Penerapan Teori Bilangan pada Algoritma Pembangkit Bilangan Prima untuk Kriptosistem RSA

Nicholas Liem - 13521135[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13521135@mahasiswa.itb.ac.id

***Abstrak* — Algoritma pada kriptosistem RSA memerlukan dua kunci utama, yakni kunci publik dan kunci privat. Kedua dari kunci ini haruslah dua bilangan prima yang sangat besar, setidaknya harus sepanjang 309-617 digit bilangan desimal atau 1024-2048 bit. Oleh sebab itu, diperlukan suatu algoritma yang dapat membangkitkan bilangan prima tersebut. Pada penerapannya, algoritma pembangkit prima memerlukan suatu sistem pengujian yang akurat terhadap bilangan prima. Sistem pengujian ini terbagi menjadi dua fase, pengujian bilangan terhadap bilangan prima tingkat rendah (*Low-Level Primality Test*) dan dilanjutkan dengan sistem pengujian Miller-Rabin (*High-Level Primality Test*) yang merupakan pengujian bilangan prima yang bersifat probabilistik.**

***Kata Kunci* —Enkripsi, Dekripsi, Miller-Rabin, Prima, RSA**

I. PENDAHULUAN

Di tengah perkembangan teknologi yang begitu pesat, pertukaran informasi menjadi hal yang sangat penting bagi saat ini. Pada waktu Perang Dunia II, kerahasiaan informasi menjadi hal yang krusial khususnya jika melibatkan tentang strategi militer. Oleh sebab itu, sudah sejak dahulu, sistem pertukaran informasi yang aman menjadi hal yang sangat menentukan.

Dalam zaman ini, signifikansi dari sistem pertukaran informasi juga adalah hal yang tidak kalah penting. Jika pada zaman perang dunia informasi yang dipertukarkan adalah rantai komando atau strategi militer, pada zaman ini, informasi yang dipertukarkan bisa dalam bentuk *password*, pesan pribadi, data kartu kredit, dan sebagainya. Oleh sebab itu, dalam setiap aksi yang melibatkan data penting di internet, diperlukan suatu sistem yang memiliki tingkat keamanan tinggi untuk mencegah penyalahgunaan data atau informasi. Sistem ini harus dapat berfungsi untuk menjamin keamanan dan privasi antara pengirim dan penerima.

Kriptologi adalah ilmu yang mempelajari tentang mekanisme keamanan komunikasi dan pertukaran informasi. Ilmu ini membahas tentang bagaimana data dipertukarkan dan ditransmisikan secara aman melalui proses enkripsi dan deskripsi. Enkripsi adalah suatu proses mengkodekan suatu informasi sehingga hanya penerima khusus yang dapat mengaksesnya. Proses enkripsi pada hakekatnya menggunakan suatu algoritma untuk melakukan randomisasi dan penguncian informasi sehingga hanya penerima dengan kunci yang dapat mengakses informasi tersebut. Di sisi lain, dekripsi adalah proses untuk mengonversi pesan terenkripsi (*cipher text*) ke dalam informasi yang bisa dibaca (*plain text*).

Salah satu sistem enkripsi-dekripsi yang sampai saat ini masih menjadi sistem enkripsi-dekripsi yang paling aman adalah kriptosistem RSA. RSA dibentuk dari tiga kata yang merupakan nama dari penemunya, yakni Ron Rivest, Adi Shamir, dan Len Adleman. Mereka mengembangkan suatu kriptosistem yang hingga saat ini dipakai karena kebergunaannya. Selain itu, kriptosistem ini tergolong sebagai kriptosistem yang bersifat asimetrik yang artinya diperlukan dua buah kunci publik dan privat untuk melakukan proses enkripsi maupun dekripsi data.

Dalam penggunaannya, sistem RSA memerlukan dua bilangan prima yang sangat besar (untuk kunci publik dan privat), dalam industri, ukuran dari panjang bilangan prima tersebut berkisar dari 1024-2048 bit (sekitar 309-617 digit desimal). Dengan begitu, muncullah sebuah pertanyaan, bagaimana caranya untuk dapat menghasilkan suatu bilangan prima yang begitu panjang dan memastikan bahwa bilangan yang panjang itu merupakan bilangan prima.

