IMPLEMENTASI DELEGATED COMPUTATION PADA PUBLIC CLOUD MENGGUNAKAN ALGORITMA FULLY HOMOMORPHIC ENCRYPTION

Annisa Muzayana, Aisyatul Karima, S.Kom, M.Cs, *Fakultas Ilmu Komputer, Universitas Dian Nuswantoro*

Abstract - ***Today the use of cloud computing has become a trend among organizations and the wider community. it is because of the easiness way of sharing and saving economically. But the easiness of sharing is can cause a security problem on cloud computing. The possibility of data stolen or compromised may occur. In addition, security issues also arise from the provider side. Fully Homomorphic Encryption become one of the solutions where easiness way of data sharing and data security will be maintained without having to sacrifice anything. The issue became the basis of this research. By using the Delegation scheme Computation (delegating calculation) it can be very possible to happen. This research applies the scheme for the computation delegated may delegate the processing of data to a second party who has been granted access and possess the public key without knowing the original data. And Fully Homomorphic Encryption algorithm to enable the scheme can be used to process data by the second ciphertext given access and has a public key. So that the safety and comfort of data sharing is reached, and can be applied to daily life - today. However, the coverage required operating more like subtraction and division in order to be more effective data processing.***

***Keyword*—Cryptography, Homomorphisme, Encryption, Fully Homomorphic Encryption, Delegation Computation**

I. PENDAHULUAN

Cloud computing merupakan model komputasi yang memungkinkan usernya untuk menggunakan resource (networks, servers, storage, applications, dan services) yang ada dalam sebuah jaringan cloud (internet) sehingga dapat dibagi dan digunakan bersama. Secara ekonomis, penerapan cloud computing mampu menghemat pengeluaran karena, tidak perlu mengalokasikan untuk biaya hardware maupun software. Organisasi cukup menyewa sebuah layanan cloud computing pada penyedia layanan saja. Model pembayaran dapat berdasarkan pemakaian atau dengan berlangganan. Sehingga dengan penerapan teknologi cloud computing ini mampu menghemat pengeluaran organisasi. [1]. Namun dari sekian banyak kelebihan yang telah dijelaskan diatas, terdapat beberapa kelemahan. Salah satunya adalah dari sisi keamanan data.

Karena dasar cloud computing adalah share resource, ini dapat mengakibatkan keamanan data rawan dibobol [1]. Dalam praktiknya user cloud computing memiliki risiko yang mungkin dihadapi sebagai berikut:

• Provider penyedia jasa cloud computing mengalami kebangkrutan sehingga server berhenti bekerja dan data hilang lalu tidak dapat dipertanggungjawabkan provider.

• Pihak lain (yang tidak ada hubungannya dengan user) melakukan penggugatan pada provider jasa layanan cloud, yang kemudian memiliki hak akses kepada seluruh server cloud dan mengancam kerahasiaan data user.

• Kegagalan pihak penyedia layanan cloud dalam melakukan perawatan infrastruktur dan fisik akses kontrol. [2]

Dengan adanya masalah keamanan ini memunculkan kekhawatiran diantara para pengguna layanan cloud computing. Layanan cloud computing yang awalnya menguntungkan karena tidak membutuhkan biaya tambahan untuk penyimpanan data malah menjadi ancaman serius bagi penggunanya.

Keterkaitan antara kenyamanan dan privasi adalah pertanyaan dimana sekarang kita berada pada masa cloud computing sudah digunakan secara meluas. Menyimpan data pada cloud tanpa dienkripsi, bisa menimbulkan banyak resiko, termasuk pencurian data dan penyalahgunaan data. Untuk beberapa jenis data seperti catatan medis, menyimpannya pada cloud tanpa dienkripsi bisa disebut illegal.

