Homomorphic Encryption dalam MapReduce di Teknologi *Big Data*

Jevant Jedidia Augustine - 13520133
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13520133@std.stei.itb.ac.id

***Abstract*—Perkembangan teknologi *big data* yang pesat membawa banyak peluang untuk analisis data yang lebih mendalam di berbagai industri. Namun, perkembangan tersebut masih harus memperhatikan permasalahan keamanan dan privasi data sensitif. Metode enkripsi dan dekripsi tradisional menjadi solusi yang efektif untuk data yang disimpan (*at rest*) tetapi kurang efektif untuk analisis data. *Homomorphic encryption* menjadi solusi bagi permasalahan tersebut. Dengan enkripsi homomorfik, proses analisis dan pengolahan data pada bidang *big data* seperti MapReduce menjadi lebih aman dan efisien.**

Keywords—homomorphic encryption; MapReduce; CKKS;

I. PENDAHULUAN

Dalam beberapa tahun terakhir, pertumbuhan pesat teknologi big data telah merevolusi berbagai industri, memungkinkan perkembangan wawasan dan kemajuan yang belum pernah terjadi sebelumnya. Namun, sama seperti teknologi data tradisional, akumulasi data sensitif telah menjadi masalah besar yang menyangkut keamanan data dan privasi. Kriptografi tradisional untuk data, meskipun efektif dalam melindungi data saat penyimpanan, tidak mampu melakukan komputasi dan analisis data terenkripsi. Masalah ini melahirkan sebuah solusi kriptografi yang dikenal sebagai *homomorphic encryption*.

Enkripsi homomorfik merupakan sebuah konsep yang berlandaskan konsep homomorfisme dalam matematika. Enkripsi homomorfik memungkinkan untuk dilakukannya operasi seperti penjumlahan dan perkalian terhadap *ciphertext* secara langsung tanpa harus dilakukan dekripsi terlebih dahulu terhadap *ciphertext*. Dengan enkripsi homomorfik, maka data-data sensitif masih dapat diproses tanpa harus mengekspos data yang sesungguhnya kepada pihak yang memroses data tersebut sehingga keamanan data sensitif dapat dijaga dengan baik.

Penggunaan enkripsi homomorfik dapat diterapkan dalam MapReduce, yaitu sebuah paradigma pemrosesan yang populer dan sering digunakan dalam teknologi Big Data. MapReduce memungkingkan kita untuk melakukan penarikan sebuah *insight* dari sebuah data dalam skala yang besar. Pengintegrasian enkripsi homomorfik terhadap MapReduce akan menjamin keamanan data-data sensitif tanpa harus mengorbankan efisiensi pemrosesan data dalam skala besar dengan melakukan MapReduce.

II. LANDASAN TEORI

A. *Public Key Cryptography*

Kriptografi kunci publik, atau yang biasa dikenal sebagai kriptografi asimetris, merupakan sebuah sistem kriptografi yang menggunakan sebuah pasangan kunci yang berbeda dalam proses enkripsi dan dekripsi. Pasangan kunci tersebut terdiri atas kunci publik dan kunci privat. Ide dari kriptografi kunci publik adalah kunci publik dapat didistribusikan secara luas dan tersedia bagi siapapun yang ingin berkomunikasi dengan pemilik kunci publik tersebut, sedangkan kunci privat disimpan dan hanya dimiliki oleh pemilik pasangan kunci. Biasanya, kunci publik digunakan untuk enkripsi pesan dan kunci privat digunakan untuk mendekripsi pesan, akan tetapi penggunaan kriptografi kunci publik pada tanda tangan digital menggunakan kunci privat untuk menghasilkan tanda tangan dan kunci publik untuk memverifikasi tanda tangan tersebut.

Keamanan kriptografi kunci publik terdapat pada sulitnya menurunkan kunci privat dari kunci publik yang dimiliki. Hal tersebut dapat dicapai dengan menggunakan beberapa persoalan *integer* klasik yang sulit dipecahkan seperti pemfaktoran, logaritma diskrit, dan *elliptic curve discrete logarithm problem*. Walaupun demikian, komputasi yang dilakukan untuk enkripsi dan dekripsi hanya memakan waktu relatif yang sedikit.

