

4.1 Automorphism AUT_w in the NTT_w Domain

While automorphisms are not used in GINX blind rotation, the AP^+ blind rotation does rely on automorphisms. Since ciphertexts are stored as in NTT_w domain, we propose the AUT_w algorithm, which allows the ciphertext in the NTT_w domain to undergo automorphism σ_i. Note that, AUT_w is a critical algorithm because it can be used as a unit operation to any lattice cryptosystem using 16-bit integer operations.

Let primitive $2d/w$-th root of unity $\zeta_{2d/w}$ be given. From Eq. (2), the NTT_w domain can be expressed as $\bigoplus_{i\in\mathbb{Z}^*_{2d/w}} \mathbb{Z}_q[X]/\langle X^w - \zeta^i_{2d/w}\rangle$. Then, Lemma 2 shows how the automorphism σ_i is applied in the NTT_w domain.

Lemma 2 (Discussion in Section 2.2 of [4]) *Let d be a power of two and let $\sigma_i \in \mathsf{Aut}(K_m)$ be an automorphism. Then for any $i, j \in \mathbb{Z}^*_{2d}$ with multiplicative inverse $i^{-1} \in \mathbb{Z}^*_{2d}$, the following isomorphism $\hat{\sigma}_i$ is induced from σ_i:*

$$\hat{\sigma}_i : \mathbb{Z}_p[X]/\langle X^w - \zeta^j_{2d/w}\rangle \to \mathbb{Z}_p[X]/\langle X^w - \zeta^{ji^{-1}}_{2d/w}\rangle \text{ as } \mathbf{a} \mapsto \sigma_i(\mathbf{a}).$$

Algorithm 2 Proposed automorphism algorithm AUT_w in the NTT_w domain

Input: A ring element $\boldsymbol{a}$ saved on NTT_w domain as $\mathsf{NTT}_w(\boldsymbol{a}) \in \bigoplus_{j\in\mathbb{Z}_{2d/w}} \mathbb{Z}_q[X]/\langle X^w - \zeta^j_{2d/w}\rangle$, an automorphism σ_i for $i \in \mathbb{Z}^*_{2d}$
Output: $\check{\boldsymbol{r}} = \mathsf{NTT}_w(\sigma_i(\boldsymbol{a}))$

1: Calculate $i^{-1} \mod 2d/w$
2: Set $\vec{t} \leftarrow (0, 0, ..., 0) \in \mathbb{Z}_p^w$
3: **for** $j \in \mathbb{Z}^*_{2d/w}$
4: $\quad j' \leftarrow ji^{-1} \mod 2d/w$ ▷ If $i = 1 \mod 2d/w$, then $j = j'$
5: $\quad$ Set $\vec{t} \leftarrow \boldsymbol{a}_j$ ▷ Assign the coefficients of polynomial $\boldsymbol{a}_j$ as a vector $\vec{t}$.
6: $\quad$ **for** $k = 0$ to $w - 1$
7: $\quad\quad k' \leftarrow ki \mod w$
8: $\quad\quad z \leftarrow kj' \mod 2d/w$
9: $\quad\quad r_{j'k'} \leftarrow \mathsf{SM}(t_k, 2^\beta \zeta^z_{2d/w})$ ▷ $r_{j'k'}$ refers to the k'-th coefficient of the j'-th polynomial in $\check{\boldsymbol{r}}$
10: $\quad$ **end for**
11: **end for**
12: **return** $\check{\boldsymbol{r}}$

AUT_w in Algorithm 2 utilizes Lemma 2 to perform the following steps: (i) move $\boldsymbol{a}_j$ stored in the j-th space of $\mathsf{NTT}_w(\mathbf{a})$ (i.e., $\mathbb{Z}_q[X]/\langle X^w - \zeta^j_{2d/w}\rangle$) to the j'-th space (i.e., $\mathbb{Z}_q[X]/\langle X^w - \zeta^{j'}_{2d/w}\rangle$), where j' is computed in Line 4; (ii) In Line 7, compute the index k' where the k-th coefficient of $\boldsymbol{a}_j$ in the j'-th

space should be stored; (*iii*) In Line 8, compute $\zeta_{2d/w}$ to the power of z to do polynomial modulo operation and in Line 9, the k'-th coefficient is obtained by multiplying the k-th coefficient t_k by $\zeta^z_{2d/w}$.

Note on Algorithm 2. In Line 9, modular exponentiation is performed with respect to $\zeta_{2d/w}$ using z. However, since the exponentiation result can be pre-calculated and stored in the form of $2^\beta\zeta^z_{2d/w}$, which is an element in the Montgomery domain, the exponentiation is not required during algorithm execution and modular multiplication can be performed using SM. Therefore, AUT_w can be implemented with only $O(d)$ additions and multiplications.

4.2 Gadget Decomposition Decom without Floating-point Arithmetic

Since all the elements in MLWE ciphertext are stored in the RNS form, it is necessary to convert these elements into the canonical form of the gadget basis to perform the external product $\boxdot$. Specifically, in order to implement Eq. (7), two methods have been proposed: one using floating-point arithmetic [22] and another taking an explicit CRT approach [9]. The former requires heuristic assumptions for the correctness due to its floating-point arithmetic, while the latter uses fixed-point arithmetic and needs additional conditions for ensuring correctness. In Algorithm 3, we introduce both algorithms and propose a modified version of the method in [9], which replaces fixed-point operations with integer arithmetic. Here, $\hat{q}_i$ is the dummy variable defined in Eq. (6).

Algorithm 3 Existing gadget decompositions [22] and [9], and proposed decomposition

Input: 16-bit numbers $q_1, ..., q_n$ with $q = q_1 \cdots q_n$, $a \in \mathbb{Z}_q$ with RNS form $(a^*_1, ..., a^*_n) \in \prod_{i=1}^n \mathbb{Z}_{q_i}$, $L = \lceil \log_2(q) - \beta \rceil$, s such that $2^s \geq 2n$, and $\bar{s}$ such that $2^{-\bar{s}} n \max_i q_i < 1/2$

Output: An $(0, 2^\beta, L)$-MRS number $(\tilde{c}_1, ..., \tilde{c}_l)$ of a

1: **for** $i \leftarrow 1 : L$

2: $v \leftarrow \begin{cases} (\mathsf{int16})\lfloor \sum_{i=1}^n (\mathsf{float})(a^*_i \hat{q}_i \bmod q_i)/q_i \rceil & \text{, (i) method in [22]} \\ \lfloor 3/4 + 2^{-s} \sum_{i=1}^n \lfloor 2^s (a^*_i \hat{q}_i \bmod q_i)/q_i \rfloor \rfloor & \text{, (ii) method in [9]} \\ \lfloor 2^{-\bar{s}} \sum_{i=1}^n (a^*_i \hat{q}_i \bmod q_i) \lfloor 2^{\bar{s}}/q_i \rfloor \rceil & \text{, (iii) proposed method} \end{cases}$

3: $u_i \leftarrow \left(\sum_{i=1}^n [a^*_i \hat{q}_i \bmod q_i] \cdot q^*_i \right) - vq \mod 2^\beta$ ▷ Output

4: $(a^*_1, ..., a^*_n) \leftarrow ((a^*_1 - u_i)2^{-\beta}, ..., (a^*_j - u_i)2^{-\beta}) \mod 2^\beta$

5: **end for**

6: **return** $(u_1, ..., u_L)$

Correctness of Line 2-(i) and (ii) in Algorithm 3 when single precision is used. The correctness Line 2-(i) and (ii) are well analyzed in [22, 9], or see Supplementary Material G.

Correctness of Line 2-(iii) in Algorithm 3. We now prove the following technical lemmas to show the correctness.

Lemma 3 *For given reals a, b and an integer $s > 0$, $ab - 2^{-s}a\lfloor 2^s b \rfloor < 2^{-s}a$.*

Proof. For any real b, we know $b - 1 < \lfloor b \rfloor \leq b$. Therefore, $ab = 2^{-s}a2^s b \geq 2^{-s}a\lfloor 2^s b \rfloor > 2^{-s}a(2^s b - 1) = ab - 2^{-s}a$. □

Lemma 4 *Let $q_1, ..., q_n$ be positive integers, $a_1, ..., a_n$ be integers such that $0 \leq a_i < q_i$, $q = \max(q_1, ..., q_n)$, $0 < \epsilon < 1/2$, s be an integer such that $2^{-s}nq < \epsilon/2$, and $v = \lfloor \sum_{i=1}^n a_i/q_i \rceil$. If $\left| v - \sum_{i=1}^n a_i/q_i \right| < \epsilon/2$, then $v = \left\lfloor 2^{-s} \sum_{i=1}^n \lfloor 2^s/q_i \rfloor a_i \right\rceil$.*

The proof is included in Supplementary Material H.

Theorem 5 *If $|a| < q/4$,the output of* Algorithm 3 *run with the proposed method in Line 2-(iii) is an* MRS *number.*

The proof is included in Supplementary Material H. However the elements of ciphertexts in FHE cannot be guaranteed to be smaller than $q/4$, which is the reason why Algorithm 4 is proposed as an alternative to Algorithm 3.