Sebenarnya, untuk menghasilkan bilangan yang sangat besar itu cukup mudah, tetapi untuk memastikan bahwa bilangan yang sangat besar itu adalah suatu bilangan prima sangat sulit. Terlebih lagi, akan sangat inefisien jika algoritma yang digunakan menggunakan metode-metode deterministik untuk menguji bilangan tersebut. Hal ini akan memakan waktu yang sangat lama. Oleh sebab itu, langkah yang dapat diambil untuk memastikan bahwa bilangan yang panjang itu adalah bilangan prima, akan digunakan pengujian prima dengan metode probabilistik.

II. LANDASAN TEORI

A. Enkripsi-Dekripsi

Dalam suatu sistem perpindahan data dan informasi, tentunya membutuhkan suatu sistem yang menjamin keamanan data. Biasanya, akan digunakan suatu sistem kriptografi. Ide fundamental dari kriptografi sendiri ialah memanfaatkan sebuah kunci yang dapat digunakan untuk mengenkripsi data maupun mendekripsikan data. Proses mengenkripsi data ada berbagai macam jenis, dua di antaranya ialah enkripsi simetrik dan enkripsi asimetrik.

1. Enkripsi Simetrik

   Enkripsi simetrik adalah suatu teknik enkripsi yang hanya membutuhkan satu kunci untuk proses enkripsi dan dekripsinya.

Gambar 2.1 Sistem enkripsi simetrik
Sumber: Dokumen pribadi

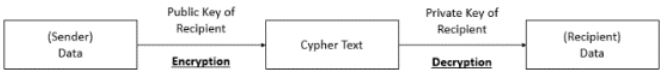
Beberapa contoh sistem enkripsi simetrik adalah AES (Advanced Encryption Standard), DES (Data Encription Standard), IDEA (International Data Encryption Algorithm), Blowfish, RC (Rivest Cipher 4, 5, dan 6).

Enkripsi simetrik biasanya lebih dipilih untuk digunakan karena pemrosesannya sangat cepat dan lebih efisien dari sistem enkripsi yang lainnya. Biasanya, pemrosesan enkripsi simetrik ini digunakan untuk mengirim data yang sangat besar karena efisiensinya. Penggunaannya dalam sehari-hari adalah untuk aplikasi pembayaran, validasi pengirim pesan, dan pembangkit bilangan acak atau sistem *hashing*.

Namun, terlepas dari kelebihan yang dipunyai oleh sistem enkripsi simetrik. Sistem ini juga memiliki kelemahan yang cukup krusial, yakni pertukaran dari kuncinya itu sendiri harus dilakukan secara aman antara kedua pihak. Hal tersebut nyatanya merupakan suatu hal yang cukup sulit untuk dijamin.

2. Enkripsi Asimetrik

   Enkripsi asimetrik adalah suatu teknik enkripsi yang membutuhkan dua macam kunci, yakni kunci publik dan kunci pribadi. Jika sebuah data dienkripsi menggunakan kunci publik, maka cara mengekstrak informasinya adalah dengan menggunakan kunci pribadi.



Gambar 2.2 Sistem enkripsi asimetrik
Sumber: Dokumen pribadi

Beberapa contoh sistem enkripsi asimetrik adalah RSA (Rivest-Shamir, DSA (Digital Signature Algorithm), dan PKCS (Public-Key Cryptography Standards), SSL/TLS (Digital Certificates), dan lain sebagainya.

Biasanya, enkripsi asimetrik dipilih untuk digunakan bukan karena pemrosesannya yang cepat, melainkan keamanannya yang sangat tinggi karena menggunakan dua kunci yang berbeda. Hal ini disebabkan karena pengguna tidak perlu menunjukkan kunci privat mereka dan hanya menampilkan kunci publiknya saja.

Keterbatasan dari sistem asimetrik ini ialah jumlah data yang sangat besar akan menyebabkan enkripsi asimetrik menjadi sangat lama karena pemrosesan algoritmanya cenderung lebih kompleks dan lebih berat dibandingkan dengan enkripsi simetrik. Oleh sebab itu, biasanya penggunaan dari enkripsi asimetrik digunakan untuk data-data yang berukuran kecil.

Kabar baiknya, penggunaan dari enkripsi asimetrik dan simetrik digunakan secara bersamaan untuk meningkatkan efisiensi dari sistem enkripsi-dekipsi. Untuk menutupi kelemahan dari sistem enkripsi asimetrik, data yang dienkripsi adalah sebuah kunci pada enkripsi simetrik (ukuran kunci enkripsi simetrik tentunya lebih kecil dari besar data), dan kelemahan dari enkripsi simetrik ditutup dengan penggunaan enkripsi asimetrik dalam pertukaran kunci tersebut.

