Accelerating PEGASUS by Applying NTRU-Based GSW-Like Encryption

Shusaku Uemura and Kazuhide Fukushima[a]
KDDI Research, Inc., Saitama, Japan

Keywords: Fully Homomorphic Encryption, Scheme Switching, PEGASUS, NTRU-Based GSW-Like Encryption, Look-up Table Evaluation.

Abstract: Fully homomorphic encryption (FHE) enables to execute operations on ciphertexts without decryption. This leads to an expectation on FHE to be applied to analyses of confidential data. Each of FHE schemes proposed thus far has its own strengths such as an ability to handle real numbers or execute arbitrary functions. Scheme switching enables to switch one FHE ciphertext to another, and enables to utilize both strengths. However, as scheme switching is computationally expensive, it is sometimes more efficient to use one scheme with approximation. Xiang et al. (CRYPTO23) proposed an efficient blind rotation technique which is used in a scheme switching method named PEGASUS. They use the NTRU-based GSW-like encryption to accelerate blind rotation, which can be applied to schemes that use the GSW encryption. This paper investigates the effects of the application of the NTRU-based GSW-like encryption to PEGASUS. We found that applying the NTRU-based GSW-like encryption to PEGASUS theoretically reduces the key size required to evaluate a look-up table by 43% and the number of multiplications of integers by 96%. We also confirmed through experiments that using the NTRU-based GSW-like encryption in PEGASUS accelerates the evaluation of a look-up table by 1.55 times.

1 INTRODUCTION

Fully homomorphic encryption (FHE) is an encryption scheme that enables operations on ciphertexts without decryption. FHE is expected to be useful for analyzing sensitive data and conducting collaborative analyses of confidential data among several organizations. Multi-party computation (MPC) is also known as one of secure computation schemes that enable computation without the computers knowing the confidential information. MPC carries a risk that the computers such as servers may know the confidential information if they maliciously cooperate. Thus, it requires the assumption that the servers do not maliciously cooperate. On the other hand, FHE does not require such an assumption because the computers cannot know the values of the plaintext unless they have access to the secret key. Although FHE tends to be more computationally expensive than MPC, it incurs lower communication costs. FHE is expected to be applied to various fields for the above reasons.

The noise of a FHE ciphertext grows as operations are performed on it. The decrypted value of it becomes wrong when the noise exceeds a certain threshold. Homomorphic operations can be performed on ciphertexts as long as the noise is less than this threshold. Homomorphic encryption schemes with a limited number of operations are called somewhat homomorphic encryption or leveled homomorphic encryption. To make the number of operations unlimited, FHE adopts a noise-reducing operation called bootstrapping. Many FHE schemes proposed thus far (Cheon et al., 2017; Chillotti et al., 2020) are based on learning with errors (LWE) encryption (Regev, 2005). They adopt different operations for multiplication and bootstrapping.

One major FHE scheme proposed thus far is the Cheon-Kim-Kim-Song (CKKS) scheme (Cheon et al., 2017), which enables handling approximate numbers as plaintext and homomorphically evaluating polynomials. CKKS is used for data analyses (Geva et al., 2023) and implemented in FHE libraries (Al Badawi et al., 2022).

FHEW (Ducas and Micciancio, 2015) and TFHE (Chillotti et al., 2020) are also major FHE schemes. They utilize Gentry-Sahai-Waters (GSW) encryption (Gentry et al., 2013) to implement homomorphic multiplication. They also realize bootstrapping with GSW

[a] https://orcid.org/0000-0003-2571-0116

Uemura, S. and Fukushima, K.
Accelerating PEGASUS by Applying NTRU-Based GSW-Like Encryption.
DOI: 10.5220/0013555300003979
In *Proceedings of the 22nd International Conference on Security and Cryptography (SECRYPT 2025)*, pages 650-657
ISBN: 978-989-758-760-3; ISSN: 2184-7711

ciphertexts. Their bootstrapping scheme is called programmable bootstrapping (PBS) because it enables the evaluations of any functions using a look-up table (LUT) while reducing noise. As PBS can homomorphically evaluate any function, TFHE is expected to be applied in fields of machine learning and data analysis where evaluations of nonlinear functions are necessary. There are several libraries of TFHE (Zama, 2022) and machine learning using TFHE (Meyre et al., 2022).

There are several FHE schemes each of which has its own strengths, and it is necessary to select an appropriate scheme for the intended purpose. These schemes sometimes have their own weaknesses in executing complicated operations. Thus, it is useful to combine their strengths and mitigate their weaknesses by switching one ciphertext to another This switching technique is called scheme switching.

Although scheme switching enables mutual switching of ciphertext and utilize the strengths of each scheme, it requires extra keys and operations. The size of the key is generally larger than that of the secret key and ciphertexts. The computational cost of switching between schemes is relatively high. Some research of applying FHE to machine learning does not adopt scheme switching and instead use approximations of complicated functions or plaintext (Geva et al., 2023; Azogagh et al., 2022) due to the above reason. Therefore, it is important to reduce the key size, space complexity and time complexity for the applications of fully homomorphic encryptions.

