Complexity Analysis of Encoding in CKKS-Fully Homomorphic Encryption Algorithm

Infall Syafalni*†, Daniel M. Reynaldi*, Rinaldi Munir*, Trio Adiono*†, Nana Sutisna*†, and Rahmat Mulyawan*†
*School of Electrical Engineering and Informatics, Bandung Institute of Technology, Indonesia
†University Center of Excellence on Microelectronics, Bandung Institute of Technology, Indonesia

2022 International Symposium on Electronics and Smart Devices (ISESD) | 979-8-3503-9660-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/ISESD56103.2022.9980695

***Abstract*—In this paper, we analyze the complexity of encoding in CKKS algorithm. The encoding process in CKKS requires some steps such as Inversion of PI π^{-1}, Scaling, Discretization, and Inverse of Sigma σ^{-1}, and Rounding. However, the decoding requires Sigma function to convert the encoded message back to the plaintext. In this paper, we evaluate each function and determine the complexities. Experimental results show that the complexity of the encoding is $O(n^3)$ and the complexity of the decoding is $O(n^2)$. The work is useful for optimization and speeding up the CKKS computation on custom computing platforms such as GPUs and FPGAs.**

I. Introduction

Recently, the information technology has been rapidly developed by the advancement of high speed computing and communication. However, vast amounts of data sent to the cloud make opportunities for hackers to steal the data. For example, in banking alone, billion transactions occur in a day requiring robust global security to ensure the transactions secured, fast and accurate. Moreover, with fast development of social media, privacy concerns related to personal data such as location, health, financial are urgently necessary.

In the other hand, a new technology called fully homomorphic encryption (FHE) allows us to compute the encrypted data without decrypting it. FHE keeps the integrity of the data even in the cloud. FHE was proposed firstly in 2009 [1]. After that, some schemes such as Brakerski/Fan-Vercauteren (BFV) [2], Brakerski-Gentry-Vaikuntanathan (BGV) [3], (Torus Fully Homomorphic Encryption) TFHE [4] as well as Cheon-Kim-Kim-Song (CKKS) [5] are proposed. However, the computation of the FHE is heavy, thus it requires accelerator to be implemented *e.g.,* in [6].

In this paper, we analyze the complexity of encoding in CKKS algorithm. The encoding process in CKKS requires some steps such as Inversion of PI π^{-1}, Scaling, Discretization, and Inverse of Sigma σ^{-1}, and Rounding. However, the decoding requires Sigma function to convert the encoded message back to the plaintext. In this paper, we evaluate each function and determine the complexities. Experimental results show that the complexity of the encoding is $O(n^3)$ and the complexity of the decoding is $O(n^2)$. The work is useful for optimization and speeding up the CKKS computation on custom computing platforms such as GPUs and FPGAs.

This paper organization consists of 5 sections: Section I explains the background of the work. Section II explains the definitions and basic properties for math in FHE and CKKS. Section III shows the proposed encoding complexity analysis in CKKS. Section IV shows the experimental results. Finally, Section V summarizes the work.

II. Basic Definitions and Properties

A. Mathematical Representation of FHE

Definition 2.1. *Root of unity [7] is a complex number z iff z^n is $1+0j$. Note that the imaginary value is 0. In euler form, z is $e^{\frac{2\pi ik}{n}}$ or $\cos\frac{2\pi k}{n}+i\sin\frac{2\pi k}{n}$.*

Definition 2.2. *Cyclotomic polynomial [8] is a polynomial that is formed from the primitive roots of unity so that the n-th cyclotomic polynomial is able to be divided by $x-1$ and is unable to be divided by x^k-1 where $k<n$.*

$$\Psi_n(x)=\prod_{1<k<n}^{n} x-e^{\frac{2\pi ik}{n}}$$

Definition 2.3. *Polynomial ring is a polynomial that is formed from a sent of values. The variables from the polynomial is isomorphism and has bijective set of values. Suppose that R is a ring and x is a variable in the polynomial. Thus, we have a polynomial ring:*

$$R(x)=a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$$

Lemma 2.1. *Cyclotomic polynomial ring is derived from Definitions 2.2 and 2.3. It is the subset of cyclotomic polynomial that has isomorphism in the roots. Let $Z(x)$ be a polynomial ring and x^n+1 is the cylotomic polynomial. Thus, we form a cyclotomic polynomial ring by the polynomial ring in the finite field such that:*

$$R=\frac{Z(x)}{x^n+1}$$

Definition 2.4. *Lattice a set of points in n dimension of Euclidean. Lattice-based cryptography is a conversion of a message to a set of lattice values with uniform basis.*

