JUSIKOM PRIMA (Junal Sistem Informasi Ilmu Komputer Prima)
Vol. 3 No. 2, Februari 2020 e-ISSN : 2580-2879

Perancangan Aplikasi Enkripsi Data Menggunakan Algoritma XXTEA

*Bayu Angga Wijaya[1], Mawwadah Harahap[1], Siti Aisyah[2]
[1]Program Studi Tehnik Informatika
[2]Program Studi Sistem Informasi
Fakultas Teknologi dan Ilmu Komputer Universitas Prima Indonesia
E-mail : *bayuanggawijaya@unprimdn.ac.id

ABSTRAK- Keamanan telah menjadi aspek yang sangat penting untuk mengamankan data. Salah satu upaya pengamanan data adalah dengan kriptografi. Kriptografi adalah ilmu yang mempelajari bagaimana supaya pesan atau dokumen tetap aman, tidak dapat dibaca oleh pihak yang tidak berhak *(anauthorized persons)*. Pada penelitian ini algoritma kriptografi yang digunakan adalah algoritma *XXTEA (Corrected Block Tiny Encryption Algorithm)* untuk melakukan pengamanan data pada proses pengiriman data aplikasi E-Surat. Sistem ini dibuat dengan menggunakan *Node.js* dengan bahasa pemrograman *Javascript* dan *Express* sebagai kerangka kerjanya. Uji performa pada penelitian ini dibagi menjadi dua kasus, yaitu uji performa aplikasi dan uji performa algoritma. Uji performa aplikasi menunjukan jumlah *request per second* yang dapat dihasilkan pada penggunaan 10 *node CPU* tidak stabil dan memiliki ruang nilai yang rendah. Nilai tertinggi *request per second* yang dapat dihasilkan adalah sebesar 26 *request* yaitu saat nilai *concurent*-nya 4 dan 256. Sedangkan nilai terendah *request per second*-nya adalah 20 yaitu saat nilai *concurent*-nya 512 Dari hasil uji performa algoritma *XXTEA*, dapat disimpulkan bahwa waktu enkripsi dan dekripsi pada algoritma *XXTEA* relatif cepat. Rata-rata waktu yang digunakan *XXTEA* untuk mengenkripsi pesan adalah 2.23987272 *ms*. Sedangkan, rata-rata waktu yang dihasilkan algoritma XXTEA untuk mendekripsi pesan adalah 2.05297956 *ms.*

Kata kunci : Kriptografi, algoritma XXTEA, enkripsi, dekripsi.

1. PENDAHULUAN

Data merupakan kumpulan fakta-fakta dari sumber yang telah didapatkan. Sebuah data pada umumnya hanya ditujukan bagi segolongan individu tertentu.

Salah satu cara untuk melindungi data adalah dengan teknik enkripsi (encryption) yaitu sebuah proses yang melakukan perubahan sebuah kode dari yang bisa dimengerti menjadi sebuah kode yang tidak bisa dimengerti, sehingga data yang sudah dienkripsi tersebut tidak akan dapat dimanipulasi oleh orang yang tidak berhak untuk mengakses data tersebut.

Salah satu upaya pengamanan data adalah dengan kriptografi. Kriptografi adalah ilmu yang mempelajari bagaimana supaya pesan atau dokumen tetap aman, tidak dapat dibaca oleh pihak yang tidak berhak (anauthorized persons). Kriptografi terdiri dari dua proses yaitu enkripsi dan dekripsi. Proses enkripsi dilakukan agar data tidak dapat diketahui oleh pihak yang berwenang, dengan cara mengubah data asli menjadi suatu kode dengan suatu kunci tertentu yang tidak dapat dipecahkan oleh orang lain. Data yang telah dienkripsi kemudian dikirimkan kepada pihak penerima, penerima kemudian mendekripsikan data yang telah dienkripsi oleh pihak pengirim agar mengetahui isi data yang sebenarnya. Pada penelitian ini algoritma kriptografi yang digunakan adalah algoritma XXTEA (Corrected Block Tiny Encryption Algorithm) untuk melakukan pengamanan data pada proses pengiriman data aplikasi E-Surat.

