PEGASUS: Bridging Polynomial and Non-polynomial Evaluations in Homomorphic Encryption

Wen-jie Lu*, Zhicong Huang*, Cheng Hong*, Yiping Ma† and Hunter Qu*
* *Gemini Lab, Alibaba Group,* † *University of Pennsylvania*

2021 IEEE Symposium on Security and Privacy (SP) | 978-1-7281-8934-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/SP40001.2021.00043

***Abstract*—Homomorphic encryption (HE) is considered as one of the most important primitives for privacy-preserving applications. However, an efficient approach to evaluate both polynomial and non-polynomial functions on encrypted data is still absent, which hinders the deployment of HE to real-life applications. To address this issue, we propose a practical framework *PEGASUS*. PEGASUS can efficiently switch back and forth between a packed CKKS ciphertext and FHEW ciphertexts without decryption, allowing us to evaluate arithmetic functions efficiently on the CKKS side, and to evaluate look-up tables on FHEW ciphertexts. Our $\mathrm{FHEW} \rightarrow \mathrm{CKKS}$ conversion algorithm is more practical than the existing methods. We improve the computational complexity from linear to sublinear. Moreover, the size of our conversion key is significantly smaller, e.g., reduced from 80 gigabytes to 12 megabytes. We present extensive benchmarks of PEGASUS, including sigmoid/ReLU/min/max/division, sorting and max-pooling. To further demonstrate the capability of PEGASUS, we developed two more applications. The first one is a private decision tree evaluation whose communication cost is about two orders of magnitude smaller than the previous HE-based approaches. The second one is a secure K-means clustering that is able to run on thousands of encrypted samples in minutes that outperforms the best existing system by $14\times$ – $20\times$. To the best of our knowledge, this is the first work that supports practical K-means clustering using HE in a single server setting.**

***Keywords*-Homomorphic Computation, Floating Point Computation, CKKS/FHEW**

I. Introduction

Homomorphic encryption (HE) is a cryptosystem that enables homomorphic operations on encrypted data and is considered as one of the most important building blocks for privacy-preserving applications. A potential application of HE is secure outsourcing [3], [36], where all the data comes from clients. That is, the client(s) encrypts his/her data using HE and uploads the ciphertexts to the server that performs all the computations over encrypted data. The server then returns the result in the form of ciphertext to the client that can decrypt to get the computed result. Another potential application of HE is secure two-party computation. The difference from secure outsourcing is that the server also holds its private databases, and the clients send encrypted queries to the server. Many applications fall into this setting, e.g., the private information retrieval [42] and the decision-tree evaluation [37], [51].

Most of the current HE schemes can be categorized into word-wise HE (such as BFV [26], BGV [8] and CKKS [14]) and bit-wise HE (such as FHEW [24] and TFHE [18]). Each type has particular advantages and disadvantages. Word-wise HEs support efficient single-instruction-multiple-data (SIMD) style homomorphic operations (i.e., addition and multiplication) by packing multiple plaintexts into a single ciphertext [49]. However, it becomes difficult to compute non-polynomial functions such as sigmoid, min/max, and division on the ciphertexts of word-wise HEs. As a compromise, the existing word-wise HE-based approaches approximate non-polynomial functions using low-degree polynomials [29], [39] or simply avoid them, e.g. replacing the max-pooling with average-pooling [22]. Moreover, in applications such as K-means clustering, there exist inevitable non-polynomial functions (i.e., min-index and division) that are hard to approximate via low-degree polynomials.

Contrary to word-wise HEs, bit-wise HEs support arbitrary functions presented as boolean circuits by encrypting each bit of the plain values using some representatives from their message space. However, as shown in [17], [18], bit-wise HEs are barely practical for addition and multiplication circuits, especially when the boolean circuit consists of thousands of fan-in bits and a large circuit depth. For example, [18] took about a half minute to multiply two encrypted 16-bit integers. Also, the expansion ratio of bit-wise HEs is usually several orders of magnitude larger than the word-wise HEs which could lead to a higher communication cost.

When applying HEs to real-world applications such as secure neural network inferences [22], [29], the problem is even more challenging. That is because the inference procedure computes many instances of arithmetic functions such as convolution, and non-polynomial functions such as sigmoid and max-pooling. It is then natural to ask the following question:

Can we evaluate both polynomial functions and non-polynomial functions on encrypted data efficiently and effectively?

Unfortunately, there are few practical approaches and frameworks that achieve this target.

A. Related Work

Cheon et al. [15], [16] presented an efficient (in the amortized sense) method to compute min/max over the

† Yiping participated in this work as an internship at Alibaba.

DOI 10.1109/SP40001.2021.00043

Table I: Memory cost and computation costs (number of polynomial multiplications) of the repacking algorithms. N (resp. n) denotes the dimension of RLWE (resp. LWE). q is the ciphertext modulus and ℓ denotes the number of LWE ciphertexts to repack. We have $1 \le \ell \le N/2$ and $N \gg n$.

	Space (bit-size)	Computation
[7]	$O(Nn \log q)$	$O(n)$
[11, §3.4]	$O(N \log N \log q)$	$O(\ell + \log(N/\ell))$
Ours	$O(N \log q)$	$O(\min(\sqrt{n}, \sqrt{\ell} + \log(\lceil n/\ell \rceil)))$

CKKS scheme. Their method is suitable to compute the min/max on a large batch (e.g., $> 2^{16}$) of encrypted integer pairs, but is less suitable for a recursive comparison such as sorting an encrypted vector. Moreover, their method is designed specifically for the min/max operations, and it is not clear how to use it to compute other non-polynomial functions, e.g., square-root and division.

Micciancio et al. [43] presented a flexible method for non-polynomial functions by evaluating look-up tables (LUT) on ciphertexts. Indeed, their method uses a learning-with-error (LWE) -based HE scheme. However, their method demands the plaintext modulus to be a divisor of the LWE dimension, which is usually chosen as a small value (e.g., an integer of 10–12 bits) for the sake of efficiency. As a result, one can not apply their LUT methods to the scenario that requires a large plaintext domain. Not to mention that LWE-based HE schemes do not support SIMD style arithmetic computations.

The most relevant study is by [7]. They designed the CHIMERA framework to switch between TFHE ciphertexts and ciphertexts of the torus variant of CKKS/BFV schemes. On one hand, CHIMERA enables to perform SIMD style arithmetic operations on the CKKS/BFV side, and to compute LUTs with a large domain on the TFHE side. However, their conversion $\mathrm{TFHE} \to \mathrm{CKKS/BFV}$ (designated as *repacking*) is costly in terms of key size and computation costs. Although a concurrent work from [11] can improve the computation efficiency of CHIMERA, this improvement is limited to a small repacking size, i.e., $\ell = \Omega(\log N)$. The computation costs of [11, §3.4] are still high when $\ell \approx N$ (see Table I). Moreover, CHIMERA uses multi-precision floating-point values of hundreds of bits to maintain a proper precision when exporting CKKS/BFV to the torus. As a result, CHIMERA needs Multi-Precision Fast Fourier Transform (MP-FFT) for the polynomial operations in its torus variant of CKKS/BFV, which can significantly deteriorate the efficiency of their framework. MP-FFT can be several orders of magnitude slower compared to their integer counterparts using Number Theoretic Transform (NTT) and Residue Number System (RNS).

In Table II, we summarize the insufficiency of the current arts for evaluating polynomial functions and non-polynomial functions on encrypted data. A practical approach that supports SIMD style homomorphic operations and provides high flexibility for non-polynomial functions is still absent.

Table II: Insufficiency of existing approaches. The efficiency in the last column is considered as "high" if the method provides either a high throughput or a low latency computation.

	SIMD?	Flexibility	Key	Efficiency
[16]	Yes	Very limited (min/max)	$\approx$ 4GB	High†
[43]	No	Limited (LUT on a small domain)	$>$ 5GB	High
[7]	Yes	High (LUT on large domains)	$\gg$ 10GB	Low
Ours	Yes	High (LUT on large domains)	$\approx$ 1GB	High

† In the sense of amortization.

B. Contribution

In this work, we present PEGASUS, a highly optimized framework that supports both SIMD style operations and LUT evaluations on large input domains (i.e., the last row of Table II). Our contributions can be summarized as follows.

1) We ease the constraint in the LUT method [43] to accept a significantly larger input (e.g., > 40 bits) at the cost of introducing some approximation errors. stress that a such large input domain is sufficient enough for many applications such as privacy-preserving machine learning. We provide empirical and theoretic analyses on the errors.

2) We present a memory-efficient and fast repacking algorithm. In brief, our repacking key consists of one CKKS ciphertext which is tremendously smaller than that of CHIMERA which consists of thousands of CKKS ciphertexts. We compare our repacking algorithm with the existing methods in Table I.

3) We implemented PEGASUS using SEAL [47]. Different from CHIMERA, we do not export CKKS to the torus. As a result, Pegasus can benefit from the efficiency of the underlying optimized NTT/RNS. Our implementations are publicly available in https://github.com/Alibaba-Gemini-Lab/OpenPEGASUS.

4) We present extensive empirical results including min/max, sorting, division, square-root, and decision tree evaluation on encrypted data. To further demonstrate the potential of Pegasus, we also implement a viable application that runs K-means clustering on thousands of encrypted samples in minutes. To the best of our knowledge, we are the first to achieve practical secure K-means clustering using purely HE in a single-server setting.

II. Preliminaries

A. Notations

For a 2-power number n, we write $R_n = \mathbb{Z}[X]/(X^n + 1)$ and $R_{n,q} = R_n/qR_n \equiv \mathbb{Z}_q[X]/(X^n+1)$. We use lower-case letters with a "hat" symbol such as $\hat{a}$ to represent elements in R_n, and a_j to denote the j-th coefficient of $\hat{a}$. We use the dot symbol $\cdot$ such as $\hat{a} \cdot \hat{b}$ to represent the multiplication of ring elements. We use bold lower-case letters symbol such as $\mathbf{a}$ to represent vectors, and use $\mathbf{a}[j]$ to denote the j-th component of $\mathbf{a}$, and use $\mathbf{a} \| \mathbf{b}$ to denote the concatenation of vectors. We use $\mathbf{a} \lll k$ to denote the left-hand-side rotation of the vector components. We use $\mathbf{a}^\top \mathbf{b}$ to denote inner product of vectors and $\mathbf{a} \circ \mathbf{b}$ for the Hadamard product of vectors. We use bold upper-case letters such as $\mathbf{M}$ to denote matrices, and $\mathbf{M}[i, j]$ to denote the (i, j) entry of $\mathbf{M}$. We denote by $\langle n \rangle$ the set $\{0, \cdots, n-1\}$ for $n \in \mathbb{N}$. We use $\lceil \cdot \rfloor$ to denote the rounding function. Function $\mathcal{I}(P)$ returns 1 if the predicate P is true, and otherwise it returns 0. All logarithms are to base 2.

B. LWE, Ring-LWE Encryption

We use the notation $\mathsf{LWE}^{n,q}_{\mathbf{s}}(m)$ to denote the set of the possible LWE encryption of the message $m \in \mathbb{Z}_q$ under the secret $\mathbf{s} \in \mathbb{Z}^n_q$. $(b, \mathbf{a}) = (m + e - \mathbf{a}^\top \mathbf{s}, \mathbf{a}) \in \mathsf{LWE}^{n,q}_{\mathbf{s}}(m)$, where $\mathbf{a} \in \mathbb{Z}^n_q$ is chosen uniformly at random, and the error e is chosen from an error distribution χ_{err}. The decryption formula for the LWE ciphertext is $b + \mathbf{a}^\top \mathbf{s} \approx m$.

Similarly, a basic RLWE encryption of a message $\hat{m} \in R_{n,q}$ under the key $\hat{s} \in R_n$ is given as $(\hat{b}, \hat{a}) = (\hat{m} + \hat{e} - \hat{a} \cdot \hat{s}, \hat{a}) \in \mathsf{RLWE}^{n,q}_{\hat{s}}(\hat{m})$, where $\hat{a} \in R_{n,q}$ is chosen uniformly at random, and the error $\hat{e}$ is chosen by sampling its coefficients from the error distribution χ_{err} independently. To lighten the notation, we identify the secret key of RLWE as a vector $\mathbf{s}$ where $\mathbf{s}[j] = s_j$ for all $j \in \langle n \rangle$, and we write the RLWE encryption of $\hat{m}$ as $\mathsf{RLWE}^{n,q}_{\mathbf{s}}(\hat{m})$ from now on. Also, PEGASUS can work with an asymmetric RLWE encryption, but we use the notations of symmetric RLWE encryption for the sake of simplicity.

The RLWE encryption supports the following homomorphic addition and multiplication operations. We provide the details of those operations in Appendix A.

- **Addition** $(+)$**.** Given RLWE ciphertexts ct_0 and ct_1, which encrypts ring elements $\hat{p}_0$ and $\hat{p}_1$, the operation $\mathsf{ct}_0 + \mathsf{ct}_1$ results in a ciphertext that encrypts the sum, i.e., $\hat{p}_0 + \hat{p}_1$.
- **Small Plaintext Multiplication** $(\cdot)$**.** Given ciphertext ct that encrypts a ring element $\hat{p}$, and given a "small" plain element $\hat{r}$, the operation $\hat{r} \cdot \mathsf{ct}$ results in a ciphertext that encrypts $\hat{r} \cdot \hat{p}$. Notice that this multiplication can only be used with a low-norm ring element $\hat{r}$.
- **Arbitrary Plaintext Multiplication** $(\diamond)$**.** In order to support multiplication by arbitrary ring elements, one defines an extended encryption $\widetilde{\mathsf{RLWE}}(\cdot; \mathbf{g})$ using a gadget vector $\mathbf{g} \in \mathbb{Z}^d$. Given the extended ciphertext $\tilde{\mathsf{ct}} \in \widetilde{\mathsf{RLWE}}(\hat{p}; \mathbf{g})$ and an arbitrary ring element $\hat{r}$, the operation $\hat{r} \diamond \tilde{\mathsf{ct}}$ results in an RLWE ciphertext that encrypts $\hat{r} \cdot \hat{p}$.
- **External Multiplication** $(\odot)$**.** Furthermore, one can define a GSW-like encryption $\mathsf{RGSW}(\cdot; \mathbf{g})$ to support multiplication by RLWE ciphertexts [28]. Given an RLWE ciphertext ct that encrypts a ring element $\hat{p}_0$, and a GSW ciphertext $\ddot{\mathsf{ct}} \in \mathsf{RGSW}(\hat{p}_1; \mathbf{g})$, the external multiplication $\mathsf{ct} \odot \ddot{\mathsf{ct}}$ results in an RLWE ciphertext that encrypts $\hat{p}_0 \cdot \hat{p}_1$.

Note that the LWE and RLWE encryption include errors whose magnitude will grow along with homomorphic operations on the ciphertexts. To preserve the precision of the decryption, we need to keep the magnitude of errors relatively small. For instance, the gadget vector $\mathbf{g}$ is used to prevent the noise from growing too large. We defer the concrete choice of gadget vectors to the full protocol of PEGASUS. Also if the gadget vector $\mathbf{g}$ is unimportant or clear from the context, we simply omit it in the notation.

C. Coefficients and Slots Manipulation

The ring structure of $\mathbb{Z}[X]/(X^n + 1)$ allows us to encode a real vector $\mathbf{v} \in \mathbb{R}^\ell$ as a ring element of $R_{n,q}$. We use $\mathsf{Ecd}(\mathbf{v}, \Delta) \in R_{n,q}$ to denote the encoding of $\mathbf{v}$ with a scaling factor $\Delta > 0$, and use $\mathsf{Dcd}(\hat{v}, \Delta, \ell) \in \mathbb{R}^\ell$ to denote the decoding of $\hat{v}$ with a scaling factor $\Delta > 0$ and a length $\ell > 0$. We introduce the following properties and functions of the encoding and refer the details to [14], [32].