Xiang et al. (Xiang et al., 2023) proposed efficient blind rotation technique. Blind rotation is often used in TFHE-like schemes to evaluate a look-up table. Their research proposed a vector of NTRU ciphertexts that enables homomorphic multiplication with an NTRU ciphertext. Their scheme makes an evaluation of a blind rotation more efficient by using this homomorphic multiplication.

This paper investigates the effects of applying Xiang's method to PEGASUS, a scheme switching method of FHE. Most of FHE schemes proposed thus far can be switched to another since they are based on LWE encryption (Regev, 2005). Scheme switching among LWE-based FHE schemes uses GSW encryption to execute homomorphic multiplication. As Xiang's method requires less space complexity and fewer operations, we discuss the effect of applying Xiang's method to PEGASUS.

2 PRELIMINARIES

This section defines the notations used in the rest of this paper and quickly reviews homomorphic encryptions and scheme switching.

2.1 Notations

The set of integers is denoted by $\mathbb{Z}$. We define $\mathbb{Z}_q$ for a positive integer q by $\mathbb{Z}_q := \mathbb{Z}/q\mathbb{Z}$. The set of n-dimensional vectors each entry of which is an integer is denoted by $\mathbb{Z}^n$. For a vector v, we use v_i to represent the i-th entry of v. $\langle u, v\rangle$ denotes an inner product of two vectors u, v. For a positive integer q and a power of two N, we define $R := \mathbb{Z}[X]/(X^N+1)$ and $R_q := R/qR$.

2.2 Fully Homomorphic Encryption

Fully homomorphic encryption (FHE) is an encryption scheme that enables operations such as addition and multiplication on ciphertexts without decryption. Major FHE schemes are Brakerski-Fan-Vercauteren (BFV) (Fan and Vercauteren, 2012), Cheon-Kim-Kim-Song (CKKS) (Cheon et al., 2017), TFHE (Chillotti et al., 2020). These schemes are based on Learning with Errors (LWE) encryption (Regev, 2005) and its variant using polynomial, Ring-LWE (Lyubashevsky et al., 2013). Each of them has its own strength such as the ability to perform SIMD, to handle a real number as a message and to evaluate an arbitrary function. As all of the above schemes are based on the LWE encryption, it is possible to combine their strengths by mutually switching between them. Such a method of switching schemes is called scheme switching and several methods have been proposed (Boura et al., 2020; Lu et al., 2021).

FHE is mainly composed of three algorithms: key generation, encryption and decryption, similar to public-key encryption. Addition of LWE-based FHE scheme is conducted by simply adding two ciphertexts. Multiplication of them is realized with a special key called an evaluation key, which contains the information of the secret key. These operations on a ciphertext increase a noise in it, which increases the decryption failure rate. Thus, it is necessary to reduce the noise using a special operation called bootstrapping. The details of bootstrapping differ among each scheme, but they have in common that all schemes realize bootstrapping with homomorphic operation with a special key containing information about the secret key.

As CKKS and FHEW/TFHE schemes are based on LWE encryption, we will describe LWE encryption at first. The parameters of LWE are positive integers q and n. Let $\mathcal{D}_e$ be an error distribution. An LWE ciphertext with a secret key $\mathbf{s} \in \mathbb{Z}_q^n$ is given by

$$\mathrm{LWE}(m) := (\mathbf{a}, -\langle \mathbf{a}, \mathbf{s}\rangle + \Delta m + e) \in \mathbb{Z}_q^{n+1}$$

where e is sampled from $\mathcal{D}_e$ and Δ is a scale parameter. For an LWE ciphertext $(\mathbf{a}, b) = \mathrm{LWE}(m)$, decryption can be performed by removing the error from $b + \langle \mathbf{a}, \mathbf{s} \rangle$ by division and rounding. An RLWE ciphertext is similarly defined by replacing $\mathbb{Z}_q^n$ in the definition of an LWE ciphertext with R_q.

2.3 CKKS

CKKS scheme is an RLWE-based FHE scheme proposed by Cheon et al. and can handle a real or complex number as a message by encoding it into a plaintext. Secret keys are sampled from R_q and each entry of the key belongs to $\{0, \pm 1\}$ with Hamming weight equal to h. The error distribution is a discrete Gaussian distribution with the mean equal to 0 and the standard deviation parameterized by σ. CKKS scheme uses an evaluation key of RLWE containing information about the secret key for homomorphic multiplication.

The CKKS scheme encodes a message before encryption so that it can handle a real number as a message and decodes the decrypted text to obtain the message. Decoding and encoding are sets of embeddings from polynomial space R_q to complex numbers and their inverses. Thus decoding is represented by a map $\phi : R_q \to \mathbb{C}^{N/2}$. Concretely, for $f(X) \in R_q$, the embedding ϕ is given by $\phi(f) = \left(f(\zeta), f(\zeta^3), \ldots, f(\zeta^{N-1})\right) \in \mathbb{C}^{N/2}$ where $\zeta = e^{2\pi i/N}$. Encoding is performed by evaluating the inverse ϕ^{-1} and multiplying a scaling factor on a message vector. Note that not all the complex vectors are necessarily encoded to a polynomial with integer coefficients, thus rounding is required. Decoding can be performed by the inverse of encoding, in other words, multiplying by the inverse of the scaling factor and applying ϕ so that the complex vector is obtained. By rounding the obtained vector to an appropriate accuracy, the decoded message is obtained. As each entry of ϕ, a map that substitutes X with ζ^{2i+1}, is a homomorphism, addition and multiplication after encoding correspond to those before encoding. This is the reason that CKKS can handle a real or complex number as a message while using polynomials with integer coefficients as a ciphertext.