2. METHOD

Adapun metode yang diperlukan penulis untuk penelitian ini yaitu :

a. Studi Lapangan
   Dengan melakukan pengumpulan data dan informasi dengan cara tanya jawab kepada pihak instansi terkait dan perwakilannya.
b. Studi Pustaka
   Metode ini dilakukan untuk mendapatkan sumber-sumber kajian, landasan teori, pengumpulan data, informasi pengolahan data,penarikan kesimpulan, saran dan impikasinya sebagai bahan penelitian.

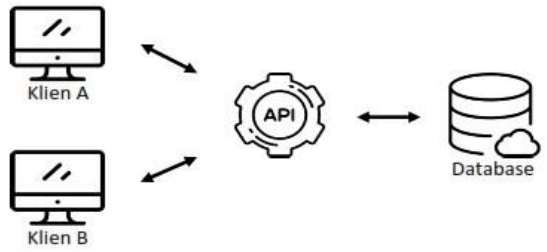
Implementasi Data Mining ini dapat digunakan sebagai penunjang keputusan pada pemilik perusahaan dalam mengambil keputusan untuk menentukan persediaan baja ringan pada bulan berikutnya berdasarkan data-data penjualan barang yang laris

3. LANGKAH-LANGKAH PENELITIAN

a. Desain Sistem

Desain umum pada sistem ini adalah membuat aplikasi surat elektronik sederhana yang dalam proses pengirimannya terenkripsi menggunakan algoritma *XXTEA*. Sistem ini dibuat dengan menggunakan *Node.js* dengan bahasa pemrograman *Javascript* dan *Express* sebagai kerangka kerjanya. Skenario penggunaan aplikasi sebenarnya tidak berbeda dengan pengiriman *email* biasa. Klien A membuat pesan baru dan mengirimkan pesan tersebut kepada klien B. Kemudian klien B membuka pesan yang telah masuk di *inbox* klien B.

Apabila pada pesan tersebut terdapat lampiran, maka klien B dapat mengunduh lampiran yang tertera.



Gambar 1. Desain Sistem Secara Umum

Proses dibalik layar yang terjadi saat klien A mengirimkan pesan ke klien B adalah klien A melakukan *HTTP request* ke *server REST* untuk mengirim pesan. Kemudian *REST* merespon dan terjadi proses enkripsi untuk mengamankan isi pesan yang dikirim. Ketika klien B ingin membuka pesan yang baru masuk, klien B melakukan *request* ke *server* untuk melakukan dekripsi pesan. Kemudian *server REST* akan merespon dengan memberikan pesan dari basis data yang telah terdekripsi kepada klien B. Proses enkripsi ini dilakukan pada *client-side* atau sebutannya adalah *end-to- end encryption*. Sebelumnya, perlu diketahui bahwa kunci yang digunakan untuk mengenkripsi dan mendekripsi didapatkan dari rumus berikut:

Email A + *Email* B + Waktu pesan + TA2017

Keterangan:

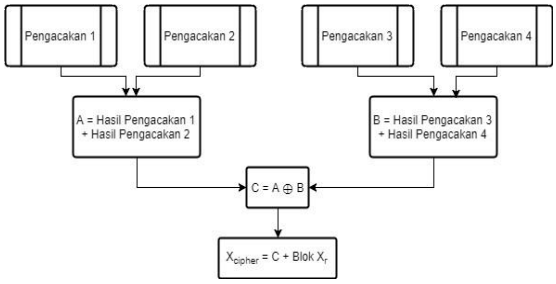
1. *Email* A : alamat *email* pengirim
2. *Email* B : alamt *email* penerima
3. Waktu pesan : waktu pesan tersebut dikirim
4. TA2017 : suatu *string* tetap sebagai penanda

Waktu pesan terkirim dijadikan *variabel* agar kunci yang dihasilkan pada setiap pengiriman pesan berbeda-beda. *String* TA2017 dijadikan sebagai *variabel* tetap agar *string* kunci nantinya memiliki identitas tersendiri.

Empat komponen ini kemudian digabung atau di-*concat* untuk dijadikan sebuah *string* yang panjang. Setelah itu *string* ini diurutkan dari yang secara alfabetik. Setelah diurutkan, *string* ini digunakan sebagai kunci untuk mengenkripsi dan mendekripsi isi pesan.