- **Self-repeating.** $\mathsf{Dcd}\left(\mathsf{Ecd}\left(\mathbf{v} \| \cdots \| \mathbf{v}, \Delta\right), \Delta, \ell\right) = \mathbf{v}$. In other words, the encoding of some self-repeating vectors can be viewed as the encoding of a single copy.
- **Slot-wise Addition and Multiplication.** Given the ring elements $\hat{u}$ and $\hat{v}$ that encode the vectors $\mathbf{u}$ and $\mathbf{v}$, respectively, the addition $\hat{u} + \hat{v}$ (resp. multiplication $\hat{u} \cdot \hat{v}$) results in a ring element that encodes the vector $\mathbf{u} + \mathbf{v}$ (resp. $\mathbf{u} \circ \mathbf{v}$). In the realm of ciphertexts, this property enables us to perform slot-wise addition and multiplication over encrypted vectors.
- **Rotation.** Given the RLWE ciphertext ct which encrypts $\mathsf{Ecd}\left(\mathbf{v}, \Delta\right)$, an integer $k \in \mathbb{N}$, and a rotation key RotK, the operation $\mathsf{RotL}^k(\mathsf{ct}; \mathsf{RotK})$ results in an RLWE ciphertext that encrypts the left-hand-side rotated vector $\mathsf{Ecd}\left(\mathbf{v} \lll k, \Delta\right)$.
- **Rescale.** Given the RLWE ciphertext ct which encrypts $\mathsf{Ecd}\left(\mathbf{v}, \Delta\right)$, and a factor $\Delta' \in \mathbb{R}$, the operation $\mathsf{Rescale}(\mathsf{ct}, \Delta')$ results in a ciphertext (with a smaller modulus) that encrypts $\mathsf{Ecd}\left(\mathbf{v}, \Delta/\Delta'\right)$.
- **Slots To Coefficients.** Given the RLWE ciphertext ct which encrypts $\mathsf{Ecd}\left(\mathbf{v}, \Delta\right)$, the operation $\mathsf{S2C}(\mathsf{ct})$ results in an RLWE ciphertext that encrypts a ring element $\hat{v}$ whose coefficients are $v_i = \Delta \mathbf{v}[i]$ for all possible positions. Indeed, $\mathsf{S2C}$ evaluates the decoding function homomorphically.
- **Coefficients Extraction.** Given the ciphertext $\mathsf{ct} = \mathsf{RLWE}^{n,p}_{\mathbf{s}}(\hat{m})$, and an integer $k \in \langle n \rangle$, the operation $\mathsf{Extract}^k(\mathsf{ct})$ results in $\mathsf{LWE}^{n,p}_{\mathbf{s}}(m_k)$, i.e., an LWE encryption of the k-th coefficient of $\hat{m}$ under the same secret key.

PEGASUS uses the combination of $\mathsf{S2C}$ and $\mathsf{Extract}$ to convert an RLWE ciphertext of an encoded vector to a set of LWE ciphertexts of the vector elements.

D. System and Security Model

We give an overview of our desired security properties. We consider three stakeholders: encryptor(s), cloud, and decryptor. We assume all stakeholders behave semi-honestly and the cloud does not collude with the decryptor. Let x be a private input of the encryptor and y be a private input of the cloud and f be a known function. If the cloud do not provide any private input then we simply set $y = \perp$. We consider the following model. The encryptor(s) sends the ciphertext $\mathsf{Enc}(x)$ to the cloud for the computation of a particular function f. The cloud operates specified homomorphic operations on $\mathsf{Enc}(x)$ and y and sends the resulting ciphertext $\mathsf{Enc}(f(x, y))$ to the decryptor who decrypts to learn $f(x, y)$ but nothing else. Under this model, the security against a semi-honest cloud follows from the fact that the view of the cloud consists of ciphertexts only. Also, even if the decryptor knows x, e.g., the encryptor and decryptor are the same entity, it learns nothing about the input of the cloud except the result $f(x, y)$.

Recently, Li et al. [41] point out that the approximated decryption results of CKKS can leak additional information of the decryption keys. They successfully constructed passive attacks that could recover the decryption keys if given access to the decryption results. We warn that the decryptor in PEGASUS *should not* reveal the decrypted values of CKKS ciphertexts to the encryptor and anyone else without doing any counter-measurement such as [13].

In the following descriptions, we describe the computation on the cloud side and omit the decryption phase of the decryptor and the encryption phase of the encryptor since these operations are either simple or application dependent.

III. Building Blocks of PEGASUS

In this section, we propose PEGASUS, a novel framework that stays in the RLWE form for efficient SIMD computation (e.g., addition, multiplication, and rotation) and transforms to LWE for evaluating a wide range of other complex functions via a fine-grained look-up table approximation (e.g., sigmoid, ReLU, max/min). The PEGASUS transformation consists of four core functions including key-switching ($\mathcal{F}_{\mathrm{KS}}$ of Fig. 1a), look-up-table evaluation ($\mathcal{F}_{\mathrm{LUT}}$ of Fig. 1b), linear transform ($\mathcal{F}_{\mathrm{LT}}$ of Fig. 1c), and approximated modulo ($\mathcal{F}_{\mathrm{mod}}$ of Fig. 1d), which are detailed in this section. It is noteworthy that the LUT function in PEGASUS is not exact and would introduce some (bounded) approximation errors.

A. Overview

PEGASUS works on a set of (R)LWE-based schemes parameterized by different (R)LWE dimensions and arguments. Particularly, we use the notations $\underline{n}, n$, and $\overline{n}$ to denote different (R)LWE dimensions and we write the secret keys in these dimensions as $\underline{\mathbf{s}}$, $\mathbf{s}$, and $\overline{\mathbf{s}}$, respectively. Moreover, the "bar" mark is used to indicate the magnitude of them, i.e., $\underline{n} < n < \overline{n}$. In this section, we use q to denote the modulus of the (R)LWE ciphertexts in a general way, i.e., q might be a large value with thousands of bits or just a machine word-sized integer. We give the specific value of q in the full description of PEGASUS in § IV.

Function $\mathcal{F}_{\mathrm{KS}}$

Input: $\mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}^{\overline{n},q}_{\overline{\mathbf{s}}}(m)$.
Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}^{\underline{n},q}_{\underline{\mathbf{s}}}(m)$ such that $\underline{n} \le \overline{n}$.

(a) Key-switching Function $\mathcal{F}_{\mathrm{KS}}$.

Function $\mathcal{F}_{\mathrm{LUT}}$

Input: $\mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}^{\underline{n},q}_{\underline{\mathbf{s}}}(\lceil \Delta m \rfloor)$ of $m \in \mathbb{R}$. A look-up table function $T(x) : \mathbb{R} \mapsto \mathbb{R}$.
Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}^{n,q}_{\mathbf{s}}(\lceil \Delta T(m) \rfloor + e)$ with a small approximation error e.

(b) Look-up table Function $\mathcal{F}_{\mathrm{LUT}}$.

Function $\mathcal{F}_{\mathrm{LT}}$

Input: $\mathsf{ct}_{\mathsf{in}} \in \mathsf{RLWE}^{\overline{n},q}_{\overline{\mathbf{s}}}(\mathsf{Ecd}(\mathbf{z}, \Delta))$ of $\mathbf{z} \in \mathbb{R}^{\overline{n}}$. A matrix $\mathbf{M} \in \mathbb{R}^{\ell \times \underline{n}}$ and a vector $\mathbf{t} \in \mathbb{R}^{\ell}$ such that $\ell, \underline{n} \le \overline{n}/2$.
Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{RLWE}^{\overline{n},q'}_{\overline{\mathbf{s}}}(\mathsf{Ecd}(\mathbf{M}\mathbf{z} + \mathbf{t}, \Delta))$.

(c) Linear transform Function $\mathcal{F}_{\mathrm{LT}}$.

Function $\mathcal{F}_{\mathrm{mod}}$

Input: $\mathsf{ct}_{\mathsf{in}} \in \mathsf{RLWE}^{\overline{n},q'}_{\overline{\mathbf{s}}}(\hat{z} + q\hat{e})$ for a polynomial $\hat{z} \in R_{\overline{n},q}$ and some small norm polynomial $\hat{e} \in R_{\overline{n}}$.
Output: $\mathsf{ct}_{\mathsf{out}} \in \mathsf{RLWE}^{\overline{n},q''}_{\overline{\mathbf{s}}}(\hat{z} + \hat{e}')$ with an approximation error $\hat{e}$ of a small norm.

(d) Mod q Function $\mathcal{F}_{\mathrm{mod}}$.

Figure 1: Core functions used in PEGASUS.

PEGASUS can start with an RLWE ciphertext, e.g., $\mathsf{ct} \in \mathsf{RLWE}^{\overline{n},q}_{\overline{\mathbf{s}}}(\mathsf{Ecd}(\mathbf{v}, \Delta))$ for $\mathbf{v} \in \mathbb{R}^{\ell}$. On a high level, it can be summarized in a three-step transformation.

1) Extract the elements of the encoded vector and obtains a set of LWE ciphertexts.

$$\mathsf{ct} \to \mathsf{ct}_i \in \mathsf{LWE}^{\overline{n},q}_{\overline{\mathbf{s}}}(\Delta \mathbf{v}[i]) \text{ for } i \in \langle \ell \rangle.$$

2) Evaluate a look-up table $T(x)$ on each LWE ciphertext.

$$\mathsf{ct}_i \to \mathsf{ct}'_i \in \mathsf{LWE}^{\overline{n},q}_{\overline{\mathbf{s}}}(\Delta T(\mathbf{v}[i])) \text{ for } i \in \langle \ell \rangle.$$

3) Repack the set of LWE ciphertexts to a single RLWE ciphertext that encrypts the encoded vector $T(\mathbf{v})$.

$$\{\mathsf{ct}'_i\}_{i \in \langle \ell \rangle} \to \mathsf{ct}'' \in \mathsf{RLWE}^{\overline{n},q''}_{\overline{\mathbf{s}}}(\mathsf{Ecd}(T(\mathbf{v}), \Delta)).$$

In practice, we usually switch down the LWE ciphertext of dimension $\overline{n}$ to a smaller dimension $\underline{n} < \overline{n}$ of the same message before the second step, aiming to reduce the computation cost of the look-up table evaluation. Security is also guaranteed for the smaller lattice dimension $\underline{n}$. The repacking step is used to maintain consistent encoding, taking as input an encoded vector and outputting an evaluated vector in the same encoding form, supporting further arithmetic operations seamlessly.

The four functions in Fig. 1 have been investigated by many existing approaches. Particularly, Chen et al. [11, §3.2] recently presented a key-switching algorithm for LWE ciphertexts of a same lattice dimension. We modify their algorithm for the key-switching function PEGASUS.KS in which the dimension $\overline{n} > \underline{n}$. The details of PEGASUS.KS are given in Fig. 9 in Appendix. Also, we adapt the RNS-friendly approach from [33] for the $\mathcal{F}_{\rm mod}$ function.

However, the existing methods for the other two functions are either insufficient to support a wide range of input or inefficient in terms of computation and memory complexity. Micciancio et al.'s method [43] for $\mathcal{F}_{\rm LUT}$ is confined to a relatively small domain, e.g., $|\Delta \mathbf{v}[i]| \le 2^{10}$. Boura et al.'s method [7] for $\mathcal{F}_{\rm LT}$ might generate a giant key of hundreds of gigabytes. In the following, we present more efficient and flexible algorithms for the $\mathcal{F}_{\rm LUT}$ and $\mathcal{F}_{\rm LT}$ functions. Then we bring them all together to form the full PEGASUS.

B. Look-up Table Evaluation on a Larger Domain

The existing methods that evaluate a look-up table $T(x)$ on an LWE ciphertext $(b, \mathbf{a}) \in \mathsf{LWE}^{\underline{n},q}_{\underline{\mathbf{s}}}(m)$ basically follow [24] which requires $\epsilon q = 2n$ for some positive $\epsilon \in \mathbb{N}$. These methods use an RLWE scheme $\mathsf{RLWE}^{n,q}_{\mathbf{s}}(\cdot)$ to compute the (scaled-up) LWE decryption function modulo the polynomial degree, i.e., $\mathsf{RLWE}^{n,q}_{\mathbf{s}}(X^{\epsilon b + \epsilon \mathbf{a}^\top \underline{\mathbf{s}} \bmod n})$ which equals to $\mathsf{RLWE}^{n,q}_{\mathbf{s}}(X^{\epsilon m})$. Then multiplying a polynomial (constructed according to $T(x)$ and ϵ) can obliviously bring the evaluation $T(m)$ to the 0-th coefficient of the resulting polynomial. Finally, the extraction function $\mathsf{Extract}^0(\cdot)$ gives an LWE ciphertext which encrypts the result $T(m)$.

As we have mentioned, the Ducas-like methods demand $q|2n$ and commonly use a small q, e.g., $q = 2^9$ was used in the implementation [43]. This constraint limits the usability of the Ducas-like LUT methods for applications that involve large integers or fixed-point values of proper precision. For example, the HE-based privacy-preserving machine learning [29], [38], [39] use a scaling factor $\Delta > 2^{30}$ to preserve the precision of fixed-point values before the encryption, demanding a larger modulus $q > \Delta$.

Our Solution via Approximate Decryption. The LWE decryption formula $b + \mathbf{a}^\top \underline{\mathbf{s}}$ is computed in modulo q but we can only perform modulo n in the RLWE scheme. To use a larger ciphertext modulus $q \gg n$ for larger plaintexts, our insight is to use an approximate LWE decryption formula which is computed in modulo n. To do so, we first scale down the modulus from q to $\tilde{\epsilon} n$, i.e., $\tilde{b} = \lceil \frac{\tilde{\epsilon} n}{q} b \rfloor$ and $\tilde{\mathbf{a}} = \lceil \frac{\tilde{\epsilon} n}{q} \mathbf{a} \rfloor$ for some even value $\tilde{\epsilon} \in \mathbb{N}$ such as $\tilde{\epsilon} = 2$. Then the approximate decryption formula is defined as:

$$\tilde{b} + \tilde{\mathbf{a}}^\top \underline{\mathbf{s}} \bmod n \approx \lceil \tilde{\epsilon} n m / q \rfloor. \tag{1}$$

We also need to modify the look-up table as $\tilde{T}(x) = T(\frac{q}{\tilde{\epsilon} n} x)$ to take inputs from the range $x \in [-n/2, n/2) \cap \mathbb{Z}$. By doing so, the origin LUT $T(\cdot)$ can be homomorphically evaluated within a subset of the wider range $[-q/2\tilde{\epsilon}, q/2\tilde{\epsilon})$. The approximation error is bounded by $\mathcal{L}q/(2\tilde{\epsilon} n)$ where $\mathcal{L} > 0$ is the smoothness of the function $T(\cdot)$ (e.g., Lipschitz continuity). Some works have shown that such modification has only a slight effect when $T(x)$ is smooth enough [6]. Also, in our applications of private decision tree and K-means clustering, we do not observe a large precision loss due to this approximation.

Input: $(b, \mathbf{a}) \in \mathsf{LWE}^{\underline{n},q}_{\underline{\mathbf{s}}}(\lceil \Delta m \rfloor)$ of a message $m \in \mathbb{R}$ with a scaling factor Δ such that $|\lceil \Delta m \rfloor| < q/4$. Look-up table function $T(x) : \mathbb{R} \mapsto \mathbb{R}$. The evaluation key EK is a set of RGSW ciphertexts of the key $\underline{\mathbf{s}} \in \{0, \pm 1\}^{\underline{n}}$.

$$\mathsf{EK}_{j,0} \in \mathsf{RGSW}^{n,q}_{\mathbf{s}}(I^+(\underline{\mathbf{s}}[j])), \mathsf{EK}_{j,1} \in \mathsf{RGSW}^{n,q}_{\mathbf{s}}(I^-(\underline{\mathbf{s}}[j])),$$

for $j \in \langle \underline{n} \rangle$. $I^+(x) = \mathcal{I}(x \ge 0)$ and $I^-(x) = \mathcal{I}(x \le 0)$.
Output: A LWE ciphertext $\mathsf{ct}_{\sf out} \in \mathsf{LWE}^{n,q}_{\mathbf{s}}(\cdot)$.