Beberapa kelebihan dari kriptografi kunci publik adalah kunci yang perlu dirahasiakan hanyalah kunci privat dan penyebaran kunci publik dapat dilakukan bahkan pada saluran yang tidak aman. Pasangan kunci tersebut juga tidak perlu untuk sering diubah, berbeda dengan kriptografi kunci privat dimana kunci harus sering diubah agar keamanan tetap terjaga. Kelemahan dari kriptografi kunci publik adalah penggunaan bilangan yang besar serta perpangkatan yang besar dalam proses enkripsi dan dekripsi yang secara langsung menyebabkan kedua proses tersebut menjadi relatif lebih lambat bila dibandingkan dengan kriptografi simetris. Kunci yang dihasilkan juga relatif lebih besar akibat hal tersebut.

B. *Homomorphic Encryption*

Secara tradisional, apabila ingin dilakukan komputasi terhadap sebuah *ciphertext*, hal yang dilakukan adalah mendekripsi *ciphertext* menjadi *plaintext*, lakukan komputasi terhadap *plaintext*, kemudian enkripsi *plaintext* menjadi *ciphertext*. Proses tersebut tidak aman karena terdapat langkah dimana *ciphertext* didekripsi menjadi *plaintext* dan penyadapan pada proses tersebut akan menjadi ancaman terhadap privasi dan keamanan data. Penggunaan enkripsi homomorfik merupakan solusi dari permasalahan tersebut.

Enkripsi homomorfik dilandasi oleh konsep homomorfisme pada aljabar abstrak yang mengatakan apabila diberikan dua grup $(G, \lozenge)$ dan $(H, \circ)$, sebuah homomorfisme grup dari $(G, \lozenge)$ ke $(H, \circ)$ adalah sebuah fungsi $f: G \rightarrow H$ dimana untuk semua g dan g' di G menyatakan bahwa:

$$f(g \lozenge g') = f(g) \circ f(g')$$

Dari definisi tersebut, dapat diketahui bahwa sebuah enkripsi dapat dikatakan homomorfik apabila untuk semua a dan b dengan skema enkripsi E dalam kunci k, baik kunci publik maupun kunci privat, memenuhi:

$$Ek(a) \circ Ek(b) = Ek(a \lozenge b)$$

Terdapat 2 jenis enkripsi homomorfik:

1. *Partially homomorphic encryption*

Enkripsi yang hanya memungkinkan satu operasi saja yang dilakukan terhadap *ciphertext*, perkalian atau penjumlahan.

2. *Fully homomorphic encryption*

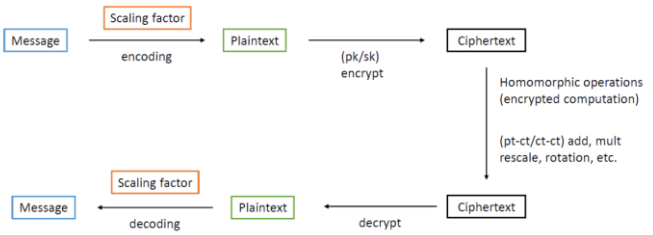
Enkripsi yang memungkinkan untuk melakukan operasi perkalian dan penjumlahan terhadap *ciphertext.*

Contoh algoritma enkripsi homomorfik sebagian adalah algoritma RSA, algoritma ElGamal, dan algoritma Pallier, sedangkan salah satu contoh dari skema enkripsi homomorfik penuh (FHE) adalah CKKS.

C. *CKKS*

CKKS (Cheon-Kim-Kim-Song) merupakan sebuah skema enkripsi homomorfik yang diperkenalkan pada tahun 2017. Salah satu fitur utama CKKS adalah kemampuannya untuk mendukung operasi aritmatika aproksimasi pada *ciphertext.* CKKS mendukung operasi penjumlahan dan perkalian pada *ciphertext* dengan menggunakan prosedur *rescaling* untuk mengurangi nilai eror terhadap hasil dekripsi.

