ANALISIS PERFORMA KRIPTOGRAFI RSA PADA WINDOWS 11 DENGAN PYTHON DAN APLIKASI NZXT CAM

Yusuf Atha Gunawan[1], Marutha Wira Yuda[2], Salsabila Wahyu Bintari[3], Aqwam Rosadi Kardian[4]

[1, 2]Rekayasa Kriptografi, Politeknik Siber dan Sandi Negara
[4]Sistem Informasi, STMIK Jakarta STIK
[1]yusuf.atha@student.poltekssn.ac.id, [2]marutha.wira@student.poltekssn.ac.id,
[3]salsabila.wahyu@student.poltekssn.ac.id, [4]marutha.wira@student.poltekssn.ac.id,

Diterima pada 10 Desember 2023; disetujui pada 10 januari 2024; dan diterbitkan pada 31 Januari 2024.

Abstrak

Penelitian ini bertujuan untuk menganalisis kinerja kriptografi RSA pada sistem operasi Windows 11 dengan implementasi menggunakan Python dan aplikasi NZXT CAM. Metode penelitian yang digunakan adalah pendekatan kuantitatif, dengan spesifikasi prosesor 2.80GHz, 8 inti, dan RAM 16 GB. Data dikumpulkan melalui aplikasi benchmark NZXT CAM, yang mengukur penggunaan CPU dan memori selama proses generasi kunci, enkripsi, dan dekripsi dengan algoritma RSA. Hasil penelitian menunjukkan korelasi langsung antara panjang kunci RSA dan beban CPU serta waktu yang dibutuhkan untuk proses kriptografi. Ditemukan bahwa kunci RSA yang lebih panjang mengakibatkan peningkatan penggunaan sumber daya komputasi. Misalnya, kunci RSA 1024 bit menggunakan rata-rata 10% beban CPU dan 53,1 MB memori, sedangkan kunci 2048 bit meningkat menjadi 11,6% beban CPU dan 53,4 MB memori. Kunci 4096 bit memerlukan 10,6% beban CPU dengan penggunaan memori yang serupa, namun waktu proses meningkat dari 2,5 detik menjadi 27,7 detik. Kesimpulan dari penelitian ini memberikan wawasan penting bagi pengembangan aplikasi keamanan di Windows 11, khususnya dalam keseimbangan antara panjang kunci dan penggunaan sumber daya. Disarankan penelitian selanjutnya untuk mengevaluasi kinerja RSA pada berbagai konfigurasi perangkat keras.

Kata Kunci: Kriptografi RSA, Windows 11, Python, NZXT CAM, Penggunaan Sumber Daya.

Abstract

This research aims to analyze the performance of RSA cryptography on the Windows 11 operating system, implemented using Python and the NZXT CAM application. A quantitative approach was employed, utilizing a processor with specifications of 2.80GHz, 8 cores, and 16 GB RAM. Data were collected through the NZXT CAM benchmark application, measuring CPU and memory usage during RSA key generation, encryption, and decryption processes. The findings reveal a direct correlation between the length of the RSA key and both CPU load and the time required for cryptographic processes. It was found that longer RSA keys result in increased computational resource usage. For instance, a 1024-bit RSA key averages 10% CPU load and 53.1 MB memory, while a 2048-bit key increases to 11.6% CPU load and 53.4 MB memory. A 4096-bit key requires 10.6% CPU load with similar memory usage, but the processing time increases from 2.5 seconds to 27.7 seconds. The conclusion of this study provides significant insights for the development of security applications on Windows 11, particularly in balancing key length and resource usage. Further research is recommended to evaluate RSA performance on various hardware configurations.

***Key Words*:** RSA Cryptography, Windows 11, Python, NZXT CAM, Resource Utilization.

1. Pendahuluan

Keamanan memiliki peran penting dalam memfasilitasi adopsi teknologi dan aplikasi yang lebih luas dalam berbagai aspek kehidupan [1]. Pada saat yang sama, ilmu kriptografi saat ini sangat berfokus pada dua hal utama, yaitu keamanan data dan kecepatan dalam mentransmisikan informasi [2]. Pada saat yang sama, ilmu kriptografi saat ini sangat berfokus pada dua hal utama, yaitu keamanan data dan kecepatan dalam mentransmisikan informasi [3].

