Kelayakan Implementasi Enkripsi Homomorfik Total Skema CKKS Pada Komputasi Awan

Studi Kasus: Implementasi Enkripsi Homomorfik Pada Pengembangan Toko *Online*

Muchammad Ibnu Sidqi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Bandung, Indonesia
muchammadibnu@gmail.com

Rinaldi Munir
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Bandung, Indonesia
rinaldi@informatika.org

Infall Syafalni
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Bandung, Indonesia
infall@staff.stei.itb.ac.id

***Abstrak*—Berkembangnya teknologi semakin cepat membuat ancaman terhadap teknologi semakin berkembang salah satunya adalah kebocoran data. Untuk mencegah kebocoran data dapat menggunakan enkripsi homomorfik total yang memungkinkan adanya komputasi pada data terenkripsi. Terdapat empat generasi enkripsi homomorfik total, salah satunya adalah CKKS yang dapat melakukan komputasi hingga bilangan kompleks sehingga cocok digunakan untuk permasalahan toko *online*. Pada skema CKKS, terdapat beberapa proses, yaitu pembangkitan kunci, *encoding*, *decoding*, enkripsi, dekripsi, evaluasi, *rescaling*, *relinearization*, dan *bootstrapping*. Skema CKKS dibuat berdasarkan permasalahan RLWE sehingga polinomial yang digunakan adalah *cyclotomic polynomial ring* sehingga proses NTT dapat diimplementasikan untuk mempersingkat waktu komputasi. Solusi penggunaan CKKS menggunakan kakas *lattigo* karena kecepatan dalam eksekusi program. Terdapat solusi arsitektur API yang digunakan, yaitu REST yang digunakan karena mudah untuk diimplementasikan. Terdapat lima solusi modul yang diimplementasikan, yaitu activate untuk pembukaan saldo, checkBalance untuk mengecek saldo, topup untuk mengisi saldo, transact untuk menyimpan riwayat transaksi, dan getTransact untuk mendapatkan riwayat transaksi. Pengujian dilakukan dengan menggunakan dua komponen, yaitu waktu eksekusi dan ukuran variabel. Waktu eksekusi yang diperlukan untuk proses enkripsi homomorfik skema CKKS meningkat secara linier mengikuti parameter, dengan waktu proses maksimum pada proses bootstrapping dan waktu minimum pada proses pembangkitan *secret key*. Waktu eksekusi total pada enkripsi homomorfik yang menggunakan NTT lebih rendah dibanding waktu eksekusi total tanpa menggunakan NTT. Mempertimbangkan waktu *bootstrapping* yang mencapai rata-rata satu menit untuk seluruh parameter yang diuji, maka enkripsi homomorfik belum bisa diimplementasikan untuk kondisi nyata.**

Kata kunci—kebocoran data; CKKS; RLWE; NTT; bootstrapping

I. PENDAHULUAN

Berkembangnya teknologi yang semakin cepat dan canggih mempengaruhi berbagai sektor kehidupan, salah satunya ada di sektor perdagangan. Dengan adanya teknologi ini memungkinkan proses perdagangan, khususnya jual-beli suatu barang, dapat dilakukan secara daring. Hal ini mengubah paradigma jual-beli yang dulunya hanya dibatasi toko fisik saja menjadi sebuah toko online. Pembuatan toko online tidak terlepas dari adanya komputasi awan. Komputasi awan adalah ketersediaan sumber daya sistem komputer sesuai dengan permintaan terutama pada penyimpanan data maupun pada penggunaan komputasi tanpa pengelolaan oleh pengguna secara langsung [1].

Penggunaan komputasi awan tidak terlepas dari unsur keamanan. Badan Siber dan Sandi Negara (BSSN) mengumumkan bahwa serangan siber meningkat yang terjadi pada tahun 2020 sebanyak dua kali lipat dari tahun 2019 yang mana terdapat peningkatan sebesar 267,06 juta serangan menjadi 495,33 serangan [2]. Salah satu serangan siber yang banyak terjadi di Indonesia adalah kebocoran data. Beberapa contoh kebocoran data di Indonesia terjadi pada KPU saat 22 Mei tahun 2020 silam, BPJS Kesehatan pada Mei 2021 silam, dan eHAC (*electronic Health Alert Card*) pada 15 Juli 2021 [3]. Penyebab kebocoran data disebabkan oleh gagalnya implementasi aspek keamanan kerahasiaan pada data yang tersimpan dan digunakan sehingga data yang didapatkan dapat langsung terbaca oleh peretas [4, 5].

Enkripsi Homomorfik (HE) adalah enkripsi yang memungkinkan terjadinya operasi aritmatik pada data yang sudah terenkripsi. Terdapat dua jenis HE yakni secara parsial dan total [6, 7, 8]. Saat ini, terdapat empat generasi HE dengan generasi terbaru atau generasi keempat menggunakan skema CKKS [9]. Dengan adanya teknologi tersebut dapat mencegah adanya kasus kebocoran data di Indonesia. Selain menyajikan manfaat yang besar, teknologi HE memiliki kekurangan, yakni dalam kompleksitas komputasi, *noise* yang dihasilkan saat melakukan operasi aritmatik berulang, dan penggunaan penyimpanan yang besar dalam membangkitkan kunci publik serta saat melakukan operasi aritmatik [10, 11, 12]. Untuk mengatasi kekurangan enkripsi homomorfik, yaitu kompleksitas komputasi dapat dilakukan proses *Number Theoritic Transform* (NTT) yang dapat mereduksi kompleksitas komputasi dengan melakukan operasi perkalian secara parallel [13]. Selain itu,

dapat dilakukan proses *bootstrapping* untuk dapat menyegarkan cipherteks agar cipherteks dapat diproses kembali.

II. PENELITIAN TERKAIT

A. Enkripsi Homomorfik Terhadap Komputasi Awan

Pada skema komputasi awan yang dibuat oleh Tebaa et al. terdapat dua komponen utama yakni server klien sebagai server yang mengelola masukan pengguna dan membentuk pesan yang sudah terenkripsi untuk selanjutnya akan dikirim kepada penyedia layanan awan dan terdapat server awan yang menyediakan layanan awan homomorfik berupa operasi aritmatik dan pengelolaan data yang sudah terenkripsi pada basis data [14].

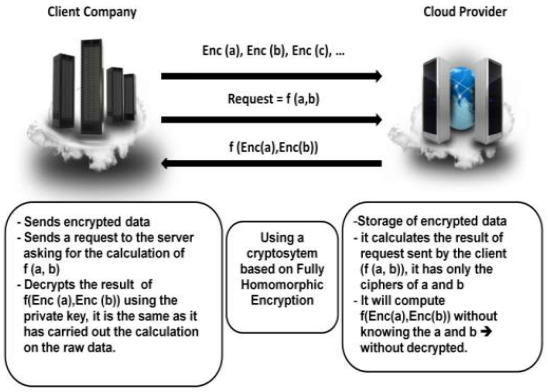

Fig. 1. Enkripsi Homomorfik Pada Komputasi Awan [14].

Untuk keseluruhan kalkulasi yang dilakukan pada seluruh data yang tersimpan pada layanan awan. Pemilihan skema enkripsi homomorfik total harus dilakukan agar keseluruhan operasi kalkulasi pada data dapat dilakukan pada data yang sudah terenkripsi tanpa harus dilakukan dekripsi telebih dahulu. Untuk kalkulasi pada data yang rahasia dapat dilakukan dengan menggunakan komputasi pada komputasi awan dengan menyimpan kunci rahasia untuk dilakukan dekripsi pada hasil komputasi [14].

