Application Analysis of Advanced Encryption Standard (AES-256) used by Google

Brian Albar Hadian - 13523048[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]*bri.hadian@gmail.com*, *13523048@std.stei.itb.ac.id*

***Abstract*—Dengan semakin pentingnya informasi dan moda distribusi informasi melalui media digital, diperlukan cara untuk melindungi informasi yang diimplementasikan melalui penerapan algoritma kriptografi. Salah satu sumber informasi dan penyedia jasa yang sering digunakan adalah Google yang menerapkan algoritma kriptografi *Advanced Encryption Standard* (AES-256) sebagai standar perlindungan dalam mengolah data. Melalui analisis lanjut dan penyederhanaan skema terhadap proses kriptografi, algoritma AES yang diterapkan Google memiliki beragam kelebihan yang patut diakui, yakni tingkat keamanan yang tinggi dan durasi generasi yang cukup rendah. Namun, terdapat kekurangan pada algoritma AES-256, diantaranya memiliki tingkat kerumitan algoritma yang tinggi dan membutuhkan daya komputasi yang cukup kuat dalam mengolah data dan struktur data.**

***Keywords*—AES-256-GCM, Algoritma, Enkripsi, SHA-256, Informasi.**

I. Introduction

Seiring berjalannya waktu, manusia telah mengembangkan berbagai hal untuk mempermudah kehidupannya. Beberapa contoh adalah teknologi dan ilmu sains. Kedua hal ini merupakan hal yang berkaitan erat karena memiliki peran yang signifikan pada kehidupan manusia. Teknologi merupakan terapan dari ilmu sains dan ilmu sains adalah dasar untuk mengembangkan cabang ilmu lainnya. Dampak dari kedua hal ini diantaranya adalah meningkatnya produktivitas manusia, berkurangnya waktu yang diperlukan untuk berpindah dari satu tempat ke tempat lainnya, dan memperluas kemampuan manusia untuk saling berbagi dan mengembangkan informasi.

Perkembangan teknologi tidak terlepas dari perkembangan metode distribusi dan informasi itu sendiri. Teknologi dapat dikembangkan karena informasi terkait teknologi dan ilmu-ilmu lain dapat disebarkan dengan baik. Seiring berkembangnya peradaban manusia, dapat dilihat bahwa informasi memiliki pengaruh yang semakin signifikan bagi manusia hingga pada tahap informasi dapat diperlakukan sebagai komoditas. Salah satu contoh adalah penerapan metode pembelajaran daring yang memanfaatkan metode distribusi informasi melalui internet sehingga pengguna dapat mengakses informasi yang diinginkan kapanpun. Dengan begitu, tidak dapat dipungkiri bahwa informasi merupakan hal yang signifikan dalam kehidupan manusia.

Namun, dengan berkembangnya berbagai kepentingan oleh berbagai pihak, situasi tersebut dapat mendorong manusia untuk memanfaatkan informasi tersebut sebagai alat yang dapat mengakibatkan bahaya bagi kehidupan masyarakat. Hal ini dapat dilihat dari adanya usaha manipulasi informasi oleh pihak media pada saat keadaan genting ataupun pengembangan senjata untuk meningkatkan daya rusak bagi lawan suatu pihak. Dengan adanya contoh-contoh tersebut, informasi dapat mengakibatkan hal yang tidak diinginkan apabila informasi tersebut dimiliki oleh pihak yang salah. Dengan begitu, diperlukan adanya usaha untuk melindungi informasi secara terbuka.

Perlindungan informasi dapat diterapkan menggunakan berbagai cara, salah satu contohnya adalah menjaga kerahasiaan informasi dan melindungi sarana distribusi informasi. Menjaga kerahasiaan suatu informasi biasa dilakukan dengan melakukan usaha kriptografi terhadap pesan tersebut dan memastikan bahwa pesan tersebut tidak mengalami perubahan pada saat dipindahtangankan. Sementara itu, melindungi sarana distribusi informasi dapat dilakukan dengan cara membuat suatu sarana distribusi khusus antar pihak yang dikehendaki. Diantara kedua konsep sebelumnya, penerapan kriptografi dalam menjaga kerahasiaan informasi lebih sering digunakan pada kehidupan sehari-hari. Beberapa contoh pihak yang menggunakan adalah perusahaan teknologi dan informasi (Google, Yahoo, Netflix, dsb), Lembaga negara, dll.

Terkait dengan pentingnya informasi dalam kehidupan sehari-hari, Google merupakan salah satu mesin pencari informasi yang paling sering digunakan oleh manusia. Google dikembangkan dari tahun 1998 oleh Sergey Brin dan Larry Page untuk menjadi salah satu alat yang dapat membantu kehidupan manusia dalam mengelola dan memanfaatkan informasi. Hingga saat ini, dampak dari adanya Google sangat terlihat di kehidupan sehari-hari, dimulai dari penggunaan dalam lingkungan belajar hingga saran untuk mencari permainan bagi pihak-pihak tertentu. Di zaman modern seperti ini, dapat dinyatakan bahwa kebutuhan terkait informasi semakin meningkat yang dibantu dengan adanya Google.

Karena Google merupakan salah satu alat yang dapat mengelola informasi, maka Google memerlukan cara untuk memastikanbahwa informasi yang dikelola merupakan informasi yang utuh dan hanya dapat dilihat oleh pihak-pihak tertentu. Salah satu cara Google melakukan hal tersebut adalah dengan melakukan usaha kriptografi terhadap informasi yang diberikan oleh pengguna. Beberapa metode kriptografi yang digunakan oleh Google diantaranya adalah Secure Hash Algorithm (SHA-256) dan Advanced Encryption Standard in

Galois Counter Mode (AES – 256 – GCM). Metode ini biasa digunakan pada data yang disimpan oleh pihak Google pada suatu infrastruktur khas yang berperan untuk mengelola seluruh data yang dikirimkan pada Google. Terdapat dua usaha pengamanan informasi oleh Google : Encryption at Rest dan Encryption in Transit