Lemma 2.2. *Learning with errors (LWE) is a computational problem for producing a ciphertext using a secret key $s\in Z^n=q$. The secret key is then used to generate the public key:*

$$a,b=(s,a)+e \mod q,$$

where $(a, b) \in Z_q^n \times Z_q$, *and* e *is the error with uniform random value. The LWE is derived from closest vector problem in lattice.*

Lemma 2.3. *Ring learning with errors (RLWE) is LWE with cyclotomic polynomial ring in Lemma 2.1:*

$$R_q = \frac{Z_q(x)}{x^n + 1},$$

such that the values of a *and* b *in the ring of* R_q*. Thus, RLWE is* $a, b = (s, a) + e \mod q$ *where* $s \in R_q$ *and* $(a, b) \in R_q \times R_q$*, and* e *is error with Gaussian uniform.*

B. CKKS-FHE

This section explains the basic properties of CKKS algorithm [5].

Definition 2.5. *By Lemma 2.2, the secret key generation of CKKS is* $s_k = s$*. Moreover, the public key is represented by:*

$$p_k = (a, b),$$
$$b = -a \cdot s + e,$$

where $(a, b) \in R_q \times R_q$*, and* e *is error with Gaussian uniform. Note that* $s_k \in R_q$ *is a cyclotomic polynomial ring and* $p_k \in R_q^2$ *is a cyclotomic polynomial ring with 2 dimensions.*

Definition 2.6. *Encoding in CKKS (Ecd) is a map of a complex number of the message* z *to* m *such that:*

$$Ecd : z \rightarrow m(X),$$

where X *is the polynomial variable,* $m(X) \in Z[X]/(X^N + 1)$*, and* $z \in C^{N/2}$*. Note that* C *is complex domain with dimension* $N/2$*.*

Lemma 2.4. *Inversion of* π *(*π^{-1}*) is a conversion of a complex number* z *into a matrix* H*:*

$$\pi^{-1} : z \rightarrow H,$$

where $H = z \in C^N$*. Note that the vector elements of* H *is the combination of* z_j *and the conjugate of* z_j^**.*

Definition 2.7. *Scale* Δ *is a constant, usually in the form of* 2^m*, that scales the* z*. In this case, the matrix* H *is also scaled by* Δ*, thus we have* $\Delta \cdot H$*.*

Lemma 2.5. *Sigma inverse (*σ^{-1}*) is required in Encoding. The inputs of sigma inverse is* b_i *that is calculated by the following:*

$$b_i = \sigma(X^{i-1}),$$
$$= ((\zeta_M^i)^{i-1}, \ldots, (\zeta_M^{N-1})^{i-1}),$$

where ζ *is the results of sigma inverse. Furthermore, discretization is represented by:*

$$z_i = \frac{(z \cdot b_i)}{||b_i||^2},$$

where $z \in C^N$*. Finally,* z_i *is rounded to the closest integer such that* $\lfloor z_i \rceil$*.*