III. DASAR TEORI

A. Kriptografi

Skema yang digunakan untuk enkripsi merupakan bidang studi yang dikenal sebagai kriptografi. Skema yang digunakan dapat disebut sebagai sistem kriptografi atau cipher [4]. Terdapat algoritma yang menggunakan lebih satu kunci atau biasa disebut sebagai enkripsi asimetrik atau kriptografi kunci publik adalah algoritma yang menggunakan setidaknya satu kunci untuk melakukan enkripsi dan setidaknya satu kunci untuk dekripsi yang mana jumlah kunci yang unik lebih dari satu kunci. Algoritma ini sering digunakan untuk autentikasi, mekanisme non-repudasi dan penjamin integritas pesan [4, 8].

1) Kriptografi Berbasis Latis

Latis merupakan kumpulan berbagai titik pada dimensi n ruang *euclidean.* Salah satu contoh latis yang mudah dipahami adalah latis dengan set anggota bilangan bulat.

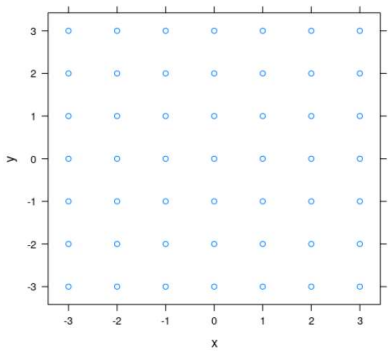

Fig. 2. Contoh Latis Set Bilangan Bulat [15]

Setiap set anggota latis memiliki independensi linier dengan basis sehingga latis merupakan kombinasi linier dari setiap set anggota latis tersebut. Dengan menggunakan latis, maka kriptografi dapat dilakukan dengan mengubah suatu pesan menjadi set anggota dari suatu latis dengan basis yang dibuat secara uniform sehingga probabilitas kemungkinan akan menjadi sulit untuk dipecahkan. Kriptografi berbasis latis dianggap sebagai suatu solusi baru untuk menghindari penyerangan dengan menggunakan komputasi kuantum [16].

a) Learning With Errors

Learning with Errors (LWE) adalah permasalahan komputasi yang mana diperlukan pemulihan suatu pesan rahasia $s \in \mathbb{Z}_q^n$ dari kunci publik $(a, b) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$ acak uniform dengan *error* e dengan nilai kecil acak uniform sehingga menghasilkan $(a, b = \langle s, a \rangle + e \bmod q)$. Bila ditelaah, persolaan LWE berasal dari permasalahan *Closest Vector Problem* dengan mencari suatu nilai terdekat dari aproksimasi nilai target yang sudah diketahui. Dengan memanfaatkan permasalahan LWE, terdapat banyak skema enkripsi yang memanfaatkan LWE karena kesulitan dalam memulihkan nilai s [15].

b) Ring Learning With Errors

Walaupun permasalahan LWE digunakan di berbagai aplikasi kriptografi karena asumsi keamanan yang kuat, tetapi terdapat masalah pada pembangkitan kunci serta kompleksitas operasi perkalian. Pembangkitan kunci LWE dapat mencapai orde n^2, dengan kompleksitas perkalian dapat mencapai $O(n^2)$. Untuk itu, terdapat permasalahan baru sehingga menghasilkan RLWE yang berbasis *polynomial ring* dengan memanfaatkan *cyclotomic polynomial ring* $R_q = \frac{Z_q(x)}{x^n+1}$ sehingga nilai a dan b berbentuk R_q. Dengan itu, maka permasalahan RLWE adalah permasalahan komputasi yang mana diperlukan pemulihan suatu pesan rahasia $s \in R_q$ dari kunci publik $(a, b) \in R_q \times R_q$ acak uniform dengan *error* e dengan nilai kecil acak uniform sehingga menghasilkan $(a, b = \langle s, a \rangle + e \bmod q)$. Dengan menggunakan bentuk *cyclotomic polynomial* maka ukuran kunci akan sebesar R_q dan operasi perkalian dapat

direduksi menjadi $O(n \log(n))$ dengan menggunakan metode *Number Theoritic Transform* [15].

B. Cyclotomic Polynomial

Cyclotomic polynomial adalah polinomial yang dibentuk oleh akar-akar dari *root of unity* yang mana pembentuk akar merupakan *root of unity* primitif sehingga *cyclotomic polynomial* ke-*n* dapat dibagi dengan $x^n - 1$ dan tidak dapat dibagi $x^k - 1$ dengan $k < n$.

$$\Phi_n(x) = \prod^{n}_{\substack{1<k<n \\ \gcd(k,n)=1}} (x - e^{\frac{2\pi ik}{n}}) \quad (1)$$

Cyclotomic polynomial memiliki akar-akar primitif sehingga seluruh *root of unity* yang menjadi akar *cyclotomic polynomial* merupakan relatif prima dengan nilai *n*. Pada skema ckks, polynomial yang digunakan adalah *cyclotomic polynomial* yang digunakan berdasarkan permasalahan *rlwe*. Untuk memudahkan komputasi, maka akan digunakan nilai *n* merupakan kelipatan 2 karena hasil *cyclotomic polynomial* dengan kelipatan 2 memiliki bentuk seragam, yaitu $x^n + 1$ dengan jumlah akar dari *cyclotomic polynomial* tersebut berjumlah $n/2$ dengan *root of unity* yang menjadi akar tersebut adalah urutan ganjil [15].

1) Cyclotomic Polynomial Ring

Cyclotomic polynomial ring adalah *polynomial ring* dengan subset *cyclotomic polynomial* sehingga polinomial yang dibentuk bersifat isomorfis dengan akar-akar dari *cyclotomic polynomial*. Misalkan $\mathbb{Z}(x)$ merupakan *polynomial ring* dan $x^n + 1$ merupakan *cyclotomic polynomial*, maka dapat dibentuk *cyclotomic polynomial ring* dengan *polynomial ring* yang berada pada *finite field* dari *cyclotomic polynomial* sebagai $R = \frac{\mathbb{Z}(x)}{x^n+1}$ [15].

C. Enkripsi Homomorfik Total Skema CKKS

Skema CKKS merupakan skema enkripsi homomorfik untuk pendekatan hasil dari operasi aritmatik sehingga mendukung operasi hingga pada bilangan kompleks. Skema CKKS mendukung operasi pertambahan serta perkalian pada pesan yang terenkripsi beserta dengan prosedur *rescaling* baru untuk mengatur nilai dari plainteks. Prosedur tersebut memotong sebuah cipherteks menjadi modulus yang lebih kecil yang menyebabkan nilai plainteks mendekati nilai aslinya [9].

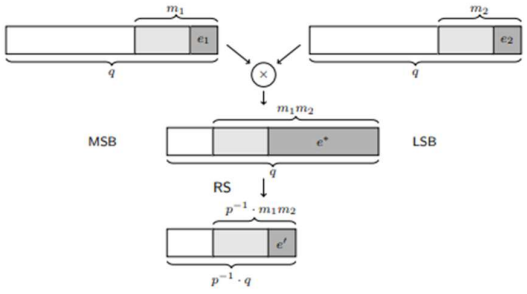

Fig. 3. Bentuk Cipherteks Pada Skema CKKS [9].