Kriptografi merupakan suatu cabang ilmu yang mempelajari bagaimana melakukan penyimpanan informasi secara baik dan aman. Tujuan dari kriptografi adalah melakukan perubahan pada informasi sedemikian sehingga informasi tersebut tidak adapat digunakan secara langsung oleh pihak yang tidak dikehendaki dan dibutuhkan suatu pemahaman tertentu untuk memperoleh informasi yang sebenarnya. Kriptografi pertama kali digunakan pada masa Mesir Kuno (± 2000 Tahun yang lalu) sebagai cara untuk menyampaikan pesan antar monarki secara rahasia. Salah satu metode kriptografi yang terkenal dalam penggunaannya adalah Caesar Cipher yang digunakan oleh Kaisar Romawi Julius Caesar (100 – 44 SM) dalam memerintah kekaisaran romawi dan juga digunakan pada masa Revolusi Perancis oleh Jean-Baptiste de Saint-Just untuk memerintahkan rencana pembunuhan terhadap salah satu keluarga bangsawan Perancis, tetapi pesan tersebut berhasil dipecahkan oleh salah satu anggota monarki dan berakhir tragis bagi Jean-Baptiste. Dengan demikian, kriptografi merupakan ilmu praktis yang dapat diterapkan untuk menjaga kerahasiaan informasi.

II. Landasan Teori

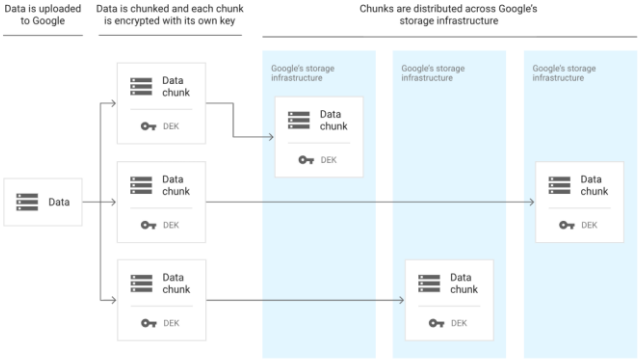
A. Skema Enkripsi Google

Google memiliki menerapkan kriptografi dalam setiap informasi yang dimilikinya. Setiap pengguna layanan Google memiliki metode autentikasi sedemikian sehingga informasi yang diberikan pada pihak Google hanya dapat diakses oleh pihak yang memiliki akses. Pada penerapannya, Google memiliki tiga prinsip dalam menjaga kerahasiaan informasi yang ada, diantaranya adalah Authentication, Integrity, dan Encryption. Aspek authentication memastikan bahwa informasi yang dimiliki oleh pengguna tertentu hanya dapat diakses oleh pengguna lain yang memiliki akses yang sama dan tidak ada tumpeng tindih antar akses sedemikian sehingga terdapat informasi yang tersebar secara tidak sengaja. Aspek Integrity memastikan bahwa informasi yang diberikan oleh pengguna disimpan dan diakses tanpa adanya perubahan pada isi maupun struktur dari informasi tersebut. Terakhir, aspek Encryption memastikan bahwa informasi yang dimiliki dan digunakan oleh pengguna merupakan informasi yang tidak dapat dimengerti maupun dimanipulasi oleh pihak yang tidak dikehendaki.

Google memiliki dua mode pengamanan informasi yang bergantung pada tipe informasi : Encryption at Rest dan Encryption in Transit. Encryption at Rest merupakan usaha enkripsi informasi terhadap informasi yang terdapat pada infrastruktur Google dan sedang digunakan dalam melakukan suatu proses / layanan. Sementara itu, Encryption in Transit merupakan usaha enkripsi informasi terhadap informasi pengguna yang sedang berpindah dari satu infrastruktur ke tujuan akhir. Tujuan akhir tersebut dapat berupa infrastruktur lain dari Google ataupun pengguna lain sehingga diperlukan perlindungan terhadap informasi agar tidak dapat disalahgunakan oleh pengguna lain dalam skema man-in-the-middle.

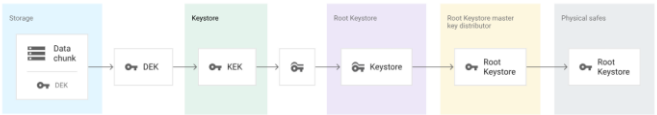
B. Encryption at Rest

Encryption at Rest melindungi data pengguna yang berada pada infrastruktur Google. Metode ini memanfaatkan penerapan algoritma kriptografi Advanced Encryption Standard (AES – 256) dan Hash Message Authentication Code – Secure Hash Algorithm (HMAC – SHA – 256). Metode memiliki skema utama sebagai berikut



Untuk setiap data yang diberikan oleh pengguna, data akan dipecah menjadi beberapa bagian sesuai dengan jumlah partisi tertentu. Kemudian, setiap partisi data akan memiliki kunci khusus yang berkaitan dengan metode kriptografi yang digunakan untuk menjaga informasi tersebut. Setiap partisi data memiliki kunci yang berbeda sehingga kunci tidak dapat diduplikasi dan digunakan untuk partisi data yang lain. Selanjutnya, setiap data dipastikan tersebar secara acak pada infrastruktur penyimpanan informasi yang dapat diperoleh kembali oleh suatu algoritma manajemen informasi.

Selanjutnya, untuk setiap infastruktur penyimpanan data, Google menerapkan kebijakan pengamanan berlapis, yakni setiap infrastruktur memiliki kunci tersendiri untuk mengakses informasi yang ada pada infrastruktur tersebut. Kunci ini kemudian dikelola oleh layanan manajemen yang disebut Keystore. Keystore juga merupakan suatu infrakstruktur Google sehingga ia juga memiliki suatu kunci khusus untuk mengakses kunci-kunci pada infrastruktur penyimpanan data. Selanjutnya, proses ini dilakukan secara berlapis hingga terdapat lima infrastruktur yang melakukan manajemen terkait informasi kunci yang dimiliki, diantaranya Root Keystore, Root Keystore master key distributor, dan infrastruktur fisik untuk menyimpan informasi kunci utama. Proses tersebut digambarkan melalui skema dibawah.



C. AES-256 – GCM

Advanced Encryption Standard merupakan suatu algoritma enkripsi yang digunakan dalam penyandian informasi. AES-256 merupakan metode AES khusus yang menangani data dalam format 256-bit. Dalam penjelasan berikut, akan dilakukan penyederhanaan terkait ilustrasi untuk mempermudah penggambaran skema dan algoritma. Algoritma AES-256-GCM untuk suatu pesan *m* dinyatakan sebagai berikut.