Skema CKKS terdiri atas beberapa tahap, yaitu: *encoding*, *encryption*, *ciphertext operation*, *decryption*, dan *decoding.*



Gambar 2.1 Skema umum CKKS

Sumber:
https://yongsoosong.github.io/files/slides/intro_to_CKKS.pdf

Skema CKKS merupakan skema kunci publik yang berarti skema CKKS akan membangkitkan sebuah pasangan kunci publik dan kunci privat dalam algoritmanya. Selain itu, skema CKKS akan membangkitkan sebuah *eval key* yang diperlukan untuk proses perkalian *ciphertext.* Kunci privat yang dihasilkan merupakan sebuah polinomial sedangkan kunci publik p dan kunci eval E yang dihasilkan adalah:

$$p = (b,a) = (-A.s + e, A)$$

$$E = (b', a') = (-A.s + e + Ps^2, A)$$

dengan,

A = polinom acak

s = kunci privat

e = polinom eror

P = bilangan besar

Pesan masukan skema CKKS merupakan sebuah vektor yang berisikan bilangan riil. Terhadap pesan tersebut akan dilakukan *encoding* untuk memetakannya menjadi sebuah polinomial berdasarkan parameter yang telah ditentukan. Proses *encoding* yang dilakukan pada CKKS merupakan *encoding* aproksimasi, yang berarti hasil *encoding* yang di *decode* akan menghasilkan nilai perkiraan dari nilai awal dan bukan nilai yang persis sama. Maka dari itu, pada proses *encoding*, polinom akan dikalikan dengan *scaling factor* yang menentukan presisi *encoding* untuk mendapatkan aproksimasi nilai akhir dengan presisi yang diinginkan.

Hasil dari proses *encoding* tersebut kemudian akan dienkripsi menggunakan kunci publik. *Ciphertext* yang didapat dari hasil enkripsi merupakan sepasang polinomial. Skema enkripsi m dengan menggunakan kunci p adalah:

$$c = (m, 0) + p = (m - A.s + e, A) = (c0, c1)$$

Proses penjumlahan dan perkalian dapat dilakukan pada *ciphertext.* Penjumlahan kedua *ciphertext* dapat dilakukan dengan menjumlahkan polinom kedua *ciphertext* secara langsung. Untuk *ciphertext c* dan *ciphertext c',* penjumlahan akan dilakukan dengan persamaan:

$$c + c' = (c0, c1) + (c'0, c'1) = (c0 + c'0, c1 + c'1)$$

Sedangkan perkalian kedua *ciphertext* yang sama dilakukan dengan persamaan:

$$c + c' = (c0, c1) * (c'0, c'1) = (c0*c'0, c0*c'1 + c1*c'0, c1*c'1) = (d0, d1, d2)$$

Proses perkalian pada *ciphertext* akan menghasilkan *ciphertext* dengan ukuran yang lebih besar dari sebelumnya (dari 2 polinom menjadi 3 polinom). Operasi perkalian selanjutnya bila menggunakan polinom yang telah didapat akan menambah jumlah polinom yang didapat yang akan menggangu proses dekripsi. Untuk mengatasi hal tersebut, perlu dilakukan relinearisasi untuk tetap menjaga ukuran *ciphertext c = (c0, c1, c2)* tetap konsisten dengan persamaan:

$$c = (c0, c1) + (c2 * evalKey * P^{-1})$$

Rescaling perlu dilakukan setiap kali proses perkalian 2 *ciphertext* dilakukan untuk menjaga nilai *scaling factor* dari *ciphertext* untuk tetap stabil dan tidak bertambah seiring dengan bertambahnya jumlah operasi perkalian. Hal tersebut juga perlu dilakukan agar nilai eror dari *plaintext* tidak menjadi terlalu besar sehingga proses dekripsi menghasilkan *plaintext* yang tidak sesuai.