Algorithm 4 Gadget decomposition Decom without floating-point arithmetic

Input: 16-bit numbers $q_1, ..., q_n$ with $q = q_1 \cdots q_n$ such that $q_1 < ... < q_n$, gadget parameter (D, B, l) such that $D, B < q_1$, $a \in \mathbb{Z}_q$ with the RNS form $(a_1^*, ..., a_n^*) \in \prod_{i=1}^n \mathbb{Z}_{q_i}$, $l = \lceil \log_2(q/D) / \log_2(B) \rceil$

Output: An (D, B, l)-MRS number $(u_1, ..., u_l)$ of a

1: $\hat{n} \leftarrow n$
2: **for** $i \leftarrow 0 : l - 1$
3: Calculate the MRS number $(\tilde{a}_1, ..., \tilde{a}_{\hat{n}})$ from $(a_1^*, ..., a_{\hat{n}}^*)$ [19]
4: $u_i \leftarrow \begin{cases} \sum_{k=1}^{\hat{n}} \tilde{a}_k q_1 \cdots q_k \bmod D, \ |u_i| \leq D/2 & \text{, if } i = 0 \\ \sum_{k=1}^{\hat{n}} \tilde{a}_k q_1 \cdots q_k \bmod B, \ |u_i| \leq B/2 & \text{, otherwise} \end{cases}$ ▷ Output
5: $\hat{n} \leftarrow \text{argmin}_{\hat{n}} \{q_1 \cdots q_{\hat{n}} D B^i > q\}$
6: $(a_1^*, ..., a_{\hat{n}}^*) \leftarrow \begin{cases} \left((a_1^* - u_i)D^{-1}, ..., (a_{\hat{n}}^* - u_i)D^{-1}\right), \text{ if } i = 0 \\ \left((a_1^* - u_i)B^{-1}, ..., (a_{\hat{n}}^* - u_i)B^{-1}\right), \text{ otherwise} \end{cases}$
7: **end for**
8: **return** $(u_1, ..., u_l = a_{\hat{n}}^*)$ ▷ Note that u_0 is discarded

Correctness of Algorithm 4. Let $i = 0$ at Line 2. Since Line 3 produces an MRS number, it satisfies the following:

$$0 \leq a = \tilde{a}_1 + \tilde{a}_2 q_1 + \tilde{a}_3 q_1 q_2 + \cdots + \tilde{a}_{\hat{n}} q_1 \cdot \cdots \cdot q_{\hat{n}-1} < q. \tag{17}$$

Line 4 performs a modular reduction of the right-hand side of Eq. (17) by D or B, yielding the least significant $\log_2 D$ bits or $\log_2 B$ bits of u_i, respectively. Then, in Line 6, we compute $a - u \in \mathbb{Z}_q$, which is divisible by D or B, division by D or B can be replaced by multiplication by D^{-1} or B^{-1}. Finally, the output $(u_1, ..., u_l)$ clearly satisfies the following:

$$a = u_0 + \sum_{i=1}^{l} u_i D B^{i-1}. \tag{18}$$

The output $(u_1, \ldots, u_{l-1})$ of Algorithm 4 and the discarded value u_0 are in canonical form so that $|u_0| \leq D/2$ and $|u_1|, \ldots, |u_{l-1}| \leq B/2$. However, since the last output element u_l is $a^*_{\hat{n}}$, we cannot directly see that $|u_l| \leq B/2$. The following Lemma 6 shows that by adjusting D, it is possible to ensure that $|u_l| \leq B/2$.

Lemma 6 *Suppose that q and the gadget parameter (D, B, l) satisfy $q \leq DB^l(\epsilon - 0.5/B)$. Then, the output $(u_1, \ldots, u_l)$ of* Algorithm 4 *and discarded value u_0 satisfies $|u_0| \leq D/2$, $|u_i| \leq B/2$ for $i = 1, \ldots, l-1$ and $0 \leq u_l < \epsilon B$.*

The proof is included in Supplementary Material H.

Using Lemma 6, set $\epsilon = 0.5$ to achieve $u_l \leq B/2$. For a sufficiently large B, $\epsilon - 0.5/B \approx \epsilon$ and so approximately $q \leq 0.5DB^l$ is sufficient. That is, for D' and B satisfying $\lceil \log_2(q) \rceil = D'B^l$, if $D = 2D'$, one additional bit ensures that the last bit falls within $B/2$. Additionally, as will be confirmed in Section 5.2, setting D larger than B reduces the error after bootstrapping.

4.3 NTT_w and INTT_w

If a single prime factor q_i of $q = q_1 \cdots q_n$ does not satisfy $2d_2 \mid q_i - 1$, then NTT_1 cannot be performed. In such case, we employ NTT_w and INTT_w in Algorithms 6 and 7 of Supplementary Material B, which are modified from the implementation methods in [36].

4.4 MUL_w from Karatsuba Algorithms in the Montgomery Domain

After applying the NTT_w to a polynomial, the output is in the domain $\bigoplus_{i \in \mathbb{Z}^*_{2d/w}} \mathbb{Z}_q[X]/\langle X^w - \zeta^i_{2d/w} \rangle$. Therefore, in order to multiply two polynomials, one must perform d/w multiplications of two polynomials of degree at most $w-1$, and then obtain the remainder from dividing each result by the corresponding $X^w - \zeta^i_{2d/w}$. We use, Karatsuba polynomial multiplication is used as described in Algorithms 8 and 9 of Supplementary Material C.

5 MIMC-enabled Blind Rotation, Analysis of Bootstrapping Error, and Parameter Optimization

In this section, we propose a blind rotation for FHE16 based on NTT_w, INTT_w, Mul_w, and Decom. Specifically, the proposed blind rotation achieves performance gain when simultaneously rotating multiple ciphertexts and also enables MIMC operations that allow for different computation on each ciphertext.

5.1 MIMC-enabled Blind Rotation

In this section, we modify the existing GINX blind rotation in Algorithm 10 of Supplementary Material E as given in Algorithm 5, while demonstrating how MIMC operation is performed.

Algorithm 5 Proposed parallel bootstrapping of GINX blind rotation

Input: MLWE integer modulus $q = q_1 \cdots q_n$ where $q_i's$ are distinct primes, secret key alphabet U_t, L LWE ciphertexts $\mathsf{ct}_i = (a_{i1}, ..., a_{ik_1}, b_i)$, $i = 1, ..., L$, encrypted by the secret key $\mathsf{sk} = (s_1, ..., s_{k_1}) \in U_t^{k_1}$, accumulating polynomials acc_1,...,acc_L, and a GINX bootstrapping key BL

Output: L MLWE ciphertexts $\mathsf{MLWE}[\mathbf{acc}_i X^{\sum_j a_{ij}s_j - b_i}]$, $i = 1, ..., L$

1: $\mathsf{CT} \leftarrow \left[\mathsf{ct}_1 | ... | \mathsf{ct}_L\right]$ such that $\mathsf{ct}_i \leftarrow (0, .., 0, \mathsf{acc}_i(X)X^{-b_i})$
2: **for** $i \leftarrow 0; i < n; i \leftarrow i + 1$
3: $\quad \mathsf{CT}' \leftarrow \mathsf{Decom}(\mathsf{CT}) \in R_{q,d}^{L\times(l_a k_2 + l_b)}$ ▷ $\mathsf{CT}' = (\mathsf{CT}_1, ..., \mathsf{CT}_n)$ are saved as CRT representation
4: $\quad$ **for** $r = 1 : n$
5: $\quad\quad$ **for** $j = 1; j \leq L; j = j + 1$ ▷ Following lines are operated in $\mathbb{Z}_{q_r}$
6: $\quad\quad\quad \mathsf{CT}'_r \leftarrow \mathsf{NTT}_w(\mathsf{CT}'_r)$ ▷ Point 1
7: $\quad\quad\quad$ **for** all $u \in U_t$
8: $\quad\quad\quad\quad \mathsf{CT}_r \leftarrow \mathsf{MUL}_w(\mathsf{MUL}_w(\mathsf{CT}'_r, (\mathsf{BL}_{i,u})_r),\ X^{a_{ij}u} - 1)$ ▷ Point 2
9: $\quad\quad\quad$ **end for**
10: $\quad\quad\quad \mathsf{CT}_r \leftarrow \mathsf{INTT}_w(\mathsf{CT}_r) \in R_{q_r,d}^{L\times(k_2+1)}$ ▷ Point 3
11: $\quad\quad$ **end for**
12: $\quad$ **end for** ▷ Sync point so that $\mathsf{CT} = (\mathsf{CT}_1, ..., \mathsf{CT}_n)$
13: **end for**
14: **return** CT

Algorithm 5 performs blind rotation using L LWE ciphertexts ct_i and their corresponding accumulating polynomials acc_i. The external products $\boxdot$ is made up of NTT_w, INTT_w, MUL_w, and Decom corresponding to Lines 3-12. While the existing Algorithm 10 performs blind rotation separately for each ciphertext ct_i, Algorithm 5 reorders the for-loop so that all ciphertexts are multiplied by the same $\mathsf{BL}_{i,u}$ in Line 8. This reordering does not change the number of operations but, from a hardware perspective, $\mathsf{BL}_{i,u}$ is loaded into cache memory and reused

repeatedly, reducing memory access frequency and improving the speed of blind rotation.

Furthermore, Line 4 shows that the for-loop over the integer modulus $q = q_1 \cdots q_n$ splits the computations across n factor, q_i. The GINX blind rotation in Algorithm 10 performs parallel processing by dividing unit operations at Points 1, 2, and 3, which is the only GPU acceleration strategy for blind rotation [38]. However, Algorithm 5 performs all computations using only $\mathbb{Z}_{q_r}$ operations, within the for-loop at Line 4, allowing independent computations across different $\mathbb{Z}_{q_r}$. As long as the sync point at Line 12 aligns when the operations over each $\mathbb{Z}_{q_r}$ are completed, the correct results are obtained, which enables better acceleration on GPUs, utilizing multiple streaming multiprocessors to further speedup. In the next section, we derive accurate model for the errors that arise after bootstrapping and propose parameter optimization techniques that reduce key size while improving the bootstrapping speed.

5.2 Bootstrapping Error Analysis

In this section, we calculate the size of the error e obtained after bootstrapping and the size of the BL key required for bootstrapping. First, we introduce the error model from [17, 26] and provide accurate calculation results when two gadget parameters (B_2^A, l_2^A) and (B_2^B, l_2^B) are used for the MGSW ciphertext. Finally, we derive the parameters for the experiments in Section 6.

To begin with an error model in [17, 26], we assume Heuristic 1 for the external product $\boxdot$.