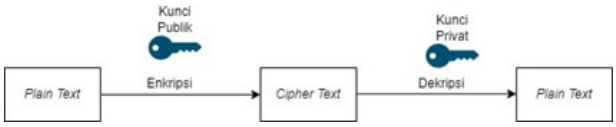
Di era digital saat ini, keamanan data menjadi aspek kritis yang tidak dapat diabaikan. Dengan berkembangnya teknologi informasi, metode kriptografi asimetris seperti RSA (*Rivest–Shamir–Adleman*)

semakin banyak digunakan untuk mengamankan data [4]. RSA sendiri merupakan gabungan dari ketiga nama penemu algoritma kriptografi tersebut [5]. RSA, sebagai salah satu algoritma kunci publik yang paling dominan, penting untuk diamati, terutama dalam hal penggunaan sumber daya komputasi seperti alokasi memori. Selain itu, keamanan juga merupakan aspek penting dalam implementasi algoritma RSA [6]. Algoritma RSA menggunakan dua jenis kunci, yaitu kunci publik dan kunci rahasia, di mana kunci publik dapat diketahui oleh siapa saja dalam proses enkripsi dan digunakan untuk mengamankan data, sedangkan kunci rahasia harus dijaga kerahasiaannya dan hanya boleh diketahui oleh pihak tertentu yang berwenang [7].

Studi yang dilakukan oleh Anwar et al. (2018) membandingkan performa keamanan antara algoritma AES 256 bit dan RSA, yang dapat memberikan pemahaman lebih mendalam terkait keamanan data [8]. Selain itu, implementasi skema SecLaaS-RW dalam pembuatan aplikasi *secure logging* oleh Victoaraya dan Cahyono juga memberikan perspektif terkait infrastruktur keamanan yang relevan dengan analisis performa kriptografi RSA [9].

Dari perspektif implementasi, penelitian yang dilakukan oleh Anwar et al. (2018) mengenai pengujian penyandian data yang disimpan dalam file dokumen dengan metode RSA dan AES menggunakan Python memberikan gambaran terkait implementasi kriptografi dalam bahasa pemrograman Python [8]. Studi yang dilakukan oleh Gupta (2023) membahas penggunaan algoritma konsensus berbasis RSA untuk komunikasi dan konsensus yang aman di antara perangkat terdistribusi. Penelitian ini menunjukkan penerapan langsung dari algoritma RSA dalam konteks komunikasi yang aman [10]. Hal ini menunjukkan bahwa Python dapat menjadi pilihan yang baik dalam mengimplementasikan algoritma RSA. Dengan demikian, implementasi algoritma RSA menggunakan Python merupakan topik yang menarik dan terus berkembang dalam bidang keamanan komputer.

Perumusan masalah penelitian ini terfokus pada bagaimana operasi kriptografi RSA mempengaruhi alokasi memori di Windows 11, serta implikasinya terhadap efisiensi dan keamanan sistem. Tingkat keamanan tinggi dari mekanisme enkripsi RSA, yang sangat penting dalam memahami implikasi RSA terhadap keamanan sistem [11]. Hal ini mendukung diskusi mengenai implikasi keamanan RSA dalam sistem Windows 11. Selain itu, penelitian ini bertujuan untuk memberikan analisis mendalam mengenai interaksi tersebut, memberikan pemahaman bagi pengembangan aplikasi keamanan yang lebih efisien dan efektif dalam penggunaan memori.