Proses dekripsi dari *ciphertext c = (c0,c1)* dilakukan dengan menggunakan kunci privat *s* dengan persamaan:

$$m = c0 + c1.s = m - A.s + e + A.s = m + e$$

Hasil dekripsi akan menghasilkan *plaintext* dengan eror. Bila nilai eror ini cukup kecil, maka akan didapatkan *plaintext* yang awal dengan melakukan pembulatan setelah proses *decoding* dilakukan terhadap *plaintext* hasil dekripsi.

D. MapReduce

MapReduce merupakan sebuah paradigma pemrograman yang memungkinkan kita untuk melakukan pemrosesan data dalam skala yang besar. MapReduce diciptakan untuk dapat menangani data dalam jumlah yang sangat besar secara paralel. Proses yang dilakukan dalam MapReduce itu sendiri dibagi menjadi 2 tahap, yaitu tahap *map* dan tahap *reduce*.

Tahap *map* akan menerima data masukan dan memroses setiap data menjadi sebuah pasangan *key-value*. Pengolahan data awal menjadi pasangan *key-value* akan menggunakan sebuah *mapper*, yaitu fungsi yang telah didefinisikan oleh pengguna untuk menghasilkan pasangan *key-value* yang diinginkan. Hasil dari tahap *mapping* ini kemudian akan diberikan kepada *reducer* pada tahap *reduce*.

Pada tahap *reduce*, hasil yang didapat dari tahap *map* akan diagregasikan dengan menggunakan *reducer*, yaitu fungsi yang telah didefinisikan oleh pengguna. Pada tahap ini, data yang memiliki *key* yang sama akan dikelompokkan dan diagregasikan *value*-nya. Operasi agregasi yang paling umum dilakukan adalah penjumlahan, akan tetapi dapat dilakukan juga operasi yang lain sesuai dengan *reducer* yang telah ditentukan.

III. Implementasi

Implementasi MapReduce dengan enkripsi homomorfik akan dilakukan dalam bahasa Python dengan bantuan library Tenseal yang akan menyediakan algoritma CKKS. Secara umum, alur dari implementasi MapReduce adalah sebagai berikut:

1. Menerima data masukan.
2. Membangun kunci publik dan kunci privat CKKS berdasarkan parameter yang ditentukan.
3. Melakukan *encoding* dan *enkripsi* terhadap data yang ingin dijaga kerahasiaannya dengan kunci publik yang telah dibangun.
4. Melakukan *mapping* terhadap data hasil enkripsi sesuai dengan *key-value* yang ditetapkan.
5. Melakukan *reducing* terhadap data yang telah di-*map* dengan operasi yang ditentukan.
6. Mendekripsi dan men-*decode* data hasil *reduce* dengan kunci privat yang telah dibangun kemudian menampilkannya untuk mengetahui hasil dari proses MapReduce.

A. Data Input

Data masukan yang akan digunakan dalam implementasi ini akan disimpan dalam sebuah file .txt. Data yang disimpan dalam file .txt adalah [namaOrang, jumlahUang]. Isi dari file .txt yang akan digunakan sebagai *testcase* dasar adalah:

```
budi,1000
budi,2000
andi,4000
budi,1000
andi,2000
cika,5000
cika,1000
andi,1500
```

Nantinya akan dilakukan beberapa modifikasi terhadap data masukan untuk melakukan analisis terhadap proses enkripsi homomorfik pada MapReduce. File .txt akan dibaca per baris kemudian dipisah setiap barisnya dengan *delimiter* ‘,’. Hasil dari pemisahan dengan *delimiter* akan disimpan dalam sebuah *array*.

B. Key Generation

Dengan bantuan library Tenseal, kunci publik dan kunci privat dapat dibangkitkan sesuai dengan parameter yang ditentukan. Kunci publik dan kunci privat kemudian disimpan untuk melakukan enkripsi dan dekripsi terhadap *plaintext* dan *ciphertext*.

C. Encoding and Encryption

Pada skema CKKS, proses *encoding* dan *encryption* merupakan 2 proses yang terpisah, akan tetapi library Tenseal menggabungkan kedua proses tersebut menjadi 1 fungsi enkripsi. Fungsi akan menerima input berupa *array* bilangan riil dan mengubahnya menjadi *ciphertext*. *Ciphertext* yang dihasilkan dengan menggunakan fungsi enkripsi Tenseal merupakan dalam bentuk *object* CKKSVector. Apabila data ingin disimpan, maka fungsi serialize() harus dilakukan terhadap *object* kemudian men-*encode* hasilnya menjadi bytes dengan base64.