B. Teori Bilangan

1. Bilangan Prima dan Bilangan Komposit

   Suatu bilangan p disebut sebagai bilangan prima jika dan hanya jika $p > 1$ dan tidak memiliki bilangan positif pembagi selain dari 1 dan p sendiri. Bilangan komposit adalah bilangan selain bilangan prima.

2. Bilangan Relatif Prima

   Suatu bilangan p dan q disebut sebagai pasangan bilangan relatif prima jika dan hanya jika $FPB(p, q) = 1$ atau tidak ada pembagi lain selain satu antara bilangan p dan q.

3. Teorema Kecil Fermat

   Untuk suatu bilangan prima p dan suatu bilangan a maka $a^p - a$ selalu habis dibagi p.

$$a^{p-1} \equiv 1 \ (mod\ p) \qquad (2.1)$$

4. Fungsi Totient Euler

   Fungsi totient Euler adalah jumlah bilangan non-negatif yang kurang dari n dan relatif prima terhadap n. Suatu teorema yang cukup penting dari fungsi totient Euler adalah sebagai berikut.

$$\phi(n) = (p-1)(q-1) \qquad (2.2)$$

   dengan p dan q adalah prima.

5. Invers Modulo

   Diberikan suatu bilangan p dan q, x disebut invers modulo dari p jika dan hanya jika memenuhi kongruensi di bawah ini.

$$px \equiv 1\ mod(q) \qquad (2.3)$$

   Salah satu cara untuk mencari invers modulo dari suatu bilangan adalah dengan menggunakan algoritma Euclidean.

6. *Probabilistic Primality Test*

   Uji prima probabilistik adalah suatu teknik pengujian bilangan prima yang memberikan suatu hasil pengujian terhadap suatu bilangan apakah bilangan tersebut adalah bilangan yang mungkin prima atau bukan

prima. Uji prima probabilistik disebut probabilistik karena pengujian ini tidak sepenuhnya selalu menghasilkan luaran yang benar, melainkan ada *margin of error* dalam perhitungannya. Ada suatu kemungkinan yang menyebabkan suatu bilangan komposit dapat lolos pada pengujian prima ini, biasanya bilangan-bilangan tersebut disebut sebagai *pseudoprime* atau bilangan prima semu.

Beberapa algoritma pengujian prima probabilistik antara lain, *Fermat primality test, Solovay-Strassen primality test,* dan masih banyak yang lain.

C. Mekanisme Umum Enkripsi-Dekripsi RSA

Persamaan utama dari algoritma RSA adalah sebagai berikut.

$$(m^e)^d = m\ (mod\ n) \qquad (2.4)$$

Dalam hal ini,

1. m: pesan yang ingin dienkripsi
2. e: kunci publik sekaligus kunci enkripsi
3. d: kunci pribadi sekaligus kunci dekripsi
4. n: perkalian dua bilangan prima yang membentuk kunci publik dan kunci pribadi.

Prosedur dalam menggunakan RSA adalah sebagai berikut:

1. Pilihlah dua bilangan prima p dan q yang besar, panjangnya kurang lebih 309 sampai 617 digit (atau 1024-2048 bit).
2. Kalikan kedua bilangan tersebut $n = pq$ untuk menghasilkan n.
3. Pilihlah suatu kunci publik (e) di mana $1 < e < \phi(n)$ dan $FPB\big(e, \phi(n)\big) = 1$.
4. Hitunglah kunci pribadi (d) sedemikian sehingga $ed = 1\ mod(\phi(n))$, atau dalam hal ini, d adalah invers modulo dari e.
5. Untuk mengenkripsi suatu pesan, kita cari teks cipher $c = m^e\ (mod\ n)$.
6. Untuk mendekripsi suatu teks cipher, pesan asli dapat dicari menggunakan persamaan ini, $m = c^d\ (mod\ n)$.

C. Algoritma Sieve of Eratosthenes

Sieve of Eratosthenes adalah suatu algoritma yang digunakan untuk mencari suatu bilangan prima sampai suatu bilangan n. Algoritma ini adalah pertamakali ditemukan oleh seorang matematikawan Yunani bernama Eratosthenes sekitar 2300 tahun yang lalu.

Alur algoritmanya secara garis besar adalah sebagai berikut:

1. Daftarkan semua bilangan dari 2 hingga n pada suatu list.
2. Iterasi k di mana k adalah bilangan dari 2 hingga $\sqrt{n}$.
3. Hapus semua kelipatan dari bilangan $(k^2, k^2 + k, k^2 + 2k, \dots)$ hingga n pada list.