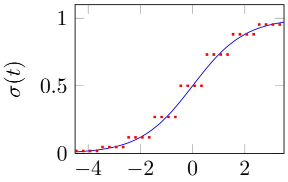
```
Let $\eta_k = kq/(2n\Delta) \in \mathbb{R}$ for $1 \le k \le n/2$. Define a polynomial $\hat{f} \in R_{n,q}$ whose coefficients are
    $$f_j = \begin{cases} \lceil \Delta T(0) \rfloor \text{ if } j = 0 \\ \lceil \Delta T(-\eta_j) \rfloor \text{ if } 1 \le j \le n/2 \\ \lceil -\Delta T(\eta_{n-j}) \rfloor \text{ if } n/2 < j < n \end{cases}.$$
Let $\tilde{b} = \lceil \frac{2n}{q} b \rfloor$ and $\tilde{\mathbf{a}} = \lceil \frac{2n}{q} \mathbf{a} \rfloor$.
Initialize $\mathsf{AC}_0 = (\hat{f} \cdot X^{\tilde{b} \bmod n}, 0) \in R^2_{n,q}$.
for $j = 0, 1, \cdots, \underline{n} - 1$ do
    $\mathsf{t}_j = ((X^{\tilde{\mathbf{a}}[j] \bmod n} - 1) \cdot \mathsf{AC}_j) \odot \mathsf{EK}_{j,0} + \mathsf{AC}_j$
    $\mathsf{AC}_{j+1} = ((X^{-\tilde{\mathbf{a}}[j] \bmod n} - 1) \cdot \mathsf{t}_j) \odot \mathsf{EK}_{j,1} + \mathsf{t}_j$
end for
Output $\mathsf{Extract}^0(\mathsf{AC}_{\underline{n}})$ as $\mathsf{ct}_{\sf out}$.
```

Figure 2: Look-up Table Evaluation PEGASUS.LUT

Our algorithm for $\mathcal{F}_{\rm LUT}$ is shown in Fig. 2. The major cost of this algorithm lies in the $2\underline{n}$ ciphertext multiplications ($\odot$) The size of the our evaluation key EK is $O(|\mathbf{g}|\underline{n}n \log q)$ bits. Note that in the algorithm, RGSW is used to simply encrypt binary values which can be viewed as degree-0 polynomials. Given a complex function $G(x)$ to be evaluated, we can discretize the domain of x to small intervals and map each interval $[u, v]$ to the function value $G(\frac{u+v}{2})$. Fig. 2 builds a plain look-up polynomial $\hat{f}$ whose coefficients are the function values corresponding to the intervals, whereas the decryption $\tilde{b} + \tilde{\mathbf{a}}^\top \underline{\mathbf{s}} \bmod n$ serves as a look-up index to select the appropriate coefficient (function

Complex function: $\sigma(t) = \frac{1}{1+\exp(-t)}$

t	-4	-3	-2	-1
$T(t)$	0.018	0.047	0.119	0.269
t	0	1	2	3
$T(t)$	0.5	0.731	0.881	0.953



Look-up polynomial:
$\hat{f} = \sigma(0) + \sigma(-1)X + \sigma(-2)X^2 + \sigma(-3)X^3 + \sigma(-4)X^4 - \sigma(3)X^5 - \sigma(2)X^6 - \sigma(1)X^7$

Look-up index: $i = \tilde{b} + \tilde{\mathbf{a}}^\top \underline{\mathbf{s}} \bmod n \in [-4, 4)(n = 8)$.

Ouput: $-f_{n-i}$ if $i > 0$ else output $f_{|i|}$

Figure 3: Look-up table example.

value). A toy example is given in Fig. 3 to illustrate the process.

Theorem 1: Given the input $\mathsf{ct}_{\mathsf{in}} \in \mathsf{LWE}^{\underline{n},q}_{\underline{\mathbf{s}}}(\lceil \Delta m \rfloor)$ and for any ternary secret $\underline{\mathbf{s}} \in \{0, \pm 1\}^{\underline{n}}$, Fig. 2 outputs $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}^{n,q}_{\mathbf{s}}(\lceil \Delta T(m) \rfloor + e_{\mathrm{lut}})$ with a bounded error $e_{\mathrm{lut}} \in \mathbb{R}$.

The upper bound of the error e_{lut} depends on the noise of the input ciphertext, the concrete choice of gadget vector and the smoothness of the function $T(\cdot)$. We will introduce the concrete choice of gadget vectors used by PEGASUS in the next section. We defer the correctness proof of Theorem 1 and give a detailed noise analysis of e_{lut} to Appendix.

C. More Efficient Repacking

PEGASUS keeps consistent encoding by repacking a set of LWE ciphertexts, i.e., from Fig. 2, to an RLWE ciphertext that encrypts an encoded vector of the LUT evaluations. One advantage of this consistency is that, the existing approaches that uses CKKS can benefit from the capability of PEGASUS. For example, PEGASUS can replace the activation function in EVA [21] and keep other parts unchanged.

The basic idea of repacking $\{\mathsf{ct}_i \in \mathsf{LWE}^{\underline{n},q}_{\underline{\mathbf{s}}}(m_i)\}_i$ to $\mathsf{ct} \in \mathsf{RLWE}^{\overline{n},q'}_{\overline{\mathbf{s}}}(\mathsf{Ecd}\,(m_0, \cdots))$ consists of two steps. First we homomorphically evaluate the partial LWE decryption and obtain $\mathsf{ct}' \in \mathsf{RLWE}^{\overline{n},q'}_{\overline{\mathbf{s}}}(\mathsf{Ecd}\,(m_0 + qr_0, \cdots))$, given the secret key $\underline{\mathbf{s}}$ in some form of encryption. Then we can evaluate modulo q homomorphically using the methods such as [4], [9], [33].

CHIMERA suggest to perform the partial decryption step using a repacking key that consists of $\underline{n}$ $\widetilde{\mathsf{RLWE}}$ ciphertexts, that is $\{\mathsf{RK}_i \in \widetilde{\mathsf{RLWE}}^{\overline{n},Q}_{\overline{\mathbf{s}}}(\underline{\mathbf{s}}[i]; \mathbf{g})\}_{i \in \langle \underline{n} \rangle}$ $(Q \gg q' \geq q)$. As a result, the size of this key is $O(2|\mathbf{g}|\underline{n}\overline{n} \log Q)$ bits which can be tremendously large. For instance, the size of their repacking key can be more than 80 gigabytes for a parameter set that $\overline{n} = 2^{16}, \underline{n} = 2^{10}$, $|\mathbf{g}| = 7$, and $\log Q \approx 735$.

Our Solution via Linear Transform. We present a faster and memory-efficient algorithm for the partial decryption step. The insight is that the partial LWE decryption of a batch of LWE ciphertexts can be written as a linear transform $\mathbf{A}\underline{\mathbf{s}} + \mathbf{b}$ where the i-th row of $\mathbf{A} \in \mathbb{Z}_q^{\ell \times \underline{n}}$ comes from an LWE ciphertext. Instead of encrypting elements of $\underline{\mathbf{s}}$ separately, we suggest to use encoding and export the repacking key as a single RLWE ciphertext $\mathsf{RK} \in \mathsf{RLWE}^{\overline{n},q}_{\overline{\mathbf{s}}}(\mathsf{Ecd}\,(\underline{\mathbf{s}}, \Delta_r))$. The scaling factor Δ_r is used to preserve the precision because the secret key $\underline{\mathbf{s}}$ is commonly a low-norm vector. Our repacking key is about $O(2\overline{n} \log q)$ bits (cf. Table I), e.g., about 12 MB for the same parameter set $\overline{n} = 2^{16}, \underline{n} = 2^{10}$, and $\log q \approx 735$. Also, the LWE dimension $\underline{n} \ll \overline{n}$ is fixed in advance but the repacking size ℓ can change within the range $1 \leq \ell \leq \overline{n}/2$. In other words, we need to evaluate the linear transform $\mathbf{A}\underline{\mathbf{s}} + \mathbf{b}$, given the ciphertext of the encoded vector $\mathsf{Ecd}\,(\underline{\mathbf{s}}, \Delta_r)$ and "plain" value of $\mathbf{A}$ and $\mathbf{b}$ such that the matrix $\mathbf{A}$ can be a "tall" matrix (i.e., $\ell > \underline{n}$) or a "short" matrix (i.e., $\ell < \underline{n}$).

Table III: Computation complexity of different methods for the transform function.

	Method	#Rotation	Depth
$\ell < \underline{n}$	[30]	$O(\sqrt{\underline{n}})$	1
	[38]	$O(\lceil \ell\underline{n}/\overline{n} \rceil + \log(\overline{n}/\underline{n}))$	1
	[10, §3.4]	$O(\epsilon + \lceil \ell\underline{n}/(\epsilon\overline{n}) \rceil \log(\underline{n}/\epsilon))^\dagger$	2
	Ours	$O(\sqrt{\ell} + \log(\underline{n}/\ell))$	1
$\ell \geq \underline{n}$	[30]	$O(\sqrt{\ell})$	1
	Ours	$O(\sqrt{\underline{n}})$	1

† $1 \leq \epsilon \leq \ell$ in [10].

There are some existing approaches to the linear transform function $\mathcal{F}_{\mathrm{LT}}$. Their efficiency greatly depends on the shape of the matrix $\mathbf{A}$. As shown in Table III, the diagonal encoding method of Halevi et al. [30] using $O(\sqrt{\max(\ell, \underline{n})})$ homomorphic rotations is thus more suitable for square matrices. Juvekar et al. [38] improved [30] especially for "short" matrices. Chen et al. [10] further improved the efficiency of [38] at the cost of more multiplicative depths. These existing approaches are not sufficient to cover both "tall" and "short" matrices efficiently. We now present a new algorithm for the $\mathcal{F}_{\mathrm{LT}}$ function in Fig. 5 which is faster and more flexible than the existing approaches.

Theorem 2: On the input $\mathsf{ct}_{\mathsf{in}} \in \mathsf{RLWE}^{\overline{n},q}_{\overline{\mathbf{s}}}(\mathsf{Ecd}\,(\mathbf{z}, \Delta_r))$, a plain matrix $\mathbf{M} \in \mathbb{R}^{\ell \times \underline{n}}$ such that $\ell, \underline{n} < \overline{n}$, and a vector $\mathbf{t} \in \mathbb{R}^{\ell}$, Fig. 5 outputs an RLWE ciphertext $\mathsf{ct}_{\mathsf{out}} \in \mathsf{RLWE}^{\overline{n},q/\Delta_r}_{\overline{\mathbf{s}}}(\mathsf{Ecd}\,(\mathbf{Mz} + \mathbf{t}, \Delta'_r))$.

As the example shown in Fig. 4, the main idea in Fig. 5 is first to convert, conceptually, the rectangular matrix $\mathbf{M}$ to a square matrix by repeating the rectangular matrix itself (called *tiling*). We said "conceptually" because, we only use a subset of the diagonals of the tiling matrix, which are constructed in Step 1 by looping through the rows and columns of $\mathbf{M}$. This tiling is always possible without zero-padding because the number of rows and columns of $\mathbf{M}$ is always a power-of-2 value. The baby-step-giant-step (BSGS) technique [31] in Step 2 aims to sum up some products of plaintext–ciphertext with a specific offset of homomorphic rotations. Indeed, Step 2 results a ciphertext $\tilde{\mathsf{ct}}$ that already encrypts the result vector $\mathbf{Mz}$ for the "tall" case. On the other hand, when $\ell < \underline{n}$, intuitively, $\tilde{\mathsf{ct}}$ can be viewed as

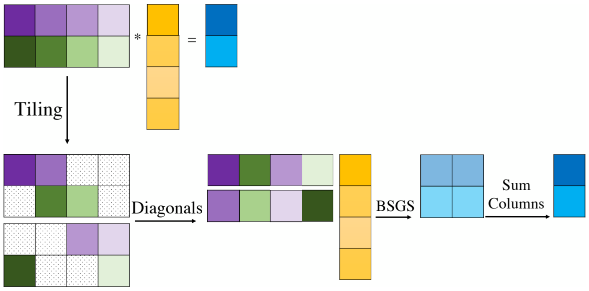


Figure 4: "Short" matrix example of PEGASUS.LT. $\ell = 2$ and $\underline{n} = 4$. Dotted cells would be omitted in the tiling step.

a ciphertext that encrypts a matrix of ℓ rows and $(\underline{n}/\ell)$ columns where the sum of the *column vectors* equals to $\mathbf{Mz}$ (see Fig. 4). The post-possessing from Step 6 to Step 7 aims to sum up the encrypted columns, resulting in ct_γ that encrypts a self-repeating vector $\mathsf{Ecd}\left(\mathbf{Mz}\|\cdots\|\mathbf{Mz}, \Delta_r\Delta_r'\right)$. It can just be viewed as $\mathsf{Ecd}\left(\mathbf{Mz}, \Delta_r\Delta_r'\right)$ according to the property of the encoding function. Finally, we use $\mathsf{Rescale}(\cdot, \Delta_r)$ to reach the specified scaling factor Δ_r', and then we add $\mathbf{t}$ at the end. The proof of Theorem 2 is deferred to Appendix B.

We compare the computation complexity of Fig. 5 with the existing approaches in Table III. It is worthy to note that the computation costs of our method are *independent* of the repacking size ℓ when $\ell \geq \underline{n}$. In other words, we can repack a large amount of LWE ciphertexts to an RLWE ciphertext without introducing extra overheads, which is confirmed empirically in §VI.

Remark: Fig. 5 can also be used for other interests. For example, the matrix $\mathbf{M}$ can also be given in the form of encryption as long as it follows the tiling-style encoding. Indeed, in our secure K-means clustering application (§V), we reuse Fig. 5 to compute the product of an *encrypted* matrix and encrypted vectors.

IV. Our Framework

The full algorithm of PEGASUS includes many specific optimization techniques that are already investigated in previous work. We briefly introduce these techniques here.