Untuk setiap baris data yang dibaca dan dipisah dengan *delimiter*, akan dienkripsi nilai dari jumlahUang dengan menggunakan kunci publik yang telah dibangun.

D. *Mapper*

Proses *mapping* akan dilakukan terhadap data yang telah dibaca dari file .txt dan telah dienkripsi. *Key* yang dipilih adalah namaOrang dan *value*-nya adalah nilai dari jumlahUang yang telah dienkripsi. Setiap pasangan *key-value* akan disimpan dalam sebuah *array* yang nantinya akan di-*reduce*.

Pseudocode dari 4 tahap diatas adalah sebagai berikut:

```
publicKey, privateKey =
generate_CKKS_Key(CKKS parameter)

data = readFile(inputFile)
mapperResult = []

for item in data:
   line = item.split(‘,’)
   enc = encryptCKKS(line[1], publicKey)
   mapperResult.append([line[0],enc])
```

E. *Reducer*

Proses *reducing* dilakukan terhadap *array* yang telah didapat dari hasil *mapping*. Dilakukan iterasi terhadap setiap elemen dari *array* tersebut kemudian akan dilakukan agregasi, yaitu proses penjumlahan atau perkalian, terhadap *value* dari pasangan *key-value* sesuai dengan *key*-nya. Nilai dari agregasi akan disimpan dalam sebuah *dictionary*. Bila *key* tidak terdapat di dalam *dictionary*, maka akan ditambahkan ke dalam *dictionary* pasangan *key-value* tersebut, bila *key* terdapat di dalam *dictionary*, maka *value* dari *key* di *dictionary* akan ditambah/dikali dengan nilai yang baru.

F. *Decryption and decoding*

Untuk menampilkan hasil dari proses MapReduce, akan dilakukan *decryption* dan *decoding*. Sama seperti proses enkripsi dan *encoding*, library Tenseal menggabungkan proses dekripsi dan *decoding* ke 1 fungsi. Proses dekripsi membutuhkan kunci privat yang telah dibangun. Proses dekripsi dan *encoding* akan dilakukan terhadap setiap *value* yang ada di *dictionary* hasil *reduce*.

Pseudocode dari 2 tahap diatas dengan proses agregasi penjumlahan adalah sebagai berikut:

```
res = {}
for item in mapperResult:
   if item[0] in res:
      res(item[0])= res.get(item[0]) +
item[1]
   else:
      res[item[0]] = item[1]

for key in res:
   res[key] = decryptCKKS(res.get(key),
privateKey)
```

IV. Pengujian dan Analisis

Terhadap data input yang telah telah dicantumkan pada bagian 3, akan dilakukan MapReduce untuk melihat jumlah total uang yang dimiliki oleh setiap orang.

```
budi [3999.9999999988213]
andi [7499.999999999223]
cika [6000.000000001068]
Time taken: 0.4028482437133789
```

Gambar 4.1 Hasil MapReduce file input operasi penjumlahan

Dengan hasil yang diharapkan sebagai berikut:

Nama Orang	Hasil
Budi	4000
Andi	7500
Cika	6000

Untuk menguji performa MapReduce seiring bertambahnya jumlah data yang harus diproses, akan digunakan data sebagai berikut:

```
budi,1000
andi,2000
cika,3000
dodi,4000
```

Data tersebut akan diduplikasi sebanyak 5 kali lipat (20 data), 25 kali lipat (100 data), dan 100 kali lipat (400 data). Berikut hasil dari percobaan tersebut:

Jumlah Data	Hasil
20	budi [5000.00000000013] andi [10000.000000000222] cika [15000.000000001195] dodi [19999.999999997657] Time taken: 0.4859600067138672
100	budi [25000.00000000338] andi [50000.00000000884] cika [75000.00000000275] dodi [99999.99999998708] Time taken: 0.8738617897033691