Heuristic 1 *The result of* Decom *performed on an* MLWE *ciphertext is independent and uniformly random.*

Under Heuristic 1, the variance of each Decom output is $B^2/12$, leading to the following error variance:

$$l_2^A k_2 d \left(\frac{(B_2^A)^2}{12} \right) \sigma_{\mathsf{bl}}, \tag{19}$$

where Eq. (19) is derived under the condition that $D_2^A = D_2^B = B_2^A = B_2^B$. Now, we recalculate the error variance when D_2^A, D_2^B, B_2^A, and B_2^B are all different. Let the internal error in $\mathsf{MLWE}[\Delta \boldsymbol{m}_1]$ be denoted as $\boldsymbol{e} \in R_{q,d}$, and the internal error in $\mathsf{MGSW}[\boldsymbol{m}_2]$ be denoted as $(\vec{\boldsymbol{f}}^{(1)}, \vec{\boldsymbol{f}}^{(2)}) \in R_{q,d}^{l_2^A k_2} \times R_{q,d}^{l_b}$. After performing the external product, the resulting error variance is known as follows [25].

$$\sum_{i=1}^{l_2^A k_2} \boldsymbol{u}_i^a \boldsymbol{f}_i^{(1)} + \sum_{i=1}^{l_b} \boldsymbol{u}_i^b \boldsymbol{f}_i^{(2)} + \sum_{i=1}^{k_2} \boldsymbol{v}_i^a \mathsf{sk}_i + \boldsymbol{v}^b + \boldsymbol{m}_2 \boldsymbol{e}. \tag{20}$$

In Eq. (20), $\boldsymbol{u}_i^a$ and $\boldsymbol{u}_i^b$ are the Decom outputs for $\boldsymbol{a}_i$ and $\boldsymbol{b}$ parts of $\mathsf{MLWE}[\Delta \boldsymbol{m}_1]$, respectively, which are bounded by $B_2^A/2$ and $B_2^B/2$. $\boldsymbol{v}_i^a$ and $\boldsymbol{v}^b$ are the discarded

elements from $\boldsymbol{a}_i$ and $\boldsymbol{b}$ parts of Decom outputs, which are bounded by $D_2^A/2$ and $D_2^B/2$, respectively. Thus, when $\boldsymbol{m}_2$ is a zero polynomial and Heuristic 1 is applied to Eq. (20), the variance in Eq. (20) becomes as follows [5].

$$\sigma_{\boxdot}^2 = \left[\frac{k_2 d}{12}\Big(l_2^A (B_2^A)^2 \sigma_{\mathsf{bl}}^2 + D_2^A \sigma_{\mathsf{sk}_2}^2\Big) + \frac{1}{12}\Big(l_2^B d (B_2^B)^2 \sigma_{\mathsf{bl}}^2 + D_2^B\Big)\right]. \tag{21}$$

It is known that the variance of the error after performing the GINX blind rotation is $\sigma_{\mathsf{acc},\mathsf{GINX}}^2 = k_1 \sigma_{\boxdot}^2$ [26]. For the AP^+ blind rotation, since automorphism keyswitching is performed $d(1-(1-v^{-1})\exp(-k_1/d))$ times [26], the following additional term is added:

$$\sigma_{\mathsf{acc},\mathsf{AP}^+}^2 = k_1 \sigma_{\boxdot}^2 + k_1 d\big(1-(1-v^{-1})\exp(-k_1/d)\big)\sigma_{\mathsf{aut}}^2,$$

where σ_{aut}^2 is the variance of the error in the automorphism key AK. Therefore, the error variance σ_{acc}^2 varies depending on the blind rotation method.

Let us consider two terms $l_2^A (B_2^A)^2 \sigma_{\mathsf{bl}}^2$ and $D_2^A \sigma_{\mathsf{sk}_2}^2$) in Eq. (21). Then, since we select $l_2^A \geq 2$ and $\sigma_{\mathsf{bl}}^2 \geq \sigma_{\mathsf{sk}_2}$, the impact of B_2^A on $\sigma_{\boxdot}^2$ is greater than that of D_2^A. Therefore, if we choose the parameters such that $D_2^A + l_2^A B_2^A \approx \log_2(q_2)$, the error can be reduced by setting D_2^A slightly larger than or equal to B_2^A. This effect stand out even more for D_2^B and B_2^B. In practice, we select the parameters such that $d \geq 2^9$ and $D_2^B + l_2^B B_2^B \approx \log_2(q)$, making significant increase of D_2^B so that the $\boldsymbol{b}$ part of the MLWE ciphertext can be approximated with a small number of bits while experiencing minimal error amplification. Thus, the proposed error model in Eq. (21) allows for more precise and compact parameter tuning than the previous model in Eq. (19).

Finally, the total error variance model after bootstrapping is as follows [26][6].

$$\sigma_{\mathsf{tot}}^2 = \frac{(2d)^2}{q_2^2} \cdot 2\sigma_{\mathsf{acc}}^2 + \frac{(2d)^2}{q_1^2}(\sigma_{\mathsf{ks1}}^2 + \sigma_{\mathsf{ms1}}^2) + \sigma_{\mathsf{ms2}}^2. \tag{22}$$

Here, σ_{acc}^2 is either from $\sigma_{\mathsf{acc},\mathsf{GINX}}^2$ or $\sigma_{\mathsf{acc},\mathsf{AP}^+}^2$ depending on the blind rotation, $\sigma_{\mathsf{ks1}}^2 = \sigma_{\mathsf{ks}}^2 k_2 d_2 l_{\mathsf{ks}}$ is the variance of the error from the $\mathsf{KeySwitching}$ [7], and $\sigma_{\mathsf{ms1}}^2 = \frac{||\mathsf{sk}_2||_2^2+1}{12}$ and $\sigma_{\mathsf{ms2}}^2 = \frac{||\mathsf{sk}_1||_2^2+1}{12}$ are the errors when the integer modulus changes from q_2 to q_1 and from q_1 to $2d$, respectively, as in Fig. 2. $||\mathsf{sk}_1||_2$ and $||\mathsf{sk}_2||_2$ are assumed to be $\sqrt{k_2 d/2}$ and $\sqrt{k_1/2}$ for binary or ternary support, and $\sqrt{k_2 d\sigma^2}$ and $\sqrt{k_1\sigma^2}$ for discrete Gaussian distribution with variance σ^2 [26].

Suppose that a t-bit message representing one of 2^t symbols is encrypted in an LWE ciphertext with modulus $2d$. If we set the symbol distances to their maximum, 2^t messages are mapped to $0d/2^t$, $2d/2^t$, $4d/2^t$, ..., $(2^t-1)2d/2^t$, and hence the size of error in the ciphertext must be smaller than half the distance

[5] If the support size $t = |U_t|$ of sk_1 is not 2, the variance increases by a factor of $4(|U_t|-1)$ [25, 26].

[6] In [26], the modulus switching parameter is changed from q_1 to q, but for simplicity, we fix $q = 2d$ in this paper.

[7] In $\mathsf{TFHE\text{-}rs}$, the error increases by $\sigma_{\mathsf{ks1}}^2 B_{\mathsf{ks}}^2$, but the key size is reduced by $1/B_{\mathsf{ks}}$ [12].

between symbols, i.e. $d/2^t$. The probability of bootstrapping failure, calculated based on this, is as follows [17, 26]:

$$p_{\mathsf{fail}} = \mathsf{erfc}\left(\frac{d}{2^t\sqrt{2}\sigma_{\mathsf{tot}}}\right), \tag{23}$$

where erfc is the complementary error function.

Note on FFT-based GINX blind rotation. On the other hand, when performing polynomial multiplication using double-precision FFT, approximation errors arise. Since the precision is 53 bits for double precision, using an integer modulus 2^{64} inevitably leads to the loss of $64 - 53 = 11$ bits of information. In [8], the variance of the error from FFT is experimentally obtained as follows.

$$\sigma^2_{\mathsf{acc}} = k_1 2^{2(64-53)-2.6} l_2^A (B_2^A)^2 d^2 (k_2 + 1), \tag{24}$$

where the exponent 2.6 is an empirically selected compensation factor. Since the gadget parameters l_2^A and B_2^A, which control the error magnitude, are positive integers, Eq. (24) shows a minimum error floor of $k_1 k_2 2^{19.4} d^2$ when both values are set to 1. In Section 6.3, we explain the reason why Eq. (24) is a primary reason that multiple message cannot be packed into a single ciphertext with TFHE-rs contrary to FHE16.

5.3 Parameter Optimization Strategy: OPENFHE, TFHE-rs, and FHE16

In this section, we examine how the FHE parameters in Table 1 are related to the bootstrapping speed, key size, and security level. We also compare the parameter selection strategies of OpenFHE and TFHE-rs with the criteria for selecting the parameters of FHE16.

Given cryptographic parameters (k_1, q_1) and (k_2, d, q_2) that satisfy the (computational) security level λ bits, the blind rotation speed is proportional to the number of NTT_w operations in the external product, that is $k_1(l_2^A k_2 + l_2^B)$. Therefore, optimization of (k_1, q_1) and (k_2, d, q_2) involves minimizing l_2^A and l_2^B, while meeting the minimum requirement of p_{fail} in Eq. (23).

On the other hand, the blind rotation key size (bits) for GINX and AP$^+$ is calculated from parameters as follows.

$$\begin{aligned}\mathsf{SIZE}_{\mathsf{BL-GINX}} =& k_1 \lceil \log(q_2) \rceil d(k_2 + 1)(l_2^A k_2 + l_2^B) \\ \mathsf{SIZE}_{\mathsf{BL-AP^+}} =& \mathsf{SIZE}_{\mathsf{BL-GINX}} + (v+1)\lceil \log(q_2) \rceil d(k_2+1)(l_2^A k_2).\end{aligned} \tag{25}$$

Table 2 lists the parameter values of OpenFHE v1.2 and TFHE-rs v0.8, alongside the proposed parameter values for FHE16. Note that the parameters in Table 2 are also used for experiments in Section 6.

The security parameter λ is obtained by using the LWE estimator [2], assuming the BKZ, which is currently the fastest known method to attack lattice-based cryptography. The Gram-Schmidt norm of the reduced basis in BKZ is assumed

-	LWE						MLWE									λ(bits)
	sk_1	k_1	q_1	B_1	l_1	σ_{ks}	sk_2	d	k_2	q_2	B_2^A	B_2^B	l_2^A	l_2^B	σ_{bl}	
$\mathsf{p}_G^{\mathsf{Open}}$	T	503	14	2^5	3	3.19	T	1024	1	27	2^9	2^9	2	2	3.19	120.9
$\mathsf{p}_A^{\mathsf{Open}}$	3.19	447	14	2^5	3	3.19	3.19	1024	1	28	2^{10}	2^{10}	2	2	3.19	126.5
$\mathsf{p}_G^{\mathsf{TFHE}}$	B	811	32	2^3	5	$2^{14.47}$	B	512	3	32	2^{10}	2^{10}	2	2	4.0	132.0
$\mathsf{p}_G^{\mathsf{FHE16}}$	B	585	14	2^5	3	3.19	Q	512	2	28	2^9	2^{10}	2	1	3.59	128.2
$\mathsf{p}_A^{\mathsf{FHE16}}$	2.19	472	14	2^5	3	3.19	1.15	512	2	28	2^9	2^{10}	2	1	3.19	128.4

Table 2: FHE parameters of OpenFHE, TFHE-rs, and FHE16. B,T, and Q stands for binary, ternary, and quinary supports, respectively, and real numbers in sk stands for standard deviation of discrete Gaussian distribution

to follow the geometric series assumption (GSA) [35], and the cost of BKZ algorithm is computed either using the Core-SVP model [3] or the list-decoding enhanced enumeration-based cost [30] [8]. Note that in this paper, we use the latter, more recent method, and determine the parameter values to ensure $p_{\mathsf{fail}} < 2^{-32}$, which is adapted for bootstrapping speed competition in many FHEs [27, 39].