Skema CKKS mendukung operasi pertambahan serta perkalian pada pesan yang terenkripsi beserta dengan prosedur *rescaling* baru untuk mengatur nilai dari plainteks. Prosedur tersebut memotong sebuah cipherteks menjadi modulus yang lebih kecil yang menyebabkan nilai plainteks mendekati nilai aslinya [9].

1) Pembangkitan Kunci

Terdapat $a, s, e \in R_q = \frac{Z_q(x)}{x^n+1}$, dimana R_q adalah *cyclotomic polynomial ring*, maka dapat dirumuskan pembangkitan kunci *secret key* sk dan *public key* pk.

$$sk = s \quad (2)$$

$$pk = (a, b = -a.s + e) \quad (3)$$

Dengan mengikuti permasalahan RLWE, yang mana $sk \in R_q$ yang merupakan *cyclotomic polynomial ring* dan $pk \in R_q^2$ yang merupakan vektor *cyclotomic polynomial ring* dimensi dua [15].

2) Encoding dan Decoding

Terdapat $z \in \mathbb{C}^{\frac{n}{2}}$ yang merupakan vektor yang dibentuk dari bilangan kompleks berjumlah $\frac{n}{2}$, yang disesuaikan dengan pesan yang akan dienkripsi, dan terdapat Δ sebagai suatu bilangan bulat *scale* yang berasosiasi dengan presisi plainteks, maka dapat didefinisikan algoritma *encoding*.

$$\mu \leftarrow Ecd(\Delta, z) \quad (4)$$

Dimana $\mu \in R = \frac{\mathbb{Z}(x)}{x^n+1}$, sebagai plainteks, yaitu *cyclotomic polynomial ring* yang dibuat berdasarkan nilai target z. Bila terdapat $\mu \in R$ dan Δ yang merupakan *scale*, maka dapat dirumuskan algoritma *decoding*.

$$z' \leftarrow Dcd(\Delta, \mu) \quad (5)$$

Dimana $z' \in \mathbb{C}^{\frac{n}{2}}$ merupakan vektor kompleks dengan dimensi $\frac{n}{2}$. Perhatikan bahwa $z' \approx z$ [15].

3) Enkripsi dan Dekripsi

Terdapat plainteks $\mu \in R$ dengan $pk = (a, b = -a.s + e) \in R_q^2$ sebagai *public key* dan s sebagai *secret key*, maka dapat dirumuskan algoritma enkripsi.

$$c \leftarrow Enc(pk, \mu), c = (\mu - a \cdot s + e, a) = (c_0, c_1) \in R_q^2 \quad (6)$$

Dimana cipherteks $c \in R_q^2$ merupakan vektor *cyclotomic polynomial ring*. Dengan $sk = s$ sebagai *secret key* maka dapat dirumuskan algoritma dekripsi dari cipherteks.

$$\mu' \leftarrow Dec(sk, c),\ \mu' = c_0 + c_1 \cdot s \quad (7)$$

Dimana $\mu \approx \mu' \in R$ sebagai plainteks [15].

4) Evaluasi

a) Penjumlahan

Pada operasi penjumlahan antara cipherteks dengan cipherteks, terdapat dua cipherteks $c = (c_0, c_1)$, $c' = (c'_0, c'_1) \in R_q^2$, maka penjumlahan cipherteks.

$$c_{add} = c + c' = (c_0 + c'_0, c_1 + c'_1) \quad (8)$$

Pembuktian atas operasi tersebut dapat dibuktikan dengan melakukan dekripsi pada cipherteks.

$$Dec(sk, c_{add}) = c_0 + c'_0 + (c_1 + c'_1) \cdot s \approx \mu + \mu' \quad (9)$$

Dari operasi tersebut dapat dibuktikan dengan melakukan dekripsi pada cipherteks sehingga nilai yang dihasilkan akan sama dengan nilai pesan sebelum terenkripsi [15].

b) Perkalian

Pada operasi perkalian antara cipherteks dengan cipherteks, terdapat dua cipherteks $c = (c_0, c_1)$, $c' = (c'_0, c'_1) \in R_q^2$, maka perkalian cipherteks.

$$\begin{aligned} c_{mult} &= (c_0 \cdot c'_0, c_0 \cdot c'_1 + c_1 \cdot c'_0, c_1 \cdot c'_1) \\ &= (d_0, d_1, d_2) \end{aligned} \quad (10)$$

Sedangkan untuk melakukan dekripsi dapat dilakukan dengan menggunakan cipherteks hasil perkalian, berikut operasi dekripsi pada cipherteks hasil perkalian antar cipherteks.

$$Dec(sk, c_{mult}) = d_0 + d_1 \cdot s + d_2 \cdot s^2 \approx \mu \cdot \mu' \quad (11)$$

Sehingga didapatkan aproksimasi nilai perkalian cipherteks mendekati nilai perkalian dua plainteks [15].

5) Relinearization

Proses *relinearization* adalah proses reduksi polinomial pada perkalian polinomial agar polinomial yang dihasilkan tidak melebihi dari dua polinomial yang mana pada proses perkalian cipherteks dapat menghasilkan lebih dari dua polinomial sehingga berpotensi memperbesar ukuran dan kompleksitas dari cipherteks.

Terdapat $c_{mult} \in R_q^3$ yang dihasilkan dari proses perkalian cipherteks dan terdapat evk yaitu *evaluation key* yang digunakan untuk proses *relinearization*. Perhatikan bahwa proses *relinearization* digunakan untuk mereduksi polynomial sehingga $c_{mult} \in R_q^3 \rightarrow c_{relin} \in R_q^2$. Untuk itu, diperlukan suatu bentuk cipherteks baru, yaitu $(d'_0, d'_1) = (d_0, d_1) + P$ dimana P adalah polinomial yang bila dilakukan dekripsi akan menghasilkan $Dec(sk, P) = d_2 \cdot s^2$. Untuk menemukan nilai P maka diperlukan $evk = (-a + e + p \cdot s^2, a) \; mod \; p \cdot q$ dengan p merupakan bilangan bulat besar acak uniform dan $a, s, e \in R_{p \cdot q}$ sehingga nilai $P = \lfloor p^{-1} \cdot d_2 \cdot evk \rceil$ sehingga bila dilakukan dekripsi, maka akan menghasilkan persamaan berikut ini.

$$\begin{aligned} Dec(sk, P) &= p^{-1} \cdot d_2 \cdot (-a + e + p \cdot s^2) + p^{-1} \cdot d_2 \cdot a \cdot s \\ &\approx d_2 \cdot s^2 \end{aligned} \quad (12)$$

Proses *relinearization* memperkenalkan metode baru, yaitu asumsi *circular security* yang mana skema yang dihasilkan harus menjamin keamanan enkripsi masing-masing *key* yang digunakan [15].