2.4 FHEW and TFHE

FHEW and TFHE are called the third generation of FHE. They enables homomorphic multiplication by utilizing bit decomposition. FHEW (Ducas and Micciancio, 2015) is an FHE scheme proposed by Ducas et al., which accelerate bootstrapping by bit operations. TFHE is a scheme proposed by Chilloti et al. that defines its ciphertext over a torus. These schemes utilize GSW encryption to realize homomorphic multiplication. A GSW ciphertext concsists of a vector of LWE ciphertexts with a gadget matrix that corresponds to a bit decomposition of the plaintext. An operation called external product of an LWE ciphertext and a GSW ciphertext enables homomorphic multiplication. A GSW ciphertext is composed as follows. Let $\mathbf{g} \coloneqq (q/B, \ldots, q/B^l)$ be a gadget vector for positive integers B, l. We define a gadget matrix G as $G \coloneqq \mathrm{diag}(\mathbf{g}, \mathbf{g}, \ldots, \mathbf{g}) \in \mathbb{Z}_q^{l(n+1)\times(n+1)}$. A GSW ciphertext is defined by $\mathrm{GSW}(m) = L + m \cdot G$ where L is a vector of $l(n+1)$ LWE ciphertexts: $L \coloneqq (\mathrm{LWE}(0), \ldots, \mathrm{LWE}(0))$. Homomorphic multiplication of an LWE ciphertext and a GSW ciphertext is given by

$$\mathrm{GSW}(m_1) \odot \mathrm{LWE}(m_2) \coloneqq \langle \mathrm{GSW}(m_1), \mathrm{BitDecomp}_B(\mathrm{LWE}(m_2)) \rangle$$

where $\mathrm{BitDecomp}_B$ stands for a bit decomposition by a base B for each entry of the LWE ciphertext. The inner product of a GSW ciphertext and bit decomposition of an LWE ciphertext is defined by the sum of the multiplication of the component LWE ciphertext of the GSW ciphertext and each bit-decomposed entry of the LWE ciphertext. This operation gives an LWE ciphertext of $m_1 m_2$, i.e. $\mathrm{LWE}(m_1 m_2)$. A ring GSW (RGSW) ciphertext is defined similarly as above except it is defined over R_q instead of $\mathbb{Z}_q$.

TFHE scheme utilizes RGSW ciphertexts to bootstrap an LWE ciphertext by blind rotation. Blind rotation multiplies X^i to a polynomial $r(X)$ without decryption and not knowing the value of i. Concretely, blind rotation obtains $r(X) \times X^{-(b+\langle \mathbf{a}, \mathbf{s} \rangle)}$ for an LWE ciphertext $(\mathbf{a}, b)$ for message m. Denoting r_i the i-th coefficient of $r(X)$, one can obtain r_m by taking the constant term of the blind rotated value. This procedure gives a LWE ciphertext of $f(m)$ by setting the coefficients of $r(X)$ so that they correspond to $f(X)$.

TFHE uses RGSW ciphertexts as the bootstrapping key to perform the above procedure. The bootstrapping key is a set of RGSW ciphertexts of each entry of the secret key $s_i \in \{0, 1\}$. Bootstrapping can be executed as follows. First, an initial RLWE ciphertext is prepared by computing $\mathrm{Acc}_0 = \mathrm{RLWE}\left(r(X) \times X^{-b}\right)$. Then, Acc_i for $i = 1, \ldots, n$ are computed by

$$\mathrm{Acc}_i = \mathrm{RGSW}(s_i) \odot (\mathrm{Acc}_{i-1} \cdot (X^{a_i} - 1)) + \mathrm{Acc}_{i-1}.$$

recursively. The value of Acc_n is an RLWE ciphertext $\mathrm{RLWE}\left(r(X) \times X^{-(b+\langle a, s \rangle)}\right)$ because s_i equals 0 or 1. By applying a method called sample extraction to the obtained RLWE ciphertext, one can obtain the desired

LWE ciphertext $\mathrm{LWE}(f(m))$. Blind rotation is an important component of the bootstrapping procedure of TFHE because it reduces the noise of the LWE ciphertext through the procedure. The above procedure is also called a look-up table (LUT) evaluation as it can evaluate a prepared function.

2.5 Scheme Switching

It is necessary for users to choose an appropriate FHE scheme depending on the purpose as each scheme has its own strengths and weaknesses. On the other hand, it is often difficult to determine the appropriate FHE scheme for data analyses or machine learning as they require many kinds of processes. For instance, on the one hand, the CKKS scheme is useful for neural networks as it can handle a real number as a message and process multiple messages simultaneously though it requires approximating activation functions with polynomials (Choi et al., 2024). On the other hand, TFHE can evaluate any activation function though it requires to scale messages to integers (Stoian et al., 2023). Considering these situations, it is useful to combine multiple schemes to implement complicated computations such as neural networks.