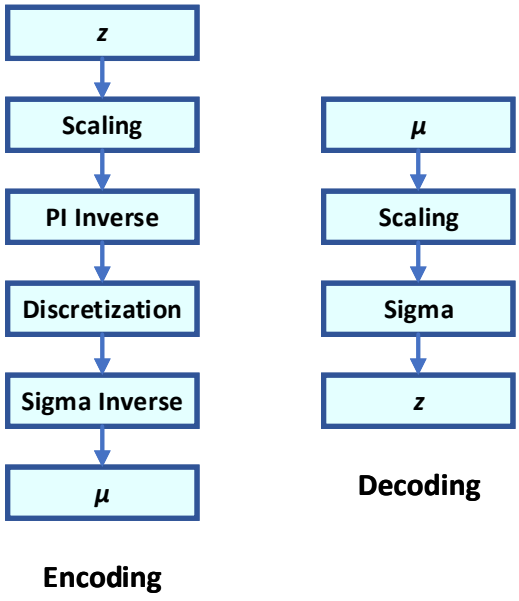

Fig. 1: Overview of encoding and decoding processes in CKKS

Lemma 2.6. *Encryption of CKKS uses encoding* $\mu = Ecd(\Delta, z)$ *and* $p_k \in R_q^2$ *as a public key and* $s \in R_q$ *as a secret key. The ciphertext* c *is calculated by the following:*

$$c = (c_0, c_1), c_0 = \mu - a \cdot s + e, \text{and } c_1 = a,$$

where $c \in R_q^2$ *is in the form of cyclotomic polynomial ring.*

Lemma 2.7. *Decryption of CKKS is calculated by the following:*

$$\mu' = c_0 + c_1 \cdot s,$$
$$= \mu - a \cdot s + e + a \cdot s,$$
$$= \mu + e,$$
$$\approx \mu,$$

where μ *is the decrypted message,* c_0 *and* c_1 *are the ciphertexts, and* s *is the secret key. Note that the value of* e *is less than* q*.*

III. Complexity of Encoding in CKKS

In this section, we observe the complexity of Encoding and Decoding processes in CKKS. The Encoding consists of inversion of π, scaling, discretization, and inverse of σ. Meanwhile, the Decoding consists of unscaling and multiplication of σ. Fig. 1 shows the overview of Encoding and Decoding processes in CKKS that will be analyzed in this work.

A. Encoding Analysis

Vandermode matrix is used to find the root of unity.

Lemma 3.1. *Suppose that we have a root:*

$$x_k = e^{2\pi i k/M},$$

where M is the half of polynomial degree N, i is the imaginary unit, and k is the index with $1 < k < N/2$ and k is odd. Note that e is the base of logarithm number.

The generation of Vandermode matrix for root of unity is represented as follows:

$$V = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{M-1} \\ 1 & x_3 & x_3^2 & \cdots & x_3^{M-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{M-1} & x_{M-1}^2 & \cdots & x_{M-1}^{M-1} \end{bmatrix},$$

where x_k is the root of unity with k is odd and $1 < k < N$.

Lemma 3.2. *The complexity of Vandermode matrix for root of unity is $O(n^2)$.*

Proof. From Alg. 1, to generate the root of unity $x_k = e^{2\pi ik/M}$, it requires $M \times M$ loops. Thus, we have the lemma. □

Algorithm 1: Vandermonde Matrix

Input: Polynomial degree N and $x_0 = e^{2\pi i/M}$.
Output: Vandermode matrix V.
begin
  $V \leftarrow [v_{0,0}, v_{0,1}, \ldots, v_{M-1,M-1}]$
  for $j = 0$ *to* $M - 1$ **do**
    for $k = 0$ *to* $M - 1$ **do**
      $v[j,k] \leftarrow x[0]^{(2j+1)^k}$

Next, in Encoding as explained in Lemma 2.4, the inversion of π is required. By Alg. 2, we convert each element of z into it's conjugate.

Algorithm 2: π inverse

Input: Complex number z.
Output: Complex number with conjugate z^*.
begin
  for $j = 0$ *to* $M - 1$ **do**
    $z^*[j] \leftarrow (z[j])^*$

Lemma 3.3. *The complexity of π inverse in Alg. 2 is $O(n)$.*

Proof. It is clear from Alg. 2 that runs in a loop. □

Algorithm 3: Discretization

Input: Scaled polynomial z_s with scale Δ.
Vandermonde matrix root of unity V
Output: Sigma inverse of ζ, $\sigma^{-1}(\zeta)$.
begin
  for $j = 0$ *to* $M - 1$ **do**
    $z_s[j] \leftarrow \frac{z_s[j] \cdot b[j]}{||b[j]^2||}$
  $z_s \leftarrow \lfloor z_s \rceil$ $z_r \leftarrow V^T \cdot z_s$

The next process required for the encoding is discretization. The mathematical representation of discretization is in Lemma 2.5. The process includes coordinate computing and a matrix multiplication between the coordinates and the transposed Vandermode matrix.