6) Rescaling

Pada perkalian Homomorfik, terdapat nilai target z dan z', bila terdapat *scale* Δ, maka operasi perkalian akan menghasilkan nilai $\Delta^2 \cdot z \cdot z'$ sehingga akan terjadi *overflow* untuk operasi perkalian berikutnya karena besar *scale* akan terus meningkat. Untuk menghindari hal tersebut, maka dibuat proses *rescaling* yang berguna untuk mereduksi hasil perkalian sehingga hasil perkalian mendekati *scale* yang seharusnya. Terdapat permasalahan *rescaling* bila menggunakan pembagian *scale* dengan Δ untuk menjaga nilai *scale* karena dapat memicu pengurangan besar cipherteks sehingga berpotensi dapat merusak nilai target yang disimpan pada cipherteks. Untuk itu, terdapat $Q_l = q_0 \cdot q_1 \cdot q_2 \dots q_l, 1 \leq l \leq L$ untuk seluruh $q_1 \dots ql$ merupakan prima unik yang digunakan sebagai *modulo chain* sehingga untuk setiap operasi perkalian dapat direduksi dengan melakukan pembagian antara hasil cipherteks dan q_l untuk mendapatkan bentuk cipherteks yang dapat dikontrol. Berikut persamaan yang digunakan pada *rescaling*.

$$c' \leftarrow RS_{l \rightarrow l'}(c), c' = \left(c \frac{Q_l}{q_l}\right) \quad (13)$$

Sehingga $Q'_l = \frac{Q_l}{q_l}$ dengan nilai $l' = l - 1$ [15].

7) Bootstrapping

Proses *Bootstrapping* adalah proses pada enkripsi homomorfik total yang melakukan prosedur evaluasi pada fungsi dekripsi cipherteks dari suatu skema enkripsi homomorfik yang digunakan agar bisa mendukung operasi homomorfik dengan menghasilkan nilai yang sama dengan operasi pada plainteks [17].

8) Number Theoritic Transform

Number Theoritic Transform (NTT) adalah polinomial pada *Discrete Fourier Transform* (DFT) yang memiliki koefisien berdasarkan bidang terbatas dan bisa diselesaikan dengan menggunakan *Fast Fourier Transform* (FFT) sesuai dengan bidang terbatas tersebut. Perkalian polinomial pada bidang terbatas sudah menjadi operasi fundamental pada skema kriptografi berdasarkan masalah pada *Ring Learning With Errors* (RLWE), sehingga dengan menggunakan NTT dapat mengurangi kompleksitas waktu saat melakukan operasi perkalian polinomial.

Pemilihan ψ sebagai primitif N_{th} *root of unity* dengan $\psi^N = 1 \; mod \; p$ untuk N yang sudah diberikan dan sebuah bilangan prima $p = kN + 1$. Untuk keseluruhan algoritma NTT pada saat melakukan operasi modular perkalian bilangan bulat sebagai berikut.

$$Xk = (\sum_{n=0}^{N-1} x_n \psi_{N,p}^{n(k)}) \; mod \; p \qquad (14)$$

Dengan $0 \leq k < N$, $\psi_{N,p}$ merupakan primitif N_{th} pada NTT untuk z_P, x_n adalah koefisien masukan polinomial untuk indeks n dan Xk merupakan keluaran koefisien polynomial dengan indeks k [18].

IV. DESAIN DAN IMPLEMENTASI SOLUSI

A. Kebutuhan Fungsional Pengguna

Terdapat beberapa kebutuhan fungsional yang sudah didefinisikan sebagai berikut.

TABLE I. KEBUTUHAN FUNGSIONAL PENGGUNA

ID	Kebutuhan Fungsional	Operasi Homomorfik
SRS-01	Pengguna dapat melakukan transaksi sesuai dengan pesanan.	Melakukan enkripsi terhadap meta transaksi bersesuaian dengan id produk, kuantitas produk, dan voucher yang digunakan.
SRS-02	Pengguna dapat mengisi dan menggunakan saldo.	Operasi penjumlahan dan pengurangan homomorfik
SRS-03	Pengguna dapat menggunakan *voucher* .	Operasi penjumlahan dan perkalian homomorfik

B. Pemilihan Kakas Enkripsi Homomorfik Skema CKKS

Terdapat beberapa kakas *open-source* yang telah mengimplementasikan skema enkripsi homomorfik. Berikut kakas *open-source* yang banyak digunakan pada impelementasi komputasi awan.

TABLE II. IMPLEMENTASI SKEMA HE PADA KAKAS *OPEN-SOURCE*

Kakas	BGV	BFV	THFE	CKKS
HELib	✓			✓
SEAL		✓		✓
PALISADE	✓	✓	✓	✓
Lattigo		✓		✓
HEANN				✓

Berdasarkan pada operasi polynomial dengan parameter $Log(n) = 14, L = 6$, dan skala 2^{40} pada *kernel.*

Library	Packing Method	Confussion Matrix		Run time per packet (s)	Packet size (Mb)
-	-	140964	1620	10 (total)	-
		4944	107		
SEAL	Simple	140964	1620	308.1	2.00
		4944	107		
SEAL	Multi	140964	1620	278.5	64.0
		4944	107		
Lattigo	Simple	140964	1620	315.7	2.33
		4944	107		
Lattigo	Multi	140964	1620	118.8	74.8
		4944	107		

Fig. 4. Hasil Pengujian Kakas Pada Kernel Polinomial [15].

Berdasarkan hasil uji tersebut, terlihat bahwa kecepatan proses kakas *Lattigo* memiliki kecepatan lebih cepat dibanding SEAL pada *packing method* Multi, tetapi berbanding terbalik dengan besar paket yang dihasilkan jauh lebih besar disbanding SEAL. Oleh karena itu, dipilihlah kakas Lattigo yang berbasis *Go-Language* untuk digunakan pada tugas akhir ini.

C. Desain Alur Enkripsi Homomorfik

Pada enkripsi homomorfik, terdapat banyak proses yang dapat dilakukan sehingga memungkinkan terjadi kebingungan bagi pengguna yang akan menggunakannya.

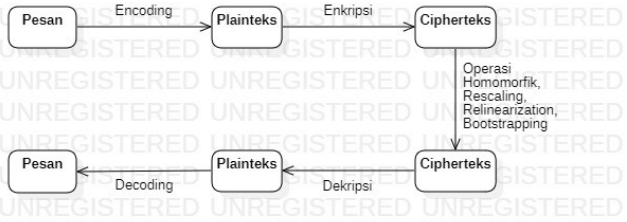

Fig. 5. Desain Alur Enkripsi Homomorfik Pada Komputasi Awan

Untuk itu, terdapat alur enkripsi homomorfik yang menyesuaikan kebutuhan fungsional yang sudah dijelaskan sebelumnya.

D. Desain dan Implementasi Arsitektur Sistem

Terdapat beberapa layanan yang dibangun pada *layer business logic* komputasi awan yang menggunakan enkripsi homomorfik. Layanan saldo menangani operasi homomorfik pada saldo pengguna. Berikut modul pada layanan saldo.

TABLE III. DAFTAR MODUL LAYANAN SALDO

Nama Layanan	*Business Logic Function*	Penjelasan
Saldo	*Activate*	Fungsi *Activate* melakukan proses enkripsi homomorfik saldo nol pada pengguna.
	CheckBalance	Fungsi untuk mendapatkan saldo pengguna dengan melakukan dekripsi pada cipherteks.
	Topup	Fungsi *Topup* melakukan proses penjumlahan homomorfik pada saldo pengguna.

Sedangkan layanan transaksi menangani operasi homomorfik pada transaksi yang dilakukan oleh pengguna. Berikut modul pada layanan transaksi.