Proses enkripsi dimulai dari mengambil blok sebelum (X_{r-1}) dan sesudah (X_{r+1}) blok yang sedang diacak, kata kunci dan DELTA sebagai *input*. Kemudian terjadi proses pengacakan pertama yaitu blok sebelum di-*shift left* 2 ($X_{r-1} << 2$) di-*XOR*-kan dengan blok sesudah yang di-*shift right* 5 ($X_{r+1} >> 5$).

Proses enkripsi dimulai dari mengambil blok sebelum (X_{r-1}) dan sesudah (X_{r+1}) blok yang sedang diacak, kata kunci dan DELTA sebagai *input*. Kemudian terjadi proses pengacakan pertama yaitu blok sebelum di-*shift left* 2 ($X_{r-1} << 2$) di-*XOR*-kan dengan blok sesudah yang di-*shift right* 5 ($X_{r+1} >> 5$). Hasil dari pengacakan pertama disimpan dengan nama hasil pengacakan 1. Proses pengacakan kedua juga diawali dengan mengambil blok sebelum (X_{r-1}) dan sesudah (X_{r+1}) blok yang sedang diacak, kata kunci dan DELTA sebagai *input*. Kemudian terjadi proses pengacakan kedua yaitu blok sebelum di-*shift right* 3 ($X_{r-1} >> 3$) di-*XOR*-kan dengan blok sesudah yang di-*shift left* 4 ($X_{r+1} << 4$). Hasil dari pengecekan pertama disimpan dengan nama hasil pengacakan 2. Sama seperti proses pengacakan pertama dan kedua, proses pengacakan ketiga dimulai dengan mengambil blok sebelum (X_{r-1}) dan sesudah (X_{r+1}) blok yang sedang diacak, kata kunci dan DELTA sebagai input. Kemudian terjadi proses pengacakan ketiga yaitu blok sebelum (X_{r-1}) di-*XOR*-kan dengan DELTA yang merupakan perkalian antara konstanta DELTA yang bernilai 0x9E3779B dengan jumlah iterasi pertama yang telah dilakukan (DELTA = q * 0x9E3779B). Hasil dari pengacakan ketiga disimpan dengan nama hasil pengacakan 3. Dengan awal yang sama seperti pengacakan sebelum - sebelumnya, blok sesudah (X_{r+1}) di-*XOR*-kan salah satu blok kata kunci ke-r yang sebelumnya telah di-*XOR*-kan dengan DELTA yang di-*shift right* 2 (K_r *XOR* (DELTA >> 2)). Hasil dari pengacakan keempat disimpan dengan nama hasil pengacakan 4.



Gambar 2. Proses Enkripsi

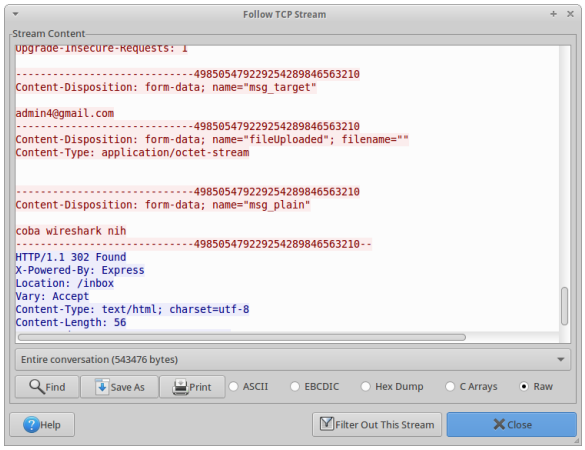
Gambar 2 merupakan gambar yang menunjukkan keseluruhan tahap dari proses enkripsi. Diagram alir ini merupakan gabungan dari seluruh tahap enkripsi. Pertama, hasil dari proses pengacakan 1 ditambahkan dengan hasil dari proses pengacakan 2 dan disimpan dengan variabel A. Hal yang sama juga terjadi pada hasil dari proses pengacakan 3 dan 4. Hasil dari proses pengacakan 3 ditambahkan dengan hasil dari proses pengacakan 4 dan disimpan dengan variabel B. Kedua, hasil jumlah pengacakan 1 dan 2 (variabel A) di-*XOR*-kan dengan hasil jumlah pengacakan 3 dan 4 (variabel B) yang kemudian disimpan variabel C. Terakhir, hasil *XOR* tersebut (variabel C) ditambahkan dengan blok yang sedang diacak. Pada proses dekripsi, iterasi pengacakan dimulai dari blok yang paling terakhir pada deret blok dan iterasi pengacakan berjalan mundur. Untuk proses pengacakan pada dekripsi tidak berbeda dengan proses pengacakan pada enkripsi seperti yang telah dijelaskan sebelumnya.