400	budi [99999.99999998446] andi [199999.99999998743] cika [299999.9999999765] dodi [399999.9999999965] Time taken: 2.45361328125

Dengan hasil yang diharapkan sebagai berikut:

Jumlah Data	Nama Orang	Hasil
20	Budi	5000
	Andi	10000
	Cika	15000
	Dodi	20000
100	Budi	25000
	Andi	50000
	Cika	75000
	Dodi	100000
400	Budi	100000
	Andi	200000
	Cika	300000
	Dodi	400000

Akan dilakukan juga proses MapReduce dengan operasi perkalian terhadap data yang dicantum pada bagian 3.

```
budi [367177.32372589124]
andi [105911.93883965859]
cika [5000000.670549649]
Time taken: 0.42597508430480957
```

Gambar 4.2 Hasil MapReduce file input operasi perkalian (1)

Dengan hasil yang diharapkan sebagai berikut:

Nama Orang	Hasil
Budi	2×10^9
Andi	12×10^9
Cika	5×10^6

Akan dilakukan percobaan ulang dengan jumlah uang yang lebih kecil sebagai berikut:

```
budi,3
budi,2
andi,4
budi,4
andi,2
cika,5
cika,1
andi,4
```

```
budi [24.000019320789256]
andi [32.00002573157523]
cika [5.000000677141462]
Time taken: 0.40552449226379395
```

Gambar 4.3 Hasil MapReduce file input operasi perkalian (2)

Dengan hasil yang diharapkan sebagai berikut:

Nama Orang	Hasil
Budi	24
Andi	32
Cika	5

Berdasarkan hasil pengujian yang telah dilakukan, proses MapReduce dengan operasi penjumlahan secara konsisten menghasilkan nilai yang diinginkan. Skema CKKS menggunakan operasi aritmatik aproksimasi, sehingga nilai yang didapat dari proses dekripsi dan *decoding* akan mengandung eror. Apabila nilai eror cukup kecil, maka pembulatan yang baik terhadap hasil dekripsi dan *decoding* akan menghasilkan nilai yang diharapkan. Waktu yang diperlukan untuk menjalankan MapReduce dengan operasi penjumlahan dapat dibilang cukup baik, bahkan dengan bertambahnya jumlah data yang diproses. Dengan ini, dapat dibilang bahwa skema CKKS dalam proses MapReduce dengan operasi penjumlahan dapat diimplementasikan dengan baik.

Proses perkalian dengan CKKS masih belum menghasilkan nilai yang diharapkan, walaupun waktu yang diperlukan untuk melakukan operasi cukup cepat. Hasil MapReduce dengan operasi perkalian akan menghasilkan nilai yang sesuai apabila angka yang dilakukan operasi kali tidak terlalu besar. Apabila angka yang dikalikan terlalu besar, maka hasil yang didapat mungkin tidak sesuai. Apabila ditinjau gambar 4.2, maka hanya jumlah uang Cika yang sesuai harapan, jumlah uang Budi dan Andi tidak sesuai. Perbedaan utama dari proses MapReduce antara kedua kasus adalah operasi perkalian adalah jumlah perkalian yang dilakukan. Jumlah perkalian yang dilakukan untuk *key* Cika adalah 2 kali sedangkan untuk *key* Budi dan Andi adalah 3 kali.

Alasan dari ketidaktepatan hasil yang didapat dari operasi perkalian adalah bertambahnya *noise* pada *ciphertext* hasil perkalian. Apabila proses perkalian hanya melibatkan angka kecil atau terjadi satu kali saja pada 2 bilangan besar, maka hasil yang didapat akan sesuai karena *noise* tidak terlalu besar.

Bila kedua syarat tersebut tidak dipenuhi, maka *noise* pada *ciphertext* akan menjadi lebih besar yang menyebabkan ketidaktepatan hasil perkalian.