A. Optimizations

1) *RNS and NTT:* A well-known technique to optimize the integer polynomial arithmetic on $R_{n,q}$ is to use a full RNS by taking the modulus q as the product of distinct and machine-word-sized primes, i.e., $q = \prod_{i \in \langle L \rangle} q_i$. One can achieve up to $L\times$ improvement in polynomial arithmetic according to the ring isomorphism $R_{n,q} \to \prod_{i \in \langle L \rangle} R_{n,q_i}$.

```
Input: $\mathsf{ct}_{\mathsf{in}} \in \mathsf{RLWE}_{\overline{\mathbf{s}}}^{\overline{n},q}(\mathsf{Ecd}\left(\mathbf{z}, \Delta_r\right))$. Rotation key RotK. A scaling factor $\Delta_r' > 0$. A plain matrix $\mathbf{M} \in \mathbb{R}^{\ell \times \underline{n}}$ such that $\ell, \underline{n} < \overline{n}$. A vector $\mathbf{t} \in \mathbb{R}^{\ell}$.
Output: An RLWE ciphertext $\mathsf{ct}_{\mathsf{out}} \in \mathsf{RLWE}_{\overline{\mathbf{s}}}^{\overline{n},q/\Delta_r}(\cdot)$.
 1: Tiling and Diagonals. Let $\ddot{n} = \max(\ell, \underline{n})$ and $\tilde{n} = \min(\ell, \underline{n})$. Define $\tilde{n}$ vectors $\{\tilde{\mathbf{m}}_j\}_{j=0}^{\tilde{n}-1}$ by going through the rows and columns of $\mathbf{M}$
        $$\tilde{\mathbf{m}}_j[r] = \mathbf{M}[r \bmod \ell, r + j \bmod \underline{n}] \text{ for } r \in \langle \ddot{n} \rangle.$$
 2: Baby-Step. Let $\tilde{g} = \lceil \sqrt{\tilde{n}} \rceil$. For $g \in \langle \tilde{g} \rangle$, compute $c_g = \mathsf{RotL}^g(\mathsf{ct}_{\mathsf{in}})$.
 3: Giant-Step. Let $\tilde{b} = \lceil \tilde{n}/\tilde{g} \rceil$. Compute
        $$\tilde{\mathsf{ct}} = \sum_{b \in \langle \tilde{b} \rangle} \mathsf{RotL}^{b\tilde{g}} \left( \sum_{g \in \langle \tilde{g} \rangle} \mathsf{Ecd}\left(\tilde{\mathbf{m}}_{b\tilde{g}+g} \gg b\tilde{g}, \Delta_r'\right) \cdot c_g \right).$$
 4: if $\ell \geq \underline{n}$ then
 5:     Output $\mathsf{Rescale}(\tilde{\mathsf{ct}}, \Delta_r) + \mathsf{Ecd}\left(\mathbf{t}, \Delta_r'\right)$ as $\mathsf{ct}_{\mathsf{out}}$.
                    $\triangleright$ $\tilde{\mathsf{ct}} \in \mathsf{RLWE}_{\overline{\mathbf{s}}}^{\overline{n},q}(\mathsf{Ecd}\left(\mathbf{Mz}, \Delta_r\Delta_r'\right))$
 6: else                $\triangleright$ Sum Columns
 7:     Let $\gamma = \log(\underline{n}/\ell)$ and $\mathsf{ct}_0 = \tilde{\mathsf{ct}}$.
 8:     Update iteratively for $1 \leq j \leq \gamma$
            $$\mathsf{ct}_j = \mathsf{RotL}^{\ell 2^j}(\mathsf{ct}_{j-1}) + \mathsf{ct}_{j-1}.$$
                    $\triangleright$ $\mathsf{ct}_\gamma = \mathsf{RLWE}_{\overline{\mathbf{s}}}^{\overline{n},q}(\mathsf{Ecd}\left(\mathbf{Mz}, \Delta_r\Delta_r'\right))$
 9:     Output $\mathsf{Rescale}(\mathsf{ct}_\gamma, \Delta_r) + \mathsf{Ecd}\left(\mathbf{t}, \Delta_r'\right)$ as $\mathsf{ct}_{\mathsf{out}}$.
10: end if
```

Figure 5: Linear Transform PEGASUS.LT

2) *Gadget Vectors:* The products ($\diamond$ and $\odot$) use a gadget vector to keep the noise magnitude small. Particularly, we use two types of gadget vectors in PEGASUS, i.e., the digit decomposition gadget $\mathbf{g}_{\text{digit}}$ and RNS decomposition gadget $\mathbf{g}_{\text{rns}}$. Given the decomposition base $B > 1$ and $d = \lceil \log_B q \rceil$, the $\mathbf{g}_{\text{digit}}$ is given as $\mathbf{g}_{\text{digit}}[i] = B^i$. The RNS gadget takes advantage of the RNS representation and the special modulus technique [27]. The $\mathbf{g}_{\text{rns}}$ vector is given as $\mathbf{g}_{\text{rns}}[i] = \frac{qq'}{q_i}((\frac{q}{q_i})^{-1} \bmod q_i)$ where q' is a special modulus. One advantage of $\mathbf{g}_{\text{rns}}$ against the digit gadget is that $\mathbf{g}_{\text{rns}}$ can render faster multiplications $\diamond$ and $\odot$. However, $\mathbf{g}_{\text{rns}}$ requires a larger modulus, which demands a larger lattice dimension to guarantee the security of the encryption scheme. Particularly, we use $\mathbf{g}_{\text{digit}}$ in the LWE switching key (i.e., SwK in Fig. 6) and use $\mathbf{g}_{\text{rns}}$ in the LUT evaluation key (i.e,. EK in Fig. 6).

B. Full Algorithm of PEGASUS

Now we present the full description of PEGASUS in Fig. 6. PEGASUS uses $L > 0$ moduli for the RNS representation and one special modulus q' for the gadget vector. Notice that our gadget vectors $\mathbf{g}_{\text{digit}}$ and $\mathbf{g}_{\text{rns}}$ only involve the first modulus q_0. Indeed, PEGASUS performs

Public Parameters:

- Ciphertext moduli $q_0, q_1, \cdots, q_{L-1}$ and a special modulus q'. Define $Q_i = \prod_{l \in \langle i \rangle} q_l$ for $1 \le i \le L$.
- Digit decomposition gadget vector $\mathbf{g}_{\text{digit}} = [1, B_{\mathsf{ks}}, \cdots, B_{\mathsf{ks}}^{d_{\mathsf{ks}}}]$ for some $B_{\mathsf{ks}}, d_{\mathsf{ks}} > 0$ and $B_{\mathsf{ks}}^{d_{\mathsf{ks}}} \ge q_0$.
- RNS decomposition gadget vector $\mathbf{g}_{\text{rns}} = [q' \bmod q_0]$.
- Rescaling factors $0 < \Delta, \Delta_r, \Delta'_r < q_0$.

Public Keys:

- Switching key $\mathsf{SwK}_{\overline{\mathbf{s}} \to \underline{\mathbf{s}}}$ is a vector of RLWE encryption of $\overline{\mathbf{s}}$ under the key $\underline{\mathbf{s}}$ where the j-th ciphertext is $\widetilde{\mathsf{RLWE}}_{\underline{\mathbf{s}}}^{\underline{n}, q_0}\left(\sum_{l \in \langle \underline{n} \rangle} \overline{\mathbf{s}}[j\underline{n} + l] X^l; \mathbf{g}_{\text{digit}}\right)$ for $j \in \langle \overline{n} / \underline{n} \rangle$. The switching key $\mathsf{SwK}_{\mathbf{s} \to \underline{\mathbf{s}}}$ is defined in a similar way.
- LWE evaluation key EK is a set of GSW encryption of the elements of $\underline{\mathbf{s}}$ under the key $\mathbf{s}$ where

$$\mathsf{EK}_{j,0} \in \mathsf{RGSW}_{\mathbf{s}}^{n, q' q_0}(I^+(\underline{\mathbf{s}}[j]); \mathbf{g}_{\text{rns}}), \mathsf{EK}_{j,1} \in \mathsf{RGSW}_{\mathbf{s}}^{n, q' q_0}(I^-(\underline{\mathbf{s}}[j]); \mathbf{g}_{\text{rns}}) \text{ for } j \in \langle \underline{n} \rangle.$$

  Specifically, $I^+(x) = \mathbf{1}\{x \ge 0\}$ and $I^-(x) = \mathbf{1}\{x \le 0\}$.
- Repacking key $\mathsf{RK} \in \mathsf{RLWE}_{\overline{\mathbf{s}}}^{\overline{n}, Q_L}(\mathsf{Ecd}(\underline{\mathbf{s}}, \Delta_r))$.
- Rotation key of the CKKS scheme RotK.

Input:

- A level-l RLWE ciphertext ($l > 1$) of an encoded vector $\mathbf{v} \in \mathbb{R}^\ell$, i.e., $\mathsf{ct}_{\mathsf{in}} \in \mathsf{RLWE}_{\overline{\mathbf{s}}}^{\overline{n}, Q_l}(\mathsf{Ecd}(\mathbf{v}, \Delta))$.
- A look-up table function $T(x) : \mathbb{R} \mapsto \mathbb{R}$.

Output: A RLWE ciphertext $\mathsf{ct}_{\mathsf{out}} \in \mathsf{RLWE}_{\overline{\mathbf{s}}}^{\overline{n}, Q_{L'}}(\mathsf{Ecd}(T(\mathbf{v}), \Delta))$, i.e., the evaluation of $T(x)$ on the elements of $\mathbf{v}$.

```
1: Slots to coefficients and drop moduli. $\mathsf{ct}' = \mathsf{S2C}(\mathsf{ct}_{\mathsf{in}})$    ▷ $\mathsf{ct}' \in \mathsf{RLWE}_{\overline{\mathbf{s}}}^{\overline{n}, q_0}(\Delta \hat{v})$
2: Extract Coefficients. $\mathsf{ct}_i = \mathsf{Extract}^i(\mathsf{ct}')$ for each $i \in \langle \ell \rangle$.    ▷ $\mathsf{ct}_i \in \mathsf{LWE}_{\overline{\mathbf{s}}}^{\overline{n}, q_0}(\Delta \mathbf{v}[i])$
3: for $i \in \langle \ell \rangle$ do    ▷ parallel
4:     Switch from $\overline{n}$ to the smaller dimension $\underline{n}$. $\dot{\mathsf{ct}}_i = \mathsf{PEGASUS.KS}(\mathsf{ct}_i, \mathsf{SwK}_{\overline{\mathbf{s}} \to \underline{\mathbf{s}}})$.
                                                       ▷ $\dot{\mathsf{ct}}_i \in \mathsf{LWE}_{\underline{\mathbf{s}}}^{\underline{n}, q_0}(\Delta \mathbf{v}[i])$
5:     Evaluate the look-up table. $\ddot{\mathsf{ct}}_i = \mathsf{PEGASUS.LUT}(\dot{\mathsf{ct}}_i, \mathsf{EK}, T(x))$.    ▷ $\ddot{\mathsf{ct}}_i \in \mathsf{LWE}_{\mathbf{s}}^{n, q_0}(\Delta T(\mathbf{v}[i]))$
6:     Switch from $n$ to the smaller dimension $\underline{n}$. $\dddot{\mathsf{ct}}_i = (b_i, \mathbf{a}_i) = \mathsf{PEGASUS.KS}(\ddot{\mathsf{ct}}_i, \mathsf{SwK}_{\mathbf{s} \to \underline{\mathbf{s}}})$.
                                                       ▷ $\dddot{\mathsf{ct}}_i \in \mathsf{LWE}_{\underline{\mathbf{s}}}^{\underline{n}, q_0}(\Delta T(\mathbf{v}[i]))$
7: end for
8: Define $\mathbf{b} = [b_0, \cdots, b_{\ell-1}]$ and $\mathbf{A} \in \mathbb{Z}_{q_0}^{\ell \times \underline{n}}$ where the $i$-th row of $\mathbf{A}$ is $\mathbf{a}_i$.    ▷ $\mathbf{A}\mathbf{s} + \mathbf{b} \bmod q_0 = \mathsf{Ecd}(T(\mathbf{v}), \Delta)$
9: Evaluate the linear transform $\tilde{\mathsf{ct}} = \mathsf{PEGASUS.LT}(\mathsf{RK}, \mathsf{RotK}, \Delta'_r, \mathbf{A}, \mathbf{b})$.    ▷ $\tilde{\mathsf{ct}} \in \mathsf{RLWE}_{\overline{\mathbf{s}}}^{\overline{n}, Q_{L-1}}(\mathsf{Ecd}(\mathbf{A}\mathbf{s} + \mathbf{b}, \Delta'_r))$
10: Evaluate the modulo $q_0$ on $\tilde{\mathsf{ct}}$ via $\mathcal{F}_{\text{mod}}$ and output the result as $\mathsf{ct}_{\mathsf{out}}$.    ▷ $\mathsf{ct}_{\mathsf{out}} \in \mathsf{RLWE}_{\overline{\mathbf{s}}}^{\overline{n}, Q_{L'}}(\mathsf{Ecd}(T(\mathbf{v}), \Delta))$
```

Figure 6: PEGASUS, Full Protocol

the key-switching (Fig. 9) and the LUT evaluation (Fig. 2) over the ring R_{*,q_0} for a various lattice dimension but a fixed modulus q_0 (reasons see below).

The input RLWE ciphertexts of PEGASUS are given in the RNS representation with l many moduli. PEGASUS first evaluates the slots-to-coefficients function $\mathsf{S2C}$, followed by $\mathsf{Extract}$ to obtain the encoded elements as a set of LWE ciphertexts (Step 1 and Step 2). As a result, it requires $l > 1$ since $\mathsf{S2C}$ itself might consume 1 or 2 ciphertext moduli. The computational costs of the following LUT evaluation and repacking depends on the modulus size of the LWE ciphertexts. To lighten the computation, PEGASUS drops all RLWE moduli but keeps the first one q_0 before Step 3. As a result, the following key-switching (Step 4 and Step 6) and LUT (Step 5) take as the input of LWE ciphertexts from $\mathsf{LWE}^{*,q_0}(\cdot)$ with a varying lattice dimension but a fixed modulus. Also by using the smallest number of moduli, it renders smaller switching keys and LUT evaluation keys (cf. Table IV). The independent LUT evaluations on LWE ciphertexts from Step 4 to Step 5 can be easily parallelized by multicores. Finally, PEGASUS repacks a set of LWE ciphertexts to an RLWE ciphertext by simply using the implementations of the $\mathcal{F}_{\text{LT}}$ and $\mathcal{F}_{\text{mod}}$ (Step 9-10).

The main computation errors in Fig. 6 come from the key-switching in Step 4 and Step 6, and the LUT evaluation in Step 5. We defer the error analysis of these two functions to Appendix due to the space limit.

Remark: The very recent work from [11] provides a different primitive for the repacking function. Briefly, given the LWE ciphertexts $\{\mathsf{ct}_i \in \mathsf{LWE}_{\underline{\mathbf{s}}}^{\underline{n}, q}(m_i)\}_{i=0}^{\ell-1}$, they first convert them to an RLWE ciphertext $\mathsf{ct} \in \mathsf{RLWE}_{\overline{\mathbf{s}}}^{\overline{n}, q}(\hat{m})$ that encrypts a polynomial such that $\hat{m} = \sum_i m_i X^i$. Then they homomorphically evaluate the encoding function and get $\mathsf{ct}' \in \mathsf{RLWE}_{\overline{\mathbf{s}}}^{\overline{n}, q'}(\mathsf{Ecd}(m_0, m_1, \cdots, m_{\ell-1}))$. Indeed, their conversion part needs $O(\ell + \log(\overline{n}/\ell))$ homomorphic rotations (cf. Table I), and they might need $O(\sqrt{\ell})$ more rotations for the encoding step. On the other hand, the number of homomorphic rotations used in PEGASUS.LT (Step 9) is about $\min(\sqrt{\underline{n}}, \sqrt{\ell}+\log(\underline{n}/\ell))$ which is independent of the largest lattice dimension $\overline{n}$. Also, the complexity of our instantiation of $\mathcal{F}_{\text{mod}}$ from [33] is independent of the repacking size ℓ. In conclusion, our method is more computation-efficient than [11] when ℓ is large, e.g., $\ell > \underline{n}$.

C. Advanced Features of PEGASUS

For the sake of simplicity, we describe PEGASUS with one LUT as input in Fig. 6. PEGASUS can provide much more flexibility. We now introduce its advanced features.

1) *Multiple LUTs:* We can repeat Step 5 and Step 6 to evaluate a sequence of LUTs. Also, we can perform additions and subtractions over the LWE ciphertexts before and after Step 5. Indeed, one of our application that computes the index of the minimum value of an encrypted vector (described in the following section) needs some basic LWE additions and subtractions between two LUT evaluations.

2) *Customizable Encoding Layout:* Moreover, we can rearrange the layout of encoding in Step 9 by just reordering the rows of $\mathbf{A}$ and $\mathbf{b}$. This property is very useful for many HE-based applications, in which different types of encoding layout are usually mixed for better performance. For instance, [38] used two encoding layouts for the secure evaluation of convolution layers and fully-connected layers. To switch between different layouts, [38] needs to decrypt ciphertexts in the middle with a random masking. On the other hand, such switching of encoding layouts is almost free in PEGASUS without interactions with the decryptor.

3) *Tunable Output Level:* The number of moduli (denoted as L') of the resulting ciphertext from Step 10 depends on the number of moduli used for constructing the repacking key RK. This parameter can be changed to a smaller one as long as it provides enough depths for the $\mathcal{F}_{\text{mod}}$, to tune a best repacking performance for the specific application. For instance, $L' = 3$ might be enough for secure convolution network inference since the polynomial operations (i.e., convolution) between two non-polynomial operations (e.g., activation and pooling) consume one multiplicative depth.