b. Hasil

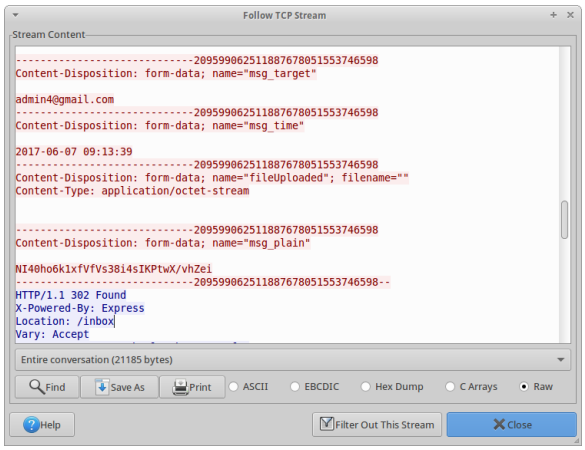
1) Hasil Uji Fungsionalitas

Uji fungsionalitas yang pertama dilakukan menggunakan peramban *Mozilla Firefox* pada komputer penguji dan yang kedua menggunakan perangkat lunak *Wireshark* untuk melakukan *sniffing* paket data.

Uji fungsionalitas yang kedua menggunakan aplikasi *sniffing* data yaitu *Wireshark*. Gambar 3.1 di bawah ini menunjukkan bahwa *msg_plain*-nya belum terenkripsi yaitu "coba *wireshark* nih". Sedangkan pada Gambar 3.2 menunjukkan bahwa *msg_plain*-nya telah terenkripsi menjadi "NI40ho6k1xfVfVs38i4sIKPtwX/vhZei" yang berarti hasil uji coba fungsionalitas ini berhasil.



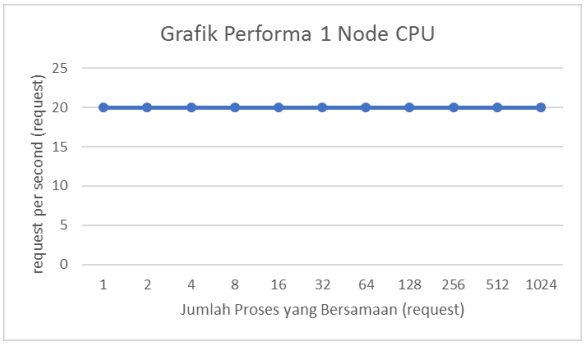
Gambar 3. Hasil Uji Fungsionalitas Menggunakan *Wireshark* Belum Terenkripsi



Gambar 4. Hasil Uji Fungsionalitas Menggunakan *Wireshark* Sudah Terenkripsi

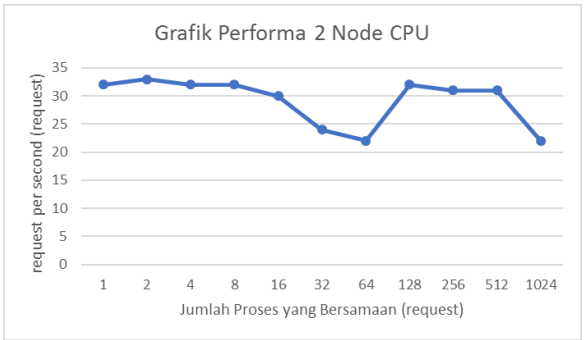
2) Hasil Pengujian Performa

Uji performa ini dilakukan dengan cara menjalankan *file loadtest.js*. Aplikasi ini akan diuji dengan menjalankan *server* yang terbagi beban pekerjaannya hingga ke 8 *node CPU virtual*. Hasil uji performa ini akan digambarkan dalam bentuk grafik. Berikut dapat dilihat grafik pada Gambar 5 hingga Gambar 12.