Gambar 1. Skema Algoritma RSA

2. Metode penelitian

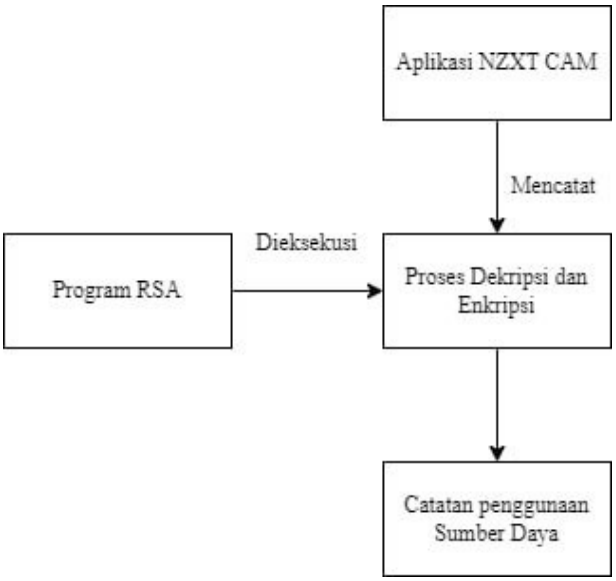
Pendekatan kuantitatif dilakukan untuk menganalisis pengaruh operasi kriptografi RSA terhadap alokasi memori pada sistem operasi Windows 11 yang memiliki spesifikasi prosesor 2.80 GHz dengan 8 inti dan memiliki RAM sebesar 16 GB. Pengumpulan data dilakukan melalui aplikasi benchmark, NZXT CAM, yang diaplikasikan untuk menilai penggunaan CPU dan memori selama proses generasi kunci, enkripsi, dan dekripsi dengan algoritma RSA. NZXT CAM dikonfigurasi untuk mengukur parameter seperti penggunaan CPU dan waktu yang diperlukan untuk proses kriptografi, memberikan wawasan yang mendalam mengenai kinerja algoritma RSA di bawah kondisi berbeda.

2.1 Metode pengumpulan data, instrumen penelitian, dan metode pengujian

Dalam rangka memperoleh data yang valid dan representatif, panjang kunci RSA yang digunakan yaitu 1024, 2048, dan 4096 bit serta panjang *plaintext* yaitu 100, 150, dan 200 kata untuk menggambarkan dampak yang lebih luas dari penggunaan algoritma RSA pada sistem. Instrumen penelitian ini mengukur penggunaan memori dan CPU untuk setiap kombinasi parameter tersebut. Metode analisis data melibatkan evaluasi kuantitatif dari hasil yang dikumpulkan, memfokuskan pada pengaruh variabel seperti panjang kunci dan ukuran *plaintext* terhadap efisiensi dan kinerja RSA. Penelitian dilakukan menggunakan pustaka Crypto dalam Python untuk melakukan enkripsi dan dekripsi dengan algoritma RSA.

RSA bergantung pada prinsip matematika yang terkait dengan kesulitan memfaktorkan bilangan besar menjadi faktor prima, yang merupakan dasar dari keamanan algoritma ini. Pada Gambar 1 menggambarkan proses dasar komunikasi yang aman dengan RSA, di mana pengirim mengenkripsi pesan dengan kunci publik penerima, dan penerima mendekripsi pesan dengan kunci privat mereka. Kedua kunci tersebut memiliki nilai yang berbeda.

Proses dimulai dengan pembuatan pasangan kunci tersebut, di mana kunci publik dan kunci privat memiliki panjang 1024 bit. Kunci-kunci ini diekspor ke dalam bentuk *string* untuk kemudian dapat disimpan atau dibagikan. Selanjutnya, kunci publik diimpor kembali untuk melakukan proses enkripsi. Pesan yang akan diamankan dibaca dari *file* teks (”plaintext3.txt”) dan dipecah menjadi beberapa *string*. Setiap *string* pesan



Gambar 2. Diagram Alir Pengumpulan Data

dienkripsi menggunakan skema *PKCS1-OAEP*, menghasilkan data terenkripsi yang disimpan dalam sebuah daftar. Proses selanjutnya adalah melakukan dekripsi pada setiap elemen dalam daftar menggunakan kunci privat.

Hasil dekripsi kemudian dicetak, mengembalikan pesan ke bentuk semula. Penggunaan modul *memory-profiler* membantu dalam memprofil penggunaan memori pada fungsi *main()*, sementara modul timeit digunakan untuk mengukur waktu eksekusi dari awal hingga akhir fungsi tersebut. Keseluruhan, kode ini memberikan gambaran komprehensif tentang implementasi enkripsi dan dekripsi RSA dalam Python, sekaligus memanfaatkan alat profilisasi dan analisis kinerja untuk memahami secara detail penggunaan memori dan waktu eksekusi program.