To wrap up, the size of the repacking key of PEGASUS is significantly smaller than CHIMERA's repacking key. Also, our repacking approach demands $O(\min(\sqrt{\underline{n}}, \sqrt{\ell} + \log(\underline{n}/\ell)))$ homomorphic rotations which is much lighter than the repacking method of CHIMERA that needs $O(\underline{n})$ rotations. Moreover, Pegasus can leverage the optimized NTT/RNS for the underlying arithmetic that is significantly faster than its Torus counterpart used by CHIMERA.

V. Applications

In this section, we present some useful applications of PEGASUS. Specifically, we compute some statistical functions such as reciprocal, min/max, max-pooling, and sorting. Also, we propose a private decision tree evaluation and a secure K-means clustering algorithm using PEGASUS.

A. Application I: Basic Non-polynomial Functions

We show some use cases for computing a wide range of common functions using the look-up tables evaluation of PEGASUS. Particularly, we compute the following functions on LWE ciphertexts.

- **sigmoid/ReLU/sqrt/reciprocal.** Many useful functions can be evaluated via one LUT, for instance sigmoid, ReLU, square-root and reciprocal. These functions are commonly used in machine learning algorithms.
- **min/max** The min/max of two values m_0 and m_1 can be computed as follows

$$\min(m_0, m_1) = 0.5(m_0 + m_1) - 0.5|m_0 - m_1|,$$
$$\max(m_0, m_1) = 0.5(m_0 + m_1) + 0.5|m_0 - m_1|.$$

Thus, we can use two LUTs $T_0(x) = 0.5x$ and $T_1(x) = 0.5|x|$ to compute the min/max on LWE encrypted values.

Also, we can find out the min/max of an array by a min/max-tree. For example, the max of 4 values $m_0, \cdots, m_3$ can be computed as $\max(\max(m_0, m_1), \max(m_2, m_3))$. To find out the min/max of t encrypted elements, in general, we need to evaluate $O(2t)$ LUTs.

- **max-pooling.** Max-pooling is a famous operation of deep learning. Although the average-pooling is more HE-friendly, it has been proved that the max-pooling can perform better in many image classification tasks [35]. Also, we notice that the number input t is usually very small for max-pooling, e.g., $t = 4$. As long as the scaled maximum value still follows the range constraint of Algorithm 2, i.e., $|t \cdot \max(m_0, m_1, \cdots, m_{t-1})| < q_0/(4\Delta)$, we can use a smaller number of LUTs to compute the max-pooling. Indeed, we replace each max-operation in the max-tree with the scaled-max $\max'(m_0, m_1) = m_0 + m_1 - |m_0 - m_1|$, resulting a t-scaled value. The factor t can then be removed by one extra LUT $T_2(x) = x/t$. In total, we can compute the max-pooling within $O(t)$ LUTs for a relatively small t.
- **sorting.** We adapt the bitonic sorting [2] which is data independent and can be parallelized easily. A bitonic sorting network uses two types of swap, i.e., ascending swap and descending swap

$$\text{ASwap}(m_0, m_1) = \min(m_0, m_1), \max(m_0, m_1)$$
$$\text{DSwap}(m_0, m_1) = \max(m_0, m_1), \min(m_0, m_1).$$

In other words, each swap needs two LUTs since the min and max function computes the same LUTs. To sort an array of t encrypted elements, our bitonic sorting network evaluates $O(2t \log_2^2 t)$ LUTs.

- **min-index/max-index.** The min-index (resp. max-index) function takes as input of an array $m_0, m_1, \cdots, m_{t-1}$ and returns a binary sequence $(b_0, b_1, \cdots, b_{t-1})$ such that $b_k = 1$

iff. m_k is the min (resp. max) value of the the array. We first compute the min (resp. max) value of the array by a min-tree (resp. max-tree). Then, we can compute the min-index (resp. max-index) using the predicate function $T(x) = \mathcal{I}(x \le 0)$ evaluated on $m_i - \min$ (resp. $m_i - \max$) with extra $O(t)$ LUT evaluations. In total the min-index/max-index evaluation needs $O(3t)$ LUTs.

Cheon et al.'s numeric method [16] can also compute the max/min-index but at the cost of a longer latency than PEGASUS which is confirmed in our experiments.

B. Application II: Private Decision Tree Evaluation

Decision tree is a fundamental and popularly used classification algorithm. Many current works have considered a cloud-based classification service using decision tree evaluation, using HEs to protect the query privacy for the user, and the model privacy for the cloud [5], [37], [40], [50], [51]. However, these approaches are either interactive, using $O(\log N)$ rounds of communication between the cloud and the user or having a large communication overhead.

On the other hand, we can achieve a single-round and communication-efficient private decision tree evaluation by using PEGASUS. Consider that each *edge* of the tree has a binary variable associated to it. The value of the binary $z_{i,j}$ is 0 if, on the input vector $\mathbf{a} \in \mathbb{R}^d$, one should go from the node ϕ_i to ϕ_j, and 1 otherwise. For each leaf node, we write v_k to denote the sum of the binary variables along a path from the root node to the leaf ϕ_{N+k}. We designate v_k as *classification predictor* as the input is classified in the leaf node ϕ_{N+k} iff. $v_k = 0$. Given the encryption of the input vector, the cloud can compute the binary values $\{z_{i,j}\}$ using N LUTs. Then the cloud aggregates the encrypted binary values along each path in the tree and it obtains the classification predictors. The cloud then calls N distinct LUTs on the classification predictors, i.e., $T_k(x) = C_k \cdot \mathcal{I}(x \le 0)$ where $C_k > 0$ is the class label in the leaf node ϕ_{N+k}. Finally, the cloud sends a ciphertext of the sum $\sum_k T_k(v_k)$ to the client. The major computation costs on the cloud side is $O(2N)$ LUTs and about $O(\lceil d/\underline{n} \rceil \underline{n} \log q_0)$ bits are sent.

C. Application III: Secure Clustering on Encrypted Data

Rao et al. [45] and Cheon et al. [17] have considered a privacy-preserving K-mean clustering using homomorphic encryption and two collusion-free servers. We consider the same K-means algorithm but in a single server setting: data are encrypted and collected to a server which performs *all the computation* for the clustering. We first introduce the basic K-means algorithm. Then we show how to perform the K-means clustering on encrypted data using PEGASUS.

Table IV: Proposed parameter sets. B_{ks} is the digit decomposition base. The scaling factors are set as $\Delta \approx 2^{36}$, $\Delta_r \approx 2^{45}$ and $\Delta'_r = 1$. The hamming weight of secret keys is set as $h = 64$. The value $\log \overline{Q}$ denotes the total bits of modulus used in SEAL (including the special modulus). All parameter sets provide at least 119-bit of security level according to [1].

Encryption	Parameters
$\widetilde{\mathsf{RLWE}}_{\underline{\mathbf{s}}}^{\underline{n},q_0}(\cdot)$	$\underline{n} = 2^{10}, q_0 \approx 2^{45}, \sigma_{\text{ks}} = 2^{13}, B_{\mathsf{ks}} = 2^7, d_{\mathsf{ks}} = 7$
$\mathsf{RGSW}_{\mathbf{s}}^{n,q' q_0}(\cdot)$	$n = 2^{12}, q' \approx 2^{60}, \sigma_{\text{lut}} = 2^{10}$
$\mathsf{RLWE}_{\overline{\mathbf{s}}}^{\overline{n},*}(\cdot)$	$\overline{n} = 2^{16}, q_i \approx 2^{45}, \sigma_{\text{ckks}} = 3.19, \log \overline{Q} = 795$
Key	**Size**
$\mathsf{SwK}_{\overline{\mathbf{s}} \to \underline{\mathbf{s}}}$	$2 d_{\mathsf{ks}} \overline{n} \log q_0$bits ≈ 5.0 MB
$\mathsf{SwK}_{\mathbf{s} \to \underline{\mathbf{s}}}$	$2 d_{\mathsf{ks}} n \log q_0$bits ≈ 315 KB
EK	$8 \underline{n} n \log(q' q_0)$ bits ≈ 420 MB
RK	$2 \overline{n} \log(\prod_{i \in \langle L \rangle} q_i)$ bits ≈ 12 MB

Given a dataset of $\mathbf{X} \in \mathbb{R}^{N \times d}$ where each row $\mathbf{x}_i$ is a data point and given an positive integer $K \ll N$, the K-means clustering algorithm aims to find out K centroids $\{\mathbf{c}_j \in \mathbb{R}^d\}_{j \in \langle K \rangle}$ that minimize the total distances between centroids and data points, i.e., $\min \sum_i \sum_j D(\mathbf{x}_i, \mathbf{c}_j)$. We consider the square Euclidean distance $D(\mathbf{x}_i, \mathbf{c}_j) = \mathbf{x}_i^\top \mathbf{c}_j$ here. To solve this optimization problem, iterative algorithms are used. We write $\mathbf{c}_j^{(t)}$ as the j-th centroid in the t-th iteration. Without loss of generality, the t-th update can be described as follows.

1) The distance between each data point and centroid is computed in a form of matrix $\mathbf{D} \in \mathbb{R}^{N \times K}$ such that $\mathbf{D}[i,j] = D(\mathbf{x}_i, \mathbf{c}_j^{(t)})$ for each $i \in \langle N \rangle$ and $j \in \langle K \rangle$.
2) Compute a matrix $\mathbf{B} \in \{0,1\}^{N \times K}$ such that $\mathbf{B}[i,j] = \mathcal{I}(j = \text{argmin}_l \mathbf{D}[i,l])$. In other words, $\mathbf{B}[i,j] = 1$ means that the j-th centroid is the closest one of the i-th data point.
3) Update the j-th centroid as $\mathbf{c}_j^{(t+1)} = \frac{1}{\sum_i \tilde{\mathbf{b}}_j[i]} \mathbf{X}^\top \tilde{\mathbf{b}}_j$, where $\tilde{\mathbf{b}}_j \in \{0,1\}^N$ is the j-th *column* of $\mathbf{B}$.

Our objective is to perform the above update procedure on encrypted data points. We assume $\mathbf{X}$ is already encoded and encrypted using RLWE encryption. The distance matrix $\mathbf{D}$ can be computed easily using the homomorphic operations of the RLWE encryption. Suppose $\mathbf{D}$ is computed as a RLWE ciphertext, we first extract the ciphertexts of the distances $\{\mathsf{ct}_{i,j} \in \mathsf{LWE}_{\underline{\mathbf{s}}}^{\underline{n},q_0}(\Delta \mathbf{D}[i,j])\}_{i \in \langle N \rangle, j \in \langle K \rangle}$. Then we can compute the min-index among the LWE ciphertexts

$$\ddot{\mathsf{ct}}_{i,0}, \cdots, \ddot{\mathsf{ct}}_{i,K-1} \leftarrow \text{min-index}(\mathsf{ct}_{i,0}, \cdots, \mathsf{ct}_{i,K-1}). \quad (2)$$

That is $\ddot{\mathsf{ct}}_{i,j} \in \mathsf{LWE}_{\underline{\mathbf{s}}}^{\underline{n},q_0}(\Delta \mathbf{B}[i,j])$. Also the reciprocal of counting $\sum_i \mathbf{b}_j[i]$ can be computed by feeding the sum $\sum_i \ddot{\mathsf{ct}}_{i,j}$ to the reciprocal look-up table $T(x) = 1/x$. Finally, we repack all the computed LWE ciphertexts and obtain RLWE ciphertexts of the encoded vectors and scalars $\mathsf{ct}_j \in \mathsf{RLWE}_{\overline{\mathbf{s}}}^{\overline{n},Q_{L'}}(\mathsf{Ecd}\left(\tilde{\mathbf{b}}_j, \Delta\right))$, and $\mathsf{ct}'_j \in \mathsf{RLWE}_{\overline{\mathbf{s}}}^{\overline{n},Q_{L'}}(\lceil \Delta / \sum_i \tilde{\mathbf{b}}_0[i] \rfloor)$ for $j \in \langle K \rangle$, which are then used to update the centroid vectors. We stress that the linear transform $\mathbf{X}^\top \tilde{\mathbf{b}}_j$ can be computed using an similar tiling idea in Algorithm 5 which needs $O(\sqrt{K} + \log_2(N/K))$ rotations on ciphertexts. Also, the number of clusters K is commonly a small value, e.g., $K < 10$. This allows us to

Table V: Single-threaded micro-benchmarks of PEGASUS. ℓ denotes the number of slots or the repacking size. We compare with the implementation of CHIMERA from [19].

	$\mathsf{KS}(\bar{\mathbf{s}} \rightarrow \underline{\mathbf{s}})$	LUT	$\mathsf{KS}(\mathbf{s} \rightarrow \underline{\mathbf{s}})$
[19]	4192ms	60s	260ms
Ours	20.14ms	0.93s	1.49ms
Speedup	208×	64×	174×

$\mathsf{S2C}$ [32]†	LT	$\mathcal{F}_{\mathrm{mod}}$ [33]†
$\log \ell = 8, 10, 12$	$\log \ell = 8, 10, 12$	
0.78s, 1.28s, 2.02s	16.76s, 44.50s, 44.65s	7.06s

† The performance of our implementations of [32] and [33].

Table VI: Benchmarks of some non-polynomial functions on Pegasus. t is the number of LWE ciphertexts. The time for extraction and repacking were not included.

	Throughput		
#Threads	**sigmoid/ReLU/sqrt/reciprocal**	**Max-Pooling** 2×2	 4×4
1	1.06/s	0.25/s	0.07/s
4	3.95s	0.91/s	0.26/s
8	7.95/s	1.97/s	0.50/s
16	15.34/s	3.90/s	0.98/s
20	20.77/s	4.83/s	1.21/s

	Latency				
#Threads	**Min-Index** $t = 2^8$	 $t = 2^7$	 $t = 2^4$	**Sort** $t = 2^6$	 $t = 2^5$
1	791.84s	395.39s	43.24s	1380.50s	493.30s
4	238.73s	124.60s	176.60s	409.09s‡	148.88s
8	118.19s	62.57s	9.72s†	199.82s	73.02s
16	66.15s	36.33s	5.17s	104.33s	38.27s
20	44.73s	25.26s	5.17s	84.03s	34.10s

† [16] took about 236s (24×). ‡ [25] took about 43 mins (6.3×).

optimize the complexity of (2) from $O(3K)$ LUTs to $O(2K)$ LUTs using the same idea in the max-pooling.

VI. Evaluation

We now report detailed performance of PEGASUS, including micro-benchmarks of the basic operations and the performance of the applications of decision tree and K-means clustering. We have implemented PEGASUS on top of the SEAL library with extra optimizations including a faster NTT [46] and a faster $(\diamond)$ operator using a lazy-reduction. Also, we have implemented the $\mathsf{S2C}$ and $\mathcal{F}_{\mathrm{mod}}$ functionalities based on [32] and [33, §4], respectively. More specifically, our implementations of $\mathsf{S2C}$ and $\mathcal{F}_{\mathrm{mod}}$ require 2 and 9 multiplicative depths, respectively. All experiments are performed on a server with an Intel Xeon Platinum 8269CY CPU (20-cores) at 2.50 GHz, compiled with gcc 7.5.0.

A. Concrete Parameters

We use $L = 16$ RNS primes for the ciphertext modulus. The first modulus is chosen $q_0 \approx 2^{45}$ and other moduli are chosen slightly larger than q_0. A special modulus $q' \approx 2^{60}$ is used. We set the secret distribution as a hamming weight distribution over the set of ternary polynomials of coefficients in $\{0, \pm 1\}$ subject to the condition that it has exactly $h = 64$ nonzero entries. Each coefficient/entry of (R)LWE error is drawn according to the discrete Gaussian distribution centered at zero with standard deviation σ. Table IV illustrates the parameter sets used in our experiments. For this parameter sets, the repacked ciphertexts from Algorithm 6 have $L' = 6$ moduli left. This enables us to run our secure clustering application without using the CKKS bootstrapping [12] because we need 2 levels for updating the centroids, 1 level for computing the distances, and 2 more levels for $\mathsf{S2C}$.