Gambar 5. Grafik Performa 1 *Node CPU*

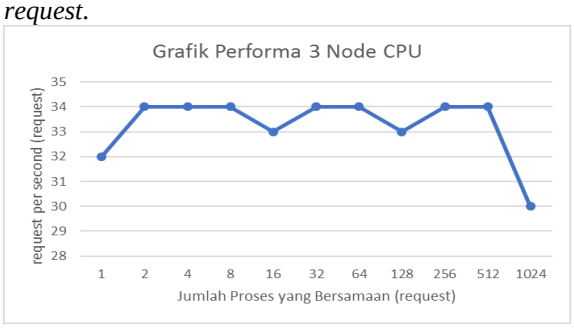
Dapat dilihat pada grafik Gambar 5 bahwa pada penggunaan 1 *node CPU* jumlah *request per second* yang dapat dihasilkan untuk setiap nilai *concurent*-nya yang telah ditentukan adalah 20.



Gambar 6. Grafik Performa 2 *Node CPU*

Grafik pada Gambar 6 menunjukan jumlah *request per second* yang dapat dihasilkan pada penggunaan 2 *node CPU* adalah tidak stabil. Nilai tertinggi *request per second* yang dapat dihasilkan adalah sebesar 33 *request* yaitu saat nilai *concurent*-nya 2. Sedangkan nilai terendah *request per second*-nya adalah 22 yaitu saat nilai *concurent*-nya 62 dan 1024.

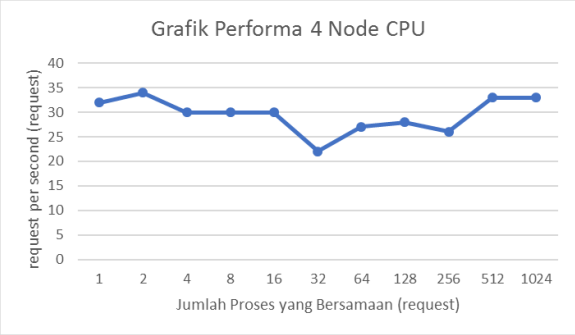
Penjelasan lebih lengkap mengenai Gambar 6 akan dijelaskan sebagai berikut. Pada saat nilai *concurent*-nya 1, 4, 8 dan 128 *request per second* yang dihasilkan adalah 32 *request*. Ketika nilai *concurrent*-nya 16, *request per second*-nya adalah 30 *request*. Saat nilai *concurrent*-nya 32, *request per second* yang dihasilkan adalah sebesar 24 *request*. Ketika nilai *concurrent*-nya 256 dan 512, *request per second* yang dihasilkan adalah sebesar 31 *request*.



Gambar 7. Grafik Performa 3 Node CPU

Dapat dilihat pada grafik gambar 7 bahwa pada penggunaan 3 *node CPU* akan menghasilkan jumlah *request per second* yang cukup stabil. Saat nilai *concurent*-nya 1 jumlah *request per second* yang dapat dihasilkan adalah 32. Jumlah *request per second* naik hingga mencapai nilai maksimalnya yaitu 34 permintaan. Kemudian nilai *request per second* tidak mengalami kenaikan dan penurunan yang drastis kecuali pada saat nilai *concurent*-nya 1024. Pada saat nilai *concurrent*-nya 1024 jumlah *request per second* dihasilkan oleh aplikasi adalah 30.

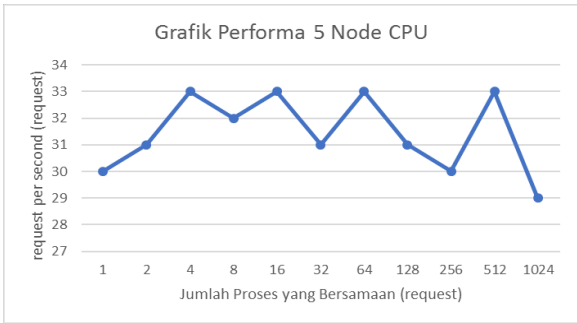
Penjelasan lebih lengkap mengenai gambar 7 akan dijelaskan sebagai berikut. Pada saat nilai *concurent*-nya 16 dan 128, *request per second* yang dihasilkan adalah 33 *request*. Ketika nilai *concurrent*-nya 2, 4, 8, 32, 64, 256 dan 512 *request per second*-nya adalah 34 permintaan per detik.