Boura et al. proposed scheme switching algorithms named CHIMERA (Boura et al., 2020). CHIMERA sets TFHE as its hub and provides algorithms to switch from TFHE to BFV or CKKS and vice versa. Transformation of TFHE to CKKS in CHIMERA packs $N/2$ TFHE ciphertexts in one polynomial and homomorphically encodes it. Finally, it evaluates the exponential function on the packed ciphertext to obtain CKKS ciphertext. This transformation performs homomorphic decryption with the GSW-based evaluation keys. Although CHIMERA provides scheme switching among three major FHE schemes, it is pointed out that it requires large-sized repacking keys (Lu et al., 2021).

Lu et al. proposed scheme switching algorithms named PEGASUS (Lu et al., 2021). PEGASUS enables to handle a real number like CKKS and to evaluate any function (LUT) homomorphically like FHEW. PEGASUS uses RGSW evaluation keys for LUT similarly to TFHE described in Section 2.4. Note that the gadget vector used to construct the evaluation key does not correspond to bit decomposition but the residue number system (RNS) for efficiency. The dimension of the vector is one. The evaluation key consists of $2n$ RGSW ciphertexts (represented as $\mathrm{EK}_{j,k}$), which contains information of the secret keys. Concretely, $\mathrm{EK}_{j,k}$ encrypts the j-th entry of the secret key (s_j) and $k=0$ (resp. $k=1$) corresponds to when $s_j = 1$ (resp. $s_j = -1$). PEGASUS obtains $f(X)\cdot X^{\tilde{b}-\langle \mathbf{a},\mathbf{s}\rangle}$ by repeating homomorphic multiplications of $X^{\tilde{a}_j s_j}$. Finally, extracting the constant term of the ciphertext gives $\mathrm{LWE}(T(m))$.

In the whole protocol of PEGASUS, it first extracts l LWE ciphertexts from the RLWE ciphertext of CKKS. It switches the parameter n and the secret key of LWE ciphertexts, applies the LUT described as above and switches n and the secret key again. It finally repacks l LWE ciphertexts into one RLWE ciphertext. Through the above procedure, PEGASUS enables to utilize the strength of CKKS in handling a real number and that of FHEW in evaluating LUT. Due to the space limitation, we have explained the sketch of PEGASUS, refer to the original paper (Lu et al., 2021, Figures 2 and 6) for the detail.

3 NTRU-BASED GSW-LIKE CIPHERTEXT

This section describes the efficient blind rotation algorithm proposed by Xiang et al. (Xiang et al., 2023). Their idea is to use the NTRU-based ciphertext (Hoffstein et al., 1998) to realize homomorphic multiplication. This algorithm shares a common feature with the GSW-based blind rotation that they both use bit decomposition to make the increase of noise in the ciphertext small.

3.1 NTRU-Based GSW-Like Ciphertext

We describe an NTRU ciphertext and an NTRU$'$ ciphertext proposed by Xiang et al. Their NTRU ciphertext is slightly different from the original NTRU ciphertext as it multiplies the inverse of the secret key by the message. An NTRU ciphertext of a message $u \in R_q$ is defined as follows:

$$\mathrm{NTRU}(m) \coloneqq \tau \cdot g/f + \Delta \cdot u/f \in R_Q.$$

where $f \in R_q$ is the secret key that is invertible in R_q and $g \in R_q$ is a polynomial with small coefficients. Parameters τ and Δ are determined depending on the used FHE scheme. For CKKS, both of them equal one. We represent by $/f$ a multiplication of the inverse of f.

An NTRU$'$ ciphertext of a message $v \in R_q$ is defined as follows:

$$\mathrm{NTRU}'(v) \coloneqq \big(\tau \cdot g_0/f + v, \tau \cdot g_1/f + B \cdot v, \ldots, \tau \cdot g_{d-1}/f + B^{d-1} \cdot v\big) \in R_Q^d$$

where $f \in R_q$ is the secret key that is invertible in R_q and $g_0, \ldots, g_{d-1} \in R_q$ are polynomials with small coefficients. τ and B are integer parameters and $d \coloneqq \lceil \log_B Q \rceil$.

Homomorphic multiplication of an NTRU ciphertext and an NTRU′ ciphertext is defined as follows. We define a bit decomposition of a polynomial for a base B. Bit decomposition of $h \in R_Q$ is given by $\mathrm{BitDecomp}(h) \coloneqq (h_0, \ldots, h_{d-1}) \in R_B^d$ so that it satisfies that $h = \sum_{i=0}^{d-1} h_i B^i$. The external product $\odot$ of a polynomial $h \in R_Q$ and a vector of polynomials $\mathbf{c} \in R_Q^d$ is defined as

$$h \odot \mathbf{c} \coloneqq \langle \mathrm{BitDecomp}_B(h), \mathbf{c} \rangle.$$

Homomorphic multiplication of an NTRU ciphertext and an NTRU′ ciphertext is given by the product $\mathrm{NTRU}(u) \odot \mathrm{NTRU}'(v)$. We can verify it by the following equation:

$$\begin{aligned}\mathrm{NTRU}(u) \odot \mathrm{NTRU}'(v) &= \sum_{i=0}^{d-1} h_i(\tau g_i/f + B^i \cdot v) \\ &= \sum_{i=0}^{d-1} h_i(\tau g_i/f) + \mathrm{NTRU}(u) \cdot v \\ &= \tau\left(g + \sum_{i=0}^{d-1} h_i g_i\right)/f + \Delta u \cdot v/f\end{aligned}$$

where $(h_i)_{i=0}^{d-1}$ is the decomposition of $\mathrm{NTRU}(u)$ with the base B. As $g + \sum_{i=0}^{d-1} h_i g_i$ is a polynomial with small coefficients under an appropriate setting, it holds that

$$\mathrm{NTRU}(u) \odot \mathrm{NTRU}'(v) = \mathrm{NTRU}(uv).$$

This procedure does not require decryption, and thus it is homomorphic multiplication.