B. Micro-benchmarks for PEGASUS

The single-threaded micro-benchmarks of PEGASUS are given in Table V. We can see that the main-loop (i.e., Step 4 to Step 6) in Algorithm 6 took about a second. In other words, the LUT evaluation throughput of PEGASUS is about 1 LUT per second which can be boosted up easily using multicores. To compare, we ran some benchmarks on the same machine using the open-sourced implementation of CHIMERA from [19]. Basically, we set the dimensions $\underline{n}$ and n as same as ours and set other parameters of CHIMERA to match the security level. However, the repacking implementation of CHIMERA is not provided in [19] so we compared the key-switching functions and the LUT function. Table V shows that our framework is about two orders of magnitude faster than CHIMERA for these basic functions. Indeed, we adapt the evaluation key idea from [43] in Fig. 2. This enables us to use a ternary secret whose security is more well studied than the binary secret used in CHIMERA. Note that, if we use a binary secret key $\underline{\mathbf{s}}$ as the CHIMERA does, the performance of our LUT evaluation would be accelerated by 2×.

C. Applications Evaluation

1) Basic Non-polynomial Functions: We evaluated the non-polynomials functions described in the previous section. We measured the throughput and latency (not including the time for encryption) for computing these functions on encrypted data. We report the performing numbers in Table VI. From this table, we can see that the performance of PEGASUS can be easily accelerated by using more machine cores. The acceleration efficiency for small fan-in functions e.g., sigmoid and max-pooling was nearly optimal, i.e., around 0.95. Also, the latency of PEGASUS for the min-index and sorting function is also superior to the state-of-the-arts. For example, [16] took more than 236s to compute the max-index of $t = 2^4$ encrypted elements using 8 threads, compared to PEGASUS that just took 9.72s. Also, [25] used

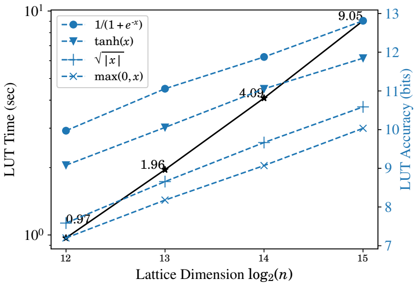
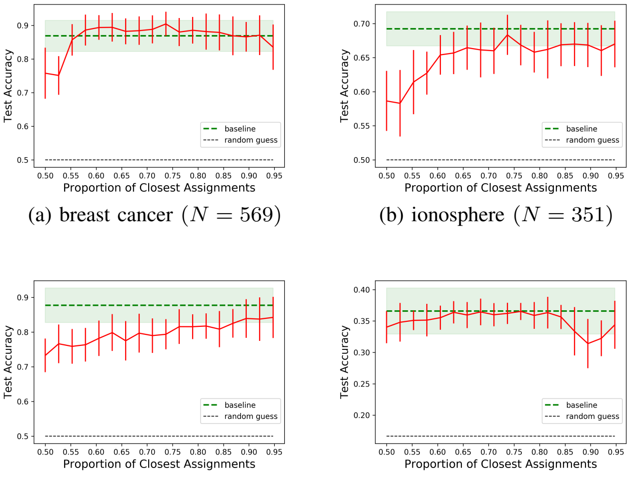


Figure 7: Trade-off between LUT accuracy and performance in PEGASUS. The LUT Accuracy is given as $-\log|f(x) - T_f(x)|$ for the function f. We demonstrate four functions $f(x) = \frac{1}{1+e^{-x}}$, $f(x) = \tanh(x)$, $f(x) = \sqrt{|x|}$, and $f(x) = \max(0, x)$ on uniform random messages from $x \in [-8, 8]$.

4 threads and took about 43 minutes to sort $t = 2^6$ encrypted integers using the bit-wise HE [48], which is about 6.3× slower than ours.

The algorithms in PEGASUS do introduce errors to the final results, including errors from the (R)LWE encryption, key-switching, and LUT. These errors can be controlled under a reasonable small value. According to our empirical results, the key-switching introduces small errors of a magnitude of 2^{-14}. Also, the approximation errors of our LUT can be within $[2^{-10}, 2^{-7}]$ under the parameters from Table IV. We consider such small errors are acceptable for many applications such as machine learning and information retrieval. Indeed, the accuracy of our LUT can be improved by using a larger lattice dimension n at a cost of a longer LUT evaluation time (see Fig. 7).

2) Private Decision Tree Evaluation: We evaluated our private decision tree evaluation algorithm on three real datasets from the UCI repository [23] that is the Iris, Housing, and Spambase dataset. We trained decision trees on randomly selected 80% of the data points using the scikit-learn library [44], and used the remaining 20% for testing. We measured the execution time on both sides of the cloud and the client and measured the communication costs counting all the ciphertexts sent by the client and the cloud. . The results are given in Table VII. We compare with the previous *single-round* HE-based methods [37], [51]. The approach [51] uses the bit-wise TFHE to evaluate the boolean circuit representation of the decision tree on inputs of $\delta = 16$-bit integers, which is sufficient precision for the used datasets. That is, [51] did not introduce accuracy loss on the classification. On the contrary, PEGASUS introduced about 3% miss-classification on the used datasets. Also, [51] was about 2× faster than ours, not counting the communication time. Indeed, they needs to exchange $O(32 \cdot 2^{10} \cdot \delta(d+N))$ bits of ciphertexts. On the other hand, for $d < \underline{n}$, our method sends only one RLWE ciphertext and one LWE ciphertext, leading to much smaller communication overhead, i.e., about 1% of that of [51]. The method [37] is fast on the cloud side but it sends $O(N)$ RLWE ciphertexts to the client to decryption, which could lead to a large communication overhead and a long decryption time when N increases. Also, [37] used 12-bit fixed point values and it might introduce some rounding errors, e.g., about 2% accuracy loss observed in our experiments.



(a) breast cancer ($N = 569$)

(b) ionosphere ($N = 351$)

(c) wine ($N = 178$)

(d) Control charts ($N = 600$)

Figure 8: Test accuracy of K-means clustering with a proportion of closest assignments. The baseline is obtained from the scikit-learn package [44]. All numbers are averaged from 20 runs of 10-fold cross-validation.

3) Secure K-means Clustering: We evaluated our secure K-means algorithm on synthesis data $\mathbf{X} \in \mathbb{R}^{N \times d}$ where the values of $\mathbf{X}$ were sampled from $[-1, 1]$ uniformly a random. The running time of our algorithm is separated into five sectors and reported in Table VIII. The $O(2KN)$ LUTs from the Min-Index sector took about 95% of the computation time, which can be easily accelerated by using more cores. Also, the running time of arithmetic computation, i.e., computing Euclidean distance (4th column) and updating cluster centroids (8th column) took just a few seconds. The repacking sector converts N LWE ciphertexts to RLWE.

We can see that the running time of repacking grows in $\sqrt{N}$ when $N \leq \underline{n}$, and it stays over 60 seconds for a larger number of data points $\underline{n} < N < \overline{n}$. We emphasize that our K-means algorithm is faster than the existing HEs-based secure K-means algorithms. For example, [34] used TFHE for the same K-means clustering problem. However, the boolean circuit representation of the K-means algorithm could consist of millions of gates which could take a long time to evaluate. To improve the efficiency, [34] suggested to use an approximate comparison by comparing the highest bits of the encrypted fixed-point values at the cost of intro-

Table VII: Compare the performance of our private decision tree evaluation algorithm with the existing approaches. N is the number of internal nodes in the tree, d is the number of features, and C is the number of classification labels. Computation time includes the evaluation time on both sides of the cloud and the client. On the cloud side, 16 threads were used for these experiments, following the same settings of [37], [51].

Setting				Computation			Communication			Classification Accuracy			
Data	N	d	C	[37]	[51]	Ours	[37]	[51]	Ours	[37]	[51]	Ours	On Plaintexts
Iris	≈ 10	4	3	0.59s	0.94s	1.87s	1.65 MB	1.19 MB	16.89 KB	97.37%	95.33%	94.74%	97.37%
Housing	≈ 100	13	2	10.27s	6.30s	10.71s	13.12 MB	6.63 MB	16.89 KB	100.0%	98.12%	98.04%	100.0%
Spambase	≈ 60	57	2	6.88s	3.66s	6.75s	11.54 MB	7.36 MB	16.89 KB	87.86%	87.06%	85.03%	87.86%

Table VIII: Running time of *one update* of our privacy-preserving clustering algorithm where N and K is the number of data points and the number of clusters, respectively. Dimension $d = 16$ and 20 threads were used.

Setting		Closest Assign Ratio ρ	Break-down Running Time (for $n = 2^{12}$)[†]							
N	K	$n = 2^{12}(2^{13})$	Distance	Extract	Min-Index & Recip.	Repacking	Centroid	Total	[34, §6][‡]	Speedup
256	2	$82.03\%(92.97\%)$	1.56s	1.83s	51.53s	25.93s	0.35s	1.35min	19.81min	14×
	4	$83.59\%(97.27\%)$	2.69s	2.60s	107.78s		0.50s	2.33min	39.61min	17×
	8	$86.72\%(94.92\%)$	3.11s	4.36s	211.31s		0.87s	4.09min	79.23min	19×
1024	2	$79.88\%(95.41\%)$	1.58s	5.33s	109.27s	55.22s	0.33s	3.66min	79.23min	21×
	4	$86.13\%(95.80\%)$	2.79s	8.95s	387.08s		0.45s	7.57min	158.45min	21×
	8	$86.52\%(95.31\%)$	3.10s	16.31s	844.58s		0.97s	15.34min	316.89min	20×
4096	2	$82.10\%(94.65\%)$	1.61s	17.78s	756.33s	60.69s	0.32s	13.95min	316.89min	22×
	4	$86.33\%(96.00\%)$	2.73s	32.48s	1500.37s		0.45s	26.61min	633.79min	23×
	8	$88.79\%(96.14\%)$	3.03s	57.04s	3000.85s		1.04s	52.04min	1267.58min	24×

[†] For $n = 2^{13}$, the runtime of the Min-Index sector (i.e,. 6th column) was increased by about 2× and other sectors were almost unchanged.

[‡] By estimation using the provided runtimes per update, per centroid, per data point, per CPU in [34, § 6].

ducing about 5% misclassification rate (cf. Fig 6 of [34]). Nevertheless, their optimized method could take up to 20× longer time than ours.

Our algorithm introduces approximation errors during the computation. A data point could be assigned to a "wrong" centroid particularly when two centroids are both close to that data point. In the 3rd column of Table VIII, we counted the percentage of data points that were properly assigned to the closest centroid. We observed that about $\rho \approx 82\%$ of the data points were assigned to the closest centroid when using the parameter $n = 2^{12}$. This proportion ρ can be improved to about 95% by using a larger $n = 2^{13}$ at the cost of increasing the time of the Min-Index sector by about 2×.

To study the effects of ρ on the classification accuracy of the K-means, we simulated on four real datasets from [23]. In brief, in each centroid update, we randomly chose a proportion of $1 - \rho$ of the training points and independently assign them to a randomly chosen centroid, excluding the closest one. From the simulation results shown in Fig 8, the classification accuracy of K-means seems quite robust. The accuracy did not drop much (within 4% when $\rho \geq 0.8$) compared to the baseline in which all data points were assigned to the closest centroid.

VII. Conclusion

In conclusion, this work presents PEGASUS, a highly optimized framework that allows us to efficiently and effectively perform both polynomial functions and non-polynomial functions on encrypted data. The main technical contribution is two-fold. We propose a new FHEW → CKKS conversion algorithm achieving better performance and a significantly smaller key compared to the prior work of CHIMERA [7]. Also, we extend the prior approach for homomorphic look-up table evaluation [43] to accept a wide range of input. Finally, we showed that PEGASUS can be applied to many real-life applications, including the private decision tree evaluation on an encrypted query, and secure outsourced K-means clustering on an encrypted dataset. We consider that PEGASUS is practical for a wide range of privacy-preserving scenarios, especially for cloud-based applications. One of our future work is to improve the performance of PEGASUS using hard-wares such as GPGPU and FPGA.

Acknowledgment

The authors would like to thank the anonymous reviewers of this paper for their insightful comments. The authors would also like to thank Hao Chen, Ph.D. from Facebook for helpful discussions on CHIMERA and TFHE.

References

[1] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *J. Mathematical Cryptology*, vol. 9, no. 3, pp. 169–203, 2015.

[2] K. E. Batcher, “Sorting networks and their applications,” in *American Federation of Information Processing Societies: AFIPS Conference Proceedings: 1968 Spring Joint Computer Conference, Atlantic City, USA, 30 April - 2 May 1968*, pp. 307–314.

[3] J. W. Bos, K. E. Lauter, and M. Naehrig, “Private predictive analysis on encrypted medical data,” *J. Biomed. Informatics*, vol. 50, pp. 234–243, 2014.

[4] J. Bossuat, C. Mouchet, J. R. Troncoso-Pastoriza, and J. Hubaux, “Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys,” *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 1203, 2020.

[5] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, “Machine learning classification over encrypted data,” in *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, USA, February 8-11, 2015*.

[6] C. Boura, N. Gama, M. Georgieva, and D. Jetchev, “Simulating homomorphic evaluation of deep learning predictions,” in *Cyber Security Cryptography and Machine Learning - Third International Symposium, CSCML 2019, Beer-Sheva, Israel, June 27-28, 2019, Proceedings*, pp. 212–230.

[7] ——, “Chimera: Combining ring-lwe-based fully homomorphic encryption schemes,” *Journal of Mathematical Cryptology*, vol. 14, no. 1, pp. 316 – 338, 2020.

[8] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “(Leveled) fully homomorphic encryption without bootstrapping,” in *Innovations in Theoretical Computer Science 2012, Cambridge, USA, January 8-10, 2012*, pp. 309–325.

[9] H. Chen, I. Chillotti, and Y. Song, “Improved bootstrapping for approximate homomorphic encryption,” in *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, pp. 34–54.

[10] H. Chen, W. Dai, M. Kim, and Y. Song, “Efficient multi-key homomorphic encryption with packed ciphertexts with application to oblivious neural network inference,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pp. 395–412.

[11] ——, “Efficient homomorphic conversion between (ring) LWE ciphertexts,” *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 15, 2020.

[12] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, “Bootstrapping for approximate homomorphic encryption,” in *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, pp. 360–384.

[13] J. H. Cheon, S. Hong, and D. Kim, “Remark on the security of ckks scheme in practice,” *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 1581, 2020.

[14] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, “Homomorphic encryption for arithmetic of approximate numbers,” in *ASIACRYPT 2017, Hong Kong, China, December 3-7, 2017*, pp. 409–437.

[15] J. H. Cheon, D. Kim, and D. Kim, “Efficient homomorphic comparison methods with optimal complexity,” in *ASIACRYPT 2020, Daejeon, South Korea, December 7-11, 2020*, pp. 221–256.

[16] J. H. Cheon, D. Kim, D. Kim, H. Lee, and K. Lee, “Numerical method for comparison on homomorphically encrypted numbers,” in *ASIACRYPT 2019, Kobe, Japan, December 8-12, 2019*, pp. 415–445.

[17] J. H. Cheon, D. Kim, and J. H. Park, “Towards a practical clustering analysis over encrypted data,” *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 465, 2019.

[18] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, “TFHE: fast fully homomorphic encryption over the torus,” *J. Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[19] “CHIMERA Implementation,” https://github.com/DPPH/chimera-iDash2018, 2020, team CEA-EPFL-Inpher: code for the iDash 2018 competition.

[20] A. Costache and N. P. Smart, “Which ring based somewhat homomorphic encryption scheme is best?” in *Topics in Cryptology - CT-RSA 2016 - The Cryptographers’ Track at the RSA Conference 2016, San Francisco, USA, February 29 - March 4, 2016, Proceedings*, pp. 325–340.

[21] R. Dathathri, B. Kostova, O. Saarikivi, W. Dai, K. Laine, and M. Musuvathi, “EVA: an encrypted vector arithmetic language and compiler for efficient homomorphic computation,” in *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation, PLDI 2020, London, UK, June 15-20, 2020*, pp. 546–561.