1. Nyatakan pesan *m* tersebut dalam suatu matriks 16x16 yang diurutkan mengurut mengikuti kolom

m = b0b1b2b3b4b5b6….b255

b_0	$b_{...}$	$b_{..}$	$b_{..}$
b_1	$b_{..}$	$b_{..}$	$b_{..}$
b_2	$b_{..}$	$b_{..}$	$b_{..}$
$b_{..}$	$b_{..}$	$b_{..}$	b_{255}

2. Tentukan kunci k

$k \in \mathbf{R}$, k adalah bilangan yang besar

Kunci k ini selanjutnya akan digunakan untuk melakukan operasi XOR pada setiap byte pada tabel sebelumnya

3. Tentukan suatu tabel substitusi

u_0	$u_{...}$	$u_{..}$	$u_{..}$
u_1	$u_{..}$	$u_{..}$	$u_{..}$
u_2	$u_{..}$	$u_{..}$	$u_{..}$
$u_{..}$	$u_{..}$	$u_{..}$	u_{255}

Buat suatu substitusi yang diperoleh dari operasi XOR pada setiap byte yang ada pada matriks pada tahap 1 dengan kunci k pada tahap 2.

4. Substitusi byte pada matriks tersebut dengan byte yang ada pada tabel substitusi

Matriks utama:

$k \veebar u_0$	$k \veebar u_4$	$k \veebar u_8$	$k \veebar u_{12}$
$k \veebar u_1$	$k \veebar u_5$	$k \veebar u_9$	$k \veebar u_{13}$
$k \veebar u_2$	$k \veebar u_6$	$k \veebar u_{10}$	$k \veebar u_{14}$
$k \veebar u_3$	$k \veebar u_7$	$k \veebar u_{11}$	$k \veebar u_{15}$

5. Tukar setiap elemen pada setiap baris ke kanan sesuai dengan indeks baris.

Lakukan penukaran setiap elemen sesuai dengan urutan indeks baris. Sebagai contoh,

$k \veebar b_0$	$k \veebar b_4$	$k \veebar b_8$	$k \veebar b_{12}$
$k \veebar b_1$	$k \veebar b_5$	$k \veebar b_9$	$k \veebar b_{13}$
$k \veebar b_2$	$k \veebar b_6$	$k \veebar b_{10}$	$k \veebar b_{14}$
$k \veebar b_3$	$k \veebar b_7$	$k \veebar b_{11}$	$k \veebar b_{15}$

↓

0	$k \veebar b_0$	$k \veebar b_4$	$k \veebar b_8$	$k \veebar b_{12}$
1	$k \veebar b_5$	$k \veebar b_9$	$k \veebar b_{13}$	$k \veebar b_1$
2	$k \veebar b_{10}$	$k \veebar b_{14}$	$k \veebar b_2$	$k \veebar b_6$
n	$k \veebar b_{15}$	$k \veebar b_3$	$k \veebar b_7$	$k \veebar b_{11}$

Pada baris 1, dilakukan pergeseran elemen dari kanan ke kiri secara sirkular sebanyak satu kali. Pada baris 2, dilakukan pergeseran elemen dari kanan ke kiri secara sirkular sebanyak dua kali. Hal ini dilakukan secara terus menerus mengikuti indeks baris hingga dicapai bilangan terakhir, yakni pergerseran sebanyak n kali.

6. Tentukan suatu matriks *N* berukuran 4 x 4 yang memiliki invers

N

b'_0	$b'_{...}$	$b'_{..}$	$b'_{..}$
b'_1	$b'_{..}$	$b'_{..}$	$b'_{..}$
b'_2	$b'_{..}$	$b'_{..}$	$b'_{..}$
$b'_{..}$	$b'_{..}$	$b'_{..}$	b'_{255}

Selanjutnya, kalikan setiap kolom dari matriks M dari kolom pertama hingga kolom terakhir dengan matriks N sedemikian sehingga membentuk skema perkalian matriks sebagai berikut.

b'_0	$b'_{...}$	$b'_{..}$	$b'_{..}$
b'_1	$b'_{..}$	$b'_{..}$	$b'_{..}$
b'_2	$b'_{..}$	$b'_{..}$	$b'_{..}$
$b'_{..}$	$b'_{..}$	$b'_{..}$	b'_{255}

x

c_0
c_2
c_3
c_{255}

Dengan demikian, akan dihasilkan suatu matriks kolom yang berukuran sama dengan matriks kolom sebelumnya. Selanjutnya, substitusikan hasil matriks kolom dengan matriks kolom sebelumnya.

7. Tentukan suatu kunci baru k2

$k_2 \in \mathbf{R}$, k_2 adalah bilangan yang besar

Selanjutnya, kunci k_2 akan digunakan untuk melakukan operasi tambah dengan hasil matriks dalam operasi *bitwise*.

8. Tambahkan setiap elemen (byte) dengan kunci k_2 dalam operasi bitwise

M'

$k_2 + b''_0$	$k_2 + b''_{...}$	$k_2 + b''_{..}$	$k_2 + b''_{..}$
$k_2 + b''_1$	$k_2 + b''_{..}$	$k_2 + b''_{..}$	$k_2 + b''_{..}$
$k_2 + b''_2$	$k_2 + b''_{..}$	$k_2 + b''_{..}$	$k_2 + b''_{..}$
$k_2 + b''_{..}$	$k_2 + b''_{..}$	$k_2 + b''_{..}$	$k_2 + b''_{255}$

9. Ulangi tahap 5-8 sebanyak 6-10 kali pada hasil yang telah diperoleh.
10. Satukan kembali setiap bit column-wise dan diperoleh hasil enkripsi 128-bit

$M' = b''_0 b''_1 b''_2 b''_3 b''_4 \ldots b''_{255}$

Untuk pesan yang memiliki Panjang lebih dari 256-byte, maka akan dilakukan partisi untuk setiap 256-byte dan selanjutnya akan disebut paket data. Paket data mengalami proses yang sama seperti yang disebutkan pada tahap 1 hingga tahap 9.