TABLE IV. DAFTAR MODUL LAYANAN TRANSAKSI

Nama Layanan	*Business Logic Function*	Penjelasan
Transaksi	*Transact*	Fungsi *Transact* melakukan proses penjumlahan dan perkalian homomorfik bila menggunakan diskon, melakukan proses pengurangan homomorfik pada saldo pengguna.
	GetTransact	Fungsi *GetTransact* melakukan proses dekripsi data riwayat transaksi yang didapatkan dari basis data.

1) Arsitektur Komputasi Awan

Dengan meninjau *request* dan *response* yang akan dilakukan oleh pengguna, maka *architecture pattern* yang cocok digunakan untuk *layer* teratas adalah *client-server pattern*.

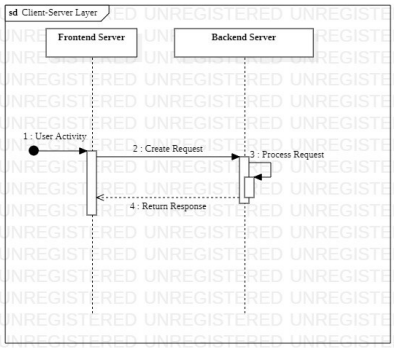

Fig. 6. Client-Server Layer Pada Implementasi Komputasi Awan

Selain itu, terdapat implementasi *business logic* pada *backend* untuk melakukan proses enkripsi homomorfik dan proses lain yang diperlukan pada toko *online*. *Architecture pattern* yang akan digunakan pada *backend layer* adalah *layered pattern*.

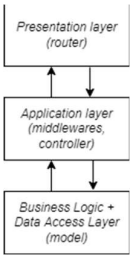

Fig. 7. Layered Architecture Pada *Backend Layer*

2) Arsitektur Layanan Saldo

a) Modul Activate

Terdapat rancangan arsitektur alur modul *activate* yang digunakan untuk mengaktifkan saldo pengguna bila pengguna belum pernah melakukan aktivasi saldo sebelumnya.

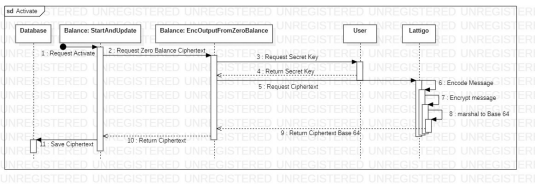

Fig. 8. Diagram Alur Modul *Activate*

Terdapat implementasi tampilan *frontend* modul *Activate* sebagai berikut.

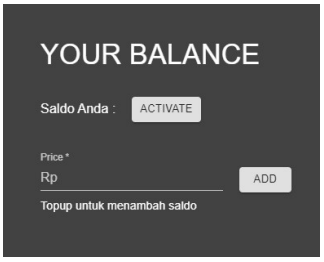

Fig. 9. Tampilan Modul *Activate* Pada *Frontend*

Pada *server backend* terdapat REST API terkait modul *Activate* yang dapat diakses melalui pranala *endpoint* berikut.

```
POST /mybalances/activate
```

b) Modul CheckBalance

Terdapat rancangan arsitektur alur modul *CheckBalance* yang digunakan untuk melakukan dekripsi pada cipherteks saldo pengguna untuk ditampilkan.

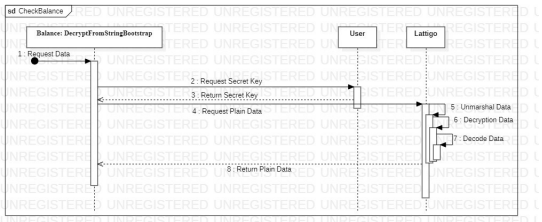

Fig. 10. Diagram Alur Modul *CheckBalance*

Terdapat implementasi tampilan *frontend* modul *CheckBalance* sebagai berikut.

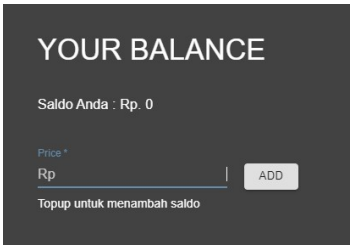

Fig. 11. Tampilan Modul *CheckBalance* Pada *Frontend*

Pada *server backend* terdapat REST API terkait modul *CheckBalance* yang dapat diakses melalui pranala *endpoint* berikut.

```
GET /mybalances/check
```

Dengan melakukan *request* pada *endpoint* tersebut maka akan didapatkan contoh *resource* sebagai berikut.

```
{
    "id": 1,
    "current_balance": "Rp. 0\n"
}
```

c) Modul Topup

Terdapat rancangan arsitektur alur modul *topup* yang digunakan untuk mengisi saldo pengguna bila pengguna sudah melakukan aktivasi saldo sebelumnya.

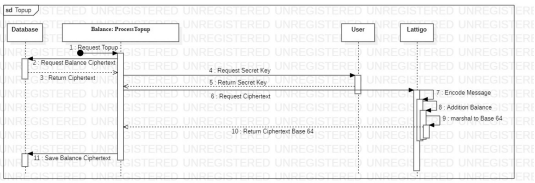

Fig. 12. Diagram Alur Modul *Topup*

Terdapat implementasi tampilan *frontend* modul *Topup* sebagai berikut.

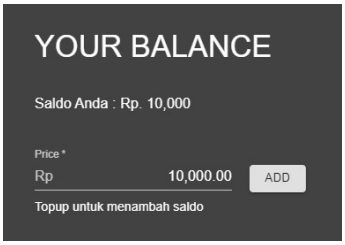

Fig. 13. Tampilan Modul *Topup* Pada *Frontend*

Pada *server backend* terdapat REST API terkait modul *Topup* yang dapat diakses melalui pranala *endpoint* berikut.

```
POST /mybalances/topup
```

Berikut contoh *request body* yang dapat diberikan untuk melakukan *request* pada *endpoint* tersebut.

```
{
    "added_balance": 10000.0
}
```

3) Arsitektur Layanan Transaksi

a) Modul Transact

Terdapat rancangan arsitektur alur modul *Transact* yang digunakan untuk melakukan transaksi produk dengan menggunakan saldo pengguna.

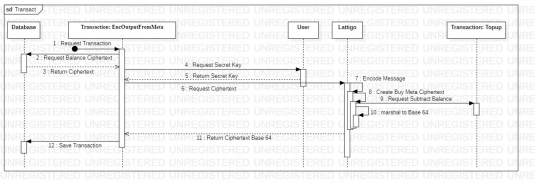

Fig. 14. Diagram Alur Modul *Transact*

Terdapat implementasi tampilan *frontend* modul *Transact* sebagai berikut.

Fig. 15. Tampilan Modul *Transact* Pada *Frontend*

Pada *server backend* terdapat REST API terkait modul *Transact* yang dapat diakses melalui pranala *endpoint* berikut.

```
POST /transacts
```

Berikut contoh *request body* yang dapat diberikan untuk melakukan *request* pada *endpoint* tersebut.

```
{
    "author_id": 1,
    "product_id": 1,
    "qty": 1,
    "disc_name": "DISCOUNT30S"
}
```

b) Modul GetTransact

Terdapat rancangan arsitektur alur modul Get*Transact* yang digunakan untuk mendapatkan transaksi produk yang sudah pernah dilakukan sebelumnya.