2.2 Tahapan Penelitian

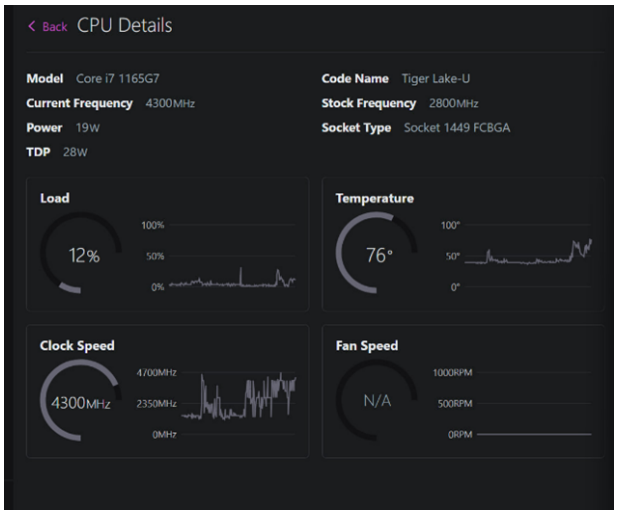
Melalui pendekatan sistematis, algoritma RSA dikembangkan dan diuji dengan memanfaatkan kunci yang panjangnya signifikan untuk mengolah data. Dengan mengintegrasikan aplikasi NZXT CAM untuk pemantauan, data dapat diakses *real-time* mengenai konsumsi sumber daya oleh CPU. Hal ini memberi pemahaman yang lebih baik tentang bagaimana operasi kriptografi memengaruhi kinerja dan manajemen memori komputer, sehingga memungkinkan analisis yang akurat dan menyeluruh atas aspek-aspek kritis dalam penggunaan algoritma RSA.

Program algoritma RSA dikembangkan dengan menggunakan Bahasa Python. Program tersebut dikonfigurasikan untuk memproses tiga *plaintext* dengan panjang yang berbeda, yaitu 100, 150, dan 200 kata, menggunakan kunci RSA dengan panjang yang bervariasi. Seiring dengan proses enkripsi dan dekripsi

```
Line #    Mem usage    Increment  Occurrences   Line Contents
=============================================================
    24     52.9 MiB     52.9 MiB           1   @profile
    25                                         def main():
    26     53.1 MiB      0.0 MiB         201       for element in plaintext:
    27     53.1 MiB      0.0 MiB         200           cipher = PKCS1_OAEP.new(key)
    28     53.1 MiB      0.1 MiB         200           ciphertext = cipher.encrypt(element)
    29     53.1 MiB      0.1 MiB         200           print(ciphertext, end=' ')
    30     53.1 MiB      0.0 MiB         200           cipher_list.append(ciphertext)
    31
    32     53.2 MiB      0.0 MiB         201       for element in cipher_list:
    33     53.2 MiB      0.1 MiB         200           private_key = RSA.importKey(privatekey)
    34     53.2 MiB      0.0 MiB         200           cipher = PKCS1_OAEP.new(private_key)
    35     53.2 MiB      0.0 MiB         200           decrypted_message = cipher.decrypt(element)
    36     53.2 MiB      0.0 MiB         200           print(decrypted_message, end=' ')

Lama eksekusi:  27.229763499984983 detik
```

Gambar 3. Program Algoritma RSA



Gambar 4. Aplikasi NZXT CAM

yang dilakukan oleh Program RSA, aplikasi NZXT CAM aktif berjalan untuk memonitor dan mencatat penggunaan CPU, sebagaimana ditunjukkan pada Gambar 2. Data yang dikumpulkan kemudian menyediakan dasar yang kuat untuk menganalisis pengaruh algoritma RSA terhadap alokasi memori pada sistem operasi Windows 11, dengan mencatat penggunaan sumber daya dan alokasi memori selama operasi kriptografi berlangsung.