11. Tentukan suatu kunci k_3

    $k_3 \in \mathbf{R}$, k_3 adalah bilangan yang besar

    k_3 selanjutnya akan digunakan untuk sebagai *counter* yang akan dienkripsi.
12. Berlakukan tahap 2-10 terhadap k_3 + indeks

    Enkripsi k_3 + indeks dilakukan untuk setiap paket data dan akan menghasilkan suatu *ciphertext. Ciphertext* ini merupakan teks yang akan digunakan untuk membentuk *tag* sebagai hasil dari enkripsi AES-256-GCM. Indeks merujuk pada urutan paket data yang akan dienkripsi.

    Hasil enkripsi setiap kunci + indeks akan disebut sebagai k_3'
13. Lakukan operasi XOR antara hasil enkripsi k_3 + indeks dengan *plaintext* paket data yang berkaitan

    $k_3' \veebar M'$

    Hasil enkripsi ini kemudian akan disebut sebagai M''. Lakukan operasi ini untuk setiap paket data yang dihasilkan dari M.
14. Nyatakan suatu *hash function* H

    $H = \text{AES-256}(0^{128})$

    H merupakan suatu nilai *hash* yang diperoleh dengan melakukan enkripsi 0 melalui tahap 2 hingga 10.
15. Lakukan perkalian memanfaatkan Galois Finite Field Theory terhadap setiap hasil enkripsi paket data dengan H

    $C' = GFF(H, M'')$

    C' merupakan hasil perkalian yang tidak melebihi suatu batas $x^{128}+x^7+x^2+x+1$ dan mengikuti ketentuan teori Galois Finite Field.
16. Lakukan operasi XOR antara hasil enkripsi paket data yang dihasilkan melalui perkalian Galois Field dengan hasil enkripsi paket data setelahnya.

    $T = C_i' \text{ XOR } C_{i+1}'$

    Hal ini dilakukan untuk setiap T yang dihasilkan dari sepasang paket data dan hasil dari operasi ini akan diterapkan kembali pada operasi yang sama dengan C' selanjutnya.

    Apabila telah mencapai bilangan terakhir, maka lakukan operasi XOR terhadap kunci pada Tahap 16 dan panjang pesan asli

    $T_n = T_{n-1} \text{ XOR } k_3 \text{ XOR } L$

    dengan L adalah panjang pesan asli.

D. HMAC – SHA256

Hash Message Authentication Code merupakan suatu kode dari hash function yang digunakan sebagai tanda autentikasi terhadap suatu informasi. Informasi tersebut biasanya digunakan pada proses transfer ataupun penyimpanan.

Algoritma HMAC-SHA-256 untuk suatu pesan M dapat dinyatakan sebagai berikut.

M = b0b1b2b3b4b5b6….b255

1. Masukkan M pada suatu fungsi SHA-256 yang mengembalikan kode SHA-256 pada pengguna
2. Tentukan suatu kunci k_1 dan k_2
3. Terapkan algoritma SHA-256 terhadap kunci k_1 yang disertakan pada objek data utama m dan sebut hasil enkripsi sebagai e_1

   $e_1 = \text{SHA-256}(k_1 \mid m)$
4. Sertakan e1 di depan objek data utama m

   $e_2 = (e_1 \mid m)$
5. Tahap x + 3 : Ulangi tahap x + 1 dan x + 2 dengan kunci k2

   $e_2' = \text{SHA-256}(e_2 \mid m)$
6. Tahap x + 4 : Lakukan enkripsi AES-256-GCM terhadap hasil enkripsi pada tahap x + 3

   $T = \text{AES-256-GCM}(e_2)$

   Dengan demikian, diperoleh hasil HMAC – SHA – 256 yang merupakan penerapan AES-256-GCM pada tahap sebelumnya.

III. METODE EKSPERIMEN

A. Metode Eksperimen

Dalam eksperimen ini, akan dibuat suatu implementasi terhadap proses *Encryption at Rest* dan *Encryption in Transit* berdasarkan dasar teori yang disebutkan sebelumnya. Eksperimen ini berfokus pada penerapan proses *Encryption at Rest* dengan beberapa modifikasi tertentu. Berikut merupakan tahap implementasi dari setiap eksperimen

1. ENCRYPTION AT REST

Tahap 1 : Mengambil suatu pesan M yang dapat berukuran berapapun.

m = j0j1j2j3j4j5j6….j255

Tahap 2 : Nyatakan M dengan menggunakan algoritma HMAC-SHA-256 sehingga diperoleh pesan baru berupa m

Tahap 3 : Nyatakan m dalam matriks 16x16

m = b0b1b2b3b4b5b6….b255

b_0	$b_{...}$	$b_{..}$	$b_{..}$
b_1	$b_{..}$	$b_{..}$	$b_{..}$
b_2	$b_{..}$	$b_{..}$	$b_{..}$
b..	$b_{..}$	$b_{..}$	b_{255}

Tahap 4 : Tentukan kunci k

k ∈ **R,** k adalah bilangan yang besar

Kunci k ini selanjutnya akan digunakan untuk melakukan operasi XOR pada setiap byte pada tabel sebelumnya

Tahap 5 : Tentukan suatu tabel substitusi

$k \veebar b_0$	$k \veebar b_4$	$k \veebar b_8$	$k \veebar b_{12}$
$k \veebar b_1$	$k \veebar b_5$	$k \veebar b_9$	$k \veebar b_{13}$
$k \veebar b_2$	$k \veebar b_6$	$k \veebar b_{10}$	$k \veebar b_{14}$
$k \veebar b_3$	$k \veebar b_7$	$k \veebar b_{11}$	$k \veebar b_{15}$

Buat suatu substitusi yang diperoleh dari operasi XOR pada setiap byte yang ada pada matriks pada tahap 1 dengan kunci k pada tahap 2.

Tahap 6 : Substitusi byte pada matriks tersebut dengan byte yang ada pada tabel substitusi

Matriks utama:

k ⊻ b_0	k ⊻ b_4	k ⊻ b_8	k ⊻ b_{12}
k ⊻ b_1	k ⊻ b_5	k ⊻ b_9	k ⊻ b_{13}
k ⊻ b_2	k ⊻ b_6	k ⊻ b_{10}	k ⊻ b_{14}
k ⊻ b_3	k ⊻ b_7	k ⊻ b_{11}	k ⊻ b_{15}

Tahap 7 : Tukar setiap elemen pada setiap baris ke kanan sesuai dengan indeks baris.

