Vol. 6 No. 1 Juni 2025 **Hal : 12 - 27** **E-ISSN: 2723-1089**
P-ISSN: 2776-7779

Implementasi *Base64*, *Beaufort Cipher*, *Vigenere Cipher*, Untuk Pengamanan *File Source Code* Dalam Bahasa Pemrograman

William Lutfir Rahman Harjo[1)], Muhammad Ikhsan[2)] , Mhd Ikhsan Rifki[3)]
[1,2,3)]Prodi Ilmu Komputer, Fakultas Sains dan Teknologi, UIN Sumatera Utara, Indonesia

*Coresponding Email: williamlutfi06@gmail.com

Abstrak

Kriptografi adalah teknik digunakan untuk menjaga kerahasiaan suatu data. Suatu data yang dianggap penting dan dirasa sensitif, sebaiknya diberikan suatu pengamanan untuk melindungi kerahasiaannya. Pada penelitian ini mengimplementasikan algoritma *Base64*, *Beaufort Cipher* dan *Vigenere Cipher* guna meningkatkan keamanan pada *file source code* dalam berbagai ekstensi bahasa pemrograman. Tahapan enkripsi yang dilakukan pada penelitian ini dengan cara merubah teks didalam *file source code* terlebih dahulu kedalam format *Base64*, kemudian mengenkripsi menggunakan algoritma *Beaufort Cipher* dan dilanjutkan menggunakan algoritma *Vigenere Cipher* untuk menghasilkan *file source* yang telah terenkripsi. Penelitian ini menghasilkan aplikasi berbasis *web*. Pengujian *avalanche effect*, *bit error rate*, *character error rate*, *entropy* dan *blackbox* pada aplikasi, telah dilakukan agar memastikan bahwa aplikasi bekerja sesuai dengan spesifikasinya dan sudah layak untuk digunakan.

Kata Kunci: *Base64*; *Beaufort Cipher*; *Vigenere Cipher*; *File Source Code*.

Abstract

Cryptography is a technique for maintain the confidentiality of data. Data that is considered important and sensitive should be given security to protect its confidentiality. This study implements the Base64, Beaufort Cipher and Vigenere Cipher algorithms to improve security in source code files in various programming language extensions. The encryption stages carried out in this study are by changing the text in the source code file first into Base64 format, then encrypting it using the Beaufort Cipher algorithm and continuing using the Vigenere Cipher algorithm to produce an encrypted source file. This study produces a web-based application. Avalanche effect, bit error rate, character error rate, entropy and blackbox testing on the application already done to ensure that the application works according to its specifications and is suitable for use.

Keywords: Base64*; *Beaufort Cipher*; *Vigenere Cipher*; *Source Code File.

PENDAHULUAN

Berdasarkan laporan *monitoring* keamanan siber dari Badan Siber dan Sandi Negara (BSSN), mencatat bahwa telah terjadi 330.527.636 juta trafik anomali serangan siber terhadap Indonesia pada tahun 2024 [1], dimana terdapat serangan yang mengeksploitasi kerentanan perangkat lunak untuk mendapatkan akses ke *server* dan mencuri data pengguna. *File source code* yang merupakan inti dari suatu sistem perangkat lunak memerlukan perlindungan ekstra untuk memastikan integritas dan kerahasiaannya. Kriptografi dapat menjadi solusi yang efektif dalam mengamankan data [2], dan penerapannya pada *file source* code dapat menjadi langkah yang signifikan dalam melindungi aset intelektual dan menjaga keberlanjutan pengembangan perangkat lunak. Pentingnya menjaga kerahasiaan dan integritas dari *file source code* tidak bisa diremehkan, mengingat risiko pencurian data, pelanggaran keamanan, penggunaan ilegal dan modifikasi tidak sah yang dapat merugikan seorang pengembang dan pemilik perangkat lunak. *File source code* tidak hanya mencakup instruksi untuk menjalankan program, tetapi juga mengandung informasi sensitif seperti algoritma, logika bisnis, dan desain aplikasi yang merupakan aset berharga bagi seorang pengembang dan suatu organisasi perusahaan.