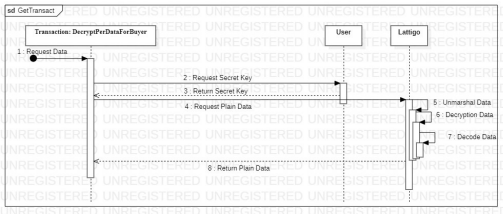

Fig. 16. Diagram Alur Modul *GetTransact*

Terdapat implementasi tampilan *frontend* modul *GetTransact* sebagai berikut.

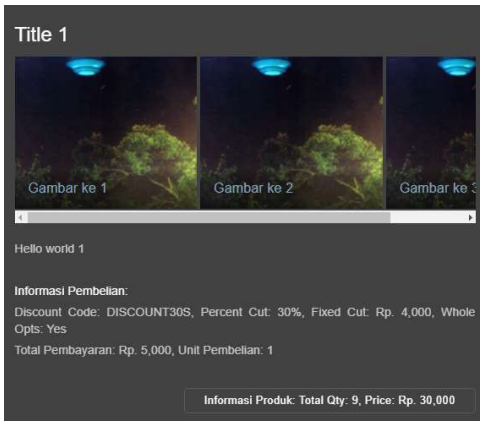

Fig. 17. Tampilan Modul *GetTransact* Pada *Frontend*

Pada *server backend* terdapat REST API terkait modul *GetTransact* yang dapat diakses melalui pranala *endpoint* berikut.

```
GET /transactmy
```

Dengan melakukan *request* pada *endpoint* tersebut maka akan didapatkan contoh *resource* sebagai berikut.

```
{
  "transacts": [
      {
        "id": 1,
        "buyer_meta": "1,1,2",
        "buyer_total_bill": "5000"
      }
  ]
}
```

V. PENGUJIAN DAN ANALISIS

Pengujian dilakukan dengan menghitung waktu eksekusi serta ukuran variabel yang digunakan untuk menampung nilai dari setiap proses enkripsi homomorfik yang dijalankan oleh program. Pengujian dilakukan sesuai dengan lingkungan implementasi, yaitu *processor* AMD Ryzen 7 5800 H dengan *memory* 8GB 3200 MHz *Dual Channel*. Pengujian tersebut dilakukan untuk mengetahui apakah enkripsi homomorfik tersebut dapat digunakan dalam keadaan nyata dengan memperhatikan waktu eksekusi yang rendah serta ukuran variabel yang relatif kecil. Terdapat dua pengujian yang dilakukan, yaitu dengan proses NTT dan tanpa proses NTT. Untuk masing masing pengujian, terdapat masing-masing proses yang dilakukan yaitu pembangkitan *secret key*, proses *encoding*, *decoding,* enkripsi, dekripsi, *addition*, *multiplication*, *rescaling,* dan *bootstrapping*.

A. *Pengujian Dengan NTT*

Pengujian terhadap NTT dilakukan untuk memastikan pengurangan waktu eksekusi dan ukuran variabel yang akan dihasilkan selama proses tersebut dilaksanakan.

TABLE V. HASIL PENGUJIAN WAKTU EKSEKUSI DENGAN NTT

Operasi	Parameter ($Log(NQ)$)	Waktu Eksekusi (ms)	Ukuran Variabel ($byte$)
Secret Key	N=15, Q=666, NQ=681	13.21	8
	N=16, Q=1248, NQ=1264	39.00	8
	N=16, Q=1303, NQ=1319	39.74	8
Encoding	N=15, Q=666, NQ=681	48.37	8
	N=16, Q=1248, NQ=1264	166.27	8
	N=16, Q=1303, NQ=1319	186.72	8
Decoding	N=15, Q=666, NQ=681	131.43	24
	N=16, Q=1248, NQ=1264	429.61	24
	N=16, Q=1303, NQ=1319	496.67	24
Enkripsi	N=15, Q=666, NQ=681	106.34	8
	N=16, Q=1248, NQ=1264	360.26	8
	N=16, Q=1303, NQ=1319	407.11	8
Dekripsi	N=15, Q=666, NQ=681	122.79	8
	N=16, Q=1248, NQ=1264	408.60	8
	N=16, Q=1303, NQ=1319	470.14	8
Add	N=15, Q=666, NQ=681	19.78	8
	N=16, Q=1248, NQ=1264	41.72	8
	N=16, Q=1303, NQ=1319	42.13	8
AddConstan	N=15, Q=666, NQ=681	11081.84	8
	N=16, Q=1248, NQ=1264	18750.16	8
	N=16, Q=1303, NQ=1319	19231.73	8
Mult	N=15, Q=666, NQ=681	110.39	8
	N=16, Q=1248, NQ=1264	417.77	8
	N=16, Q=1303, NQ=1319	419.13	8
MultConstan	N=15, Q=666, NQ=681	22310.75	8
	N=16, Q=1248, NQ=1264	27832.52	8
	N=16, Q=1303, NQ=1319	28141.26	8
Rescaling	N=15, Q=666, NQ=681	101.85	8
	N=16, Q=1248, NQ=1264	342.22	8
	N=16, Q=1303, NQ=1319	353.77	8
Bootstrapping	N=15, Q=666, NQ=681	55833.71	8
	N=16, Q=1248, NQ=1264	66701.95	8
	N=16, Q=1303, NQ=1319	79469.10	8
Total	N=15, Q=666, NQ=681	89750.27	88
	N=16, Q=1248, NQ=1264	115030.59	88
	N=16, Q=1303, NQ=1319	128796.24	88

Terlihat bahwa waktu eksekusi meningkat sesuai dengan besar parameter yang digunakan, tetapi ukuran variabel yang digunakan untuk menampung nilai masih tetap konstan dan tidak mengalami *overflow*. Ukuran Variabel proses *decoding* memiliki ukuran lebih besar karena variabel yang digunakan untuk menampung nilai asli yang sudah didekripsi sehingga tidak ada proses modulo yang dapat menurunkan ukuran variabel.

B. *Pengujian Tanpa NTT*

Dengan menggunakan metode pengujian yang sama seperti pengujian dengan NTT, berikut merupakan tabel hasil pengujian pada proses enkripsi homomorfik tanpa menggunakan proses NTT.

TABLE VI. HASIL PENGUJIAN WAKTU EKSEKUSI TANPA NTT

Operasi	Parameter ($Log(NQ)$)	Waktu Eksekusi (ms)	Ukuran Variabel ($byte$)
Secret Key	N=15, Q=666, NQ=681	14.11	8
	N=16, Q=1248, NQ=1264	39.18	8
	N=16, Q=1303, NQ=1319	39.96	8
Encoding	N=15, Q=666, NQ=681	50.01	8
	N=16, Q=1248, NQ=1264	171.31	8
	N=16, Q=1303, NQ=1319	190.57	8
Decoding	N=15, Q=666, NQ=681	134.12	24
	N=16, Q=1248, NQ=1264	431.12	24
	N=16, Q=1303, NQ=1319	517.21	24
Enkripsi	N=15, Q=666, NQ=681	111.00	8
	N=16, Q=1248, NQ=1264	373.01	8
	N=16, Q=1303, NQ=1319	432.31	8
Dekripsi	N=15, Q=666, NQ=681	131.01	8
	N=16, Q=1248, NQ=1264	412.10	8
	N=16, Q=1303, NQ=1319	494.27	8
Add	N=15, Q=666, NQ=681	20.61	8
	N=16, Q=1248, NQ=1264	42.38	8
	N=16, Q=1303, NQ=1319	44.46	8
AddConstan	N=15, Q=666, NQ=681	13111.23	8
	N=16, Q=1248, NQ=1264	19099.11	8
	N=16, Q=1303, NQ=1319	19444.89	8
Mult	N=15, Q=666, NQ=681	111.12	8
	N=16, Q=1248, NQ=1264	419.66	8
	N=16, Q=1303, NQ=1319	421.10	8
MultConstan	N=15, Q=666, NQ=681	23227.86	8
	N=16, Q=1248, NQ=1264	28444.11	8
	N=16, Q=1303, NQ=1319	28777.33	8
Rescaling	N=15, Q=666, NQ=681	104.68	8
	N=16, Q=1248, NQ=1264	347.32	8
	N=16, Q=1303, NQ=1319	356.70	8
Bootstrapping	N=15, Q=666, NQ=681	56199.80	8
	N=16, Q=1248, NQ=1264	67113.20	8
	N=16, Q=1303, NQ=1319	84533.10	8
Total	N=15, Q=666, NQ=681	93083.82	88
	N=16, Q=1248, NQ=1264	116430.46	88
	N=16, Q=1303, NQ=1319	134786.33	88