Gambar 3 menunjukkan proses Algoritma RSA yang dijalankan melalui *command prompt* pada Windows. Dalam proses pengembangan, peneliti memanfaatkan sebuah program Python yaitu *library* pycryptodome untuk mengimplementasikan algoritma RSA dengan panjang kunci yang melebihi 1024 bits. Program yang dikembangkan juga menggunakan *library* timeit untuk mengukur *runtime* saat eksekusi. Analisis penggunaan memori menggunakan *library memory-profiler*. Program tersebut merupakan operasi enkripsi dan dekripsi menggunakan algoritma RSA. Serta penggunaan memori yang tidak mengalami lonjakan yang signifikan selama proses eksekusi.

NZXT CAM adalah aplikasi *monitoring* yang digunakan untuk mengawasi kinerja komputer, khususnya CPU. Aplikasi ini menyediakan informasi tentang CPU seperti tipe, frekuensi saat ini, suhu, penggunaan daya. Tampilan Aplikasi NZXT CAM

Tabel 1. Tabel Kondisi CPU *Idle*

Plaintext	CPU Load	CPU Freq	Time	Memori
-	5%	1400 MHz	-	-

Tabel 2. Tabel Pengujian RSA 1024 bit

Plaintext	CPU Load	CPU Freq	Time	Memori
100	8%	3300 MHz	1.7s	52.9 MB
150	10%	4500 MHz	2.7s	53.3 MB
200	12%	4100 MHz	3.2s	53.1 MB

ditunjukkan pada Gambar 4. Pengguna dapat menggunakan NZXT CAM untuk memantau kondisi CPU mereka, memastikan suhu tetap dalam batas aman, dan mengoptimalkan performa komputer. Pengguna juga dapat melihat grafik perubahan kecepatan CPU seiring waktu. NZXT CAM membantu pengguna memahami dan mengelola kinerja CPU dengan mudah dan efisien.

3. Hasil dan Pembahasan

Pengaruh operasi kriptografi RSA terhadap beban CPU dan alokasi memori pada sistem operasi Windows 11 dianalisis dengan menggunakan aplikasi NZXT CAM. Tabel 1 hingga Tabel 4 di bawah ini menyajikan data yang dikumpulkan.

Tabel 1 menunjukkan kondisi dasar CPU sebelum operasi kriptografi RSA dimulai, dengan beban CPU sebesar 5% dan frekuensi CPU 1400MHz. Kondisi *idle* ini penting sebagai titik acuan untuk memahami peningkatan beban dan kecepatan CPU selama enkripsi dan dekripsi.

Hasil pada Tabel 2, pengujian dengan kunci RSA 1024 bit, menunjukkan bahwa beban CPU meningkat seiring dengan panjang *plaintext* yang dienkripsi. Untuk *plaintext* sepanjang 100 kata, CPU terbebani 8% dengan kecepatan jam 3300MHz, sedangkan untuk 200 kata, beban meningkat menjadi 12% dengan kecepatan 4100MHz. Waktu yang diperlukan untuk enkripsi dan dekripsi juga meningkat dari 1,7 detik menjadi 3,2 detik.

Hasil pada Tabel 3 menunjukkan peningkatan panjang kunci menjadi 2048 bit menghasilkan peningkatan beban CPU dan waktu proses yang lebih tinggi. Untuk 100 kata, beban CPU adalah 6% dengan waktu 4,3 detik, dan untuk 200 kata, beban meningkat menjadi 12% dengan waktu 15,8 detik, menunjukkan hubungan yang eksponensial antara panjang kunci dan sumber daya yang digunakan.

Pada Tabel 4 pengujian RSA dengan kunci 4096 bit, beban CPU dan waktu proses mencapai puncaknya, dari 9% dan 18,3 detik untuk 100 kata, hingga 12% dan 37,4 detik untuk 200 kata. Hal ini mengindikasikan bahwa panjang kunci memiliki pengaruh yang signifikan terhadap kinerja sistem selama operasi kriptografi.