3.2 Blind Rotation with NTRU Ciphertext

We describe the blind rotation technique introduced by Xiang et al. using NTRU and NTRU′ ciphertexts. In the rest of this section, we fix the parameter q of LWE with the secret key $\mathbf{s} = (s_0, \ldots, s_{n-1})$ and NTRU parameter Q with the secret key $f \in R_Q$ The degree of R_Q is a power of two N that can be divided by q. This blind rotation requires $n+1$ evaluation keys and $q-1$ key switching keys. Evaluation keys are defined as follows:

$$\mathrm{evk}_0 = \mathrm{NTRU}'(X^{s_0}/f), \ \mathrm{evk}_i = \mathrm{NTRU}'(X^{s_i})$$
$$\mathrm{evk}_i = \mathrm{NTRU}'\left(X^{-\sum_{i=0}^{n-1} s_i}\right).$$

Key switching keys are

$$\mathrm{ksk}_j = \mathrm{NTRU}'\left(f(X^j)/f(X)\right)$$

where $j \in S \coloneqq \{2N/q+1 | 1 \le i \le q-1\}$. The algorithm contains homomorphic evaluations of the automorphisms ($\mathrm{EvalAuto}_j$) that homomorphically substitutes X of an NTRU ciphertext with X^j. In other words, $\mathrm{EvalAuto}_j$ transforms $\mathrm{NTRU}(u(X))$ to $\mathrm{NTRU}\left(u(X^j)\right)$. See (Xiang et al., 2023, Section 3) for details.

We briefly review the correctness of the algorithm. The homomorphic multiplication of the accumulated value and the evaluation key (evk_i) makes the accumulated value equal to $\mathrm{NTRU}\left(r(X^{\frac{2N}{q}w'_i}) \cdot X^{-\frac{2N}{q}bw'_i + \sum_{k=0}^{i-1} s_k w_k w'_i + s_i}\right)$. By evaluating the automorphism, it obtains $\mathrm{NTRU}\left(r(X^{\frac{2N}{q}w'_{i+1}}) \cdot X^{-\frac{2N}{q}bw'_{i+1} + \sum_{k=0}^{i} s_k w_k w'_{i+1}}\right)$. Finally, the output becomes

$$\begin{aligned}&\mathrm{NTRU}\left(r(X^{\frac{2N}{q}w'_{i+1}}) \cdot X^{-\frac{2N}{q}b + \sum_{k=0}^{n-1} s_k w_k - \sum_{k=0}^{n-1} s_k}\right) \\ &= \mathrm{NTRU}\left(r\left(X^{\frac{2N}{q}}\right) \cdot X^{\frac{2N}{q}\left(-b + \sum_{i=0}^{n-1} a_i s_i\right)}\right)\end{aligned}$$

by the homomorphic multiplication of the accumulated value and the n-th evaluation key. Therefore, the blind rotation algorithm outputs an NTRU ciphertext of the blind rotated polynomial.

As the output of the blind rotation algorithm is an NTRU ciphertext of the blind rotated polynomial, it is required to revert it to an LWE ciphertext. This operation can be performed by sample extraction (Kim et al., 2024). Let $\mathbf{c} = (c_0, \ldots, c_{N-1})$ and $\mathbf{f} = (f_0, \ldots, f_{N-1})$ be coefficient vectors of the output $c \in R_Q$ of the algorithm and the secret key $f \in R_Q$. We set $\hat{\mathbf{c}} \coloneqq (c_0, -c_{N-1}, \ldots, -c_1)$, then $(\hat{\mathbf{c}}, 0) \in \mathbb{Z}_Q^{N+1}$ is a desired LWE ciphertext with the secret key $\mathbf{f}$.

The whole procedure of bootstrapping consists of three parts: blind rotation, modulus switching and key switching. It first applies the aforementioned blind rotation algorithm to the input LWE ciphertext. The input LWE ciphertext becomes an NTRU ciphertext of the blind rotated polynomial by the blind rotation and an LWE ciphertext is extracted by sample extraction. As the modulus and the secret key of this LWE ciphertext are different from those of the input LWE ciphertext, it is necessary to revert them to their original values. This is solved by modulus switching and key switching. Modulus switching from Q to q is accomplished by $\lfloor q/Q \cdot \mathrm{LWE}(m) \rceil \mod q$.