Terlihat bahwa waktu eksekusi semakin meningkat dibandingkan dengan pengujian dengan NTT, tetapi ukuran variabel masih tetap konstan tanpa mengalami *overflow*.

C. Analisis Pengujian

Dengan meninjau pengujian enkripsi homomorfik dengan NTT dan tanpa NTT, Operasi *secret key* merupakan operasi dengan waktu eksekusi minimum. Hal tersebut dapat terjadi karena pembangkitan *secret key* hanya berdasarkan pembangkitan *cyclotomic polynomial ring* acak dengan distribusi uniform sehingga tidak ada operasi khusus yang dilakukan.

Proses *Add* adalah proses operasi pertambahan pada cipherteks. Pada proses tersebut terjadi penjumlahan antara cipherteks dengan cipherteks, $c + c' = (c_0 + c'_0, c_1 + c'_1)$, sehingga waktu yang dibutuhkan hanya sedikit karena hanya terjadi penjumlahan antar polinomial.

Proses *Encoding* menjadi proses dengan eksekusi waktu tercepat ketiga setelah *Add*. Hal tersebut dapat terjadi karena untuk membentuk plainteks dari suatu pesan hanya membutuhkan operasi interpolasi untuk mengubah bilangan kompleks menjadi polinomial dengan sifat isomorfis. Selain itu, Batasan penggunaan derajat polinomial pada orde dua membuat bentuk *cyclotomic polynomial* menjadi seragam.

Proses *rescaling* adalah proses menyesuaikan *scaling factor* karena pada saat proses perkalian polinomial akan terjadi perkalian terhadap nilai *scaling* sehingga orde *scaling* akan meningkat. Untuk itu, perlu dilakukan pembagian terhadap cipherteks berdasarkan nilai *scaling factor* sehingga cipherteks memiliki nilai *scaling* yang tetap sama dengan sebelum dioperasikan. Pada proses ini, waktu eksekusi pada NTT cukup rendah karena operasi sudah teroptimasi akibat pembagian yang dilakukan. Namun pada proses tanpa NTT, waktu eksekusi meningkat akibat tidak teroptimasi dengan NTT.

Proses Enkripsi menjadi proses dengan eksekusi waktu tercepat keempat mirip dengan proses dekripsi, *decoding*, *mult*, dan *rescaling*. Hal ini disebabkan untuk membentuk cipherteks memerlukan perkalian dan penjumlahan polinomial yang berbasis *cyclotomic polynomial*.

Proses *Mult* adalah proses perkalian antar cipherteks. Pada proses ini, terjadi perkalian untuk menghasilkan tiga vektor yang akan digunakan dengan penjumlahan termasuk pada proses perkalian tersebut, $(c_0 \cdot c'_0, c_0 \cdot c'_1 + c_1 \cdot c'_0, c_1 \cdot c'_1)$, sehingga waktu yang dihasilkan menjadi lebih besar dibanding pada proses *Add*.

Proses Dekripsi menjadi proses dengan eksekusi waktu tercepat keenam setelah *encoding*. Hal tersebut dapat terjadi karena untuk melakukan dekripsi diperlukan perkalian dan pertambahan polinomial untuk mendapatkan polinomial plainteks.

Proses *Decoding* menjadi proses yang memiliki waktu mirip dengan dekripsi. Pada proses ini terdapat substitusi nilai *root of unity* serta penyesuaian *scaling factor* dari polinomial plainteks untuk menjadi pesan yang dapat dibaca oleh manusia. Berbeda dengan operasi polinomial yang memiliki koefisien bilangan riil, proses *decoding* melakukan operasi dengan bilangan kompleks hasil dari akar akar *root of unity*. Selain itu, pada proses *decoding* menyebabkan ukuran variabel meningkat menjadi 24 *bytes*. Hal itu dapat terjadi karena pada proses *encoding* terjadi proses pengembalian nilai menjadi seperti semula yang sebelumnya terbatas karena operasi modulo menjadi nilai utuh tanpa operasi modulo.

Proses *addConstant* adalah proses yang disediakan kakas lattigo untuk menambahkan konstanta pada cipherteks. Pada proses ini, diperlukan *encoding* dan enkripsi terlebih dahulu pada konstanta yang akan ditambahkan pada cipherteks. Oleh karena itu, waktu yang dibutuhkan untuk melakukan eksekusi sangat tinggi.

Proses *multConstant* adalah proses yang disediakan kakas lattigo untuk mengalikan konstanta pada cipherteks. Pada proses ini, diperlukan *encoding* dan enkripsi terlebih dahulu pada konstanta yang akan ditambahkan pada cipherteks. Setelah itu, dilakukan perkalian polinomial sehingga waktu *multConstant* menjadi lebih tinggi bila dibandingkan dengan *addConstant*.

Proses *bootstrapping* menjadi proses dengan eksekusi waktu telama. Hal ini disebabkan oleh kompleksitas operasi yang memanfaatkan rotasi pada *modulo chain* untuk mengembalikan ukuran cipherteks menjadi seperti semula.

Selanjutnya dapat dilakukan perbandingan waktu total eksekusi pada enkripsi homomorfik. Berikut grafik yang dibuat berdasarkan perbandingan waktu eksekusi total antara proses yang menggunakan proses NTT dengan yang tidak menggunakan NTT.

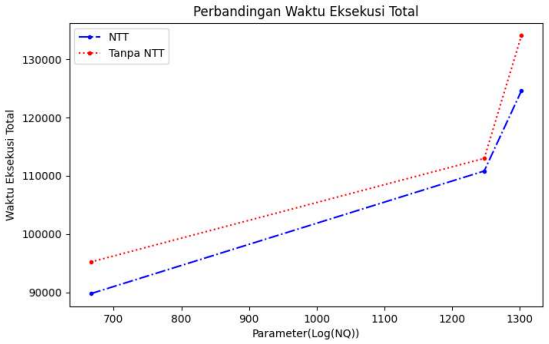

Fig. 18. Grafik Perbandingan Waktu Eksekusi Total Enkripsi Homomorfik

Dengan melihat grafik tersebut, maka dapat disimpulkan bahwa proses menggunakan NTT dapat mempersingkat waktu proses bila dibandingkan proses tanpa menggunakan NTT karena proses NTT dilakukan secara *parallel* bila dibandingkan dengan proses tanpa NTT. Terdapat ketidaksesuaian kompleksitas waktu ideal pada NTT yang seharusnya berada pada $O(n \log(n))$. Hal ini dapat terjadi disebabkan oleh keterbatasan perangkat keras yang digunakan karena hanya dapat menggunakan 8 *core* untuk melakukan proses paralel sehingga tidak mampu untuk melakukan proses paralel penuh pada parameter uji.