Pada penelitian sebelumnya telah meneliti penggunaan algoritma *Base64* dalam keamanan data, yang dilakukan oleh *Tinendung et al* [3], hasil penelitian ini memberikan opsi tambahan dalam melindungi data pribadi saat bertransaksi melalui jaringan komunikasi. Selanjutnya pada penelitian sebelumnya telah meneliti penggunaan algoritma *Beaufort Cipher* dan *Vigenere Cipher* secara bersamaan, yang dilakukan oleh *Rachmadsyah et al* [4], hasil penelitian menunjukkan kombinasi dari kedua algoritma berhasil digunakan untuk mengamankan pesan teks, implementasi tersebut memungkinkan pesan teks dienkripsi dan didekripsi dengan baik, sehingga pesan yang disampaikan menjadi aman dan rahasia. Selain itu, oleh *Darmeli et al* [5], hasil penelitian menunjukkan kombinasi kedua algoritma yang diterapkan dalam skema *three-pass protocol* meningkatkan kualitas enkripsi data penerbangan. Hasil uji coba pada data penerbangan, membuktikan bahwa data yang dikirimkan tetap terjaga kerahasiaannya selama proses pengiriman. Sementara itu, oleh *Setiadi et al* [6], hasil penelitian

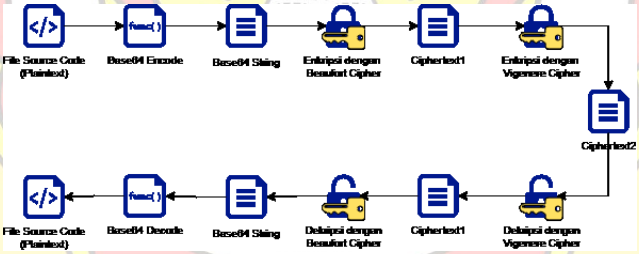
menunjukkan kombinasi kedua algoritma pada citra *grayscale* menunjukkan bahwa kualitas enkripsi meningkat dibandingkan dengan menggunakannya secara terpisah, menghasilkan kualitas enkripsi yang baik dan distribusi nilai *pixel* yang lebih seragam. Penelitian ini menggunakan nilai *Mean Square Error* (MSE), *Peak Signal-to-Noise Ratio* (PSNR), dan analisis *histogram* sebagai alat ukur.

METODE PENELITIAN

Perencanaan dan Perancangan

Tahapan perencanaan penelitian bertujuan untuk menguraikan masalah dan tujuan penelitian. Terdapat beberapa hal yang dilakukan selama proses perencanaan yakni identifikasi masalah, penentuan tujuan penelitian, studi pustaka, analisis sistem, desain sistem, implementasi dan pengujian sistem dan kesimpulan penelitian.

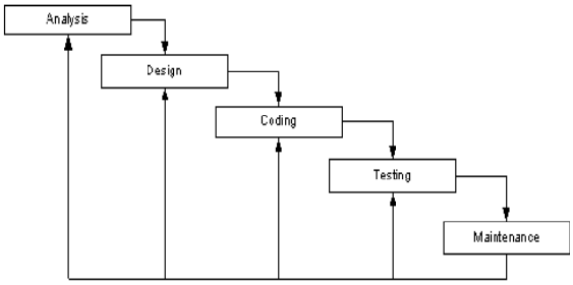
Perancangan sistem adalah proses untuk menentukan bagaimana sebuah sistem akan bekerja, termasuk struktur, dan hubungan antar komponen, sehingga mampu memenuhi kebutuhan atau menyelesaikan masalah tertentu. Perancangan konsep pengamanan *file source code*, dengan mengimplementasikan algoritma *Base64*, *Beaufort Cipher* dan *Vigenere Cipher*, dapat dilihat pada Gambar 1 :



Gambar 1 Arsitektur Sistem

Metode Pengembangan Sistem

Metode pengembagan sistem yang digunakan pada penelitian ini menggunkan metode *waterfall*. Metode *waterfall* adalah salah satu model pengembangan perangkat lunak bersifat *linier* dan berurutan, artinya setiap tahap dalam pengembangan harus diselesaikan sepenuhnya sebelum tahap berikutnya dimulai. Model ini disebut "*waterfall*" karena alur kerjanya mengalir ke bawah seperti air terjun. Dalam pengembangannya metode *waterfall* memiliki beberapa tahapan yang runtut : analisis kebutuhan, *design* sistem, *coding* dan *testing*, penerapan program, pemeliharaan [7], seperti terlihat pada Gambar 2 :

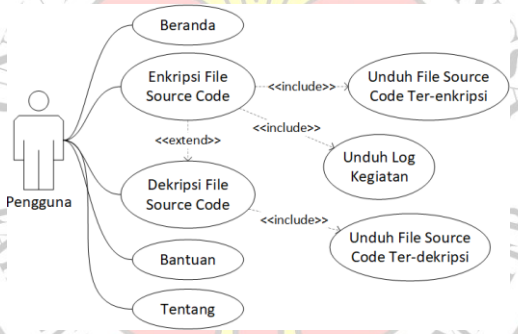


Gambar 2 Metode *Waterfall*

Unified Modelling Language (UML)