Key switching of an LWE ciphertext requires LWE ciphertexts containing information of the secret key $\mathbf{f}$ as key switching keys. The key switching keys are given by

$$\mathrm{lksk}_{i,j,v} \coloneqq \mathrm{LWE}_{\mathbf{s}}\left(-vB_{ks}^j f_j\right).$$

where $\mathbf{s}$ is the secret key after key switching, B_{ks} and Q_{ks} are integer parameters, and it hold that $i \in \mathbb{Z}_N$, $j \in \mathbb{Z}_{d_{ks}}$, $1 \le v < B_{ks}$. We set $d_{ks} \coloneqq \lceil \log_{B_{ks}} Q_{ks} \rceil$. The procedure of key switching with the above key switching keys is outlined as follows. Let

$(\hat{\mathbf{a}}, \hat{b}) = (\hat{a}_0, \ldots, \hat{a}_{N-1}, \hat{b})$ be an LWE ciphertext and $(v_{i,0}, \ldots, v_{i,d_{ks}-1})$ be a decomposition of $\hat{a}_i$ with the base B_{ks}. We write $\mathrm{lksk}_{i,j,v} = (\mathbf{a}_{i,j,v}, b_{i,j,v})$. Here, we set

$$\hat{\mathbf{a}}' \coloneqq \sum_{i=0}^{N-1} \sum_{j=0, v_{i,j} \neq 0}^{d-1} \mathbf{a}_{i,j,v_{i,j}},$$

$$\hat{b}' \coloneqq \sum_{i=0}^{N-1} \sum_{j=0, v_{i,j} \neq 0}^{d-1} b_{i,j,v_{i,j}} + \hat{b}.$$

Then, it can be verified that $(\hat{\mathbf{a}}', b')$ is an LWE ciphertext with the secret key $\mathbf{s}$.

The above procedure completes the bootstrapping on an LWE ciphertext with NTRU-based GSW-like keys. Thus, it finally outputs $\mathrm{LWE}(f(m))$ from the input $\mathrm{LWE}(m)$. Due to the space limitation, we have described the sketch of the algorithm. See (Xiang et al., 2023, Section 3) for details.

4 APPLICATION TO PEGASUS

This section discusses the effects of Xiang's NTRU-based technique as applied to PEGASUS. We discuss the effects both theoretically and experimentally.

As stated in Section 2.5, PEGASUS extracts LWE ciphertexts from an RLWE ciphertext of CKKS and evaluates a look-up table by blind rotation with RGSW-based evaluation keys. The LUT evaluation of this process can be replaced by Xiang's LUT technique. We will discuss the effects of reducing the key size and the number of computations when the LUT is replaced. In this paper, the LUT evaluation of PEGASUS will be replaced by Xiang's LUT illustrated in the previous section. The modulus q of an LWE ciphertext can be switched by modulus switching. The parameter n also can be changed through the key switching executed in the PEGASUS scheme. Thus, we use the same LWE and NTRU parameter sets as proposed in (Lu et al., 2021; Xiang et al., 2023) and skip fine-tuning of parameters. Xiang et al. proposed the 128-bit and 192-bit security NTRU parameters while Lu et al. proposed the LWE parameters at least 119-bit security. Hence, we use 128-bit security parameters for the NTRU scheme.

4.1 Comparison of Key Size

Xiang's algorithm uses $n+1$ NTRU$'$ ciphertexts as evaluation keys, i.e. evk_i, to contain information of the secret key and $q-1$ NTRU$'$ ciphertexts as key switching keys, i.e. ksk_i, to change the exponent of polynomials. Thus, it requires $n+q$ NTRU$'$ ciphertexts in total. An NTRU$'$ ciphertext consists of d entries of R_Q. As the degree of a polynomial in R_Q equals $N-1$, it requires $N\lceil \log_2 Q \rceil$ bits to represent one R_Q element. Therefore, it requires $d(n+q)N\lceil \log_2 Q \rceil$ bits in total to store the evaluation keys for blind rotation.

Additionally, to complete the bootstrapping procedure, LWE key switching keys (LKSK) are also required. They also have to be taken into account. LKSK is composed of multiple LWE ciphertexts. The number of LWE ciphertexts is $N \times d_{ks} \times (B_{ks}-1)$ since the suffixes i, j, v of $\mathrm{lksk}_{i,j,v}$ satisfy $i \in \mathbb{Z}_N$, $j \in \mathbb{Z}_{d_{ks}}$, and $1 \leq v < B_{ks}$. It requires $n+1$ elements of $\mathbb{Z}_{Q_{ks}}$ to represent a single LWE ciphertext. Therefore $N \cdot d_{ks} \cdot (B_{ks}-1) \cdot (n+1) \log_2 Q_{ks}$ bits are necessary for LKSK. The sum of the bits of the evaluation keys and the LWE key switching keys is the number of bits required to store the keys for the bootstrapping procedure. Xiang et al. proposed the following parameters: $n = 512$, $q = N = 1024$, $d = 5$, $\log_2 Q \approx 19.9$, $\log_2 B = 4$, $\log_2 Q_{ks} = 14$, $\log_2 B_{ks} = 7$, and $d_{ks} = 2$. By substituting these values into the parameters, the bits of the evaluation keys equal 18.75MB and the bits of the key switching keys equal 222.7MB, thus 241.4MB in total. The difference of 0.1 is due to the rounding error.