Hasil dari algoritma ini adalah suatu list yang berisi kumpulan bilangan-bilangan prima dari 2 hingga n. Sebagai tambahan, kompleksitas dari algoritma ini adalah $O(n \log(\log n))$.

E. Algoritma Uji Prima Miller-Rabin

Ada beberapa cara untuk menguji suatu bilangan prima atau bukan, salah satunya adalah algoritma uji prima Miller-Rabin. Awal mulanya, teknik uji prima ini ditemukan oleh Gary Lee Miller pada tahun 1976, tetapi versi yang dibuat oleh Gary Lee Miller merupakan uji prima yang deterministik dan dependen terhadap kebenaran dari hipotesis Riemann. Kemudian, Michael Oser Robin, melakukan modifikasi terhadap teknik dari Miller sehingga terciptalah algoritma uji prima probabilistik yang dikenal saat ini. Oleh sebab itu, algoritma itu sekarang disebut sebagai algoritma Miller-Rabin.

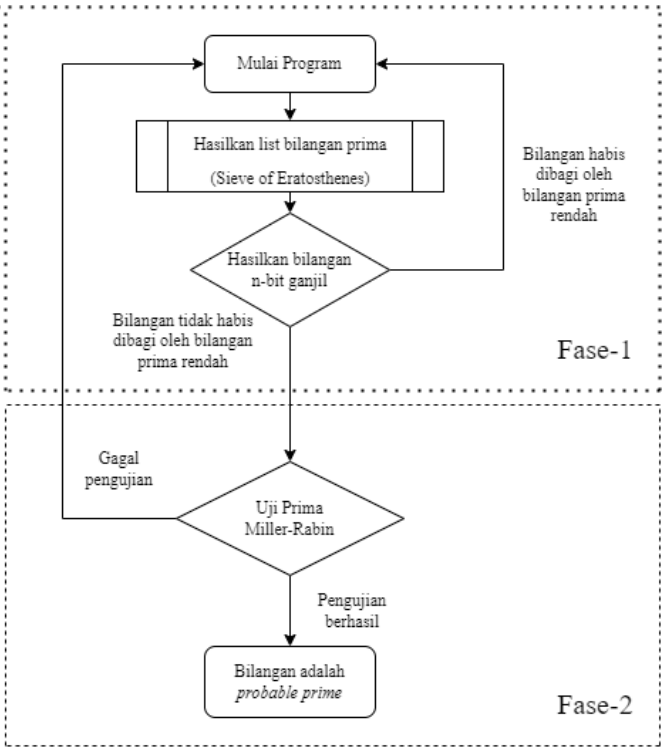
Namun, berbeda dengan pengujian-pengujian bilangan prima pada umumnya, algoritma Miller-Rabin tergolong sebagai algoritma uji prima yang bersifat probabilistik. Galat dari algoritma Miller-Rabin dapat dihitung dengan 4^{-k} di mana k adalah jumlah iterasi uji prima Miller-Rabin yang dilakukan. Untuk nilai k yang cukup besar, dapat diperkecil kemungkinan bahwa bilangan prima yang lolos uji prima tersebut adalah bilangan prima semu.

Algoritmanya adalah sebagai berikut:

1. Ambil n sebagai bilangan yang ingin diuji.
2. Hitunglah $n - 1 = 2^k \times m$
3. Pilihlah suatu angka a, dengan $1 < a < n - 1$
4. Hitunglah $b_0 = a^m\ (mod\ n)\ , \dots\ , b_i = {b_{i-1}}^2\ (mod\ n)$
5. Jika b_0 adalah 1, maka bilangan tersebut adalah bilangan komposit. Sedangkan, jika b_0 adalah -1, maka bilangan tersebut berkemungkinan sebagai bilangan prima. Selain nilai dari itu, hitung b_i hingga didapatkan hasil 1 atau -1.

III. APLIKASI

Secara garis besar, algoritma pembangkit bilangan prima yang memiliki jumlah digit di antara $309 - 617$ desimal dapat dibagi menjadi dua bagian besar, yakni pengujian prima *low-level* kemudian dilanjutkan dengan pengujian prima *Miller-Rabin*.