Gambar 8. Grafik Performa 4 *Node CPU*

Grafik pada Gambar 8 menunjukan jumlah *request per second* yang dapat dihasilkan pada penggunaan 2 *node CPU* adalah tidak stabil. Terdapat kenaikan dan penurunan nilai *request per second* yang cukup signifikan. Nilai tertinggi *request per second* yang dapat dihasilkan adalah sebesar 34 *request* yaitu saat nilai *concurent*-nya 2. Sedangkan nilai terendah *request per second*-nya adalah 22 yaitu saat nilai *concurent*-nya 32.

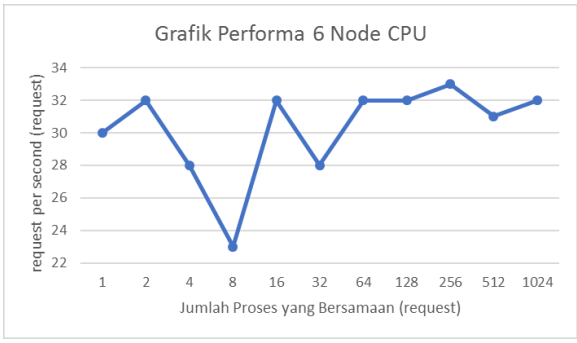
Penjelasan lebih lengkap mengenai Gambar 8 akan dijelaskan sebagai berikut. Pada saat nilai *concurent*-nya 1, *request per second* yang dihasilkan adalah 32 *request*. Ketika nilai *concurrent*-nya 4, 8, 16 *request per second*-nya adalah 30 *request*. Saat nilai *concurrent*-nya 64, *request per second* yang dihasilkan adalah sebesar 27 *request*. Pada saat nilai *concurent*-nya 128 dan 256, *request per second* yang dihasilkan adalah 28 dan 26. Ketika nilai *concurrent*-nya 512 dan 1024, *request per second* yang dihasilkan adalah sebesar 33 *request*.



Gambar 9. Grafik Performa 5 *Node CPU*

Dapat dilihat pada grafik Gambar 9 bahwa pada penggunaan 5 *node CPU* akan menghasilkan jumlah *request per second* yang cukup stabil karena nilai yang dihasilkan berkisar dari angka 29 hingga 34. Saat nilai *concurent*-nya 1 jumlah *request per second* yang dapat dihasilkan adalah 30. Jumlah *request per second* naik hingga mencapai nilai maksimalnya yaitu 33 permintaan saat nilai *concurent*-nya 4, 16, 64, dan 512. Nilai terendah drastik kecuali pada saat nilai *concurent*-nya 1024. Pada saat nilai *concurrent*-nya 1024 jumlah *request per second* dihasilkan oleh aplikasi adalah 29.

Penjelasan lebih lengkap mengenai Gambar 9 akan dijelaskan sebagai berikut. Pada saat nilai *concurent*-nya 1, 256 *request per second* yang dihasilkan adalah 30 *request*. Ketika nilai *concurrent*-nya 2, 32, 128 *request per second*-nya adalah 31 *request*. Saat nilai *concurrent*-nya 8, *request per second* yang dihasilkan adalah sebesar 32 *request*.

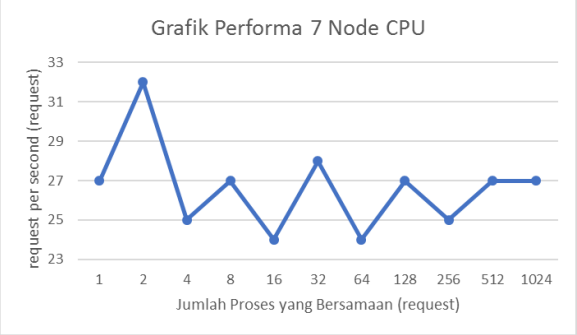


Gambar 10. Grafik Performa 6 *Node CPU*

Grafik pada Gambar 10 menunjukan jumlah *request per second* yang dapat dihasilkan pada penggunaan 2 *node CPU* adalah tidak stabil. Terdapat kenaikan dan penurunan nilai *request per second* yang cukup drastis. Nilai tertinggi *request per second* yang dapat dihasilkan adalah sebesar 33 *request* yaitu saat nilai *concurent*-nya 256. Sedangkan nilai terendah *request per second*-nya adalah 23 yaitu saat nilai *concurent*-nya 8.