Tabel 3. Tabel Pengujian RSA 2048 bit

Plaintext	CPU Load	CPU Freq	Time	Memori
100	6%	4500 MHz	4.3s	53.4 MB
150	10%	4700 MHz	9.8s	53.5 MB
200	12%	4300 MHz	15.8s	53.3 MB

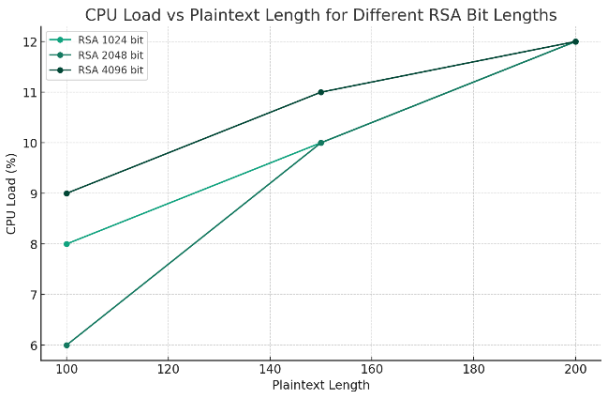
Tabel 4. Tabel Pengujian RSA 4096 bit

Plaintext	CPU Load	CPU Freq	Time	Memori
100	9%	4600 MHz	18.3s	53.5 MB
150	11%	4700 MHz	27.6s	53.4 MB
200	12%	4300 MHz	37.4s	53.5 MB

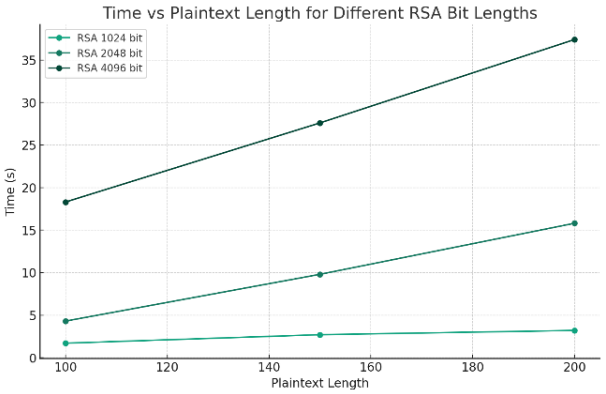
Analisis perbandingan dari Tabel 1, 2, 3, dan 4 adalah bahwa semakin besar panjang kunci RSA dan ukuran *plaintext*, semakin tinggi beban CPU yang diperlukan dan waktu yang dibutuhkan untuk memproses RSA. Selain itu, frekuensi CPU juga mempengaruhi waktu yang dibutuhkan dalam pengujian, dengan frekuensi CPU yang lebih tinggi cenderung menghasilkan waktu yang lebih singkat. Namun, perlu dicatat bahwa beban CPU dan waktu yang dibutuhkan akan bervariasi tergantung pada spesifikasi perangkat keras yang digunakan.

Beban CPU meningkat seiring dengan peningkatan panjang *plaintext* yang ditunjukkan pada Gambar 5. Pada RSA 1024 bit, kenaikan beban CPU terlihat lebih curam dibandingkan dengan RSA 2048 dan 4096 bit. Sedangkan, RSA 4096 bit menunjukkan beban CPU yang lebih stabil dibandingkan dengan yang lain, meskipun ada sedikit kenaikan.

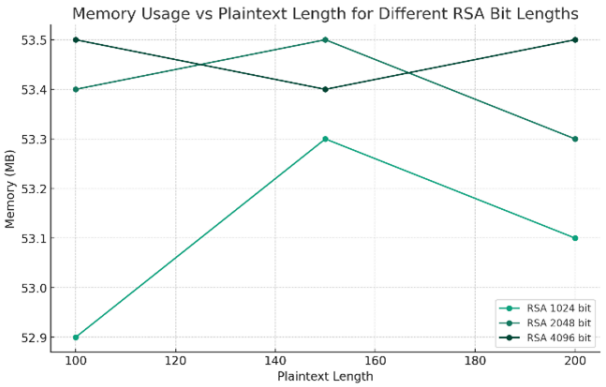
Berdasarkan grafik pada Gambar 6, waktu pemrosesan meningkat secara signifikan dengan panjang bit RSA yang lebih besar. Untuk setiap panjang bit RSA, ada peningkatan waktu pemrosesan yang linier seiring