Namun, mengenkripsi data pada cloud sama saja mengurangi manfaat dari cloud itu sendiri, kecuali memberikan kunci untuk mendekripsinya dengan mengorbankan privasi. Ini sama saja dengan mengambil kembali data yang ada pada cloud lalu didekripsi dan diproses sendiri. Namun, masalah kenyamanan dan privasi dapat diselesaikan ke pembahasan yang lebih luas. Untuk data yang dienkripsi dengan skema enkripsi biasa, secara

virtual tidak mungkin ada seseorang yang mampu memanipulasi data tersebut tanpa memiliki kunci dekripsinya. Akan tetapi, ada beberapa skema enkripsi yang homomorphic atau dapat dibentuk. Skema ini mengizinkan siapapun yang memiliki kunci publik untuk memanipulasi data yang terenkripsi, bahkan tanpa mengetahui kunci rahasianya. [3]

Dengan adanya skema yang homomorphic ini, pengguna dapat mempertahankan keamanan data mereka dan tetap nyaman untuk berbagi. Dalam hal ini berati dimungkinkan pihak ketiga yang diberi kewenangan untuk memiliki kunci publik dapat memproses data yang ter-enkripsi tanpa harus membukanya atau dapat disebut sebagai pendelegasian perhitungan terhadap data yang terenkripsi.

Sebelumnya terdapat penelitian [3] menggunakan algoritma Fully Homomorphic Encryption. Penelitian ini menghasilkan skema yang memungkinkan bagi pihak ketiga untuk dapat memanipulasi data yang ter-enkripsi tanpa harus mengetahui kunci atau membukanya.

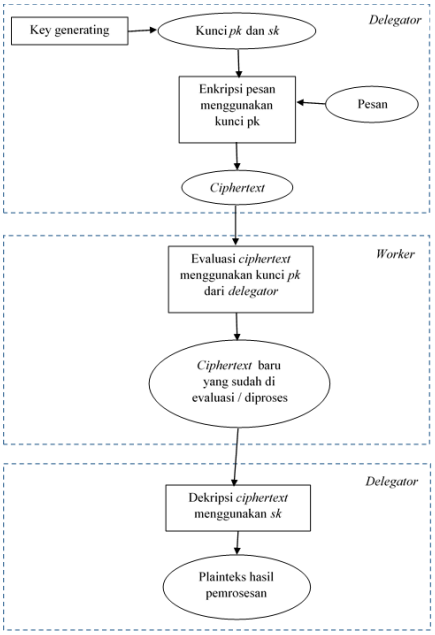
Penelitian ini akan menggunakan algoritma Fully Homomorphic Encryption sehingga dapat tercipta rancangan aplikasi yang aman dan nyaman bagi pengguna cloud computing untuk pendelegasian perhitungan data yang terenkripsi.

II. METODE YANG DIUSULKAN

Metode yang diusulkan dalam penelitian ini adalah dengan menerapkan algoritma Fully Homomorphic Encryption pada skema delegated computation "pipelined". Berikut ini tahapan yang dilaksanakan untuk menjalankan metode:

a. Menjalankan proses Keygen untuk mendapatkan satu pasang kunci public key (pk) dan secret key (sk),
b. Menjalankan proses Encrypt yang dilakukan oleh delegator dengan menggunakan pk dan plaintext untuk menghasilkan ciphertext,
c. Hasil ciphertext dan pk dikirimkan kepada worker, untuk melakukan proses terhadap data,
d. Menjalankan proses Evaluate yang dilakukan oleh worker, dengan menggunakan ciphertext dan pk, yang menghasilkan ciphertext yang sudah di proses / evaluate,
e. Delegator menerima kembali ciphertext yang sudah diproses,
f. Delegator menjalankan proses Decrypt dengan menggunakan c dan sk untuk mendapatkan hasil perhitungan berupa data asli.

Gambar 1 di bawah ini adalah gambaran metode yang diusulkan.



Gambar 1

III. IMPLEMENTASI

Dalam penelitian ini menerapkan algoritma Fully Homomorphic Encryption dengan skema somehow homomorphic encryption, dimana dalam algoritmanya terdapat empat proses yaitu, proses pembangkitan kunci, enkripsi pesan, evaluasi, dan dekripsi pesan.