Penjelasan lebih lengkap mengenai Gambar 10 akan dijelaskan sebagai berikut. Pada saat nilai *concurent*-nya 2, 16, 64, 128 dan 1024 *request per second* yang dihasilkan adalah 32 *request*. Ketika nilai *concurrent*-nya 4 dan 32 *request per second*-

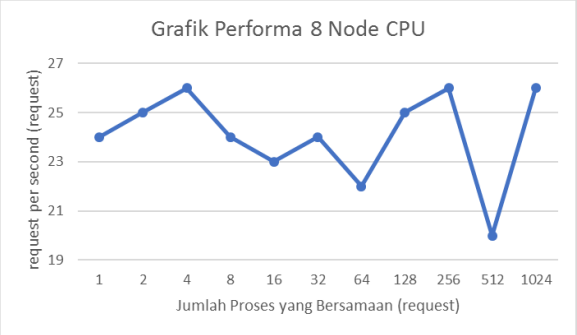
nya adalah 28 *request*. Saat nilai *concurrent*-nya 1, *request per second* yang dihasilkan adalah sebesar 30 *request*. Ketika nilai *concurrent*-nya 512, *request per second* yang dihasilkan adalah sebesar 31 *request*.



Gambar 11. Grafik Performa 7 *Node CPU*

Dapat dilihat pada grafik Gambar 11 bahwa pada penggunaan 7 *node CPU* akan menghasilkan jumlah *request per second* yang tidak stabil karena nilai yang dihasilkan berkisar dari angka 24 hingga 32. Saat nilai *concurent*-nya 1 jumlah *request per second* yang dapat dihasilkan adalah 27. Jumlah *request per second* naik hingga mencapai nilai maksimalnya yaitu 32 permintaan saat nilai *concurent*-nya 2. Nilai terendah yang dihasilkan adalah 24 yaitu pada saat nilai *concurent*-nya 16 dan 64.

Penjelasan lebih lengkap mengenai Gambar 11 akan dijelaskan sebagai berikut. Pada saat nilai *concurent*-nya 4 dan 256 *request per second* yang dihasilkan adalah 25 *request*. Ketika nilai *concurrent*-nya 8, 128, 512, dan 1024, *request per second*-nya adalah 27 *request*. Saat nilai *concurrent*-nya 32, *request per second* yang dihasilkan adalah sebesar 28 *request*.



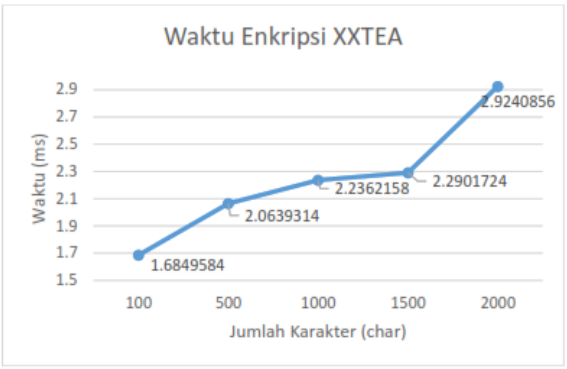
Gambar 12. Grafik Performa 8 *Node CPU*

Grafik pada Gambar 11 menunjukan jumlah *request per second* yang dapat dihasilkan pada penggunaan 10 *node CPU* tidak stabil dan memiliki ruang nilai yang rendah. Nilai tertinggi *request per second* yang dapat dihasilkan adalah sebesar 26 *request* yaitu saat nilai *concurent*-nya 4 dan 256. Sedangkan nilai terendah *request per second*-nya adalah 20 yaitu saat nilai *concurent*-nya 512.