Gambar 3.1 Flowchart algoritma pembangkit bilangan prima

Sumber: Dokumen pribadi[1]

Pada gambar di atas, fase pertama adalah fase pengujian prima *low-level* di mana akan disaring apakah bilangan prima ini habis dibagi oleh bilangan prima-prima yang kecil (tidak sampai puluhan digit panjangnya).

Fase kedua adalah fase pengujian prima menggunakan algoritma uji prima Miller-Rabin, jika terjadi gagal pengujian maka program akan dikembalikan ke awal, sedangkan jika pengujian berhasil, maka akan dikeluarkan bilangan tersebut.

A. Aplikasi Algoritma Sieve of Eratosthenes

Berikut adalah implementasinya dalam bahasa pemrograman Python sesuai dengan definisi yang ada pada landasan teori.

```
def sieve_of_eratosthenes(n):
    list_of_primes = [True for i in range(n+1)]
    for i in range(2, int(n**(1/2))+1, 1):
        if list_of_primes[i]:
            for j in range(i**2, n+1, i):
                list_of_primes[j] = False
    # Bagian untuk memasukkan bilangan prima
    # Berdasarkan index list_of_primes dan valuenya
    new_primes = []
    for k in range(2,n+1):
        if (list_of_primes[k]):
            new_primes.append(k)
    return new_primes
```

Kode 3.1 Algoritma sieve of Erathosthenes
Sumber: Dokumen pribadi

Dalam algoritma ini, dibuatlah suatu list dengan panjang n yang berisi boolean bernilai True. Indeks dari list ini akan menjadi patokan bilangan prima yang akan nanti didapatkan. Ide utama dari algoritma ini adalah mengiterasi dari nilai index pertama (yakni 2) dan mengganti semua nilai pada index yang merupakan kelipatan dari 2 dan seterusnya hingga ke atas sehingga nilai pada index kelipatan iterasi tersebut menjadi bernilai False. Dengan begitu, akan disisakan boolean bernilai True pada indeks-indeks yang merupakan bilangan prima atau bukan kelipatan dari bilangan-bilangan sebelumnya.

B. Algoritma Uji Prima Level Rendah

Pada tahap pertama, akan dilakukan uji prima level rendah untuk calon bilangan prima yang akan diuji pada level berikutnya. Implementasinya adalah sebagai berikut.

```
def check_low_prime(n, low_primes):
    for primes in low_primes:
        if n % primes == 0 and primes**2 <= n:
            return False
        else:
            return True
```

Kode 3.2 Algoritma uji prima level rendah
Sumber: Dokumen pribadi

Ide utama dari algoritma di atas adalah akan dicek apakah bilangan ini dapat dibagi dengan bilangan-bilangan prima kecil, jika sisa hasil baginya adalah 0 artinya bilangan tersebut merupakan kelipatan suatu bilangan prima yang artinya bilangan tersebut adalah bilangan komposit dan bukan prima. Sebaliknya, jika sisa hasil baginya adalah selain nol, artinya bilangan tersebut adalah bilangan prima.

C. Aplikasi Algoritma Uji Prima Miller-Rabin

Berikut adalah implementasi dari algoritma uji prima Miller Rabin sesuai dengan yang ada pada landasan teori.

```
import random
def Miller_Rabin(n_prime):
    m = 0
    e = n_prime-1
    while n_prime % 2 == 0:
        e >>= 1
        m += 1
    def isComposite(n):
        if pow(n, e, n_prime) == 1:
            return False
        for i in range(m):
            if pow(n, 2**i * e, n_prime) == n_prime-1:
                return False
        return True
    iteration = 25
    for i in range(iteration):
        random_a = random.randrange(2, e)
        if isComposite(random_a):
            return False
    return True
```

Kode 3.3 Algoritma iji prima Miller-Rabin
Sumber: GeeksForGeeks (dimodifikasi)

Dalam algoritma ini, akan dicari bilangan m dengan melakukan operasi bit shift kanan untuk membagi bilangan menjadi dua dan setiap operasi bit shift dilakukan akan ditambahkan nilai m tersebut. Setelah itu, akan dipilih suatu bilangan acak a dan lakukan pengujian apakah $a^m \ mod \ (n\_prime)$, jika hasilnya adalah 1 maka akan dilakukan iterasi sebanyak $iteration$ untuk memastikan bahwa n_prime adalah prima. Jika hasil pengujian dari fungsi $isComposite$ adalah selain -1, maka n_prime bukanlah bilangan prima.