Parameter yang digunakan adalah polynomial dalam λ (parameter keamanan):

a. γ : panjang bit integer kunci publik,
b. η : panjang bit integer kunci rahasia,
c. ρ : panjang bit integer *noise* (jarak antara elemen dari kunci public dan kelipatan terdekat dari kunci rahasia), dan
d. τ : jumlah integer dalam kunci publik.

Untuk memenuhi batasan batasan agar skema ini aman, saran yang diberikan oleh [6] dalam pemilihan parameter adalah $\rho = \lambda,\ \rho' = 2\lambda,\ \eta = O(\lambda^2),\ \gamma = O(\lambda^5)$, dan $\tau = \gamma + \lambda$. Untuk sebuah (η-bit) integer ganjil *p*, digunakan distribusi berikut untuk γ-bit integer :

$\mathcal{D}_{\gamma,\rho}(p) = \{$pilih $\overset{R}{\leftarrow} \mathbb{Z} \cap [0, 2^{\gamma}/p), r \overset{R}{\leftarrow} \mathbb{Z} \cap (-2^{\rho}, 2^{\rho})$: output $x = pq + r\}$

3.1 Analisis pembangkitan kunci

Pada bagian, ini peneliti akan menjelaskan proses pembangkitan kunci pada algoritma Fully Homomorphic Encryption. Disini dimisalkan menggunakan variabel global sebagai security parameter,

λ = 2
ρ = 2
γ = 32
η = 4
τ = 34

Dan memakai konstruksi sebagai berikut :

Keygen(λ)

Kunci rahasia, sk adalah integer ganjil dengan panjang bit η : $p \xleftarrow{R} (2\mathbb{Z}+1) \cap [0, 2^{\eta-1}, 2^{\eta})$. Pilih $x_i \xleftarrow{R} \mathcal{D}_{\gamma,\rho}(p)$ untuk i = 0,…,τ. Ganti nama variabel sehingga x_0 adalah yang paling besar. Ulangi proses kecuali x_0 ganjil dan $r_p(x_0)$ genap. Kunci public adalah pk = $\langle x_0, \dots, x_\tau \rangle$.

Berikut tahapan pembangkitan kunci :

1. Kunci rahasia didapatkan dengan mengacak angka sebanyak 4bit dengan range $2^{\eta-1}, 2^{\eta}$ selama kunci rahasia genap maka pengacakan angka akan terus dilakukan sampai didapat sk (kunci rahasia) sebagai berikut :

   sk : 15
2. Untuk membuat kunci publik yang merupakan hasil dari memilih $x_i \xleftarrow{R} \mathcal{D}_{\gamma,\rho}(p)$ untuk i = 0,…,τ. Dimana $\mathcal{D}_{\gamma,\rho}(p)$ adalah $\mathcal{D}_{\gamma,\rho}(p)$ = {pilih $q \xleftarrow{R} \mathbb{Z} \cap [0, 2^{\gamma}/p), r \xleftarrow{R} \mathbb{Z} \cap (-2^{\rho}, 2^{\rho})$: output $x = pq + r$} dimana p adalah *sk*. Dan proses diulang sampai τ. Berikut proses pembangkitannya :
   1. menentukan q yang nilainya dirandom dari 0 - $2^{\gamma}/p$ ditemukan q : 91128969,
   2. menentukan q yang nilainya dirandom dari $-2^{\rho}, 2^{\rho}$ ditemukan r : -2,
   3. lalu menentukan x yang merupakan hasil perkalian antara sk, q yang kemudian dijumlahkan dengan r,
      x = sk * q + r
      = 15 * 91128969 + (-2)
      = 1366934533
   4. hasil dari x, diulang sampai τ untuk menghasilkan pk. berikut pk.
      pk : [1366934533, 1366934533, 1366934533, 1366934533,1366934533,1366934533, 1366934533,1366934533,1366934533, 1366934533,1366934533,1366934533, 1366934533,1366934533,1366934533, 1366934533,1366934533,1366934533, 1366934533,1366934533,1366934533, 1366934533,1366934533,1366934533, 1366934533,1366934533,1366934533, 1366934533,1366934533,1366934533, 1366934533,1366934533,1366934533, 1366934533]