3) Hasil Pengujian Algoritma

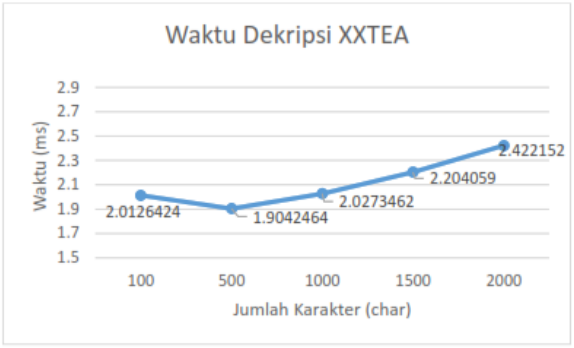
Hasil uji coba performa algoritma dilakukan dengan menguji lamanya waktu (dalam *millisecond*) untuk mengenkripsi dan mendekripsi 5 contoh teks dengan jumlah karakter yang bervariasi namun memiliki kunci yang sama. Kunci yang digunakan pada uji coba ini adalah "email3@gmail.comemail4@gmail.comTA2017!". Uji coba ini dilakukan sebanyak lima kali untuk masing-masing jumlah karakter yang telah ditentukan kemudian diambil nilai rata-ratanya.

Dari hasil uji performa algoritma *XXTEA*, dapat disimpulkan bahwa waktu enkripsi dan dekripsi pada algoritma *XXTEA* relatif cepat. Rata-rata waktu yang digunakan *XXTEA* untuk mengenkripsi pesan adalah 2.23987272 *ms*. Sedangkan, rata-rata waktu yang dihasilkan algoritma XXTEA untuk mendekripsi pesan adalah 2.05297956 *ms*.



Gambar 13. Grafik Hasil Waktu Enkripsi *XXTEA*

Pada Gambar 13 dapat dilihat bahwa waktu yang dibutuhkan *XXTEA* untuk mengenkripsi suatu teks yang panjangnya 100 karakter adalah 1,6849584 *ms* dan teks yang panjangnya 2000 karakter adalah sebesar 2,9240856 *ms*. Waktu tersebut tercatat sebagai waktu tercepat dan terlama *XXTEA* mengenkripsi sebuah teks. Dari grafik di atas dapat dilihat bahwa semakin banyak jumlah karakter yang dienkripsi semakin lama waktu yang dibutuhkan *XXTEA* untuk mengenkripsi.



Gambar 14. Grafik Hasil Waktu Dekripsi *XXTEA*

Gambar 14 menunjukkan grafik hasil waktu yang dibutuhkan *XXTEA* untuk mendekripsi suatu teks. Untuk mendekripsi teks yang panjangnya 100

karakter dibutuhkan waktu sebesar 2,0126424 *ms* sedangkan untuk teks yang panjangnya 2000 karakter diperlukan waktu sebesar 2,422152 ms.

DAFTAR PUSTAKA

[1] Azizah KN. 2017. Implementasi Metode Enkripsi Menggunakan Algoritma XXTEA (Corrected Block Tiny Encryption Algorithm) Pada Aplikasi Surat Elektronik Berbasis Web. Institut Teknologi Sepuluh Nopember.

[2] Adiputera YF. 2017. Aplikasi Enkripsi Video MPEG dengan Vidio Encryption Algorithm (VEA) yang dimodifikasi dengan Algoritma RC4. Universitas Diponegoro.

[3] Elka LK, dkk. 2014. Aplikasi Enkripsi dan Dekripsi Data Menggunakan Algoritma RC4 dengan menggunakan Bahasa Pemrograman PHP. Jurnal Media Infotama 10(1). Universitas Dehasen Bengkulu.

[4] Kara SB. 2015. Rancang Bangun Aplikasi Enkripsi dan Dekripsi Menggunakan Algoritma RC4 pada Sistem Operasi Android. Universitas Gadjah Mada.

[5] Yuricha, dkk. 2017. Implementasi Algoritma Kriptografi XXTEA untuk Enkripsi dan Dekripsi Query Database pada Aplikasi Online Test. Jurnal Sistem dan Teknologi Informasi 5 (1). Universitas Tanjung Pura

[6] E. Indra, Steffanily, and T. Dinesh, "Designing Android Gaming News & Information Application Using Java-Based Web Scraping Technique," *J. Phys. Conf. Ser.*, vol. 1230, p. 012069, Jul. 2019, doi: 10.1088/1742-6596/1230/1/012069.