Lemma 3.4. *The dicretization process in Alg. 3 has $O(n^2 + n) \approx O(n^2)$ complexity.*

Proof. From Alg. 3, we have a computation for coordinates for a loop. Furthermore, the output of the coordinates is multiplied by transposed Vandermode matrix root of unity with $O(n^2)$. □

Algorithm 4: Sigma inverse σ^{-1}

Input: Polynomial ζ. Vandermonde matrix root of unity V
Output: Sigma inverse of ζ, $\sigma^{-1}(\zeta)$.
begin
  $P \leftarrow$ *linear.algebra.solver*(ζ, V)
  $P_r \leftarrow \lfloor P \rceil$

The final process in encoding is the inversion of σ or σ^{-1}. The σ^{-1} is the most complex function in the encoding since it requires linear algebra solver between $M \times 1$ matrix (ζ) and $M \times M$ matrix (V).

Lemma 3.5. *The inversion of σ or σ^{-1} requires at most $O(n^3)$ time complexity.*

Proof. By Lemma 2.5, sigma inverse is required to find b_j. To make it clearer, suppose that we have the following matrices operation:

$$[V][B] = [Z],$$

where $[V]$ is the Vandermonde matrix and $[Z]$ is the polynomial input ζ. To get the value of $[B]$, we use the sigma inverse such that:

$$[B] = [V]^{-1}[Z].$$

Note that the $[V]^{-1}$ is the σ^{-1} function. This inversion of the matrix in linear algebra solver requires at most $O(n^3)$. Thus, we have the lemma. □

Algorithm 5: Encode

Input: Input polynomial z and scale Δ.
Output: Encoded z (μ).
begin
  $z_{pi} \leftarrow \pi^{-1}(z)$
  $z_s \leftarrow z_{pi} \cdot \Delta$
  $z_r \leftarrow$ *discretization*(z_s)
  $\mu \leftarrow \sigma^{-1}(z_r)$

Theorem 3.1. *The encoding consists of π^{-1}, scaling, discretization and σ^{-1}. Thus, the overall complexity is*

$$O(n + n^2 + n^3) \approx O(n^3).$$

Proof. As Alg. 5, the complexities of π^{-1} is $O(n)$, scaling is $O(n)$, discretization is $O(n^2)$, and σ^{-1} is $O(n^3)$. Thus, we have the theorem. □

B. Decoding Analysis

The last function is sigma σ that is used for decoding process. Note that the σ uses $[V]$ to convert back the message $[Z]$ that is explained in the proof of Lemma 3.5.

Algorithm 6: Sigma σ

Input: Polynomial C and it's degree M. Vandermonde matrix root of unity V.
Output: Sigma of C, $S = \sigma(C)$.
begin
  $C \leftarrow [c_0, \ldots, c_{M-1}]$
  $S \leftarrow [s_{0,0}, \ldots, s_{M-1,M-1}]$
  for $j = 0$ *to* $M - 1$ **do**
    for $k = 0$ *to* $M - 1$ **do**
      $s[j,k] \leftarrow c[j] \cdot v[j,k]$

Lemma 3.6. *As Alg. 5, the complexity of σ is $O(n^2)$.*

Proof. It is clear that σ process uses dot multiplication of a matrix. Thus, it costs $O(n^2)$. □

Algorithm 7: Decode

Input: Input encoded polynomial μ and scale Δ.
Output: Decoded μ (z).
begin
  $z_{pi} \leftarrow \mu \cdot 1/\Delta$
  $z \leftarrow \sigma(z_{pi})$

Theorem 3.2. *The decoding consists of division of scaling and multiplication with sigma σ. Thus, the overall complexity is $O(n + n^2) \approx O(n^2)$.*

Proof. It is clear from Alg. 7. □

IV. Experimental Results

In the experiment, we implement all the encoding and the decoding functions in Python. Next, we compute the functions in Google Collaboration with Dual CPUs of AMD EPYC 7B12 with 2.249 GHz and 13 GB Memory.