Dari proses pembangkitan kunci diatas didapatkan satu pasang kunci yaitu kunci rahasia (sk) dengan panjang 64bit (η-bit) panjang sk ini didapatkan dari $\eta = O(\lambda^2)$, dimana nilai λ adalah 2. Dan didapat kunci publik (sk) dengan panjang 4bit. Panjang pk ini didapatkan dari $\tau = \gamma + \lambda$, dimana nilai γ = 32 dan λ = 2. Panjang bit dari kedua kunci sudah memenuhi batasan – batasan dari [6] agar skema ini aman.

3.2 Analisis Enkripsi Pesan

Pada bagian, ini peneliti akan menjelaskan proses enkripsi pada algoritma Fully Homomorphic Encryption. Dalam melakukan enkripsi sebuah bit, diberikan m = 1. Dan memakai konstruksi berikut :

Encrypt(*pk*,$m \in \{0,1\}$)

Pilih sebuah subset acak $S \subseteq \{1,2, \dots \tau\}$ dan sebuah integer acak $r \in (-2^{p'}, 2^{p'})$.

Output $c \leftarrow [m + 2r + 2\sum_{i \in S} x_i]_{x_0}$

Berikut adalah proses enkripsi :

1. memilih sebuah subset acak S, ditemukan nilai S = 3
2. menentukan r yang nilainya dirandom dengan range $-2^{p'}, 2^{p'}$. Ditemukan nilai r = 1.
3. menghitung c :

c = $[m + 2r + 2\sum_{i \in S} x_i]_{x_0}$

c = (1 + 2*1 + 2*1366934533) mod 1366934533

c = -13L

Dari proses enkripsi diatas didapatkan satu *ciphertext* berbentuk array 8 item, semua *ciphertext* juga akan berisi 8 array karena dibatasi 8 item. Dimana r adalah bilangan integer yang mempunyai panjang $(-2^{p'}, 2^{p'})$. Kunci publik yang diambil dari subset adalah acak. Setelah itu pemrosesan dilakukan dengan menambahkan *plaintext* dengan perkalian 2 dan r, juga dengan perkalian 2 dan subset pk. Hasil dari penjumlahan di modulus 2.

3.3 Analisis Evaluasi

Pada bagian, ini peneliti akan menjelaskan proses enkripsi pada algoritma Fully Homomorphic Encryption. Berikut konstruksi evaluasi:

Evaluate(*pk*, C, $c_1,\dots,c_t$)

Untuk sirkuit binary *C* dengan *t* buah input, *t* buah ciphertext, lakukan operasi penjumlahan dan perkalian bilangan bulat sesuai dengan gerbang – gerbang sirkuit pada *C*. Semua operasi dilakukan dalam integer dan menghasilkan integer.

Dalam melakukan evaluasi sebuah bit terdapat dua operasi yaitu penjumlahan dan perkalian. Dalam aplikasi ini hanya terdapat operasi penjumlahanyang mewakili gerbang logika XOR dalam bilangan bulat. Untuk melakukan evaluasi hanya perlu dilakukan penghitungan nilai $c_1 + c_2$. Berikut proses evaluasi :

$c_1 + c_2$ = [-13L + -13L]

= [-26L]

3.4 Analisis Dekripsi

Pada bagian, ini peneliti akan menjelaskan proses dekripsi pada algoritma Fully Homomorphic Encryption. Dan menggunakan konstruksi sebagai berikut :

Decrypt(*sk*, *c*)

Ouput $m' \leftarrow (c \bmod p) \bmod 2$

Dalam melakukan dekripisi sebuah bit, ciphertext di modulus dengan kunci rahasia, sisa hasil baginya di modulus lagi dengan 2. Berikut proses dekripsi :