[22] R. Dathathri, O. Saarikivi, H. Chen, K. Laine, K. E. Lauter, S. Maleki, M. Musuvathi, and T. Mytkowicz, “CHET: an optimizing compiler for fully-homomorphic neural-network inferencing,” in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, USA, June 22-26, 2019*, pp. 142–156.

[23] D. Dua and C. Graff, “UCI machine learning repository,” 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[24] L. Ducas and D. Micciancio, “FHEW: bootstrapping homomorphic encryption in less than a second,” in *EUROCRYPT 2015, Sofia, Bulgaria, April 26-30, 2015*, pp. 617–640.

[25] N. Emmadi, P. Gauravaram, H. Narumanchi, and H. Syed, “Updates on sorting of fully homomorphic encrypted data,” in *2015 International Conference on Cloud Computing Research and Innovation, ICCCRI 2015, Singapore, October 26-27, 2015*, pp. 19–24.

[26] J. Fan and F. Vercauteren, “Somewhat practical fully homomorphic encryption,” *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, 2012.

[27] C. Gentry, S. Halevi, and N. P. Smart, “Homomorphic evaluation of the AES circuit,” in *CRYPTO 2012, Santa Barbara, USA, August 19-23, 2012.*, pp. 850–867.

[28] C. Gentry, A. Sahai, and B. Waters, “Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based,” in *CRYPTO 2013, Santa Barbara, USA, August 18-22, 2013.*, pp. 75–92.

[29] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing, “CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy,” in *ICML 2016, New York City, USA, June 19-24, 2016*, pp. 201–210.

[30] S. Halevi and V. Shoup, “Algorithms in HElib,” in *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, USA, August 17-21, 2014, Proceedings, Part I*, pp. 554–571.

[31] ——, “Faster homomorphic linear transformations in HElib,”

in *CRYPTO 2018, Santa Barbara, USA, August 19-23, 2018*, pp. 93–120.

[32] K. Han, M. Hhan, and J. H. Cheon, “Improved homomorphic discrete fourier transforms and FHE bootstrapping,” *IEEE Access*, vol. 7, pp. 57 361–57 370, 2019.

[33] K. Han and D. Ki, “Better bootstrapping for approximate homomorphic encryption,” in *Topics in Cryptology - CT-RSA 2020 - The Cryptographers’ Track at the RSA Conference 2020, San Francisco, USA, February 24-28, 2020, Proceedings*, pp. 364–390.

[34] A. Jäschke and F. Armknecht, “Unsupervised machine learning on encrypted data,” in *Selected Areas in Cryptography 2018, Calgary, Canada, August 15-17, 2018, Revised Selected Papers*, pp. 453–478.

[35] Jianchao Yang, Kai Yu, Yihong Gong, and T. Huang, “Linear spatial pyramid matching using sparse coding for image classification,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 1794–1801.

[36] W. jie Lu, S. Kawasaki, and J. Sakuma, “Using fully homomorphic encryption for statistical analysis of categorical, ordinal and numerical data,” in *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, USA, February 26 - March 1, 2017.*

[37] W. jie Lu, J. Zhou, and J. Sakuma, “Non-interactive and output expressive private comparison from homomorphic encryption,” in *AsiaCCS 2018, Incheon, Republic of Korea, June 04-08, 2018*, pp. 67–74.

[38] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, “GAZELLE: A low latency framework for secure neural network inference,” in *USENIX Security 2018, Baltimore, USA, August 15-17, 2018*, pp. 1651–1669.

[39] M. Kim, Y. Song, S. Wang, Y. Xia, and X. Jiang, “Secure logistic regression based on homomorphic encryption,” *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 74, 2018.

[40] Á. Kiss, M. Naderpour, J. Liu, N. Asokan, and T. Schneider, “Sok: Modular and efficient private decision tree evaluation,” *PoPETs*, vol. 2019, no. 2, pp. 187–208.

[41] B. Li and D. Micciancio, “On the security of homomorphic encryption on approximate numbers,” *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 1533, 2020.

[42] C. A. Melchor, J. Barrier, L. Fousse, and M. Killijian, “XPIR : Private information retrieval for everyone,” *PoPETs*, vol. 2016, no. 2, pp. 155–174.

[43] D. Micciancio and Y. Polyakov, “Bootstrapping in FHEW-like cryptosystems,” *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 86, 2020.

[44] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[45] F. Rao, B. K. Samanthula, E. Bertino, X. Yi, and D. Liu, “Privacy-preserving and outsourced multi-user k-means clustering,” in *Conference on Collaboration and Internet Computing 2015, Hangzhou, China, October 27-30, 2015*, pp. 80–89.

[46] M. Scott, “A note on the implementation of the number theoretic transform,” in *Cryptography and Coding - 16th IMA International Conference, IMACC 2017, Oxford, UK, December 12-14, 2017, Proceedings*, pp. 247–258.

[47] “Microsoft SEAL (release 3.5),” https://github.com/Microsoft/SEAL, Apr. 2020, microsoft Research, Redmond, WA.

[48] N. P. Smart and F. Vercauteren, “Fully homomorphic encryption with relatively small key and ciphertext sizes,” in *Public Key Cryptography 2010, Paris, France, May 26-28, 2010.*, pp. 420–443.

[49] ——, “Fully homomorphic SIMD operations,” *Des. Codes Cryptogr.*, vol. 71, no. 1, pp. 57–81, 2014.

[50] R. K. H. Tai, J. P. K. Ma, Y. Zhao, and S. S. M. Chow, “Privacy-preserving decision trees evaluation via linear functions,” in *ESORICS 2017, Oslo, Norway, September 11-15, 2017*, pp. 494–512.

[51] A. Tueno, Y. Boev, and F. Kerschbaum, “Non-interactive private decision tree evaluation,” in *DBSec 2020, Regensburg, Germany, June 25-26, 2020*, pp. 174–194.

Appendix

A. Operations of the RNS Variant of the CKKS Scheme

We give the details of the CKKS operations used in PEGASUS.

- **Setup.** Dimension n is a 2-power value. $L > 0$ distinct primes $q_0, q_1, \cdots, q_{L-1}$. Let $Q_i = \prod_{k=0}^{i} q_i$ for all $i \in \langle L \rangle$. The error distribution χ_{err} is defined as the centered discrete normal distribution of standard deviation $\sigma_{\mathrm{ckks}} > 0$. The encryption distribution χ_{enc} is defined as the uniform distribution over $\{0, \pm 1\}^n$.
- **KeyGen.** Sample the secret key $\mathbf{s}$ from $\{0, \pm 1\}^n$. Set the public key $\mathsf{pk} \in \mathsf{RLWE}_{\mathbf{s}}^{n,Q_{L-1}}(0)$, i.e., a RLWE sample of zero.
- **Encryption.** On a plaintext $\hat{m} \in R_{n,Q_{L-1}}$, computes $\mathsf{ct} = (\hat{m} + \hat{e}, 0) + \hat{u} \cdot \mathsf{pk} \in \mathsf{RLWE}_{\mathbf{s}}^{n,Q_{L-1}}(\hat{m})$, where the coefficients of $\hat{e} \in R_{n,Q_{L-1}}$ are sampled from χ_{err} and $\hat{u} \in R_{n,Q_{L-1}}$ are sampled from χ_{enc}.
- **Extended Encryption.** On a plaintext $\hat{m} \in R_{n,Q_{L-1}}$, and a gadget $\mathbf{g} \in \mathbb{Z}^d$, compute a set of RLWE samples
$$(c_0, \cdots, c_{d-1}) \in \widetilde{\mathsf{RLWE}}_{\mathbf{s}}^{n,Q_{L-1}}(\hat{m}; \mathbf{g})$$
where $c_i \in \mathsf{RLWE}_{\mathbf{s}}^{n,Q_{L-1}}(\mathbf{g}[i] \cdot \hat{m})$ for $i \in \langle d \rangle$.
- **Decryption.** On the ciphertext $\mathsf{ct} = (c_0, c_1) \in R_{n,Q_l}^2$ for any $l \in \langle L \rangle$, compute and output $c_0 + c_1 \cdot \mathbf{s} \in R_{n,Q_l}$.
- **Addition.** On the ciphertexts $(c_0, c_1) \in \mathsf{RLWE}_{\mathbf{s}}^{n,Q_l}(\hat{m})$ and $(c_0', c_1') \in \mathsf{RLWE}_{\mathbf{s}}^{n,Q_l}(\hat{m}')$ for any $l \in \langle L \rangle$, compute $(c_0 + c_0', c_1 + c_1') \in \mathsf{RLWE}_{\mathbf{s}}^{n,Q_l}(\hat{m} + \hat{m}')$.
- **Multiplication.** On the ciphertexts $(c_0, c_1) \in \mathsf{RLWE}_{\mathbf{s}}^{n,Q_l}(\hat{m})$ and $(c_0', c_1') \in \mathsf{RLWE}_{\mathbf{s}}^{n,Q_l}(\hat{m}')$ and a relinearization key $\mathsf{RelK} \in \widetilde{\mathsf{RLWE}}_{\mathbf{s}}^{n,Q_l}(\mathbf{s}^2; \mathbf{g})$, for any $l \in \langle L \rangle$, first compute $\tilde{c}_0 = c_0 \cdot c_0', \tilde{c}_1 = c_0 \cdot c_1' + c_1 \cdot c_0'$ and $\tilde{c}_2 = c_1 \cdot c_1'$. Then output $(\tilde{c}_0, \tilde{c}_1) + \tilde{c}_2 \diamond \mathsf{RelK} \in \mathsf{RLWE}_{\mathbf{s}}^{n,Q_l}(\hat{m} \cdot \hat{m}')$.
- **Rotation.** On the ciphertext $(c_0, c_1) \in \mathsf{RLWE}_{\mathbf{s}}^{n,Q_l}(\mathsf{Ecd}\,(\mathbf{m}))$ and a rotation key $\mathsf{RotK}_k \in \widetilde{\mathsf{RLWE}}_{\mathbf{s}}^{n,Q_l}(\phi^k(\mathbf{s}); \mathbf{g})$, for any $l \in \langle L \rangle$, compute $(\phi^k(c_0), 0) + c_1 \diamond \mathsf{RotK}_k \in \mathsf{RLWE}_{\mathbf{s}}^{n,Q_l}(\mathsf{Ecd}\,(\mathbf{m} \lll k))$.

The automorphism function $\phi^k : R_n \to R_n$ is defined by $X \mapsto X^{5^k}$.

- **Operator ($\diamond$).** On any ring element $\hat{r} \in R_{n,Q_l}$, and the ciphertext $\mathsf{ct} = (c_0, \cdots, c_{d-1}) \in \widetilde{\mathsf{RLWE}}_{\mathbf{s}}^{n,Q_l}(\hat{m}; \mathbf{g})$, to perform the operation $\hat{r} \diamond \mathsf{ct}$, first decompose the ring element $\hat{r} = \sum_{j=0}^{d-1} \mathbf{g}[j]\hat{r}_j$. Then compute $\sum_{i=0}^{d-1} \hat{r}_j \cdot c_j \in \mathsf{RLWE}_{\mathbf{s}}^{n,Q_l}(\hat{r} \cdot \hat{m})$.
- **Rescale.** On the ciphertext $(c_0, c_1) \in \mathsf{RLWE}_{\mathbf{s}}^{n,Q_l}(\hat{m})$, compute $c'_b = q_l^{-1} \cdot (c_b - c_b \bmod q_l) \bmod Q_{l-1}$ for $b \in \{0, 1\}$ and output $(c'_0, c'_1) \in \mathsf{RLWE}_{\mathbf{s}}^{n,Q_{l-1}}(\lceil \hat{m}/q_l \rfloor)$.

In addition to the CKKS operations, we need some operations from the GSW encryption. The GSW encryption $\mathsf{RGSW}(\cdot; \mathbf{g})$ is defined as a tuple of $\widetilde{\mathsf{RLWE}}$ ciphertexts

$$\mathsf{RGSW}_{\mathbf{s}}^{n,q}(\hat{m}; \mathbf{g}) = \left(\widetilde{\mathsf{RLWE}}_{\mathbf{s}}^{\overline{n},q}(\hat{m}; \mathbf{g}), \widetilde{\mathsf{RLWE}}_{\mathbf{s}}^{n,q}(\hat{s} \cdot \hat{m}; \mathbf{g}) \right).$$

The $\odot$ operation multiples a RLWE ciphertext with a GSW ciphertext. Specifically, given $(\hat{b}, \hat{a}) \in \mathsf{RLWE}_{\mathbf{s}}^{n,q}(\hat{m}_0)$ and $(\hat{\beta}, \hat{\alpha}) \in \mathsf{RGSW}_{\mathbf{s}}^{n,q}(\hat{m}_1)$, one computes $(\hat{b}, \hat{a}) \odot (\hat{\beta}, \hat{\alpha})$ as $\hat{b} \diamond \hat{\beta} + \hat{a} \diamond \hat{\alpha} \in \mathsf{RLWE}_{\mathbf{s}}^{n,q}(\hat{m}_0 \cdot \hat{m}_1)$.

B. Proofs

The proofs in this section show the correctness of proposed building blocks.

We first prove the correctness of Theorem 1.

Proof: (**Theorem 1**) Our proof consists of two parts. First, we will prove the following invariant:

$$\mathsf{AC}_j \in \mathsf{RLWE}_{\mathbf{s}}(\hat{f} \cdot X^{\tilde{b} + \sum_{l \in \langle j \rangle} \tilde{\mathbf{a}}[l]\underline{\mathbf{s}}[l] \bmod n})$$

for each $j \in \langle n \rangle$ and any ternary key $\underline{\mathbf{s}} \in \{0, \pm 1\}^n$. Then we prove that the 0-th coefficient of the decryption of AC_n is (approximately) the evaluation of the look up table at point m, i.e., $\Delta T(m)$.

We prove the first part using mathematical induction. In the beginning, by definition, AC_0 is a trivial RLWE ciphertext of $\hat{f} \cdot X^{\tilde{b}}$, which follows the invariant. Let us assume AC_j follows the invariant for any $0 < j < \underline{n}$. Since the key $\underline{\mathbf{s}}$ is ternary, we now prove for each case, i.e., $\underline{k}_j \in \{0, 1, -1\}$, AC_{j+1} also follows the invariant.

1) Suppose $\underline{\mathbf{s}}[j] = 0$ then $\tilde{\mathbf{a}}[j]\underline{\mathbf{s}}[j] = 0$. We note that $\mathsf{EK}_{j,0}$ and $\mathsf{EK}_{j,1}$ are both GSW encryption of 1 in this case.

$$\begin{aligned} \mathsf{AC}_{j+1} &= ((X^{-\tilde{\mathbf{a}}[j]} - 1) \cdot \mathsf{t}_j) \odot \mathsf{RGSW}_{\underline{\mathbf{s}}}(1) + \mathsf{t}_j \\ &= X^{-\tilde{\mathbf{a}}[j]} \cdot \mathsf{t}_j = X^{-\tilde{\mathbf{a}}[j]} \cdot X^{\tilde{\mathbf{a}}[j]} \cdot \mathsf{AC}_j = \mathsf{AC}_j. \end{aligned}$$

2) Suppose $\underline{\mathbf{s}}[j] = 1$ then $\tilde{\mathbf{a}}[j]\underline{\mathbf{s}}[j] = \tilde{\mathbf{a}}[j]$. We note that $\mathsf{EK}_{j,0}$ is a GSW encryption of 1 and $\mathsf{EK}_{j,1}$ is a GSW encryption of 0 in this case.

$$\begin{aligned} \mathsf{AC}_{j+1} &= ((X^{-\tilde{\mathbf{a}}[j]} - 1) \cdot \mathsf{t}_j) \odot \mathsf{RGSW}_{\underline{\mathbf{s}}}(0) + \mathsf{t}_j \\ &= \mathsf{t}_j = X^{\tilde{\mathbf{a}}[j]} \cdot \mathsf{AC}_j. \end{aligned}$$

3) Suppose $\underline{\mathbf{s}}[j] = -1$ then $\tilde{\mathbf{a}}[j]\underline{\mathbf{s}}[j] = -\tilde{\mathbf{a}}[j]$. We note that $\mathsf{EK}_{j,0}$ is GSW encryption of 0 and $\mathsf{EK}_{j,1}$ is a GSW encryption of 1 in this case.

$$\begin{aligned} \mathsf{AC}_{j+1} &= ((X^{-\tilde{\mathbf{a}}[j]} - 1) \cdot \mathsf{t}_j) \odot \mathsf{RGSW}_{\underline{\mathbf{s}}}(1) + \mathsf{t}_j \\ &= X^{-\tilde{\mathbf{a}}[j]} \cdot \mathsf{t}_j = X^{-\tilde{\mathbf{a}}[j]} \cdot \mathsf{AC}_j. \end{aligned}$$

So it is proved that, for any $j \in \langle \underline{n} \rangle$, we have $\mathsf{AC}_j \in \mathsf{RLWE}_{\mathbf{s}}(\hat{f} \cdot X^{\tilde{b} + \sum_{l \in \langle j \rangle} \tilde{\mathbf{a}}[j]\underline{\mathbf{s}}[j] \bmod n})$. In other words, after the iterative updates, $\mathsf{AC}_{\underline{n}} \in \mathsf{RLWE}_{\mathbf{s}}(\hat{f} \cdot X^{\tilde{b} + \tilde{\mathbf{a}}^\top \underline{\mathbf{s}} \bmod n})$.