PEGASUS employs RGSW ciphertexts to evaluate LUT. One RGSW ciphertext consists of four R_q elements. The degree of a polynomial in R_q is $N-1$ and PEGASUS uses $2n$ RGSW ciphertexts. Hence, it requires $8nN \log_2 q$ bits in total. Considering that the parameters of PEGASUS are $n = 2^{10}$, $N = 2^{12}$, $\log q \approx 105$, the size of the evaluation key is 420MB.

Based on the above discussion, application of NTRU-based GSW-like technique to PEGASUS reduces the LUT evaluation key size by 178.6MB (43%).

4.2 Comparisons of the Number of Operations

The most significant part of the computational cost is the external products in both PEGASUS and the NTRU-based algorithm because the other parts consist of additions and multiplications of plaintexts. For this reason, we compare the computational cost of the external products. As the degrees of the polynomials used in RGSW ciphertexts and NTRU$'$ ciphertexts differs, it is fair to compare the numbers of integer multiplications incurred by the external products, not the numbers of polynomial multiplications.

The external products are executed twice in the for loop of Xiang's algorithm. This for loop repeats the operation n times and there is another external product after the loop. Thus, it requires $2n+1$ external

Table 1: Comparisons of Key size and the number of integer multiplications when NTRU-based GSW-like encryption is applied to PEGASUS.

	Key size [MB]	#Integer Mult.
PEGASUS	420	1.37×10^{11}
PEGASUS+NTRU	241.4	5.37×10^{9}
Difference	178.6 (43%)	1.32×10^{11} (96%)

products. One external product consists of d polynomial multiplications. Taking the degree $N-1$ of the polynomial into account, the number of necessary multiplications in $\mathbb{Z}_Q$ is $d(n+1)N^2$. We obtain 5.37×10^9 by substituting the parameters with the aforementioned values.

PEGASUS executes $2n$ external products, resulting in $8n$ polynomial multiplications as one external product consists of four polynomial multiplications. Therefore, it conducts $8nN^2$ integer multiplications, which is equal to 1.37×10^{11} with the aforementioned parameters substituted.

Hence, application of NTRU-based GSW-like technique to PEGASUS reduces the number of integer multiplications by 1.32×10^{11} (96%).

4.3 Experimental Results

We implemented the external products of both the schemes that we discussed above. As described in the previous subsection, the elapsed time to execute LUT evaluation is dominated by the time to execute the external products. Therefore, we compare the elapsed time to execute as many external products as are required to execute one LUT evaluation.

We implemented the external products with SageMath 10.3 (The Sage Developers, 2024), which works on Python 3.11.8.

The execution environment was a Mac mini with an Apple M2 CPU, 24 GB RAM, and macOS Ventura 13.6.3. We employed the quotient ring structure pre-implemented in SageMath to implement RLWE, RGSW, NTRU and NTRU′ ciphertexts.

The details of our experiments were as follows. We implemented the external products of both PEGASUS and Xiang's NTRU-based algorithm. We executed as many external products as necessary for each algorithm, in other words, 2048 times for PEGASUS and 1025 times for Xiang's NTRU-based algorithm. We repeated this experiment 100 times to measure the mean time to execute one LUT evaluation.

The experimental results we obtained are shown in Table 2. The mean time for PEGASUS is 67.03 seconds while that for the NTRU is 43.28 seconds.

Table 2: Experimental results of mean time and standard deviation to execute as many external products as required to perform one LUT evaluation.

	mean time [s]	std. dev.[s]
PEGASUS	67.03	0.52
PEGASUS+NTRU	43.28	5.82

This means that replacing the GSW-based algorithm of PEGASUS with the NTRU-based algorithm improves the execution time by a factor of 1.55 times. Although our theoretical calculation showed that this replacement would reduce 96% of the execution time, the experimental results do not show such a reduction. This is considered to be because of the difference in the time required for polynomial multiplications. The 25 times acceleration of our proposed method consists of 16 times acceleration of a single polynomial multiplication and 1.6 times improvement in the number of polynomial multiplications. Concretely, the NTRU-based LUT evaluation requires 5125 polynomial multiplications and each polynomial multiplication requires 1024^2 integer multiplications while the PEGASUS scheme requires 8192 polynomial multiplications and each polynomial multiplication requires 4096^2 integer multiplications. As we implemented the NTRU and RGSW using the polynomial structure of SageMath, the polynomial multiplication is optimized. This leads to a slight difference between polynomial multiplications with different degrees. Hence, the difference of the two schemes only occurs in the number of polynomial multiplications. The theoretical improvement of the number of polynomial multiplications is approximately 1.60 ($= 8192/5125$). This explains our experimental improvement.

5 CONCLUSION

We have discussed the effects of applying Xiang's NTRU-based GSW encryption to PEGASUS. We theoretically confirmed that this application reduces the key size for LUT by 43% and the number of integer multiplications of external products by 96% by replacing the LUT evaluation of PEGASUS with Xiang's NTRU-based blind rotation technique. We also confirmed by experiments that NTRU-based GSW-like encryption improves PEGASUS scheme by a factor of 1.55 times.

We employed the original parameters to analyze the effects of the replacement of the look-up table evaluation of the original PEGASUS scheme with the NTRU-based look-up table evaluation. However, it may be possible to make the scheme more effi-

cient by optimizing the parameter set. Our future work is to optimize the parameters of both PEGASUS and NTRU-based schemes and compare their performances.