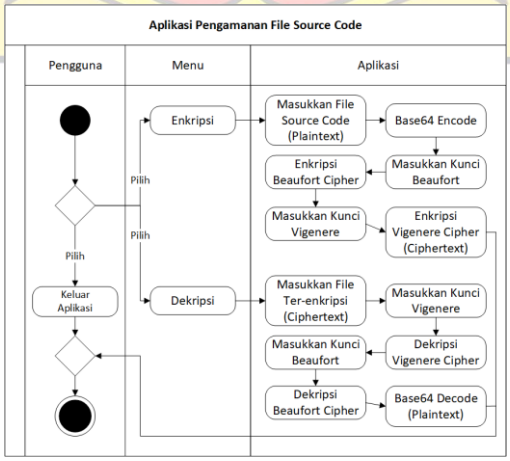
Unified Modelling Language adalah bahasa standar yang digunakan untuk mendokumentasikan, merancang, dan memodelkan suatu sistem perangkat lunak. UML memfasilitasi pengembang dalam menggambarkan alur sistem yang akan dikembangkan dengan memanfaatkan simbol-simbol pada diagram [8].

Adapun *use case diagram* yang dibuat untuk perancangan sistem pada penelitian ini, terlihat pada Gambar 3 berikut :



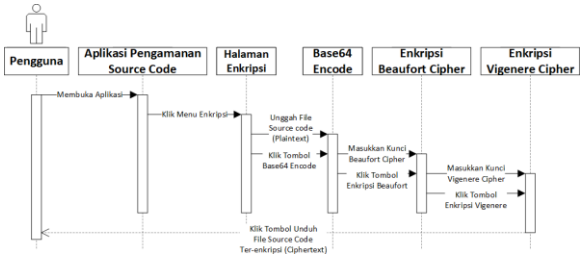
Gambar 3 *Use Case Diagram* Dari Sistem Yang Dirancang

Adapun *activity diagram* yang dibuat untuk perancangan sistem pada penelitian ini, dapat dilihat pada Gambar 4 berikut :



Gambar 4 *Activity Diagram* Dari Sistem Yang Dirancang

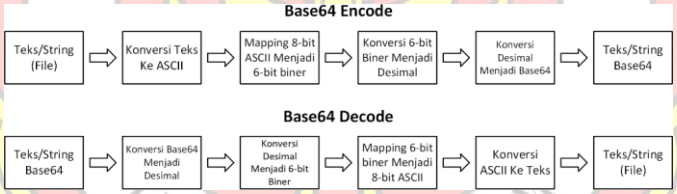
Adapun *sequence diagram* yang dibuat untuk perancangan sistem pada penelitian ini, dapat dilihat pada Gambar 5 :



Gambar 5 *Sequence Diagram* Dari Sistem Yang Dirancang

Algoritma Base64

Algoritma *Base64* adalah sebuah algoritma pengkodean yang mentransformasi data biner menjadi format teks ASCII. Algoritma *Base64* menggunakan tabel karakter yang terdiri dari 64 karakter, biasanya terdiri dari huruf besar dan kecil (A-Z, a-z), angka (0-9), dan dua karakter tambahan yaitu karakter "+", "/", serta satu karakter "=" terletak diakhir yang digunakan dalam pengisian padding atau penyesuaian dan menggenapkan data biner [3]. Adapun proses *encode* dan *decode* dari algoritma *Base64*, terlihat pada Gambar 6 :



Gambar 6 Proses Dari Algoritma *Base64*

Algoritma Beaufort Cipher

Algoritma *Beaufort Cipher* adalah algoritma kriptografi yang digunakan untuk menyandikan pesan, yang menggunakan teknik substitusi dengan kunci simetris [6]. Algoritma *Beaufort Cipher* termasuk dalam kelompok kriptografi klasik, di mana kunci pada *Beaufort Cipher* adalah urutan karakter-karakter K = k1...kd. Nilai k1 menunjukkan jumlah pergeseran dari alfabet ke-I [4]. Oleh karena itu, jumlah karakter kunci yang digunakan sama dengan jumlah karakter *plaintext* yang ingin dienkripsi. Maka, setiap karakter *plaintext* harus memiliki pasangan dengan karakter kunci. Hal ini menjadikan algoritma ini hampir serupa dengan algoritma *Vigenere Cipher* [9]. Adapun rumus dari algoritma *Beaufort Cipher* yang digunakan dalam tahapan enkripsi dan dekripsi dinyatakan melalui rumus berikut :