Figs. 2 and 3 shows our implementations of encoding and decoding. Next, we run the functions for $N = 2^{D=2} = 4$ to $N = 2^{D=10} = 1024$. Figs. 4 and 5 (in Logarithmic) show the time execution performances. As shown, σ^{-1} is the most complex with $O(n^3)$ while σ is only $O(n^2)$.

```
def encode(self, z):
  z_pi = self.pi_inverse(z)
  scaled_z = self.scale * z_pi
  rounded_z = self.discretization(scaled_z)
  polynomial = self.sigma_inverse(rounded_z)
  return p
```

Fig. 2: Encoding implementation

```
def decode(self, p):
  unscaled_p = np.multiply(np.array(p.coef), (1/
  self.scale))
  z = self.sigma(unscaled_p)
  return z
```

Fig. 3: Decoding implementation

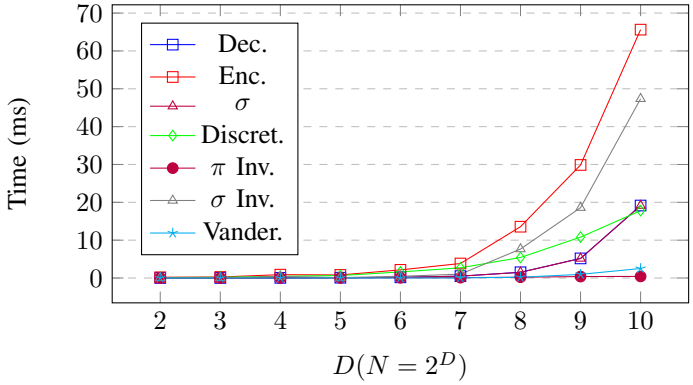

Fig. 4: Various functions execution time

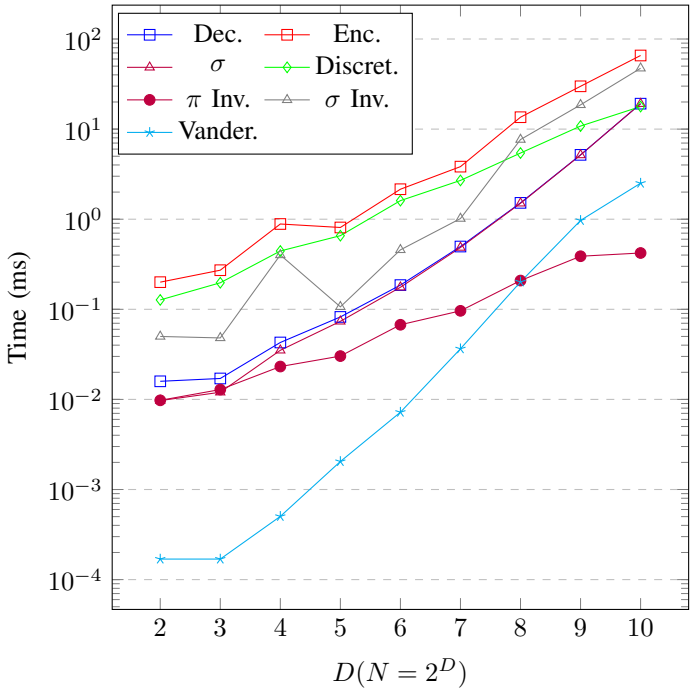

Fig. 5: Various functions execution time (in logarithmic)