D. Pembangkit Bilangan dan Uji Prima Level Rendah

Berikut adalah implementasi algoritma pembangkit bilangan acak dan sekaligus mencari suatu bilangan yang lolos uji prima level rendah.

[1] Gambar ini dimodifikasi dari GeeksForGeeks.

```
def generateRandomPrime(n, low_primes):
    while True:
        num = random.randrange(2**(n-1)+1, 2**n - 1)
        if check_low_prime(num,low_primes):
            return num
```

Kode 3.4 Bilangan acak dan iji prima tingkat rendah
Sumber: Dokumen pribadi

Ide dari algoritma di atas adalah akan dicari suatu bilangan prima yang memiliki panjang *n* bit dan juga harus lolos tahap uji prima level rendah.

E. Algoritma Pembangkit Bilangan Prima

Secara garis besar, dihasilkan suatu bilangan besar acak, kemudian akan digunakan algoritma *Sieve of Eratosthenes* untuk menghasilkan sebuah list yang berisi bilangan-bilangan prima rendah, kemudian akan dilakukan pengujian tahap satu, yakni pembagian calon bilangan prima dengan daftar bilangan prima rendah. Kemudian, jika lolos pada tahap satu, algoritma dilanjutkan ke tahap dua di mana akan dilakukan iterasi algoritma uji Miller-Rabin sebanyak yang diminta. Jika lolos pada tahap kedua, maka calon bilangan prima tersebut adalah calon yang sangat mungkin menjadi bilangan prima.

Berikut adalah implementasi program secara keseluruhan.

```
import random
# Bag. 0 Menghasilkan suatu bilangan besar acak
def generateRandomPrime(n, low_primes):
    while True:
        num = random.randrange(2**(n-1)+1, 2**n - 1)
        if check_low_prime(num,low_primes):
            return num

# Bag. 1 Membentuk list bilangan prima rendah
def sieve_of_eratosthenes(n):
    list_of_primes = [True for i in range(n+1)]
    for i in range(2, int(n**(1/2))+1, 1):
        if list_of_primes[i]:
            for j in range(i**2, n+1, i):
                list_of_primes[j] = False
    new_primes = []
    for k in range(2,n+1):
        if (list_of_primes[k]):
            new_primes.append(k)
    return new_primes

# Bag. 2 mengecek apakah bilangan tersebut dapat dibagi
# oleh bilangan-bilangan prima rendah, kalau habis
dibagi
# artinya, bilangan tersebut adalah bilangan komposit.
def check_low_prime(n, low_primes):
    for primes in low_primes:
        if n % primes == 0 and primes**2 <= n:
            return False
        else:
            return True

# Bag. 3 mengecek bilangan prima dengan algoritma
# uji prima Miller-Rabin
def Miller_Rabin(n_prime):
    m = 0
    e = n_prime-1
    while n_prime % 2 == 0:
        e >>= 1
        m += 1
    def isComposite(n):
        if pow(n, e, n_prime) == 1:
            return False
        for i in range(m):
            if pow(n, 2**i * e, n_prime) == n_prime-1:
                return False
        return True
    iteration = 25
    for i in range(iteration):
        random_a = random.randrange(2, e)
        if isComposite(random_a):
            return False
    return True

# Bagian Utama Program
if __name__ == '__main__':
    low_primes = []
    low_primes = sieve_of_eratosthenes(1000)
    while True:
        jumlah_bit = 2048
        calon_prima = generateRandomPrime(jumlah_bit,
low_primes)
        if not Miller_Rabin(calon_prima):
            # Jika hasil prima yang diambil gagal,
            # akan diulang sampai ditemukan
            continue
        else:
            # Bilangan berhasil melewati pengujian
            print("Panjang bilangan prima: ",
len(str(calon_prima)))
            print("Bilangan prima yang dihasilkan: ",
calon_prima)
            break
```

Kode 3.5 Implementasi algoritma pembangkit bilangan prima
Sumber: Dokumen pribadi