$m = (c \bmod sk) \bmod 2$
$= [-13] \bmod 11) \bmod 2$
$= 1$
$c_1 + c_2 = [-26L] \bmod 11) \bmod 2$
$= 2$

Dari proses dekripsi diatas, *c* (*ciphertext*) yang sebelumnya diproses dimodulus dengan *sk* (kunci rahasia). Sisa hasil baginya dimodulus dengan 2, dimana hasil dekripsinya adalah 1 atau *plaintext* dari *c* yang dienkripsi sebelumnya. Begitu juga hasil evaluasi $c_1 + c_2$ (*ciphertext*$_1$ dijumlahkan dengan *ciphertext*$_2$) yang hasilnya adalah 2 dimana hasilnya sama dengan hasil dari penjumlahan *plaintext* sebelum dienkripsi.

IV. HASIL PENELITIAN

Dalam penelitian ini memberikan hasil yang sesuai dengan pengujian yang menggunakan metode pengujian keberhasilan yaitu tingkat keberhasilan enkripsi – dekripsi dan enkripsi – evaluasi – dekripsi. Proses menggunakan satu pasang kunci dan 20 pesan. Sehingga dapat diketahui tingkat keberhasilannya dari 0% - 100%.

Syarat keberhasilan apabila,

1. Plaintext yang dienkripsi dengan kunci publik hasilnya sama ketika didekripsi dengan kunci rahasia.
2. Dua plaintext yang dienkripsi lalu dievaluasi hasilnya sama dengan jumlah kedua plaintext ketika didekripsi.

Pengujian dilakukan dengan dua skenario, skenario yang pertama untuk menguji syarat keberhasilan hasil dekripsi sama dengan plaintext yang belum dienkripsi. Berikut skenario pengujiannya :

1. sk, pk ← keygen(), membuat kunci rahasia (sk) dan kunci publik (pk)
2. m_i ← pesan berupa angka dan karakter yang dipakai(lihat lampiran 3) untuk i = 1, 2, …, 10,
3. c_i ← encrypt(pk, m_i), melakukan enkripsi per data sebanyak 10 kali untuk 10 pesan
4. d_i ← decrypt(sk, m_i), melakukan dekripsi per hasil enkripsi sebanyak 10 kali untuk 10 ciphertext

Berikut hasil pengujian skenario pertama :

Tabel 1

Percobaan	Data Pengujian	Hasil pengujian	
		Enkripsi	Dekripsi
1.	‘ab’	berhasil	berhasil
2.	‘saya’	berhasil	berhasil
3.	‘ti’	berhasil	berhasil
4.	‘fik’	berhasil	berhasil
5.	‘udinus’	berhasil	berhasil
6.	175	berhasil	berhasil
7.	500	berhasil	berhasil
8.	400	berhasil	berhasil
9.	190	berhasil	berhasil
10.	2000	berhasil	berhasil

Skenario yang kedua untuk menguji syarat keberhasilan yang kedua untuk menguji hasil dekripsi sama dengan hasil penjumlahan plaintext sebelum dienkripsi. Berikut skenario pengujiannya :

1. sk, pk ← keygen(), membuat kunci rahasia (sk) dan kunci publik (pk) (lihat lampiran 2),
2. m_i ← pesan berupa angka yang dipakai(lihat lampiran 4) untuk i = 1, 2, …, 10
3. c_i ← encrypt(pk, m_i), melakukan enkripsi per data sebanyak 10 kali untuk 10 pesan
4. d_i ← decrypt(sk, m_i), melakukan dekripsi per hasil enkripsi sebanyak 10 kali untuk 10 ciphertext.
5. jumlah ← jumlahInt(c1, c2), melakukan evaluasi operasi penjumlahan c_1 dan c_2, c_3 dan c_4, c_5 dan c_6, c_7 dan c_8, c_9 dan c_{10}
6. Melakukan dekripsi terhadap jumlah.