Lakukan penukaran setiap elemen sesuai dengan urutan indeks baris. Sebagai contoh,

k ⊻ b_0	k ⊻ b_4	k ⊻ b_8	k ⊻ b_{12}
k ⊻ b_1	k ⊻ b_5	k ⊻ b_9	k ⊻ b_{13}
k ⊻ b_2	k ⊻ b_6	k ⊻ b_{10}	k ⊻ b_{14}
k ⊻ b_3	k ⊻ b_7	k ⊻ b_{11}	k ⊻ b_{15}

↓

0	k ⊻ b_0	k ⊻ b_4	k ⊻ b_8	k ⊻ b_{12}
1	k ⊻ b_5	k ⊻ b_9	k ⊻ b_{13}	k ⊻ b_1
	k ⊻ b_{10}	k ⊻ b_{14}	k ⊻ b_2	k ⊻ b_6
	k ⊻ b_{15}	k ⊻ b_3	k ⊻ b_7	k ⊻ b_{11}

Pada baris 1, dilakukan pergeseran elemen dari kanan ke kiri secara sirkular sebanyak satu kali. Pada baris 2, dilakukan pergeseran elemen dari kanan ke kiri secara sirkular sebanyak dua kali. Hal ini dilakukan secara terus menerus mengikuti indeks baris hingga dicapai bilangan terakhir, yakni pergerseran sebanyak n kali.

Tahap 8 : Tentukan suatu matriks *N* berukuran 4 x 4 yang memiliki invers

N

b'_0	$b'_{...}$	$b'_{..}$	$b'_{..}$
b'_1	$b'_{..}$	$b'_{..}$	$b'_{..}$
b'_2	$b'_{..}$	$b'_{..}$	$b'_{..}$
b'..	$b'_{..}$	$b'_{..}$	b'_{255}

Selanjutnya, kalikan setiap kolom dari matriks M dari kolom pertama hingga kolom terakhir dengan matriks N sedemikian sehingga membentuk skema perkalian matriks sebagai berikut.

b'_0	$b'_{...}$	$b'_{..}$	$b'_{..}$
b'_1	$b'_{..}$	$b'_{..}$	$b'_{..}$
b'_2	$b'_{..}$	$b'_{..}$	$b'_{..}$

x

c_0
c_2
c_3
c_{255}

b'..	$b'_{..}$	$b'_{..}$	b'_{255}

Dengan demikian, akan dihasilkan suatu matriks kolom yang berukuran sama dengan matriks kolom sebelumnya. Selanjutnya, substitusikan hasil matriks kolom dengan matriks kolom sebelumnya.

Tahap 9 :Tentukan suatu kunci baru k2

$k_2 \in \mathbf{R}$, k_2 adalah bilangan yang besar

Selanjutnya, kunci k_2 akan digunakan untuk melakukan operasi tambah dengan hasil matriks dalam operasi *bitwise*.

Tahap 10 :Tambahkan setiap elemen (byte) dengan kunci k_2 dalam operasi bitwise

M'

$k_2 + b''_0$	$k_2 + b''_{...}$	$k_2 + b''_{..}$	$k_2 + b''_{..}$
$k_2 + b''_1$	$k_2 + b''_{..}$	$k_2 + b''_{..}$	$k_2 + b''_{..}$
$k_2 + b''_2$	$k_2 + b''_{..}$	$k_2 + b''_{..}$	$k_2 + b''_{..}$
$k_2 + b''_{..}$	$k_2 + b''_{..}$	$k_2 + b''_{..}$	$k_2 + b''_{255}$

Tahap 11 : Ulangi tahap 5-8 sebanyak 6-10 kali pada hasil yang telah diperoleh.

Tahap 12 : Satukan kembali setiap bit column-wise dan diperoleh hasil enkripsi 128-bit

$M' = b''_0 b''_1 b''_2 b''_3 b''_4 \ldots b''_{255}$

Untuk pesan yang memiliki Panjang lebih dari 256-byte, maka akan dilakukan partisi untuk setiap 256-byte dan selanjutnya akan disebut paket data. Paket data mengalami proses yang sama seperti yang disebutkan pada tahap 1 hingga tahap 9.

Tahap 13 : Tentukan suatu kunci k_3

$k_3 \in \mathbf{R}$, k_3 adalah bilangan yang besar

k_3 selanjutnya akan digunakan untuk sebagai *counter* yang akan dienkripsi.

Tahap 14 : Berlakukan tahap 2-10 terhadap k_3 + indeks

Enkripsi k_3 + indeks dilakukan untuk setiap paket data dan akan menghasilkan suatu *ciphertext. Ciphertext* ini merupakan teks yang akan digunakan untuk membentuk *tag* sebagai hasil dari enkripsi AES-256-GCM. Indeks merujuk pada urutan paket data yang akan dienkripsi.

Hasil enkripsi setiap kunci + indeks akan disebut sebagai k_3'

Tahap 15 : Lakukan operasi XOR antara hasil enkripsi k_3 + indeks dengan *plaintext* paket data yang berkaitan

$k_3' \veebar M'$

Hasil enkripsi ini kemudian akan disebut sebagai M''. Lakukan operasi ini untuk setiap paket data yang dihasilkan dari M.

Tahap 16 : Nyatakan suatu *hash function* H

$H = AES\text{-}256(0^{128})$

H merupakan suatu nilai *hash* yang diperoleh dengan melakukan enkripsi 0 melalui tahap 2 hingga 10.

Tahap 17 : Lakukan perkalian memanfaatkan Galois Finite Field Theory terhadap setiap hasil enkripsi paket data dengan H

$$C' = GFF(H, M'')$$

C’ merupakan hasil perkalian yang tidak melebihi suatu batas $x^{128}+x^7+x^2+x+1$ dan mengikuti ketentuan teori Galois Finite Field.

Tahap 18 : Lakukan operasi XOR antara hasil enkripsi paket data yang dihasilkan melalui perkalian Galois Field dengan hasil enkripsi paket data setelahnya.

$$T = C_i' \text{ XOR } C_{i+1}'$$

Hal ini dilakukan untuk setiap T yang dihasilkan dari sepasang paket data dan hasil dari operasi ini akan diterapkan kembali pada operasi yang sama dengan C’ selanjutnya.