In Table 2, $\mathsf{p}_G^{\mathsf{Open}}$ and $\mathsf{p}_A^{\mathsf{Open}}$ refer to the parameters of OpenFHE for GINX and AP$^+$, respectively, while $\mathsf{p}_G^{\mathsf{TFHE}}$ refers to the GINX parameters of TFHE-rs. In FHE16, $\mathsf{p}_G^{\mathsf{FHE16}}$ stands for GINX, and $\mathsf{p}_A^{\mathsf{FHE16}}$ stands for AP$^+$. The modulus q_2 in FHE16 is set as $p_{14}p_{25}$ from Table 9 in Supplementary Material A, which allows NTT_1 operations with $d = 512$ while ensuring that $4p_{14}, 4p_{25} < 2^{16}$, preventing overflow even after four additions. Thus, $q_2 = p_{14}p_{25}$ is used as a default, unless stated otherwise.

OpenFHE minimizes key size by reducing q_1 and q_2, but only considers $l_2^A = l_2^B$, $k_2 = 1$, and the same support for both sk_1 and sk_2. In contrast, TFHE-rs fixes q_1 and q_2 to machine-friendly values of 2^{32} or 2^{64}, which leads to an increase of k_2 in order to meet the security level λ, resulting in larger key sizes.

FHE16, by setting different supports for sk_1 and sk_2, can achieve a security level of at least 128 bits, while keeping sk_1 binary under the GINX blind rotation to improve the computational speed. Also, by choosing $l_2^A > l_2^B$ and reducing l_2^B, the number of external product operations is reduced, which improves the bootstrapping speed and reduces the BL key size.

In the next section, we conduct various experiments and comparison of the bootstrapping performance of FHE schemes.

6 Experiments

In this section, we experimentally analyze the bootstrapping speeds of OpenFHE, TFHE-rs, and FHE16. Table 3 lists the specifications of the machines used for experiments.

[8] In the LWE estimator, the former can be computed using Estimate.rough(), while the latter uses Estimate().

Index	CPU frequency	L1	L2	L3	Memory BW
$\mathsf{computer}_1$	3.2GHz	32KiB	1MiB	30.25MiB(shared)	7.6GB/s
$\mathsf{computer}_2$	1.1GHz	32KiB	256KiB	6MiB(shared)	12.2GB/s

Table 3: Machine specification

$\mathsf{computer}_1$ is a server computer equipped with Intel(R) Xeon(R) Gold 6240R CPU @ 2.40GHz, and $\mathsf{computer}_2$ is a laptop equipped with Intel(R) i5-8260U CPU (1.6GHz). The CPU frequency, cache size, and memory bandwidth were measured directly. The average of CPU frequency was measured by running the turbo mode using the Linux program Stress under load. $\mathsf{computer}_1$ supports both AVX2 and AVX512, whereas $\mathsf{computer}_2$ supports only AVX2. By using these two machines, we can assess the impact of machine performance on algorithm efficiency.

6.1 FHE16 vs OpenFHE

The bootstrapping speeds of OpenFHE using the parameters $\mathsf{p}_G^{\mathsf{Open}}$ and $\mathsf{p}_A^{\mathsf{Open}}$ are presented in Table 4. Also, the bootstrapping speeds of FHE16 using the same parameters $\mathsf{p}_G^{\mathsf{Open}}$ and $\mathsf{p}_A^{\mathsf{Open}}$ are presented with the only difference of using $q_2 = p_{14}p_{25}$ from Supplementary Material A, Table 9.

(millisecond)	AVX2 ($\mathsf{computer}_1$,$\mathsf{computer}_2$)	AVX512 ($\mathsf{computer}_1$)
OpenFHE GINX	(35, 66)	23
FHE16 GINX	(7.65, 8.85)	5.3
OpenFHE AP^+	(34, 72)	20
FHE16 AP^+	(7.54, 9.68)	5.08

Table 4: Comparison of bootstrapping time: FHE16 and OpenFHE with $\mathsf{p}_G^{\mathsf{Open}}$ and $\mathsf{p}_A^{\mathsf{Open}}$

Table 4 shows that AP^+ is generally slightly faster than GINX method, which aligns with the experimental results in [26], where the GINX method was slower when using a ternary secret key for LWE ciphertexts. Table 4 also shows that even when using the same parameters as OpenFHE, FHE16 achieves bootstrapping speed up to 4 times faster than OpenFHE. A notable observation is that, compared to OpenFHE whose performance highly depends on CPU speed and cache memory size, FHE16 shows a little degradation across different machines.

6.2 FHE16 vs TFHE-rs when $q_2 = 2^{32}$

Table 5 compares the bootstrapping times between TFHE-rs and FHE16 using the same parameter $\mathsf{p}_G^{\mathsf{TFHE}}$, except $q_2 = p_{14}p_{25}$ is used for FHE16. From Table 5, we can see that FHE16 is 1.4 times faster with AVX512 and up to 1.18 times faster with AVX2, compared to TFHE-rs.

(millisecond)	AVX2 ($\mathsf{computer}_1$,$\mathsf{computer}_2$)	AVX512 ($\mathsf{computer}_1$)
TFHE-rs	(16.6, 17)	14
FHE16	(14, 15.1)	10

Table 5: Comparison of bootstrapping time: FHE16 and TFHE-rs with $\mathsf{p}_G^{\mathsf{TFHE}}$

Comparison of bootstrapping time between FHE16 vs TFHE-rs: when d_2 is higher than 512. In order to perform encryption on multiple messages, both TFHE-rs and FHE16 should increase d_2, which is crucial for implementing homomorphic integer arithmetic with large message bits. Therefore, to compare the bootstrapping time between TFHE-rs and FHE16 for various d_2, we conducted experiments using the same parameter $\mathsf{p}_G^{\mathsf{TFHE}}$, while adjusting d_2. To minimize w in NTT_w, we selected $q_2 = p_{24}p_{25}$ from Table 9 in Supplementary Material A.

(millisecond)	$d_2 = 1024$	$d_2 = 2048$	$d_2 = 4096$
TFHE-rs	30	65	131
FHE16	22	57	131

Table 6: Comparison of bootstrapping time comparison: FHE16 and TFHE-rs for various d_2

In Table 6, the bootstrapping time of TFHE-rs increases approximately linearly with d_2 [9]. However, FHE16 shows steeper increase since NTT_2 is performed for $d_2 = 2048$ and NTT_4 is performed for $d_2 = 4096$. Note that, if $d \leq 4096$ FHE16 is not slower than TFHE-rs for two primes i.e., $n = 2$.

Bootstrapping time comparison model between FHE16 and TFHE-rs. In the next section, we discuss the bootstrapping speed between FHE16 and TFHE-rs when q_2 is approximately 64 bits. Before the next section, we propose a metric for the speed ratio between FHE16 and TFHE-rs based on changes in q_2 and d_2, and aim to measure the performance of both FHE when parameters

[9] Although the complexity of FFT operations is $O(d_2 \log d_2)$, the impact of $\log d_2$ appears minimal up to $d_2 = 4096$.

are increased.

$$\big[d_2(l_2^A k_2 + l_2^B) \text{ using } \mathsf{TFHE\text{-}rs} \text{ values}\big]^{-1} \times d_2(l_2^A k_2 + l_2^B)\frac{n}{2} \text{ using } \mathsf{FHE16} \text{ values}\big]. \tag{26}$$

The Eq. (26) is based on the following three experimental justification: (i) When the number of primes comprising q_2 is 2, the results in Table 6 show that FHE16 is not slower than TFHE-rs for $d_2 \leq 4096$. Since TFHE-rs uses double-precision FFT, the number of external product operations does not increase, whereas in FHE16, the number of external products increases linearly with the number of primes n. Therefore, we expect that FHE16 would be $n/2$ times slower than TFHE-rs hence when $n = 2$, FHE16 is not slower than TFHE-rs; (ii) In Table 6, the impact of d_2 is linear until $d_2 \leq 4096$; (iii) The number of NTT_w operations is $(l_2^A k_2 + l_2^B)$, which is the most time-consuming operation in bootstrapping.

6.3 FHE16 vs TFHE-rs with $q_2 \approx 2^{64}$

In this section, we compare FHE16 and TFHE-rs with $q_2 \approx 2^{64}$, in terms of implementing high-precision integer homomorphic operation [14]. When implementing integer homomorphic operations, it is crucial to choose a modulus large enough to support ciphertext multiplication. It is known that the error after multiplication is given by Eq. (27) [14] [10].

$$\sigma^2_{\mathsf{mul}} = \Omega\bigg(2^{2t} q_2^{-2} d\sigma^4_{\mathsf{tot}} + k_2 d^2 2^{2t} \sigma^2_{\mathsf{tot}}\bigg). \tag{27}$$

Hence, the Eq. (27) is used for ensuring the correctness of multiplying t-bits message homomorphically, by using the following equation $\mathsf{erfc}\bigg(\frac{q_2}{2^{t+1}\sqrt{2}\sigma_{\mathsf{mul}}}\bigg) < 2^{-32}$.

To compare FHE16 and TFHE-rs, we select parameters message4carry4pbs16 used in TFHE-rs and Propose1-Propose8 used in FHE16 as in Table 7. Here, t is the message bits being multiplied.