Gambar 5. Grafik Pengujian RSA (*CPU Load vs Plaintext Length*)



Gambar 6. Grafik Pengujian RSA (*Time vs Plaintext Length*)

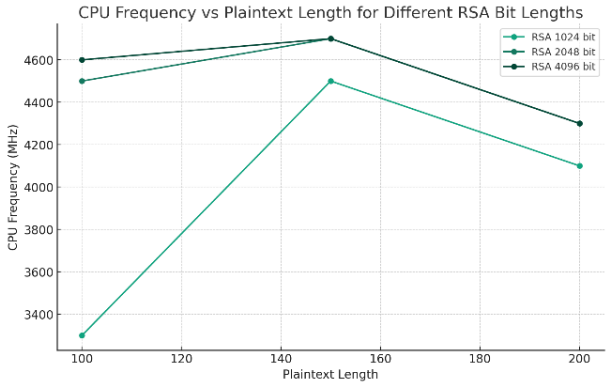


Gambar 7. Grafik Pengujian RSA (*Memory vs Plaintext Length*)

dengan bertambahnya panjang *plaintext*. RSA 4096 bit membutuhkan waktu pemrosesan yang jauh lebih lama dibandingkan dengan RSA 1024 dan 2048 bit, menunjukkan kompleksitas komputasi yang lebih tinggi.

Berdasarkan Gambar 7, penggunaan memori relatif konstan dan tidak menunjukkan variasi signifikan untuk semua panjang bit RSA dan panjang *plaintext* yang diuji. Hal ini menunjukkan bahwa penggunaan memori tidak terlalu dipengaruhi oleh panjang bit RSA atau panjang *plaintext* dalam kisaran data yang ada.

Gambar 8 menampilkan grafik perbandingan antara panjang plaintext dan frekuensi CPU untuk setiap panjang bit RSA. Frekuensi CPU dapat naik atau turun karena terdapat faktor seperti CPU yang panas sehingga sistem menerapkan perlambatas otomatis (*throttling*) untuk mencegah *overheating*, selain itu jika CPU digunakan secara intensif dan hampir mencapai kapasitas maksimal, sistem secara otomatis menyesuaikan frekuensi CPU ntuk menghindari *overloading* dan menjaga keseimbangan kinerja. Proses enkripsi dan dekripsi tidak mengakibatkan lonjakan penggunaan memori yang berlebihan.



Gambar 8. Grafik Pengujian RSA (*CPU Frequency vs Plaintext Length*)

Setiap grafik mewakili aspek yang berbeda dari metrik kinerja yang diamati selama proses enkripsi RSA dengan panjang kunci yang berbeda. Pembahasan data di atas menunjukkan bahwa terdapat korelasi langsung antara panjang kunci RSA dan beban CPU serta waktu yang dibutuhkan untuk proses kriptografi. Penelitian ini berhasil memperlihatkan bahwa dengan meningkatnya panjang kunci dan panjang *plaintext*, terdapat peningkatan signifikan pada penggunaan sumber daya sistem.

Selanjutnya, penelitian ini dapat diperluas dengan memasukkan faktor-faktor lain seperti pengaruh jumlah proses yang berjalan di latar belakang, penggunaan sistem operasi lain, serta evaluasi pada perangkat keras dengan spesifikasi yang berbeda. Langkah tindak lanjut yang akan dilakukan mencakup analisis dampak jangka panjang dari operasi RSA pada sistem, serta optimasi algoritma untuk mengurangi beban sumber daya sistem.

4. Kesimpulan

Kinerja kriptografi RSA pada Windows 11 dengan implementasi menggunakan Python dan aplikasi NZXT CAM, secara signifikan dipengaruhi oleh panjang kunci dan ukuran *plaintext*. Hasil uji coba menegaskan bahwa kunci RSA yang lebih panjang menyebabkan peningkatan penggunaan sumber daya komputasi. Misalnya, kunci RSA 1024 bit menggunakan rata-rata 10% beban CPU dan 53,1 MB memori, sementara kunci 2048 bit meningkat menjadi 11,6% beban CPU dan 53,4 MB memori. Lebih lanjut, kunci 4096 bit memerlukan 10,6% beban CPU dengan penggunaan memori yang serupa. Dalam hal waktu proses, terjadi peningkatan dari 2,5 detik untuk kunci 1024 bit menjadi 27,7 detik untuk kunci 4096 bit.