Apabila telah mencapai bilangan terakhir, maka lakukan operasi XOR terhadap kunci pada Tahap 16 dan panjang pesan asli

$$T_n = T_{n-1} \text{ XOR } k_3 \text{ XOR } L$$

dengan L adalah panjang pesan asli.

Berdasarkan ringkasan metode eksperimen yang akan dilakukan, ditentukan pula suatu metode untuk menguji berbagai kualitas dan aspek dari penerapan metode ini. Beberapa aspek pengujian yang akan diperhatikan diantaranya :

1. Tingkat kemiripan
   Semakin mirip suatu ciphertext dengan plaintext yang menjadi kondisi awalnya, semakin buruk algoritma enkripsi tersebut. Hal ini akan memanfaatkan prinsip Cosine Similarity sebagai metode pengukuran untuk setiap byte
2. Waktu
   Semakin sedikit waktu yang diperlukan untuk memperoleh ciphertext, maka semakin baik algoritma yang digunakan
3. Perbedaan jumlah perulangan implementasi algoritma AES
   Hal ini memperhatikan pengaruh jumlah perulangan implementasi algoritma AES terhadap tingkat kemiripan antara plaintext dan ciphertext

C. Kode Eksperimen

```
import hashlib
import time
import random
import numpy as np

def AES256(payload : str) :
    # payload is a 64 character string that produced from hashHex

    # generate key for XOR
    k1 = random.getrandbits(256)
    # print(f"k1: {k1}") # for debug purposes only

    # hex dictionary
    # hexDict = {
    #     '0': 0, '1': 1, '2': 2, '3': 3,
    #     '4': 4, '5': 5, '6': 6, '7': 7,
    #     '8': 8, '9': 9, 'a': 10, 'b': 11,
    #     'c': 12, 'd': 13, 'e': 14, 'f': 15
    # }

    # # Create Matrix 16x16
    matrix = [[0 for _ in range(16)] for _ in range(16)]
    colMatrix = [0 for _ in range(16)]

    # # generate binary from payload
    # bin = ""
    # for char in payload:
    #     bin += format(hexDict[char], '04b')
    # print(f"bin initial : {bin}") # for debug purposes only

    # XOR operation for bin
    binXOR = int(payload, 2) ^ k1
    binXOR = format(binXOR, '0256b')

    # Convert all the bin into matrices 16x16
    for i in range(16):
        for j in range(16):
            matrix[i][j] = (binXOR[i*16 + j])

    # subsitute elements with the substitute matrix
    k2 = random.getrandbits(16)
    k2 = str(format(k2, '016b'))
    # print(f"k2 : {(k2)}") # for debug purposes only
    for i in range(16):
        for j in range(16):
            matrix[i][j] = str(int(matrix[i][j]) ^ int(k2[j]))
    # print(f"init matrix : {matrix}") # for debug purposes

    # switch elements in the rows
    for i in range(16):
        for _ in range(i):
            temp = matrix[i][0]
            for j in range(15):
                matrix[i][j] = matrix[i][j + 1]
            matrix[i][15] = temp
    # print(f"after matrix : {matrix}") # for debug purposes

    # switch every column using matrix multiplication : generate matrix multiplication
    mulMatrix = generateMultiplicationMatrix()
    for i in range(16):

        # get colMatrix
        for j in range(16):
            colMatrix[j] = matrix[j][i]
        for l in range(len(colMatrix)):
            colMatrix[l] = int(colMatrix[l])
        colResMatrix = multiplyFiniteFieldMatrix(mulMatrix, colMatrix)

        # filling the col before
        for k in range(len(colResMatrix)):
            matrix[k][i] = str(colResMatrix[k])


    # add every row with the k3 (bitwise operation)
    k3 = random.getrandbits(16)
```

```
    k3 = str(format(k3, '016b'))
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            matrix[i][j] = str(int(matrix[i][j]) ^ int(k3[j]))
    matrixBin = ''.join(value for row in matrix for value in row)

    return matrixBin # output is in the form of string of binaries

def AES256GCM(payload : str, lengthMsg : int, iteration : int):
    dataPacket = ""
    tagArray = []
    cipherTextArray = []
    keyGCM = random.getrandbits(256)

    # hex dictionary
    hexDict = {
        '0': 0, '1': 1, '2': 2, '3': 3,
        '4': 4, '5': 5, '6': 6, '7': 7,
        '8': 8, '9': 9, 'a': 10, 'b': 11,
        'c': 12, 'd': 13, 'e': 14, 'f': 15
    }

    # converting from hex into binary
    bin = ""
    for char in payload:
        bin += format(hexDict[char], '04b')

    # generate AES-256 of payload
    for i in range(len(bin)):
        if (i % 256 == 0) and (i != 0) :

            # ------------------------------------------------------------------------
            # GCM part
            matrix = AES256(dataPacket)
            for _ in range(iteration):
                matrix = AES256(matrix)
            # print(f"matrix : {matrix}") # for debug purposes

            # Hash value
            hashVal = format(0, "0256b")
            hashVal = AES256(hashVal)
            # print(f"hasval : {hashVal}") # for debug purposes

            # generate key for GCM
            keyGCM = str(format(keyGCM + (i // 256), '0256b'))
            keyGCM = AES256(keyGCM)
            # print(f"keygcam : {keyGCM}") # for debug purposes

            # XOR keyGCM with the plaintext
            cipherText = int(keyGCM, 2) ^ int(matrix, 2)
            cipherTextArray.append(cipherText)
            # cipherText = format(cipherText, "0256b") # for debug purposes
            # print(f"cipherText : {cipherText}")

            # multiply the hashvalue with the cipherText using finite field
            tag = gf_multiply_mod(int(cipherText), int(hashVal))
            tagArray.append(tag)


        elif (i == (len(payload) - 1) and (i % 256 != 0)):
            if len(dataPacket) < 256:
                dataPacket = dataPacket.zfill(256)

            # ------------------------------------------------------------------------
            # GCM part
            matrix = AES256(dataPacket)
            for _ in range(iteration):
                matrix = AES256(matrix)
            # print(f"matrix : {matrix}") # for debug purposes

            # Hash value
            hashVal = format(0, "0256b")
            hashVal = AES256(hashVal)
            # print(f"hasval : {hashVal}") # for debug purposes

            # generate key for GCM