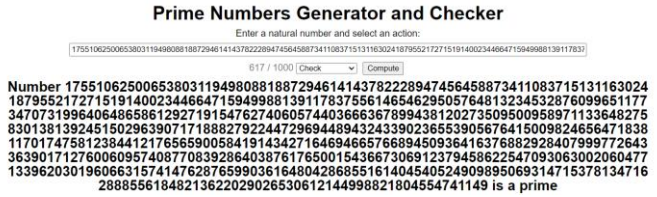
IV. HASIL IMPLEMENTASI

Berikut adalah hasil implementasi dari program yang dijalankan, pada hal ini, jumlah bit yang diambil adalah sepanjang 2048 bit atau dalam hal ini sekitar 617 digit bilangan desimal.

```
Panjang bilangan prima:  617
Bilangan prima yang dihasilkan:

17551062500653803119498088188729461414378222894745645887341108371513116302418795521727151914002344664715949988139117837
45645887341108371513116302418795521727151914002344664715949988139117837556146546295057648132345328760996511774
44664715949988139117837556146546295057648132345328760996511774
28760996511773470731996406486586129271915476274
60574403666367899438120273509500958971133648275
30138139245150296390717188827922447296944894324339023655390567641500982465647183
39023655390567641500982465647183811701747547
41217656590058419143427164694665766894509364163768829284079997726433639017127600609574087708392
68829284079997726433639017127600609574087708392864038761765001543667306912379458622547093063002060477
64038761765001543667306912379458622547093063002060477
60477133962030196066315741476287659903616480428685516140454052490989506931471537813471628885561848213622029026530612144998821804554741149
```

Kemudian, melalui website numberempire.com, akan dilakukan uji banding terhadap hasil yang didapatkan dari program. Berikut adalah hasilnya.



Gambar 4.1 Hasil uji banding dengan suatu website
Sumber: Dokumen pribadi

Hasil yang didapatkan dari hasil uji banding tersebut adalah sama. Kedua program menyebutkan bahwa bilangan prima yang dihasilkan adalah bilangan prima. Sebenarnya, untuk meningkatkan akurasi dari bilangan prima yang dihasilkan adalah dengan meningkatkan jumlah iterasi dalam operasi Miller-Rabin. Misalnya, untuk jumlah iterasi 40 kali, maka akan didapatkan tingkat galat sebesar 4^{-40} atau sekitar 8.27×10^{-25} dan dengan tinkat galat yang sangat kecil seharusnya sudah cukup untuk digunakan baik secara pribadi maupun komersial.

V. KESIMPULAN

Algoritma pembangkit bilangan prima adalah suatu algoritma yang sangat bergantung pada teori bilangan. Hal ini disebabkan karena pada badan algoritma pembangkit ini diperlukan suatu sistem uji yang akurat terhadap bilangan prima yang akan dihasilkan. Sistem uji yang sifatnya deterministik dinilai kurang tepat untuk menjadi sistem yang efisien dalam menguji suatu bilangan prima yang besar sebab algoritma uji deterministik memerlukan waktu yang sangat lama untuk memberikan hasil. Namun, harga yang harus dibayar untuk ketepatan atau akurasi yang diberikan algoritma uji deterministik harus dibayar dengan adanya galat. Algoritma yang menyediakan pertukaran ini adalah algoritma yang sifatnya probabilistik.

Sistem uji tersebut dibagi menjadi dua bagian. Bagian pertama adalah uji prima tingkat rendah yang memanfaatkan algoritma yang telah ada semenjak zaman dulu, yakni algoritma Sieve of Eratosthenes untuk mendapatkan daftar bilangan prima rendah dan kemudian akan dilakukan iterasi apakah bilangan yang ingin diuji dapat dibagi oleh bilangan-bilangan prima rendah itu, jika ya maka bilangan tersebut bukanlah bilangan prima. Bagian pertama tergolong sebagai algoritma uji prima deterministik, tetapi karena jumlah prima yang ada pada daftar berjumlah sedikit, jadi dampaknya terhadap algoritma keseluruhan tidak terlalu signifikan.

Bagian kedua adalah uji prima tingkat tinggi, yakni algoritma Miller-Rabin, kita menerima suatu bilangan prima besar dan menguranginya dengan 1. Kemudian, kita akan cari suatu nilai m di mana m adalah hasil faktorisasi bilangan prima tersebut dengan 2^k atau $n-1 = 2^k \times m$. Kemudian, akan dipilih suatu bilangan acak $a, 1 < a < n-1$ dan akan dilakukan iterasi terhadap $b_0 = a^m \ (mod\ n)$, jika bilangan yang dihasilkan adalah 1 maka bilangan n adalah bilangan komposit, tetapi jika hasilnya adalah -1, maka bilangan tersebut adalah bilangan prima. Selain dari 1 dan -1, maka akan dilakukan iterasi ulang di mana $b_i = {b_{i-1}}^2 \ (mod\ n)$.