Berikut hasil pengujian skenario kedua :

Tabel 2

Percobaan	Data Pengujian	Hasil Pengujian		
		Enkripsi	Evaluasi	Dekripsi
1.	100	berhasil	berhasil	berhasil
2.	70	berhasil		berhasil
3.	80	berhasil	berhasil	berhasil
4.	85	berhasil		berhasil
5.	65	berhasil	berhasil	berhasil
6.	95	berhasil		berhasil
7.	85	berhasil	berhasil	berhasil
8.	90	berhasil		berhasil
9.	75	berhasil	berhasil	berhasil
10.	65	berhasil		berhasil

Setelah melalui pengujian, presentase yang didapatkan dari 20 pesan data tersebut semuanya memenuhi semua syarat keberhasilan. Sehingga aplikasi mencapai keberhasilan 100% dan tidak ditemukan error pada masing-masing plaintext maupun ciphertext pada proses

V. PENUTUP

Peneliti dapat membangun aplikasi yang memungkinkan bagi pihak ketiga yang diberi kewenangan untuk memiliki kunci publik untuk dapat memanipulasi data yang ter-enkripsi tanpa harus mengetahui kunci atau membukanya dengan menerapkan algoritma *Fully Homomorphic Encryption.*

Dalam penelitian ini, dibuktikan bahwa implementasi *Fully Homomorphic Encryption* dapat diterapkan pada penggunaan sehari hari, dalam hal ini pemrosesan nilai.

Namun, dalam penelitian ini masih terdapat kekurangan yaitu operasi prosesnya hanya menggunakan penjumlahan. Sangat disarankan untuk dikembangkan ke operasi matematika yang lain, seperti pengurangan, perkalian atau pembagian.

REFERENCES

[1] O. N. Pratiwi, "ANALISIS KEAMANAN APLIKASI PENYIMPANAN DATA PADA SISTEM CLOUD COMPUTING," *e-Indonesia Initiative 2011 : Konferensi Teknologi Informasi dan Komunikasi untuk Indonesia*, pp. 137-139, 2011.

[2] G. Reese, "Cloud Application Architectures : Building Applications and Infrastructure in the Cloud," 2009.

[3] C. Gentry, "Computing arbitrary functions of encrypted data," *Magazine Communications of the ACM*, vol. 53, no. 3, pp. 97-105, 2010.

[4] C. Gentry, "Fully Homomorphic Encryption Using Ideal Lattices," *STOC '09 Proceedings of the forty-first annual ACM symposium on Theory of computing*, pp. 169-178, 2009.

[5] C. Gentry, "A fully homomorphic encryption scheme," Stanford University, 2009.

[6] M. v. Dijk, C. Gentry, S. Halevi and V. Vaikuntanathan, "Fully Homomorphic Encryption over the Integers," *EUROCRYPT'10 Proceedings of the 29th Annual international conference on Theory and Applications of Cryptographic Techniques*, pp. 1-28, 2010.

[7] I. Sofana, Cloud Computing Teori dan Praktik (OpenNebula, VMware, dan Amazon AWS), Bandung: Informatika Bandung, 2012.

[8] D. Ariyus, Pengantar Ilmu Kriptografi, Teori, Analisis, dan Implementasi, Yogyakarta: ANDI, 2008.

[9] A. R. Anggoro, "Studi Mengenai Fully Homomorphic Encryption dan Perkembangannya dari RSA sebagai Enkripsi Homomorfis Populer," 2010.

[10] P. K. Setia, "Skema Fully Homomorphic Encryption Gentry versi Integer dan impelementasinya dengan menggunakan bahasa pemrograman phyton," 2013.

[11] K.-M. Chung, Y. Kalai and S. P. Sadhan, "Improved Delegation of Computation using Fully Homomorphic Encryption," *CRYPTO'10 Proceedings of the 30th annual conference on Advances in cryptology*, pp. 483-501, 2010.

[12] P. K. Setia, "Skema Fully Homomorphic Gentry versi Integer dan implementasi pada bahasa pemrograman python versi 1 [Computer program]," 2013. [Online]. Available: https://github.com/hesahesa/IntegerFHE-Gentry. [Accessed 14 October 2014].