            keyGCM = str(format(keyGCM + (i // 256), '0256b'))
            keyGCM = AES256(keyGCM)
            # print(f"keygcam : {keyGCM}") # for debug purposes

            # XOR keyGCM with the plaintext
            cipherText = int(keyGCM, 2) ^ int(matrix, 2)
            cipherTextArray.append(cipherText)
            # cipherText = format(cipherText, "0256b") # for debug purposes
            # print(f"cipherText : {cipherText}")

            # multiply the hashvalue with the cipherText using finite field
            tag = gf_multiply_mod(int(cipherText), int(hashVal))
            tagArray.append(tag)

        else :
            dataPacket += bin[i]
    # tags have been appended into the tagArray

    # calculating final tag
    finalTag = 0
    for i in range(len(tagArray)):
        finalTag ^= cipherTextArray[i]
        finalTag = gf_multiply_mod(finalTag, cipherTextArray[i])
    finalTag ^= (lengthMsg ^ int(keyGCM, 2))

    return finalTag

def gf_multiply_mod(a, b):
    """
    Multiply two 128-bit integers in GF(2^128) modulo P(x) =
```

```
x^128 + x^7 + x^2 + x + 1.
    """
    # Modulus polynomial as an integer
    mod_poly = 0x87  # Corresponds to x^7 + x^2 + x + 1 (high bit implied for x^128)

    # Initialize result
    result = 0

    # Perform multiplication
    for i in range(128):
        # If the lowest bit of b is 1, add a to the result
        if b & 1:
            result ^= a

        # Shift b to the right
        b >>= 1

        # Check if the highest bit of a is set
        carry = a & (1 << 127)

        # Shift a to the left
        a <<= 1

        # If carry out of 128 bits occurs, reduce modulo P(x)
        if carry:
            a ^= mod_poly

    # Return the 128-bit result
    return result & ((1 << 128) - 1)  # Ensure result is within 128 bits



  def multiplyFiniteFieldMatrix(mulMatrix, colMatrix):
    # mulMatrix and colMatrix inserted is in integer form

    resMatrix = [0 for _ in range(16)]

    for i in range(len(mulMatrix)):
        res = 0
        for j in range(len(colMatrix)):
            res ^= multiplyFiniteField(mulMatrix[i][j], colMatrix[j])
        res = format(res, '016b')
        for k in range(len(res)):
            resMatrix[k] = int(res[k])

    return resMatrix

  def multiplyFiniteField(a : int, b : int):
    """
    Perform finite field multiplication in GF(2^n).

    :param a: First operand (as an integer).
    :param b: Second operand (as an integer).
    :param mod_poly: The irreducible polynomial (as an integer).
    :return: Result of the multiplication modulo the polynomial.
    """
    # COSTANTS
    mod_poly = 0x11b

    # Algorithm
    result = 0
    while b > 0:
        # If the least significant bit of b is 1, add a to the result
        if b & 1:
            result ^= a  # XOR for addition in GF(2)

        # Shift b to the right and a to the left (like traditional multiplication)
        b >>= 1
        a <<= 1

        # If a exceeds 8 bits, reduce it modulo the irreducible polynomial
        if a & 0x100:  # Check if the 9th bit is set
            a ^= mod_poly

    return result

  def generateMultiplicationMatrix() :

    # generate matrix
    mulMatrix = np.array([[random.getrandbits(16) for _ in range(16)] for _ in range(16)])
    det = (np.linalg.det(mulMatrix))

    # check for determinant
    while det == 0 :
        mulMatrix = np.array([[random.getrandbits(16) for _ in range(16)] for _ in range(16)])
        det = (np.linalg.det(mulMatrix))

    return mulMatrix.tolist()

  def EncryptionAtRest(msg : str):
    dataPacket = ''
    # hex dictionary
    hexDict = {
        '0': 0, '1': 1, '2': 2, '3': 3,
        '4': 4, '5': 5, '6': 6, '7': 7,
        '8': 8, '9': 9, 'a': 10, 'b': 11,
        'c': 12, 'd': 13, 'e': 14, 'f': 15
    }

    # determine k1 and k2
    k1 = random.uniform(1, 10**100)
    k2 = random.uniform(1, 10**100)

    # Do SHA-256 of the dataPacket with k1
    dataPacket = str(k1) + dataPacket
    hexDataPacket = hashlib.sha256()
    hexDataPacket.update(dataPacket.encode('utf-8'))
```

```
        hexDataPacket = hexDataPacket.hexdigest()

        # Do SHA-256 of the msg with k2
        hexDataPacket = str(k2) + hexDataPacket
        hexDataPacket = hashlib.sha256()
        hexDataPacket.update(dataPacket.encode('utf-8'))
        hexDataPacket = hexDataPacket.hexdigest() # Here binMsg is the HMAC of the real message

        # return AES-256-GCM of the message string
        iterations1 = 6
        iterations2 = 10
        tag = AES256GCM(hexDataPacket, len(msg), iterations1)

        # testing : similarity
        binHexDataPacket = ''
        for hex in hexDataPacket :
            binHexDataPacket += format(hexDict[hex], '04b')
        print(f"Binary Message (SHA-256): {binHexDataPacket}")
        print(f"Cipher Message (AES-256-GCM) : {format(tag, '256b')}")
        print(f"Similarity : {cosineSimilarity(format(tag, '0256b'), binHexDataPacket)}")
        print(f"Perulangan AES-256 : {iterations1}")


        return


    def cosineSimilarity(plainText: str, cipherText : str):
        # plaintext and cipherText needs to be the same length

        # Compute dot product
        dot_product = 0
        for i in range(len(plainText)):
            dot_product += int(plainText[i]) * int(cipherText[i])

        # Compute magnitudes
        normPlainText = 0
        for i in range(len(plainText)):
            normPlainText += int(plainText[i])
        normPlainText = np.sqrt(normPlainText)
        normCipherText = 0
        for i in range(len(plainText)):
            normCipherText += int(cipherText[i])
        normCipherText = np.sqrt(normCipherText)

        # Compute cosine similarity
        similarity = dot_product / (normCipherText * normPlainText)

        return similarity

    # driver
    msg = "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum."

    # testing : time
    start = time.time()
    EncryptionAtRest(msg)
    end = time.time()
    print(f"Encryption time : {end - start}")
```