Untuk mengatasi hal tersebut, maka dapat dilakukan *bootstrapping*. Penggunaan metode tersebut akan mengatasi nilai *noise* yang semakin besar seiring bertambahnya jumlah operasi perkalian terhadap *ciphertext.* Alhasil, secara teoritis, proses perkalian dalam MapReduce dapat dilakukan dengan konstrain yang lebih bebas. Perlu diketahui bahwa untuk mengimplementasikan *bootstrapping*, perlu diimplementasikan skema CKKS secara manual tanpa menggunakan library Tenseal dan implementasi skema CKKS itu sendiri merupakan tugas yang cukup berat dan memakan cukup banyak waktu sehingga penulis tidak sempat untuk mengimplementasikan solusi ini.

Hal tersebut menyinari sebuah kelebihan dari pengimplementasian skema CKKS tanpa library Tenseal. Kelebihan lain dari pengimplementasian skema CKKS secara manual adalah kemampuan untuk menyesuaikan *ciphertext* sesuai dengan keinginan bagaimana *ciphertext* akan disimpan. *Ciphertext* yang dihasilkan oleh library Tenseal merupakan dalam bentuk obyek CKKSVector. Apabila *ciphertext* tersebut ingin disimpan, obyek tersebut harus diubah terlebih dahulu menjadi bytes (dalam jumlah yang cukup besar relatif terhadap polinom *ciphertext* yang dihasilkan) yang dapat disimpan dan dibaca ulang oleh program. Dengan implementasi CKKS secara manual, maka bentuk *ciphertext* dapat disimpan dalam bentuk yang bebas dan lebih efisien untuk disimpan dan dibaca oleh program, contohnya 2 *array* yang berisikan koefisien polinomial.

V. KESIMPULAN

Skema CKKS dapat diintegrasikan ke dalam MapReduce dengan baik untuk proses penjumlahan (baik menggunakan library Tenseal atau menggunakan cara yang lain). Nilai yang didapatkan dari hasil penjumlahan apabila dibulatkan dengan baik akan menghasilkan nilai yang tepat. Untuk operasi perkalian, hanya dapat dilakukan dengan konstrain tertentu apabila menggunakan library Tenseal.

Untuk dapat mengintegrasikan skema CKKS kedalam proses MapReduce dengan operasi perkalian yang lebih baik dan memiliki konstrain yang lebih bebas, disarankan untuk mengimplementasikan skema CKKS tanpa library Tenseal yang memiliki metode *bootstrapping*. Implementasi skema CKKS tanpa menggunakan library Tenseal dapat memberikan beberapa kelebihan lainnya. Salah satu kelebihannya adalah data *ciphertext* yang dapat disimpan dengan lebih baik dikarenakan bentuk *ciphertext* yang dapat disesuaikan dengan tempat penyimpanan.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa karena atas berkat dan karunia-Nya lah penulis dapat menyelesaikan makalah "*Homomorphic Encryption* dalam MapReduce di Teknologi *Big Data"* dengan tepat waktu. Penulis juga mengucapkan terima kasih kepada orang tua penulis yang selalu mendukung penulis selama masa perkuliahan penulis di ITB. Terakhir, penulis ingin mengucapkan terima kasih kepada bapak Rinaldi Munir yang telah membimbing penulis selama satu semester ini dalam menjalankan kelas IF4020 Kriptografi.

REFERENSI

[1] Munir, Rinaldi. (2023). *Bahan Kuliah IF4020 Kriptografi.* Program Studi Informatika ITB.

[2] Yi, X., Paulet, R., & Bertino, E. (2014). "Homomorphic Encryption and Applications". Spinger.

[3] Akbar, Saiful. (2023). *Bahan Kuliah IF4044 Teknologi Big Data*. Program Studi Informatika ITB.

[4] https://blog.openmined.org/ckks-explained-part-1-simple-encoding-and-decoding/

[5] https://sefiks.com/2023/04/10/a-step-by-step-fully-homomorphic-encryption-example-with-tenseal-in-python/

[6] https://github.com/OpenMined/TenSEAL/blob/main/tutorials/Tutorial%200%20-%20Getting%20Started.ipynb

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2023

Jevant Jedidia Augustine - 13520133