To prove the second part, we argue that

$$\begin{aligned} & b + \mathbf{a}^\top \underline{\mathbf{s}} = \lceil \Delta m \rfloor + \epsilon q_0 \text{ i.e., decryption function} \\ \Rightarrow\ & \tilde{b} + \tilde{\mathbf{a}}^\top \underline{\mathbf{s}} = \delta + 2\epsilon n = \lceil \frac{2n}{q_0}(\lceil \Delta m \rfloor) \rfloor + 2\epsilon n \end{aligned}$$

where $\epsilon \in \mathbb{N}$. In other words, $X^{\tilde{b} + \tilde{\mathbf{a}}^\top \underline{\mathbf{s}} \bmod n} = X^{\delta \bmod n}$. Also it is required that $|\Delta m| < q/4$. Thus we have $\delta \in \mathbb{Z} \cap [-n/2, n/2)$, which means that

$$X^{\delta \bmod n} = \begin{cases} -X^{n-|\delta|} \text{ for } \delta \leq 0 \\ X^{\delta} \text{ for } \delta > 0 \end{cases}.$$

Then the 0-th coefficient of $\hat{f} \cdot X^{\tilde{b} + \tilde{\mathbf{a}}^\top \underline{\mathbf{s}} \bmod n}$ is

$$\begin{cases} f_{|\delta|} X^{|\delta|} \cdot -X^{n-|\delta|} = \lceil \Delta T(-\eta_{|\delta|}) \rfloor \text{ for } \delta \leq 0 \\ -f_{n-\delta} X^{n-\delta} \cdot X^{\delta} = \lceil \Delta T(\eta_\delta) \rfloor \text{ for } \delta > 0 \end{cases}.$$

By definition

$$\eta_\delta = \frac{\lceil \frac{2n}{q_0} \lceil \Delta m \rfloor \rfloor q_0}{2n\Delta} \approx m$$

(resp. $\eta_{|\delta|} \approx |m|$), we have $T(\eta_\delta) \approx T(m)$ (resp. $T(-\eta_{|\delta|}) \approx T(m)$ for $m \leq 0$). In other words, the 0-th coefficient of the decryption of AC_n is approximately the evaluation of $T(x)$ at point m. ■

Then we show the correctness of Theorem 2.

Proof: (**Theorem 2**) After Step 2, $\tilde{\mathsf{ct}}$ encrypts some vectors $\mathbf{u} \in \mathbb{R}^{\underline{n}}$ which consists of products of $\{\tilde{\mathbf{m}}_j\}$ and $\mathbf{z}$. Indeed, for any $c \in \langle \ddot{n}/\tilde{n} \rangle$ and $r \in \langle \tilde{n} \rangle$, we have

$$\begin{aligned} \mathbf{u}[c\tilde{n} + r] &= \sum_{j \in \langle \tilde{n} \rangle} \tilde{\mathbf{m}}_j[c\tilde{n} + r]\mathbf{z}[j + c\tilde{n} + r] \\ &= \sum_{j \in \langle \tilde{n} \rangle} \mathbf{M}[c\tilde{n} + r, j + c\tilde{n} + r]\mathbf{z}[j + c\tilde{n} + r]. \end{aligned}$$

We omit the modulo in the subscripts to lighten the notation.

If $\underline{n} = \tilde{n}$ then $\mathbf{u}[c\tilde{n} + r] = \sum_j \mathbf{M}[c\underline{n} + r, j + r]\mathbf{z}[j + r]$, that is the $c\underline{n}+r$-th element of the product $\mathbf{Mz}$. Since $c\underline{n}+r$ loops over $\langle \ell \rangle$, $\mathbf{u} = \mathbf{Mz}$ in this case.

To simplify the second part of our proof (i.e., $\ell = \tilde{n}$), we first define $\mathbf{M}_c \in \mathbb{R}^{\ell \times \ell}$ as a column-block matrix of $\mathbf{M}$ such that $\mathbf{M}_c[i, j] = \mathbf{M}[i, c\ell + j]$ for $i, j \in \langle \ell \rangle$, and write $\mathbf{z}_c = \mathbf{z}[c\ell : c\ell + \ell]$. By this definition, we have $\mathbf{Mz} = \sum_c \mathbf{M}_c \mathbf{z}_c$.

Input: A LWE ciphertext $(b, \mathbf{a}) \in \mathsf{LWE}^{\overline{n},q}_{\overline{\mathbf{s}}}(m)$. The LWE switching key is a set of $\widetilde{\mathsf{RLWE}}$ ciphertexts, that is

$$\mathsf{SwK}_j \in \widetilde{\mathsf{RLWE}}^{\underline{n},q}_{\underline{\mathbf{s}}} \left(\sum_{l \in \langle \underline{n} \rangle} \overline{\mathbf{s}}[j\underline{n} + l] X^l ; \mathbf{g} \right)$$

for $j \in \langle \overline{n}/\underline{n} \rangle$.

Output: A LWE ciphertext $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}^{\underline{n},q}_{\underline{\mathbf{s}}}(\cdot)$.

```
1: Define a set of polynomials $\{\hat{a}_j\}_j$ where $\hat{a}_j = \mathbf{a}[j\underline{n}] - \sum_{l=1}^{\underline{n}-1} \mathbf{a}[j\underline{n} + l] X^{\underline{n}-l}$ for $j \in \langle \overline{n}/\underline{n} \rangle$.
2: Compute $\tilde{\mathsf{ct}} = \sum_{j \in \langle \overline{n}/\underline{n} \rangle} \hat{a}_j \diamond \mathsf{SwK}_j \in R^2_{\underline{n},q}$.
3: Output $(b, \mathbf{0}) + \mathsf{Extract}^0(\tilde{\mathsf{ct}})$ as $\mathsf{ct}_{\mathsf{out}}$.
```

Figure 9: Key-Switch PEGASUS.KS (modified from [11])

Next, we view $\mathbf{u}$ as a column-major matrix $\mathbf{U} \in \mathbb{R}^{(n/\ell) \times \ell}$ such that $\mathbf{U}[r, c] = \mathbf{u}[c\ell + r]$. Since $\ell = \tilde{n}$, we have

$$\mathbf{U}[r, c] = \sum_{j \in \langle \ell \rangle} \mathbf{M}[r, j + c\ell + r] \mathbf{z}[j + c\ell + r],$$

which equals to $(\mathbf{M}_c \mathbf{z}_c)[r]$. In other words, the c-th column of $\mathbf{U}$ is the product vector $\mathbf{M}_c \mathbf{z}_c$. Then to obtain $\mathbf{Mz}$, we the need to sum the columns of $\mathbf{U}$, which is done in Step 7.

Finally, $\mathsf{Rescale}(\cdot, \Delta_r) + \mathsf{Ecd}(\mathbf{t}, \Delta'_r)$ gives the result, i.e., $\mathsf{RLWE}_{\overline{\mathbf{s}}}(\mathsf{Ecd}(\mathbf{Mz} + \mathbf{t}, \Delta'_r))$. ■

For the sake of completeness, we present the our key-switching algorithm in Fig. 9. We now show the correctness of our key-switching in Algorithm 9.

Theorem 3: On the input ciphertext $\mathsf{LWE}^{\overline{n},q}_{\overline{\mathbf{s}}}(m)$, Fig. 9 outputs a LWE ciphertext $\mathsf{ct}_{\mathsf{out}} \in \mathsf{LWE}^{\underline{n},q}_{\underline{\mathbf{s}}}(m)$ when $\underline{n}$ is a 2-power value.

Proof: (**Theorem 3**) It suffices to show that the 0-th coefficient of the decryption of $\tilde{\mathsf{ct}}$ is the inner product of $\mathbf{a}$ and $\overline{\mathbf{s}}$.

$$\begin{aligned}
\tilde{\mathsf{ct}} &\in \sum_{j \in \langle \overline{n}/\underline{n} \rangle} (\mathbf{a}[j\underline{n}] - \sum_{l=1}^{\underline{n}-1} \mathbf{a}[j\underline{n} + l] X^{\underline{n}-l}) \diamond \widetilde{\mathsf{RLWE}}_{\underline{\mathbf{s}}}(\sum_{l \in \langle \underline{n} \rangle} \overline{\mathbf{s}}[j\underline{n} + l] X^l) \\
&\in \sum_{j \in \langle \overline{n}/\underline{n} \rangle} \mathsf{RLWE}_{\underline{\mathbf{s}}} \left(\sum_{l \in \langle \underline{n} \rangle} \mathbf{a}[j\underline{n} + l] \overline{\mathbf{s}}[j\underline{n} + l] + \hat{r}_j \right) \because X^{\underline{n}} = -1 \\
&\in \mathsf{RLWE}_{\underline{\mathbf{s}}} \left(\sum_{j \in \langle \overline{n}/\underline{n} \rangle} \sum_{l \in \langle \underline{n} \rangle} \mathbf{a}[j\underline{n} + l] \overline{\mathbf{s}}[j\underline{n} + l] + \hat{r}_j \right) \\
&\in \mathsf{RLWE}_{\underline{\mathbf{s}}} \left(\mathbf{a}^\top \overline{\mathbf{s}} + \hat{r} \right),
\end{aligned}$$

where $\{\hat{r}_j\}_j$ and $\hat{r}$ are polynomials that do not contain the constant term. Then $\mathsf{Extract}^0(\tilde{\mathsf{ct}}) + (b, \mathbf{0})$ gives a valid LWE encryption of $\mathbf{a}^\top \overline{\mathbf{s}} + b \approx m$ under the target key $\underline{\mathbf{s}}$. ■

C. Error Analysis

Our analysis follows the widely used heuristic assumption that the coefficients of each polynomial behave like independent zero-mean random variables of the same variance [20], and central limit heuristic [24]. On the polynomials $\hat{a}, \hat{b} \in R_n$ whose variances of the coefficients are σ_a^2 and σ_b^2, respectively. Then the variance of the coefficients of the sum $\hat{a} + \hat{b}$ is $\sigma_a^2 + \sigma_b^2$, and that of the product $\hat{a} \cdot \hat{b}$ is $n\sigma_a^2\sigma_b^2$. Remind that, PEGASUS uses the digit decomposition gadget $\mathbf{g}_{\mathrm{digit}}$ for the key-switching function, and uses RNS gadget $\mathbf{g}_{\mathrm{rns}}$ in the LUT evaluation function.

1) Key-Switching: We can view the switching key $\mathsf{SwK}_{\overline{\mathbf{s}} \to \underline{\mathbf{s}}}$ as a $(\overline{n}/\underline{n}) \times d_{\mathsf{ks}}$ matrix over $\mathsf{RLWE}^{\underline{n},q_0}_{\underline{\mathbf{s}}}(\cdot)$. The (j, k) entry of this matrix decrypts to $B^k_{\mathsf{ks}} \hat{s}_j + \hat{e}_{j,k}$ where the variance of the coefficients of the poly $\hat{e}_{j,k}$ is σ^2_{ks}. The multiplications ($\diamond$) in Step 2 of Algorithm 9 will introduce the extra error in the form of $\sum_{j,k} \hat{a}_{j,k} \cdot \hat{e}_{j,k}$ where the coefficients of $\hat{a}_{j,k}$ is bounded by B_{ks} due to the digit decomposition gadget. By viewing the coefficients of $\hat{a}_{j,k}$ as a uniform random over $[0, B_{\mathsf{ks}})$, its variance is $B^2_{\mathsf{ks}}/12$. As a result, the variance of the extra error is $\frac{1}{12} \overline{n} d_{\mathsf{ks}} B^2_{\mathsf{ks}} \sigma^2_{\mathsf{ks}}$.

2) Look-up Table Evaluation: Suppose the input LWE ciphertext of the LUT evaluation decrypts to $b + \mathbf{a}^\top \underline{\mathbf{s}} \bmod q_0 = \Delta m + e_0$ for some error term e_0 whose variance is σ_0^2. Our LUT evaluation outputs a LWE ciphertext that decrypts to $\Delta T(m + e_1) + e_2$. We now analyze the variances of these errors. By scaling down from $[0, q_0)$ to $[0, 2n)$, in Step 2 of Algorithm 2, we obtain $\tilde{b} = \lceil (2nb)/q_0 \rfloor$ and $\tilde{\mathbf{a}}[i] = \lceil (2n\mathbf{a}[i])/q_0 \rfloor$ for $i \in \langle n \rangle$. Let define $\mathsf{idx} = \tilde{b} + \tilde{\mathbf{a}}^\top \underline{\mathbf{s}} \bmod 2n$. According to the central limit heuristic [24], $\mathsf{idx} = \lceil (2n\Delta m)/q_0 \rfloor + e$ with the error variance of $((2n\sigma_0)/q_0)^2 + (\|\underline{\mathbf{s}}\|^2 + 1)/12$. The value idx serves as a look-up index to select the coefficient of $\hat{f}$, i.e., $\Delta T(q_0/(2\Delta n) \cdot \mathsf{idx})$ which can be viewed as $\Delta T(m + e' + e_{\mathrm{rd}})$ with the rounding error e_{rd} and an error e' of a variance $(\sigma_0/\Delta)^2 + q_0^2(\|\underline{\mathbf{s}}\|^2 + 1)/(48n^2\Delta^2)$. By viewing the message m as a uniform random over $[-q_0/4\Delta, q_0/4\Delta)$, the variance of e_{rd} can be given as $q_0^2/(48n^2\Delta^2)$. As a result, the variance of $e_1 = e' + e_{\mathrm{rd}}$ can be given as $(\sigma_0/\Delta)^2 + q_0^2(\|\underline{\mathbf{s}}\|^2 + 2)/(48n^2\Delta^2)$. Moreover, the error e_2 is accumulated from the $2\underline{n}$ multiplications ($\odot$). According to the analysis of special prime technique from [10], the variance of e_2 can be given as $4\underline{n}q_0\sigma^2_{\mathrm{lut}}/q'$.

Suppose $T(\cdot)$ is a $\mathcal{L}$-Lipschitz function. Then the difference between the ground truth $T(m)$ and the approximated result $T(m + e_1) + e_2/\Delta$ can be bounded as

$$6 \cdot \left(\mathcal{L} \sqrt{(\sigma_0/\Delta)^2 + q_0^2(\|\underline{\mathbf{s}}\|^2 + 2)/(48n^2\Delta^2)} + \sqrt{4\underline{n}q_0\sigma^2_{\mathrm{lut}}/q'}/\Delta \right) \tag{3}$$

with high probability.