For Propose1, the execution time is similar to that of TFHE-rs, but the number of message bits t for multiplication of FHE16 is one bit less. For Propose2, t is the same as TFHE-rs, but the execution time is expected to be 1.2 times longer. However, like Propose3 and 4, FHE16 can adjust parameters to increase value of t, whereas TFHE-rs cannot select $t > 7$ due to the error floor caused by FFT.

Furthermore, FHE16 can encrypt more message bits than TFHE-rs by adjusting both d_2 and k_2 as seen in Propose5 through Propose8. Such flexible parameter choices of FHE16 are critical to FHE. For example, in the case of 16-bit integer homomorphic multiplication, the TFHE-rs requires splitting the 16-bit integer

[10] We calculate only the order of worst-case errors from Eq. (1) of Theorem 1 in [14]. Although the number of message bits t is conservatively estimated, it is sufficient for performance comparison.

param idx	sk_2	q_2	B_2^A	B_2^B	l_2^A	l_2^B	σ_{bl}	t	S	$\mathsf{BL}_{\mathsf{size}}$	λ
message4carry4pbs16	B	2^{64}	2^9	2^9	4	4	$2^{11.9}$	7	-	-	123.9
Propose1	Q	$2^{55.36}$	2^{14}	2^{16}	2	2	7.4	6	1.00x	0.43x	128.0
Propose2	Q	$2^{55.36}$	2^{14}	2^{12}	3	3	7.4	7	1.50x	0.65x	128.0
Propose3	Q	$2^{55.36}$	2^{11}	2^{12}	4	3	7.4	8	1.75x	0.76x	128.0
Propose4	Q	$2^{55.36}$	2^9	2^{10}	5	4	7.4	9	2.25x	0.98x	128.0
Propose5($d = 2^{10}, k_2 = 3$)	6.9	$2^{84.25}$	2^{21}	2^{21}	2	1	9.8	11	1.31x	2.32x	128.6
Propose6($d = 2^{10}, k_2 = 3$)	6.9	$2^{84.25}$	2^{21}	2^{21}	2	2	9.8	14	1.50x	2.65x	128.6
Propose7($d = 2^{10}, k_2 = 3$)	6.9	$2^{84.25}$	2^{21}	2^{21}	3	2	9.8	16	2.06x	3.65x	128.6
Propose8($d = 2^9, k_2 = 3$)	Q	$2^{41.85}$	2^9	2^{10}	5	4	6.39	5	0.89x	3.1x	128.2

Table 7: Comparison of bootstrapping time: ProposeX for FHE16 and message4carry4pbs16 for TFHE-rs when $q_2 \gg 2^{32}$. t is the number of valid message bits after blind rotation and multiplication [14]. S is a calculated ratio of expected execution time in Eq. (26). $\mathsf{BL}_{\mathsf{size}}$ is the relative key size via Eq. (25). $d = 2^{11}$ and $k_2 = 1$ except Propose6 and Propose7. q_2 is chosen $p_{20}p_{21}p_{24}p_{25}$, $p_{20}p_{21}p_{22}$, and $p_{20}p_{21}p_{22}p_{23}p_{24}p_{25}$ for Propose1-5, Propose6-8, and Propose9 , respectively.

into three ciphertexts, each of which holding a 7-bit message and hence to implement a 16-bit multiplier can be implemented using three 7-bit multipliers. In contrast, Propose7 can directly handle a 16-bit integer in a single ciphertext, enabling immediate 16-bit multiplication. Thus, FHE16, which avoids FFT errors, allows for more efficient implementation of homomorphic integer arithmetic across various precision level.

FHE16 **vs NTRU-based FHE.** We also compare FHE16 and NTRU-based FHE based on experimental results listed in Supplementary Material I.

6.4 Parameter-Optimized FHE16: MIMC and Multi-threading

We compare the bootstrapping time and bootstrapping key BL size using $\mathsf{p}_G^{\mathsf{Open}}$ and $\mathsf{p}_A^{\mathsf{Open}}$ for OpenFHE, $\mathsf{p}_G^{\mathsf{FHE16}}$ for TFHE- rs, and $\mathsf{p}_G^{\mathsf{FHE16}}$ and $\mathsf{p}_A^{\mathsf{FHE16}}$ for FHE16.

Table 8 shows that FHE16 is 3.3 times faster than OpenFHE when using AP^+ method, 6.5 times faster than OpenFHE and 4 times faster than TFHE-rs when using GINX method is used. The reason why FHE16 outperforms OpenFHE using GINX method is that the parameters have been optimized to allow sk_1 to have binary support.

The GINX BL key size of FHE16 is almost half that of OpenFHE because OpenFHE uses ternary support for sk_1, while FHE16 uses binary support. When using AP^+ method, the BL key size of FHE16 is slightly smaller than that of OpenFHE because FHE16 uses gadget parameters $l_2^A > l_2^B$, which results in a smaller bootstrapping key. Finally, the BL key size of TFHE-rs is 3.8 times larger

-	OpenFHE	TFHE-rs	FHE16
Time (GINX, ms)	23	14	3.5
Time (AP^+, ms)	20	-	6.0
-	OpenFHE	TFHE-rs	FHE16
BL Key size (GINX, MiB)	26.4	55	14.4
BL Key size (AP, MiB)	12.3	-	11.8

Table 8: Comparision of boostrapping time using $\mathsf{computer}_1$ and BL key size.

than that of FHE16 because TFHE-rs sets the integer modulus q_2 to the word size 2^{32}.

MIMC Performance. Fig. 3 shows the average amortized execution time for performing bootstrapping on 100L ciphertexts for averaging amortized time simultaneously using the proposed blind rotation in Algorithm 5 with MIMC parameter L.

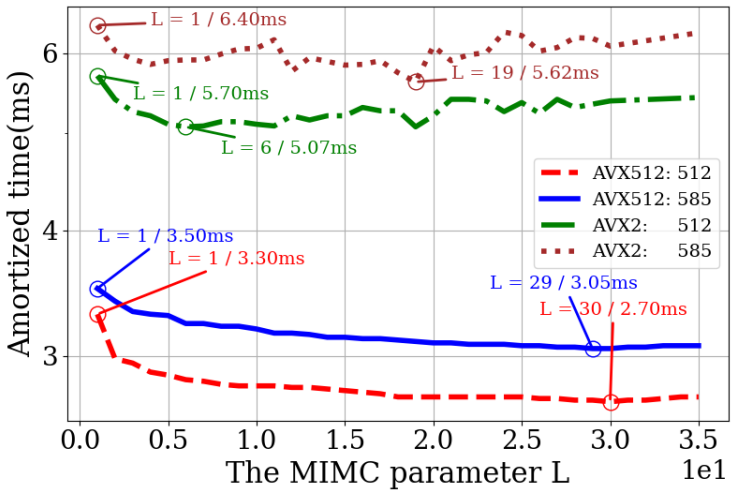

Fig. 3: Amortized bootstrapping time with L MIMC. AVX512 is used with $\mathsf{computer}_1$ and AVX2 is used with $\mathsf{computer}_2$.

In Fig. 3, the amortized the execution time for L= 29 $\sim$ 30 is 1.14x-1.2x faster than L= 1. Since the size of cache memory of $\mathsf{computer}_2$ is smaller than $\mathsf{computer}_1$, the AVX2 amortized execution time of MIMC does not converge well even after averaging over many ciphertexts. Nevertheless, it demonstrates that MIMC can improve the bootstrapping speed by up to a factor of 1.14x even if the cache size is small.

7 Conclusions and Future Works

Conclusion. This paper proposed FHE16, an FHE system using only 16-bit arithmetic operations. We showed that FHE16 outperforms current gate-based FHE, OpenFHE and TFHE-rs. Also, we confirmed that FHE16 is not only faster in gate operations but also capable of encrypting more messages than TFHE-rs. Moreover, the blind rotation key size of FHE16 is small due to parameter optimization. **Future Work 1. Implementing Homomorphic Integer Operations.** Since FHE16 can handle more messages than TFHE-rs, fewer bootstrappings are possible when implementing homomorphic integer arithmetic with various precision. Therefore, the development of optimized homomorphic integer operations based on FHE16 will be crucial as future research.

Future Work 2. GPU and Hardware Acceleration. The proposed blind rotation in Algorithm 5 uses a composite modulus $q_2 = q_1 \cdots q_n$, allowing parallelization across $\mathbb{Z}_{q_i}$ operations in NTT_w, MUL_w, and INTT_w. This opens up opportunities for optimized GPU implementations, where threads can be grouped by q_i, with each thread performing the corresponding $\mathbb{Z}_{q_i}$ operations, similar to the optimization strategies in [38]. Furthermore, since FHE16 avoids floating-point operations such as FFT and relies entirely on 16-bit arithmetic, it is well-suited for hardware implementations on ASICs or FPGAs. Therefore, high-density designs with multiple 16-bit ALUs are promissing areas, low power consumption and small area.