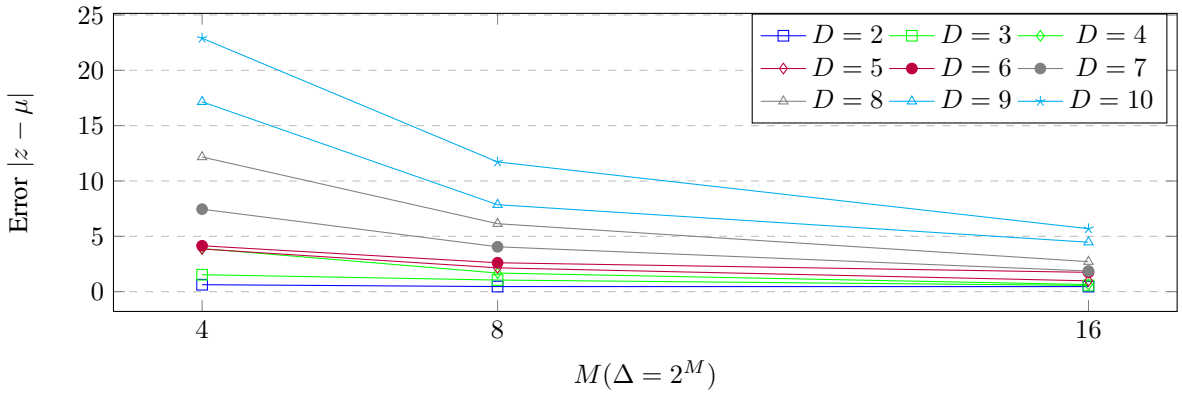

Fig. 6: Encoding-decoding error for $D = 2$ to $D = 10$

The sum of time complexities of π^{-1}, σ^{-1}, and discretization form the time complexity of Encoding (red line with square mark) in Figs. 4 and 5 (in Logarithmic). However, time complexity of $sigma$ represents most of the time complexity of Decoding.

Finally, we observe the error performance of encoding-decoding in CKKS as shown in Fig. 6 with various D. Note that $N = 2^D$, where N is the polynomial degree. We take the values of $M = [4, 8, 16]$ that indicate the number of bit of Δ. In this case, we have Δ is 16, 256 and 1024 for $M = [4, 8, 16]$, respectively. As shown in Fig. 6, the increment of degrees of $N = 2^D$ increases the errors. However, by adding some bits at Δ, the errors can be reduced.

V. Conclusions

In this paper, we analyzed the complexity of encoding in CKKS algorithm. The encoding process in CKKS requires some steps such as Inversion of PI π^{-1}, Scaling, Discretization, and Inverse of Sigma σ^{-1}, and Rounding. However, the decoding requires Sigma function to convert the encoded message back to the plaintext. In this paper, we evaluate each function and determine the complexities. Experimental results showed that the complexity of the encoding is $O(n^3)$ and the complexity of the decoding is $O(n^2)$. The work is useful for optimization and speeding up the CKKS computation on custom computing platforms such as GPUs and FPGAs.

The future works include the implementation of parallel programming of CKKS encoding-decoding together with transformations optimization for the polynomial operations in the CKKS. Furthermore, the optimization of bootstrapping in CKKS is also required to speed up the time execution. Thus, the CKKS algorithm can be practically implemented in the industries.

Acknowledgment

This work was supported by The Ministry of Education, Culture, Research, and Technology under National Competition Research grant (Program Penelitian Kompetitif Nasional).

References

[1] C. Gentry, "Fully homomorphic encryption using ideal lattices," *in Proceedings of the forty-first annual ACM symposium on Theory of computing*, pp. 169–178, 2009.

[2] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptology ePrint Archhieve*, vol. 2012/144, pp. 1-19, 2012.

[3] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014

[4] I. Chillotti, N. Gama, M. Georgieva, *et al.*, "TFHE: Fast fully homomorphic encryption over the torus," *J Cryptol*, vol. 33, pp. 34–91, 2020.

[5] J.H. Cheon, A. Kim, M. Kim, Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," *In: Takagi, T., Peyrin, T. (eds) Advances in Cryptology – ASIACRYPT 2017, Lecture Notes in Computer Science*, vol. 10624, 2017.

[6] I. Syafalni, G. Jonatan, N. Sutisna, R. Mulyawan and T. Adiono, "Efficient homomorphic encryption accelerator With integrated PRNG using low-cost FPGA," *in IEEE Access*, vol. 10, pp. 7753-7771, 2022.

[7] A. Lang. *Algebra*. Springer Science and Business Media. 2005.

[8] A. V. Saavedra. *Study and Applications of Homomorphic Encryption Algorithms to Privay Preserving SVM Inference for a Bak Fraud Detection Context*. Máster Inter-Universitario en Ciberseguridade: Universidade de Vigo. 2021.