Dengan meninjau ukuran variabel total yang digunakan untuk melakukan enkripsi homomorfik, variabel total yang digunakan untuk melakukan proses enkripsi homomorfik adalah konstan tanpa mengalami *overflow*. Berikut grafik perbandingan ukuran variabel total antara proses yang menggunakan NTT dengan proses tanpa NTT.

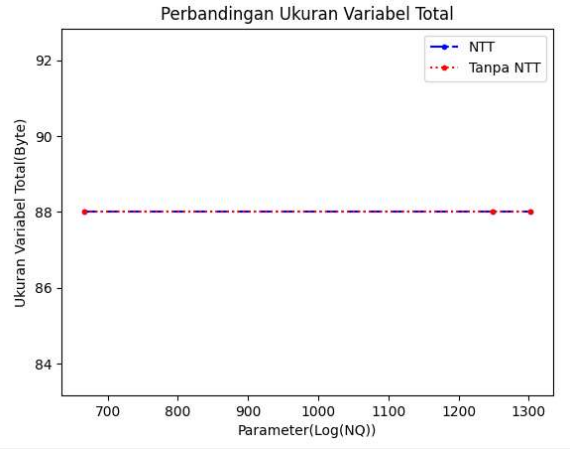

Fig. 19. Grafik Perbandingan Ukuran Variabel Total Enkripsi Homomorfik

Selanjutnya, dengan meninjau waktu eksekusi total dari semua parameter standar dari *lattigo* untuk melakukan *bootstrapping* adalah berkisar 2 menit. Hal tersebut dapat terjadi karena waktu eksekusi proses *bootstrapping* menggunakan waktu yang sangat lama hingga berkisar 1 menit. Untuk itu, perlu dilakukan modifikasi parameter standar maupun pemangkasan proses pada *bootstrapping* agar dapat digunakan pada kondisi nyata.

VI. KESIMPULAN

Enkripsi Homomorfik skema CKKS diimplementasikan pada komputasi awan dengan menggunakan kakas *lattigo* karena memiliki waktu komputasi lebih cepat dibanding kakas lainnya. Implementasi komputasi awan menggunakan *architecture pattern* untuk implementasi modul dan arsitektur API yang digunakan untuk membuat API sebagai komunikasi antar *server*. Waktu eksekusi enkripsi homomorfik dengan proses NTT lebih cepat bila dibandingkan tanpa proses NTT karena proses *parallel* yang dilakukan pada proses NTT. Ukuran variabel yang dihasilkan untuk melakukan proses enkripsi homomorfik konstan untuk parameter yang sudah diujikan tanpa mengalami *overflow*. Waktu eksekusi yang diperlukan untuk melakukan proses enkripsi homomorfik skema CKKS meningkat secara linier mengikuti parameter uji. Waktu yang dibutuhkan untuk mengeksekusi proses *bootstrapping* yang mencapai rata-rata satu menit untuk seluruh parameter yang diuji, maka enkripsi homomorfik belum bisa diimplementasikan untuk kondisi nyata sehingga perlu dilakukan modifikasi parameter standar maupun pemangkasan proses pada *bootstrapping* agar dapat digunakan pada kondisi nyata.

UCAPAN TERIMA KASIH

Penulis mengucapkan terimakasih sebesar-besarnya kepada dosen pembimbing, yaitu Bapak Dr. Ir. Rinaldi Munir, M.T. dan Bapak Infall Syafalni, S.T., M.Sc. selaku dosen pembimbing tugas akhir yang telah memberikan bimbingan disertai berbagai diskusi hangat yang membantu penulis dalam pengerjaan tugas akhir ini.

REFERENSI

[1] P. P. Ray, "An Introduction to Dew Computing: Definition, Concept and Implications - IEEE Journals & Magazine," *IEEE Access,* pp. 723-737, 2018.

[2] Biro Hukum dan Hubungan Masyarakat – BSSN, "Marak Kejahatan Siber, BSSN RI Selenggarakan Workshop Bagi Sektor IIKN di Bali," 16 Juni 2021. [Online]. Available: https://bssn.go.id/marak-kejahatan-siber-bssn-ri-selenggarakan-workshop-bagi-sektor-iikn-di-bali/.

[3] CNN Indonesia, "Deretan Kasus Bocor Data Penduduk RI dari Server Pemerintah," 1 September 2021. [Online]. Available: https://www.cnnindonesia.com/teknologi/20210901150749-185-688400/deretan-kasus-bocor-data-penduduk-ri-dari-server-pemerintah.

[4] W. Stallings, Cryptography and Network Security, Principle and Practice, 5th edition, Pearson Education, Inc., 2011.

[5] C. Zuo, Z. Lin and Y. Zhang, "Why Does Your Data Leak? Uncovering the Data Leakage in Cloud from Mobile Apps," *2019 IEEE Symposium on Security and Privacy (SP),* 2019.

[6] A. Greenberg, "Hacker lexicon: What is homomorphic encryption?," 11 Maret 2014. [Online]. Available: www.wired.com/2014/11/hacker-lexicon-homomorphic-encryption.

[7] L. Morris, "Analysis of Partially and Fully Homomorphic Encryption," *Homomorphic Encryption,* pp. 1-5, 2013.

[8] X. Yi, R. Paulet and E. Bertino, Homomorphic Encryption and Applications, Heidelberg, New York, Dordrecht, London: Springer, Cham, 2014.

[9] J. H. Cheon, A. Kim, M. Kim and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," *Takagi T., Peyrin T. (eds) Advances in Cryptology – ASIACRYPT 2017,* p. 409–437, 2017.

[10] M. Van Dijk, C. Gentry, S. Halevi and V. Vaikuntanathan, "Fully homomorphic encryption over the integers. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques," *EUROCRYPT 2010: Advances in Cryptology,* pp. 24-43, 2010.

[11] H. Y. C. L. H. X. S. B. Wang W, "Exploring the feasibility of fully homomorphic," *IEEE Trans Comput,* p. 698–706, 2015.

[12] B. S. Wei D, "cuHE: a homomorphic encryption accelerator library," *International conference on cryptography and information security in the balkans,* 2015.

[13] J. Goey, W. Lee, B. Goi and W. Yap, "Accelerating number theoretic transform in GPU platform," *The Journal of Supercomputing,* 2020.

[14] M. TEBAA, S. E. HAJJI and A. E. GHAZI, "Homomorphic Encryption Applied to the Cloud," *Proceedings of the World Congress on Engineering,* 2012.

[15] A. V. Saavedra, "Study and Applications of Homomorphic," Universidade de Vigo, Vigo, 2020.

[16] C. Peikert, "A Decade of Lattice Cryptography," *IACR Cryptol,* p. 939, 2015.

[17] X. Zhao and A. Wang, "Generalized Bootstrapping Technique Based on Block Equality Test Algorithm," *Security and Communication Networks,* 2018.

[18] S. Kim, W. Jung, J. Park and J. H. Ahn, "Accelerating Number Theoretic Transformations for Bootstrappable," 2020.