References

1. Albrecht, M.R., Davidson, A., Deo, A., Gardham, D.: Crypto dark matter on the torus: Oblivious prfs from shallow prfs and tfhe. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 447–476. Springer (2024)
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)
3. Alkim, E., Ducas, L., Pöppelmann, T., Schwabe, P.: Post-quantum key exchange—a new hope. In: 25th USENIX Security Symposium (USENIX Security 16). pp. 327–343 (2016)
4. Attema, T., Lyubashevsky, V., Seiler, G.: Practical product proofs for lattice commitments. In: Annual International Cryptology Conference. pp. 470–499. Springer (2020)
5. Badawi, A.A., Alexandru, A., Bates, J., Bergamaschi, F., Cousins, D.B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., Liu, Z., Micciancio, D., Pascoe, C., Polyakov, Y., Quah, I., R.V., S., Rohloff, K., Saylor, J., Suponitsky, D., Triplett, M., Vaikuntanathan, V., Zucca, V.: OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915 (2022), https://eprint.iacr.org/2022/915, https://eprint.iacr.org/2022/915
6. Bae, Y., Cheon, J.H., Kim, J., Stehlé, D.: Bootstrapping bits with ckks. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 94–123. Springer (2024)

7. Belorgey, M.G., Carpov, S., Gama, N., Guasch, S., Jetchev, D.: Revisiting key decomposition techniques for fhe: Simpler, faster and more generic. Cryptology ePrint Archive (2023)
8. Bergerat, L., Boudi, A., Bourgerie, Q., Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Parameter optimization and larger precision for (t) fhe. Journal of Cryptology **36**(3), 28 (2023)
9. Bernstein, D., Sorenson, J.: Modular exponentiation via the explicit chinese remainder theorem. Mathematics of Computation **76**(257), 443–454 (2007)
10. Bonte, C., Iliashenko, I., Park, J., Pereira, H.V., Smart, N.P.: Final: faster fhe instantiated with ntru and lwe. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 188–215. Springer (2022)
11. Boura, C., Gama, N., Georgieva, M., Jetchev, D.: Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. Journal of Mathematical Cryptology **14**(1), 316–338 (2020)
12. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)
13. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In: Cyber Security Cryptography and Machine Learning: 5th International Symposium. pp. 1–19. Springer (2021)
14. Chillotti, I., Ligier, D., Orfila, J.B., Tap, S.: Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for tfhe. In: Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security. pp. 670–699. Springer (2021)
15. Chung, C.M.M., Hwang, V., Kannwischer, M.J., Seiler, G., Shih, C.J., Yang, B.Y.: Ntt multiplication for ntt-unfriendly rings: New speed records for saber and ntru on cortex-m4 and avx2. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 159–188 (2021)
16. Conrad, K.: Cyclotomic extensions. Preprint (2015)
17. Ducas, L., Micciancio, D.: Fhew: bootstrapping homomorphic encryption in less than a second. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 617–640. Springer (2015)
18. Ducas, L., van Woerden, W.: Ntru fatigue: how stretched is overstretched? In: Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security. pp. 3–32. Springer (2021)
19. Gbolagade, K.A., Cotofana, S.D.: An o (n) residue number system to mixed radix conversion technique. In: 2009 IEEE International Symposium on Circuits and Systems. pp. 521–524. IEEE (2009)
20. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing. pp. 169–178 (2009)
21. Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference. pp. 75–92. Springer (2013)
22. Halevi, S., Polyakov, Y., Shoup, V.: An improved rns variant of the bfv homomorphic encryption scheme. In: Topics in Cryptology–CT-RSA 2019: The Cryptographers' Track at the RSA Conference 2019, San Francisco, CA, USA, March 4–8, 2019, Proceedings. pp. 83–105. Springer (2019)
23. Higham, N.J.: The accuracy of floating point summation. SIAM Journal on Scientific Computing **14**(4), 783–799 (1993)

24. Kim, A., Lee, Y., Deryabin, M., Eom, J., Choi, R.: Lfhe: fully homomorphic encryption with bootstrapping key size less than a megabyte. Cryptology ePrint Archive (2023)
25. Lee, S., Shin, D.J.: Overflow-detectable floating-point fully homomorphic encryption. IEEE Access (2024)
26. Lee, Y., Micciancio, D., Kim, A., Choi, R., Deryabin, M., Eom, J., Yoo, D.: Efficient fhew bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 227–256. Springer (2023)
27. Li, Z., Lu, X., Wang, Z., Wang, R., Liu, Y., Zheng, Y., Zhao, L., Wang, K., Hou, R.: Faster ntru-based bootstrapping in less than 4 ms. IACR Transactions on Cryptographic Hardware and Embedded Systems **2024**(3), 418–451 (2024)
28. Liu, Z., Wang, Y.: Amortized functional bootstrapping in less than 7ms, with $\tilde{O}(1)$ polynomial multiplications. Cryptology ePrint Archive (2023)
29. Lyubashevsky, V., Seiler, G.: Short, invertible elements in partially splitting cyclotomic rings and applications to lattice-based zero-knowledge proofs. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 204–224. Springer (2018)
30. MATZOV: Report on the security of lwe: Improved dual lattice attack. Tech. rep., The Center of Encryption and Information Securiy (2022)
31. Muller, J.M., Brisebarre, N., De Dinechin, F., Jeannerod, C.P., Lefevre, V., Melquiond, G., Revol, N., Stehlé, D., Torres, S., et al.: Handbook of floating-point arithmetic, vol. 1. Springer (2018)
32. Nguyen, N.K., Seiler, G.: Greyhound: Fast polynomial commitments from lattices. In: Annual International Cryptology Conference. pp. 243–275. Springer (2024)
33. Nussbaumer, H.J., Nussbaumer, H.J.: The fast Fourier transform. Springer (1982)
34. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009)
35. Schnorr, C.P.: Lattice reduction by random sampling and birthday methods. In: STACS 2003: 20th Annual Symposium on Theoretical Aspects of Computer Science. pp. 145–156. Springer (2003)
36. Seiler, G.: Faster avx2 optimized ntt multiplication for ring-lwe lattice cryptography. Cryptology ePrint Archive (2018)
37. Smart, N.P.: Practical and efficient fhe-based mpc. In: IMA International Conference on Cryptography and Coding. pp. 263–283. Springer (2023)
38. Wei, D., Gizem, S.C., Bogdan, O., Kitsu: Cuda-accelerated fully homomorphic encryption library (2019), https://github.com/username/repository
39. Xiang, B., Zhang, J., Deng, Y., Dai, Y., Feng, D.: Fast blind rotation for bootstrapping fhes. In: Annual International Cryptology Conference. pp. 3–36. Springer (2023)
40. Zama: TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data (2022), https://github.com/zama-ai/tfhe-rs

Supplementary Material

A List of primes whose values are less than 2^{16}

idx	p	m	θ	$\lceil \log_2(p) \rceil$	idx	p	m	θ	$\lceil \log_2(p) \rceil$	idx	p	m	θ	$\lceil \log_2(p) \rceil$
p_1	257	8	1	9bit	p_{10}	26881	8	105	15bit	p_{19}	32257	9	63	15bit
p_2	769	8	3	10bit	p_{11}	30977	8	121	15bit	p_{20}	13313	10	13	14bit
p_3	3329	8	13	12bit	p_{12}	31489	8	123	15bit	p_{21}	15361	10	15	14bit
p_4	7937	8	31	13bit	p_{13}	7681	9	15	13bit	p_{22}	19457	10	19	15bit
p_5	9473	8	37	14bit	p_{14}	10753	9	21	14bit	p_{23}	25601	10	25	15bit
p_6	14081	8	55	14bit	p_{15}	11777	9	23	14bit	p_{24}	18433	11	9	15bit
p_7	14593	8	57	14bit	p_{16}	17921	9	35	15bit	p_{25}	12289	12	3	14bit
p_8	22273	8	87	15bit	p_{17}	23041	9	45	15bit					
p_9	23297	8	91	15bit	p_{18}	26113	9	51	15bit					

Table 9: Primes in the form of $p = \theta 2^m + 1$ with $m \geq 8$

Table 9 lists the primes used to construct q_2, where each prime p_i satisfies $2^m \mid (p-1)$ for the largest possible m. This condition guarantees the existence of $a \in \mathbb{Z}_p$ such that $\mathsf{ord}_p(a) = 2^m$. Specifically, the prime p is expressed in the form $p = \theta 2^m + 1$, where we only consider the case of $m \geq 8$, and the primes are listed in ascending order of m. Note that if $m < 8$, the polynomial modulus degree w will increase when performing multiplication in the NTT_w space using the multiplication algorithm MUL_w, leading to slower computation.

B NTT_w and INTT_w

Note on Implementation of NTT_w and INTT_w. In Line 5 of Algorithm 6, and Line 8 of Algorithm 7 $\bar{k}$ refers to the value obtained by applying the bit-reversal permutation to k [33]. All $\zeta^i_{2d/w}$ values are pre-multiplied by 2^β as input to SM and stored. Note that the output of INTT_w in Algorithm 7 is scaled by $2^{d/w}$, meaning that $\mathsf{INTT}_w(\mathsf{NTT}_w(\boldsymbol{a})) \neq \boldsymbol{a}$. However, this multiplicative difference is pre-multiplied to the blind rotation key BL, and hence these discrepancies are compensated during the MUL_w operation.

C MUL_w

Note on implementation. Since the output of multiplication between $\boldsymbol{a}$ and $\boldsymbol{b}$ in Algorithm 8 is $2^{-\beta}\boldsymbol{ab}$, a scaling of $2^{-\beta}$ arises from signed Montgomery reduction SM and needs to be compensated along with the scaling factor $2^{d/w}$ from

Algorithm 6 NTT_w

Input: $\boldsymbol{a} = (a_0, ..., a_{d-1}) \in R_{q,d}$
Output: $\check{\boldsymbol{a}} = (\boldsymbol{a} \bmod X^w - \zeta_{2d/w}^i)_{i \in \mathbb{Z}_{2d/w}^*}$

$k \leftarrow 1$
for $l \leftarrow d/2; l > w/2; l \leftarrow l/2$
  for $s \leftarrow 0; s < d; s \leftarrow s + 2l$
    for $j \leftarrow s; j < s + l; j \leftarrow j + l$
      $t \leftarrow \mathsf{SM}(2^\beta \zeta_{2d/w}^{\bar{k}}, a_{j+l})$
      $a_{j+l} \leftarrow a_j - t$
      $a_j \leftarrow a_j + t$
    end for
    $k \leftarrow k + 1$
  end for
end for
return $\check{\boldsymbol{a}} = (a_0, ..., a_{d-1})$

Algorithm 7 INTT_w

Input: $\check{\boldsymbol{a}} = (a_0, ..., a_{d-1}) \in \bigoplus_{i \in \mathbb{Z}_{2d/w}^*} \mathbb{Z}_q[X]/\langle X^w - \zeta_{2d/w}^i \rangle$
Output: $\boldsymbol{a}$ where $\mathsf{NTT}_w(\boldsymbol{a}) = 2^{d/w} \check{\boldsymbol{a}}$

$k \leftarrow 1$
for $l \leftarrow 1; l < d/w; l \leftarrow 2l$
  for $s \leftarrow 0; s < d; s \leftarrow s + 2l$
    for $j \leftarrow s; j < s + l; j \leftarrow j + l$
      $t \leftarrow a_j$
      $a_j \leftarrow t + a_{j+l}$
      $a_{j+l} \leftarrow t - a_{j+l}$
      $a_{j+l} \leftarrow \mathsf{SM}(2^\beta \zeta_{2d/w}^{\bar{k}}, a_{j+l})$
    end for
    $k \leftarrow k + 1$
  end for
end for
return $\boldsymbol{a} = (a_0, ..., a_{d-1})$