REFERENCES

Al Badawi, A., Bates, J., Bergamaschi, F., Cousins, D. B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., Liu, Z., Micciancio, D., Quah, I., Polyakov, Y., R.V., S., Rohloff, K., Saylor, J., Suponitsky, D., Triplett, M., Vaikuntanathan, V., and Zucca, V. (2022). OpenFHE: Open-Source Fully Homomorphic Encryption Library. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC'22, pages 53–63, New York, NY, USA. Association for Computing Machinery.

Azogagh, S., Delfour, V., Gambs, S., and Killijian, M.-O. (2022). PROBONITE: PRivate One-Branch-Only Non-Interactive decision Tree Evaluation. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC'22, page 23–33, New York, NY, USA. Association for Computing Machinery.

Boura, C., Gama, N., Georgieva, M., and Jetchev, D. (2020). CHIMERA: Combining Ring-LWE-based Fully Homomorphic Encryption Schemes. *Journal of Mathematical Cryptology*, 14(1):316–338.

Cheon, J. H., Kim, A., Kim, M., and Song, Y. (2017). Homomorphic Encryption for Arithmetic of Approximate Numbers. In Takagi, T. and Peyrin, T., editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham. Springer International Publishing.

Chillotti, I., Gama, N., Georgieva, M., and Izabachène, M. (2020). TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33:34–91.

Choi, H., Woo, S. S., and Kim, H. (2024). Blind-Touch: Homomorphic Encryption-Based Distributed Neural Network Inference for Privacy-Preserving Fingerprint Authentication. In *Proceedings of the AAAI Conference on Artificial Intelligence*, number 2452 in AAAI'24/IAAI'24/EAAI'24, pages 21976–21985.

Ducas, L. and Micciancio, D. (2015). FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In Oswald, E. and Fischlin, M., editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 617–640, Berlin, Heidelberg. Springer Berlin Heidelberg.

Fan, J. and Vercauteren, F. (2012). Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2012/144. https://eprint.iacr.org/2012/144.

Gentry, C., Sahai, A., and Waters, B. (2013). Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In Canetti, R. and Garay, J. A., editors, *Advances in Cryptology – CRYPTO 2013*, pages 75–92, Berlin, Heidelberg. Springer Berlin Heidelberg.

Geva, R., Gusev, A., Polyakov, Y., Liram, L., Rosolio, O., Alexandru, A., Genise, N., Blatt, M., Duchin, Z., Waissengrin, B., Mirelman, D., Bukstein, F., Blumenthal, D. T., Wolf, I., Pelles-Avraham, S., Schaffer, T., Lavi, L. A., Micciancio, D., Vaikuntanathan, V., Badawi, A. A., and Goldwasser, S. (2023). Collaborative privacy-preserving analysis of oncological data using multiparty homomorphic encryption. *Proceedings of the National Academy of Sciences*, 120(33):e2304415120.

Hoffstein, J., Pipher, J., and Silverman, J. H. (1998). NTRU: A ring-based public key cryptosystem. In Buhler, J. P., editor, *Algorithmic Number Theory*, pages 267–288, Berlin, Heidelberg. Springer Berlin Heidelberg.

Kim, A., Deryabin, M., Eom, J., Choi, R., Lee, Y., Ghang, W., and Yoo, D. (2024). General Bootstrapping Approach for RLWE-Based Homomorphic Encryption. *IEEE Transactions on Computers*, 73(1):86–96.

Lu, W.-j., Huang, Z., Hong, C., Ma, Y., and Qu, H. (2021). PEGASUS: Bridging Polynomial and Non-polynomial Evaluations in Homomorphic Encryption. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1057–1073.

Lyubashevsky, V., Peikert, C., and Regev, O. (2013). On Ideal Lattices and Learning with Errors over Rings. *J. ACM*, 60(6).

Meyre, A., Chevallier-Mames, B., Frery, J., Stoian, A., Bredehoft, R., Montero, L., and Kherfallah, C. (2022). Concrete ML: a Privacy-Preserving Machine Learning Library using Fully Homomorphic Encryption for Data Scientists. https://github.com/zama-ai/concrete-ml.

Regev, O. (2005). On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 84–93, New York, NY, USA. Association for Computing Machinery.

Stoian, A., Frery, J., Bredehoft, R., Montero, L., Kherfallah, C., and Chevallier-Mames, B. (2023). Deep Neural Networks for Encrypted Inference with TFHE. In Dolev, S., Gudes, E., and Paillier, P., editors, *Cyber Security, Cryptology, and Machine Learning*, pages 493–500, Cham. Springer Nature Switzerland.

The Sage Developers (2024). *SageMath, the Sage Mathematics Software System (Version 10.3)*. `https://www.sagemath.org`.

Xiang, B., Zhang, J., Deng, Y., Dai, Y., and Feng, D. (2023). Fast Blind Rotation for Bootstrapping FHEs. In Handschuh, H. and Lysyanskaya, A., editors, *Advances in Cryptology – CRYPTO 2023*, pages 3–36, Cham. Springer Nature Switzerland.

Zama (2022). TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data. https://github.com/zama-ai/tfhe-rs.