IV. HASIL DAN PEMBAHASAN

A. Hasil Eksperimen

1. Encryption at Rest

Binary Message (SHA-256):
0111000000100001000111101000110101111011110011111001001111111100001100100100111100011110111000111101101011101010011101111110000101001111000101110011011010100101110100001001100110110001100000000101000011011000110001011001110110111100100010011100000000100001001
Cipher Message (AES-256-GCM) :
1010111011000011110010000111001111111010000011010111100010101111001101001101010111010100000100111101101110111011100010001010000110001101000101010110011011111010100111101010010000111011111100111010010110110101101100001110011110111011000011000011110001111100
Similarity : 0.5191639914323797
Perulangan AES-256 : 6
Encryption time : 0.012930154800415039

(1)

Binary Message (SHA-256):
0110110000001011011110100111000010110101111110111010111110010111000101100011011000001010000000101101100101110110100001110101010010011011101100000100000000100110011001111101100000110001101010010111110100110000101010110001011000100101100110010000100010110110
Cipher Message (AES-256-GCM) :
1010000011001111111001100100110100111011111000000000001011100000110010001101100000000101110001101011010101010111000111000001111110000000001100001010100001110001101101010110011001111100110111011000101000111010101000011100000010110101110011011111000011001000
Similarity : 0.500017362015388
Perulangan AES-256 : 6
Encryption time : 0.023041248321533203

(2)

Binary Message (SHA-256):
0000000100110111101100111000011011010111000111101000111100010011111110100101000111101010011101110000110110110111100101000110010110110101000111110001111001100010100110110111011110011001011000111011011011000010100000111110011100010100001110101001010001000011
Cipher Message (AES-256-GCM) :
1000111000101001011001100110100000110100110110010

110111010100111111000111110010111111110000110100010101010100010001010111001101000101010101000000111101110001110011010101010101011000011100001000010010001000111000010010100011100000100001000110001000000
Similarity : 0.4404825594979945
Perulangan AES-256 : 10
Encryption time : 0.02532362937927246

(3)

B. Pembahasan

Berdasarkan hasil yang diperoleh, dapat dilihat bahwa proses enkripsi yang diterapkan oleh Google memiliki tingkat kemiripan yang cenderung rendah dibawah 50%. Hal ini dipengaruhi oleh faktor permutasi dari elemen elemen pada setiap baris dan faktor permutasi dari setiap matriks kolom yang diterapkan pada algoritma AES-256-GCM. Selain itu, hal ini juga didorong dengan penggunaan mode GCM (Galois Counter Mode) yang mengenkripsi berbagai karakter dengan kunci yang berbeda-beda sehingga faktor acak semain meningkat. Namun, menilai dari operasi XOR yang cenderung cukup banyak, hal ini mendorong adanya pola terhadap enkripsi sehingga menurunkan tingkat acak dari hasil enkripsi.

Kemudian, berdasarkan hasil eksperimen, waktu yang diperoleh sangat kecil dan hal ini merupakan hal yang baik karena bermakna bahwa algoritma yang diterapkan memiliki tingkat keberhasilan yang cukup baik. Hal ini dipengaruhi oleh banyaknya operasi *bitwise* yang tidak membutuhkan banyak waktu dan dapat memberikan kendali lebih bagi mesin komputer untuk melakukan manipulasi data. Namun, menimbang dari waktu tersebut, terdapat pertimbangan mengenai kasus apabila terdapat kesalahan logika pada program sehingga akan mengakibatkan program memberikan hasil yang salah meskipun memiliki waktu generasi yang cepat.

Terakhir, menimbang dari pengujian pada variasi pengulangan terhadap algoritma AES-256, dapat dilihat bahwa variabel waktu meningkat seiring dengan meningkatnya jumlah perulangan. Hal ini dapat terlihat dari proses enkripsi variasi iterasi = 6 dan iterasi = 10 memiliki perbedaan sebesar ±0.002 sekon. Hal ini dapat memiliki pengaruh lebih besar apabila iterasi dilakukan dalam jumlah lebih banyak dan hal ini juga dapat ditelusuri dari penggunaan algoritma AES-256 yang memerlukan waktu sehingga memang wajar untuk meningkatnya variabel waktu seiring dengan variabel iterasi. Dengan demikian, peningkatan jumlah waktu sebanding dengan peningkatan jumlah iterasi

Demikian, melalui analisis yang telah diberikan, AES-256-GCM merupakan algoritma yang relatif bagus dalam penerapannya karena dapat diperoleh suatu hasil dengan tingkat kemiripan yang relatif cukup rendah dan durasi yang cukup kecil dengan konfigurasi tertentu.

V. Conclusion

Algoritma Advanced Encryption Standard (AES-256) merupakan algoritma enkripsi yang relative bagus karena memiliki tingkat kemiripan yang relatif cukup rendah, durasi generasi yang cukup kecil dan kustomisasi terhadap iterasi dapat meningkatkan performa dari algoritma ini. Meski begitu, iterasi yang digunakan dalam algoritma ini perlu dipertimbangkan lebih lanjut mengikuti kebutuhan tertentu.

VII. Acknowledgment

Dengan ini, saya menyatakan terima kasih sebesar-besarnya untuk setiap pihak yang telah mendukung Saya dalam menyelesaikan mata kuliah IF2123 Matematika Diskrit. Tak lupa, Saya juga mengucapkan terimakasih secara langsung untuk Pak Rila Mandala karena telah membimbing Saya hingga titik ini.

References

[1] Default encryption at rest | Documentation | Google Cloud
[2] Understand Key Concepts in Tink | Google for Developers
[3] FIPS 140-2 Validated - Compliance | Google Cloud
[4] Cryptographic Module Validation Program | CSRC
[5] https://www.youtube.com/watch?v=4zahvcJ9glg
[6] https://www.youtube.com/watch?v=O4xNJsjtN6E&t=661s
[7] https://www.youtube.com/watch?v=-fpVv_T4xwA&t=519s
[8] https://www.youtube.com/watch?v=g_eY7JXOc8U
[9] https://www.youtube.com/watch?v=wlSG3pEiQdc&t=339s
[10] What Is SHA-256? | Boot.dev

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, 06 Januari 2025

Brian Albar Hadian
13523048