Algorithm 8 Recursive Karatsuba $\mathsf{RK}(\boldsymbol{a}, \boldsymbol{b}, l)$

Input: a natural power of two l, two polynomials $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_q[X]$ with degree $\leq l - 1$
Output: $\boldsymbol{r} = 2^{-\beta} \boldsymbol{a}\boldsymbol{b} \in \mathbb{Z}_q[X]$

if $l = 1$
  return $\boldsymbol{r} = \mathsf{SM}(a_0, b_0)$
else
  $\boldsymbol{a} = \boldsymbol{a}_0 + \boldsymbol{a}_1 X^{l/2}, \boldsymbol{b} = \boldsymbol{b}_0 + \boldsymbol{b}_1 X^{l/2}$
  $\boldsymbol{r}_0, \boldsymbol{r}_1, \boldsymbol{r}_2 \leftarrow \mathsf{RK}(\boldsymbol{a}_0, \boldsymbol{b}_0, l/2), \mathsf{RK}(\boldsymbol{a}_0 + \boldsymbol{a}_1, \boldsymbol{b}_0 + \boldsymbol{b}_1, l/2), \mathsf{RK}(\boldsymbol{a}_1, \boldsymbol{b}_1, l/2)$
  $\boldsymbol{r}_1 \leftarrow \boldsymbol{r}_1 - \boldsymbol{r}_0 - \boldsymbol{r}_2$
  return $\boldsymbol{r} \leftarrow \boldsymbol{r}_0 + \boldsymbol{r}_1 X^{l/2} + \boldsymbol{r}_2 X^l$
end if

Algorithm 9 Karatsuba point-wise multiplication MUL_w

Input: two polynomials $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{Z}_q[X]/\langle X^w - \zeta \rangle$, and $c = 2^{\beta - (d/w)}$
Output: $\boldsymbol{r} = c\boldsymbol{a}\boldsymbol{b} \in \mathbb{Z}_q[X]/\langle X^w - \zeta \rangle$

$\boldsymbol{b} \leftarrow c\boldsymbol{b}$ ▷ This line is precomputed
$\boldsymbol{r} \leftarrow \mathsf{RK}(\boldsymbol{a}, \boldsymbol{b}, w)$ ▷ $\boldsymbol{r} = \boldsymbol{r}_0 + X^w \boldsymbol{r}_1$.
$\boldsymbol{r} \leftarrow \boldsymbol{r}_0 + \mathsf{SM}(2^\beta \zeta, \boldsymbol{r}_1)$
return $\boldsymbol{r}$

the result of INTT_w. Both scaling factors are compensated in Line 1 of Algorithm 9. The compensation factor $2^{\beta - d/w}$ is pre-multiplied to the blind rotation

key BL and stored, and hence no compensation is required during bootstrapping. Furthermore, to perform the Line 3 of Algorithm 9, the constant ζ of the irreducible polynomial $X^w - \zeta$ is expressed as a power of the primitive element $\zeta_{2d/w}$ according to Lemma 1. Since this constant values are pre-multiplied by 2^β and stored when implementing NTT_w and INTT_w, no additional computation is needed.

D Conversion of Torus-based ciphertext into MLWE ciphertext

If the coefficients of ciphertext in lattice-based cryptosystem are taken from $\mathbb{T}_q$, a torus-based cryptosystem is constructed [12]. Moreover, conversions between $\mathbb{T}_q$-based ciphertexts and standard MLWE ciphertexts have been studied [11], and it is known that $\mathbb{T}_q$ is isomorphic to $\mathbb{Z}_q$ as $\mathbb{Z}$-module [11].

To make a direct comparison between $\mathbb{Z}_q$-based and $\mathbb{T}_q$-based lattice cryptosystems, this section introduces conversion from torus-based ciphertexts to MLWE ciphertexts, as done in TFHE-rs. Since this conversion is essentially an identity map in practice, it shows that elements of TFHE-rs can be viewed as elements of $\mathbb{Z}_q$. In other words, the difference between TFHE-rs and OpenFHE lies only in how they implement polynomial multiplication. Henceforth, all $\mathbb{T}_q$-based ciphertexts are converted into $\mathbb{Z}_q$-based MLWE ciphertexts according to the method outlined below.

The $\mathbb{Z}$-module isomorphism is as follows:

$$\mathbb{T}_q \to \mathbb{Z}_q, \quad x/q + q^{-1}\mathbb{Z} \mapsto x + \mathbb{Z}_q. \tag{28}$$

If sampling is performed from a discrete Gaussian distribution with variance σ^2 in $\mathbb{T}_q$, the standard deviation changes as follows when the samples are mapped to $\mathbb{Z}_q$ space:

$$\sigma \text{ in } \mathbb{T}_q \mapsto q\sigma \text{ in } \mathbb{Z}_q. \tag{29}$$

Based on this equivalence, when storing the structure of $\mathbb{T}_q$ in a machine, we store $x \in \mathbb{T}_q$ as $x \in \mathbb{Z}_q$, so no additional steps are required when switching from $\mathbb{T}_q$ space to $\mathbb{Z}_q$ space in a program.

Fig. 4 confirms the validity of conversion $\mathbb{T}_q \to \mathbb{Z}_q$ by comparing the error variance in each number space. The circles $\circ$ in Fig. 4 represent empirically measured variance of errors after decrypting $\mathbb{T}_{2^{32}}$-based LWE ciphertexts (left) and $\mathbb{T}_{2^{32}}$-based MLWE ciphertexts (right) in the TFHE-rs program where decrypting operations are performed over $\mathbb{Z}_{2^{32}}$. The standard deviation on $\mathbb{T}_q$ is σ. The solid lines represent the predicted variance $q^2\sigma^2$, calculated by Eq. (29). Since these two types of variances match exactly, it confirms that TFHE-rs can be treated as a program implemented on $\mathbb{Z}_{2^{32}}$. Therefore, parameters of TFHE-rs

[11] Furthermore, $\mathbb{Z}_q$ multiplication can be pulled back to $\mathbb{T}_q$, and from this perspective, $\mathbb{Z}_q$ and $\mathbb{T}_q$ are isomorphic as rings [14].

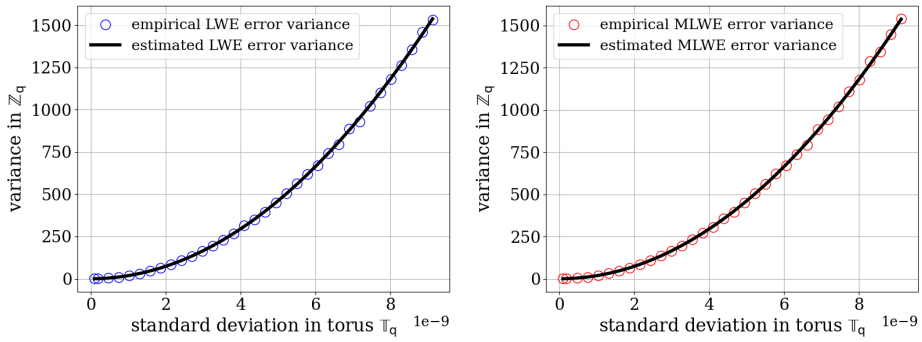

Fig. 4: Comparison of the predicted error variance $q^2\sigma^2$ after decryption with $\mathbb{Z}_q$ operations given a standard deviation σ on $\mathbb{T}_q$

can be equally compared with those of FHE16 in $\mathbb{Z}_q$. Therefore, in this paper, we describe the algorithms using $\mathbb{Z}_q$ instead of $\mathbb{T}_q$ and the standard deviation and variance will be described based on the transformation in Eq. (29).

E GINX blind rotation

Algorithm 10 GINX blind rotation [12]

Input: A secret key alphabet U_t, an LWE ciphertext $(a_1, ..., a_{k_1}, b)$ encrypted by $\mathsf{sk}_1 = (s_1, ..., s_{k_1}) \in U_t^{k_1}$, an accumulating polynomial $\mathsf{acc} \in R_{q_2,d}$, a GINX-type bootstrapping key $\mathsf{BL}_{i,j}$

Output: $\mathsf{MLWE}[\mathsf{acc}X^{\sum_i a_i s_i - b}]$

1: $\mathsf{ct} \leftarrow (0, ..., 0, \mathsf{acc}X^{-b}) \in R_{q_2,d}^{(k_2+1)\times 1}$
2: **for** $i = 0; i < n; i = i + 1$
3: **for** all $u \in U_t$
4: **if** $t = 2$
5: $\mathsf{ct} \leftarrow \mathsf{acc} + ((X^{a_i u} - 1)\mathsf{ct}) \boxdot \mathsf{BL}_{i,u}$
6: **else**
7: $\mathsf{ct} \leftarrow \mathsf{acc} + (X^{a_i u} - 1)(\mathsf{ct} \boxdot \mathsf{BL}_{i,u})$
8: **end if**
9: **end for**
10: **end for**
11: **return** ct

If sk_1 is a binary random vector (i.e., $|U_t| = 2$), the ciphertext ct is multipliled by the polynomial $X_i^{a_i u} - 1$ and then the external product is performed, as in Line 5. The error generated in Line 5 has a variance reduced by a factor of 2

compared to the error in Line 7 [12]. If $t > 2$, the computation in Line 7 results in an error variance that is also twice as large as that of Line 5, but has the advantage of reducing the number of NTT_w operations, thus speeding up the bootstrapping.