Jika bilangan n berhasil melewati uji-uji tersebut, maka dapat dikatakan bahwa bilangan n adalah bilangan yang sangat mungkin adalah prima. Kemudian, hasil dari bilangan ini akan diimplementasikan ke dalam algoritma kriptosistem RSA untuk pembangkitan kunci publik dan kunci privat.

VII. UCAPAN TERIMA KASIH

Puji syukur kepada Tuhan Yang Maha Esa atas berkat dan penyertaanNya sehingga penulis bisa menyelesaikan makalah ini. Penulis ingin mengucapkan terima kasih kepada Ibu Fariska Zakhralativa Ruskanda, S.T., M.T selaku dosen mata kuliah IF2120 Matematika Diskrit yang telah membantu penulisan makalah ini melalui berbagai hal dan Dr. Rinaldi Munir yang telah menyediakan website yang dapat diakses untuk bahan belajar dan informasi.

REFERENSI

[1] Brush, Kate, et al. “What Is Asymmetric Cryptography? Definition from Searchsecurity.” Security, TechTarget, 27 September 2021, https://www.techtarget.com/searchsecurity/definition/asymmetric-cryptography. Diakses pada 10 Desember 2022.

[2] Harmenin, James T. “Symmetric Encryption.” Symmetric Encryption - an Overview, https://www.sciencedirect.com/topics/computer-science/symmetric-encryption. Diakses pada 10 Desember 2022.

[3] “How to Generate Large Prime Numbers for RSA Algorithm.” GeeksforGeeks, 3 Desember 2022, https://www.geeksforgeeks.org/how-to-generate-large-prime-numbers-for-rsa-algorithm/. Diakses pada 10 Desember 2022.

[4] Pauli, Sebastian. “MAT 112 Integers and Modern Applications for the Uninitiated.” Sieve of Eratosthenes, https://mathstats.uncg.edu/sites/pauli/112/HTML/seceratosthenes.html. Diakses pada 10 Desember 2022.

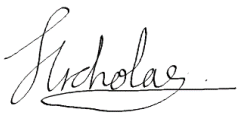
[5] Peter Smirnoff & Dawn M. Turner (guests). “Symmetric Key Encryption - Why, Where and How It's Used in Banking.” Cryptomathic, https://www.cryptomathic.com/news-events/blog/symmetric-key-

encryption-why-where-and-how-its-used-in-banking. Diakses pada 10 Desember 2022.
[6] "Primality Test: Set 3 (Miller–Rabin)." GeeksforGeeks, 14 November 2022, https://www.geeksforgeeks.org/primality-test-set-3-miller-rabin/. Diakses pada 10 Desember 2022.
[7] "Primality Tests." Primality Tests - Algorithms for Competitive Programming, 8 Juni 2022, https://cp-algorithms.com/algebra/primality_tests.html. Diakses pada 10 Desember 2022.
[8] "Python Program for Sieve of Eratosthenes." GeeksforGeeks, 12 Juli 2022, https://www.geeksforgeeks.org/python-program-for-sieve-of-eratosthenes/. Diakses pada 10 Desember 2022.
[9] "Python: Sieve of Eratosthenes Method, for Computing Prime Number." w3resource, https://www.w3resource.com/python-exercises/list/python-data-type-list-exercise-34.php. Diakses pada 10 Desember 2022.
[10] Savvy Security. "What Is Asymmetric Encryption? Read Symmetric vs. Asymmetric Encryption Diversity." Savvy Security, 21 Juli 2021, https://cheapsslsecurity.com/blog/what-is-asymmetric-encryption-understand-with-simple-examples/. Diakses pada 10 Desember 2022.
[11] "Time Complexity of Sieve of Eratosthenes Algorithm." Stack Overflow, 1 Mei 2011, https://stackoverflow.com/questions/2582732/time-complexity-of-sieve-of-eratosthenes-algorithm. Diakses pada 10 Desember 2022.
[12] David M. Burton. Elementary Number Theory. McGraw-Hill, 2010. ISBN: 9780073383149. Diakses pada 10 Desember 2022.
[13] Leighton Eric Lehman and Albert R. Meyer. "Mathematics for Computer Science". In: MIT, 2015. Chap. 2.1. Diakses pada 10 Desember 2022.
[14] Dr. Justin Wyss-Gallifent. Math 406: Multiplicative Functions and φ. url: http://www.math.umd.edu/~immortal/MATH406/lecturenotes/ch7-1.pdf. Diakses pada 10 Desember 2022.

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, 10 Desember 2022



Nicholas Liem
13521135