Temuan ini memberikan wawasan penting bagi pengembangan aplikasi keamanan di Windows 11, khususnya dalam hal keseimbangan antara panjang kunci dan penggunaan sumber daya. Walaupun peningkatan panjang kunci RSA berkontribusi pada peningkatan

keamanan, ini juga berarti tuntutan yang lebih besar terhadap sumber daya komputasi. Oleh karena itu, sangat penting untuk mengeksplorasi keseimbangan antara keamanan dan efisiensi sumber daya dalam pengembangan aplikasi keamanan yang menggunakan RSA. Disarankan pula agar penelitian selanjutnya mengevaluasi kinerja RSA pada berbagai konfigurasi perangkat keras untuk menentukan kondisi optimal dalam kinerja kriptografi, memperhatikan peran Python sebagai alat implementasi yang fleksibel dan efisien dalam lingkungan Windows 11.

Daftar Pustaka

[1] M. Suresh and M. Neema, "Hardware implementation of blowfish algorithm for the secure data transmission in internet of things," *Procedia Technology*, vol. 25, pp. 248–255, 2016.

[2] D. Smekal, J. Frolka, and J. Hajny, "Acceleration of aes encryption algorithm using field programmable gate arrays," *IFAC-PapersOnLine*, vol. 49, no. 25, pp. 384–389, 2016.

[3] T. Arianti and B. Nadeak, "Perancangan aplikasi pembelajaran kriptografi algoritma gost dengan menggunakan metode computer based instruction," *KAKIFIKOM (Kumpulan Artikel Karya Ilmiah Fakultas Ilmu Komputer)*, vol. 01, no. 01, pp. 40–46, 2019. [Online]. Available: https://ejournal.ust.ac.id/index.php/KAKIFIKOM/article/view/626

[4] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 2019.

[5] F. Ariyanto and S. Suprihadi, "Implementasi digital signature dan quick response code pada aplikasi kuitansi digital," *JIKO (Jurnal Informatika dan Komputer)*, vol. 5, pp. 125–131, 2022. [Online]. Available: https://ejournal.unkhair.ac.id/index.php/jiko/article/view/4770

[6] I. B. G. Sarasvananda and I. B. A. I. Iswara, "Tanda tangan elektronik menggunakan algoritma rivest shamir adleman (rsa) pada sistem informasi surat menyurat lpik instiki," *Jurnal Sisfokom (Sistem Informasi dan Komputer)*, vol. 11, no. 2, pp. 289–296, 2022.

[7] R. Y. Rifai, Y. Christyono, and I. Santoso, "Implementasi algoritma kriptografi rivest code 4, rivest shamir adleman, dan metode steganografi untuk pengamanan pesan rahasia pada berkas teks digital," *Transient: Jurnal Ilmiah Teknik Elektro*, vol. 5, no. 1, pp. 86–91, 2016. [Online]. Available: https://ejournal3.undip.ac.id/index.php/transient/article/view/12032

[8] N. Anwar, M. Munawwar, M. Abduh, and N. B. Santosa, "Komparatif performance model keamanan menggunakan metode algoritma aes 256 bit dan rsa," *Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi)*, vol. 2, no. 3, pp. 783–791, 2018.

[9] G. VictoAraya and S. Cahyono, "Implementasi skema seclaas-rw dalam membuat aplikasi secure logging," *Info Kripto*, vol. 16, no. 3, pp. 85–94, 2022.

[10] N. Gupta and A. K. Jain, "Rsa based consensus algorithm for resource-constrained distributed devices," *Internet Technology Letters*, vol. 6, no. 6, 2023.

[11] Z. Y. Fu, "Computer cyberspace security mechanism supported by cloud computing," *PLoS ONE*, vol. 17, no. 10, 2022.