F AP$^+$ blind rotation

Algorithm 11 AP$^+$ blind rotation [26]

Input: A secret key alphabet U_t, an LWE ciphertext $(a_1, ..., a_{k_1}, b)$ encrypted by $\mathsf{sk}_1 = (s_1, ..., s_{k_1}) \in U_t^{k_1}$, an accumulating polynomial $\mathsf{acc} \in R_{q_2,d}$, an AP$^+$-type bootstrapping key BL, automorphism key AK, an window size v

Output: $\mathsf{MLWE}[\mathsf{acc}X^{\sum_i a_i s_i - b}]$

1: $z \leftarrow 0$ and $\mathsf{ct} \leftarrow (0, ..., 0, \sigma_{-g}(\mathsf{acc}X^b)) \in R_{q_2,d}^{(k_2+1)\times 1}$
2: **for** $v \in \{1, -1\}$
3: **for** $k \leftarrow N/2 - 1; k > 0; k \leftarrow k - 1$
4: **for** j such that $\psi_1(a_j) = v$ and $\psi_2(a_j) = k$
5: $\mathsf{ct} \leftarrow \mathsf{ct} \boxdot \mathsf{BL}_j$
6: **end for**
7: $z \leftarrow z + 1$
8: **if** $\{ \exists j$ such that $\psi_1(a_j) = v$ and $\psi_2(a_j) = k\} \vee \{z = v\} \vee \{k = 1\}$
9: $\mathsf{ct} \leftarrow \mathsf{Aut}(\mathsf{ct}, \mathsf{AK}_{g^z})$ ▷ See Section 3.3
10: $z = 0$
11: **end if**
12: **end for**
13: **for** j such that $\psi_1(a_j) = v$ and $\psi_2(a_j) = 0$
14: $\mathsf{ct} \leftarrow \mathsf{ct} \boxdot \mathsf{BL}_j$
15: **end for**
16: **if** $v = 1$
17: $\mathsf{acc} \leftarrow \mathsf{Aut}(\mathsf{acc}, \mathsf{AK}_{g^{-1}})$
18: **end if**
19: **end for**
20: **return** ct

The window size v is a parameter that provides a trade-off between the size of the automorphism keys and the computation speed. Experimentally, it has been shown that setting $v = 10$ strikes a balance between these factors.

[12] Line 5 refers to the original GINX method [12], while Line 7 refers to a modified version proposed by the authors [25].

G Correctness of Line 2-(i) and Line 2-(ii) in Algorithm 3

Correctness of Line 2-(i) in Algorithm 3 when single precision is used. Let η denote the floating-point precision and let ϵ represent the error that occurs after performing an arbitrary arithmetic operation op on floating-point elements x and y. It is known that this error satisfies the bound $|\epsilon| \leq |(x \ \mathsf{op} \ y)|2^{1-\eta}$ [31]. Now, we analyze the error magnitude resulting from the computation in Line 2-(i). Let x_i denote the true value after the floating division $(a_i^* \hat{q}_i \mod q_i)/q_i$, and let ϵ_i represent the error during this operation. Since $|x_i| \leq 1$, the computed floating-point value $\tilde{x}_i = x_i + \epsilon_i$ satisfies $|\tilde{x}_i| \leq (1 + 2^{1-\eta})$.

Finally, the error from the sum $\sum_{i=1}^{n} \tilde{x}_i$ is known to be $(n-1)2^{-\eta} \sum_{i=1}^{n} |\tilde{x}_i| + O(2^{-2\eta})$ [23], and hence an upper bound of the total error can be computed as follows.

$$\begin{aligned} &(n-1)2^{1-\eta} \sum_{i=1}^{n} |(x_i + \epsilon_i)| + O(2^{-2\eta}) \\ \leq &n(n-1)2^{1-\eta}(1 + 2^{-\eta-1}) + O(2^{-2\eta}) = n(n-1)2^{1-\eta} + O(n^2 2^{-2\eta}). \end{aligned} \tag{30}$$

Since we consider up to 25 candidates for 16-bit primes as listed in Supplementary Material A, Table 9, we use up to 25 16-bit primes with single precision $\eta = 23$, and hence the error in Eq. (30) is bounded by $2^{-19.35}$. Therefore, the computation in Line 2-(i) will output a correct result under the heuristic assumption of 16-bit precision.

Correctness of Line 2-(ii) in Algorithm 3. If the input satisfies $|a| < q/4$, Line 2-(ii) can be computed exactly [9]. However, a drawback of Line 2-(ii) is that it requires fixed-point division for the internal flooring operation. Therefore, the method in Line 2-(iii) is proposed, which replaces the division with precomputed values, and performs only integer addition and multiplication.

H Proofs

This section provides the proofs omitted in the main text.

Proof of Lemma 4 By Lemma 3,

$$\sum_{i=1}^{n} a_i/q_i - 2^{-s} \sum_{i=1}^{n} \lfloor 2^s/q_i \rfloor a_i = \sum_{i=1}^{n} \left[a_i/q_i - 2^{-s} \lfloor 2^s/q_i \rfloor a_i \right] < 2^{-s} \sum_{i=1}^{n} a_i < \epsilon/2.$$

Therefore,

$$\begin{aligned} &\left| v - 2^{-s} \sum_{i=1}^{n} \lfloor 2^s/q_i \rfloor a_i \right| \\ \leq &\left| v - \sum_{i=1}^{n} a_i/q_i \right| + \left| \sum_{i=1}^{n} a_i/q_i - 2^{-s} \sum_{i=1}^{n} \lfloor 2^s/q_i \rfloor a_i \right| < \epsilon < 1/2. \end{aligned}$$

Hence, $v = \left\lfloor 2^{-s} \sum_{i=1}^{n} \left\lfloor 2^s/q_i \right\rfloor a_i \right\rceil$. □

Proof of Lemma 6 All $u_i's$ in Eq. (18) satisfy $|u_0| \leq D/2$ and $|u_1|, \ldots, |u_{l-1}| \leq B/2$ from Line 4 of Algorithm 4. From Eq. (18), we have $u_l DB^{l-1} = a - u_0 - \sum_{i=1}^{l-1} u_i DB^{i-1} \geq 0$, and hence u_l is always non-negative. An upper bound of u_l is derived as follows.

$$\begin{aligned} u_l DB^{l-1} &= a - u_0 - \sum_{i=1}^{l-1} u_i DB^{i-1} < a + |u_0| + \Big| \sum_{i=1}^{l-1} u_l DB^{i-1} \Big| \\ &\leq q + \frac{1}{2} \sum_{i=0}^{l-1} DB^i \leq D(B^l - 1)(\epsilon - \frac{1}{2B}) + \frac{D}{2} \frac{(B^l - 1)}{(B-1)} \\ &\leq DB^l(\epsilon - \frac{1}{2B}) + \frac{1}{2} DB^{l-1} = \epsilon DB^l, \end{aligned}$$

thus, $u_l < \epsilon B$. □

Proof of Theorem 5 Dividing both sides of Eq. (7) by q and taking the absolute value yields

$$|a/q| = \left| \sum_{i=1}^{n} (a_i^* \hat{q}_i \mod q_i)/q_i - \lfloor \sum_{i=1}^{n} (a_i^* \hat{q}_i \mod q_i)/q_i \rceil \right| < 1/4.$$

Thus, by Lemma 4, with $\epsilon = 1/4$, Line 2-(iii) of Algorithm 3 gives the correct result. After performing Line 4, a is updated to $\lfloor a 2^{-\beta} \rfloor$. Since the updated a still satisfies $|a| < q/4$, Algorithm 3 returns the correct MRS number. □

I FHE16 vs NTRU-based FHE

In this section, we compare the bootstrapping times of FHE16 and the NTRU-based FHE [27]. Specifically, we compare the AVX2 and AVX512 execution times of NTRU-based FHE presented in [27] with the GINX bootstrapping times of FHE16 using $\mathsf{p}_G^{\mathsf{FHE16}}$ in Table 2. In [27], the secret key sk_1 is a binary with $k_1 = 512$. For a fair comparison, we set $k_1 = 512$ and measured the execution time. Note that the benchmark performance of $\mathsf{computer}_1$ used in this paper is slightly worse than the machine used in the experiments for NTRU-based FHE in [27], and hence our comparision is more conservative.

(millisecond)	AVX2	AVX512
NTRU-based FHE [27]	5.5	3.8
FHE16 with $\mathsf{p}_G^{\mathsf{FHE16}}$	4.9	3.3

Table 10: Comparision of bootstrapping time: [27] and FHE16 with $\mathsf{p}_G^{\mathsf{FHE16}}$

Table 10 shows that FHE16 achieves an overall speed improvement of 1.1x. Therefore, it is concluded that the speed of LWE-based FHE is now as fast as the fastest NTRU-based FHE.

J Multi-threading performance of FHE16

Fig. 5 shows the amortized bootstrapping time for dividing $100L$ data into L multi-threaded operations.

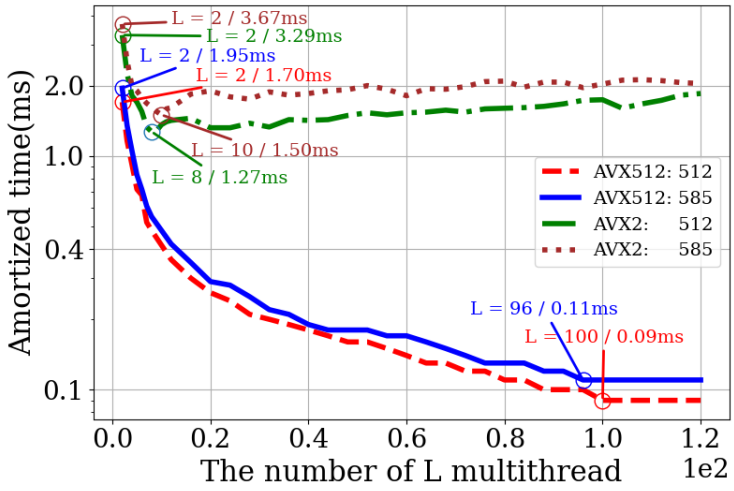

Fig. 5: Amortized bootstrapping time using L multi-threads. AVX512 is used with $\mathsf{computer}_1$ and AVX2 is used with $\mathsf{computer}_2$

Although $\mathsf{computer}_1$ has 48 cores, the speed does not increase proportionally with the number of threads due to potential bottlenecks, such as inconsistencies in the L3 cache line when multiple threads access the same memory address (e.g., where BL is stored). Nevertheless, FHE16 achieves an amortized execution time of 0.11ms for $k_1 = 585$ at $L = 96$ and 0.09ms for $k_1 = 512$ at $L = 100$, 31x and 38x, faster than the single-threaded bootstrapping of 3.5ms execution time presented in Table 8. Since the number of core of $\mathsf{computer}_2$ is 4, amortized bootstrapping times are slower than $\mathsf{computer}_1$, but $\mathsf{computer}_2$ also shows 2.5x performance enhancement.