$C_c = (k_i - P_c)\ mod\ 26$.....(1)

$P_c = (k_i - C_c)\ mod\ 26$.....(2)

Algoritma Vigenere Cipher

Algoritma *Vigenere Cipher* adalah algoritma kriptografi yang digunakan untuk mengenkripsi dan mendekripsi pesan dengan menggunakan kunci yang terdiri dari sebuah kata atau frasa [10], kemudian algoritma *Vigenere* cipher merupakan kriptografi klasik yang memakai metode substitusi abjad-majemuk, dengan mengenkripsi setiap huruf dalam *plaintext* menggunakan kunci, dimana jika panjang kunci lebih pendek dari panjang *plaintext*, maka kunci akan diulang secara periodik [11]. Adapun rumus dari algoritma *Vigenere Cipher* yang digunakan dalam tahapan enkripsi dan dekripsi dinyatakan melalui rumus berikut :

$C_i = (P_i + k_i)\ mod\ 26$.....(3)

$P_i = (C_i - k_i)\ mod\ 26$.....(4)

Avalanche Effect

Avalanche effect merupakan metode pengujian dalam kriptografi yang digunakan untuk menilai kualitas algoritma dengan mengukur persentase perubahan pesan selama proses enkripsi. Perubahan kecil pada *plaintext* (misalnya, mengubah satu bit) akan menghasilkan perubahan yang signifikan pada *ciphertext* (misalnya, lebih dari setengah bit *ciphertext* berubah). Jika perubahan bit berada pada kisaran 45-60%, maka pengujian *avalanche effect* dianggap baik, dimana 50% sebagai hasil yang dianggap ideal dalam pengujian [12]. Adapun rumus *avalanche effect* sebagai berikut :

$$AE = \frac{Jumlah\ Perubahan\ Bit}{Jumlah\ Total\ Bit} \times 100\%$$.....(5)

Bit Error Rate

Bit error rate (BER) merupakan metode untuk mengevaluasi performa dari sistem yang menerapkan algoritma kriptografi. BER mengukur frekuensi kesalahan bit dalam transmisi data dan menilai kualitas enkripsi dan dekripsi data. BER dianggap baik jika nilainya mendekati 0, yang artinya tidak ada perbedaan antara plaintext asli dengan hasil dekripsi *ciphertext* [12]. Adapun rumus *bit error rate* sebagai berikut :

$$BER = \frac{Jumlah\ Bit\ Error}{Jumlah\ Total\ Bit} \times 100\%$$.....(6)

Character Error Rate

Character error rate (CER) merupakan metode pengujian untuk mengukur presentase tingkat akurasi sebuah hasil dekripsi dengan cara mencocokan dan membandingkan *character* (huruf, angka, simbol) dengan *plaintext*-nya. CER merujuk pada tingkat kesalahan dalam proses dekripsi pesan. CER mengukur seberapa sering karakter yang dihasilkan setelah proses dekripsi berbeda dari karakter aslinya [13]. Adapun rumus *character error rate* sebagai berikut :

$$CER = \frac{Jumlah\ Kesalahan\ Karakter}{Jumlah\ Total\ Karakter} \times 100\% \quad (7)$$

Entropy

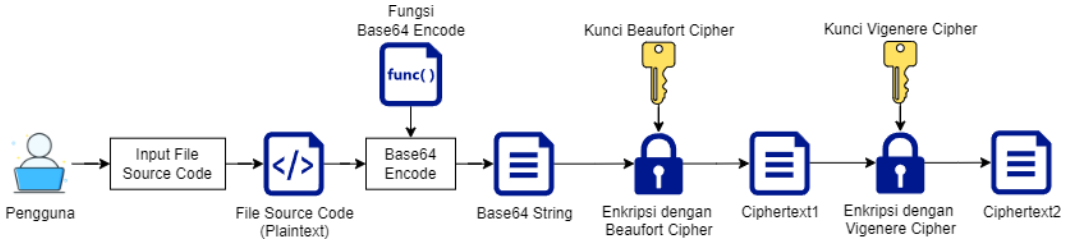
Entropy dalam konteks kriptografi merujuk pada ukuran keacakan dalam sistem kriptografi. *Entropy* yang tinggi menunjukkan bahwa *ciphertext* yang dihasilkan dalam proses kriptografi sangat acak dan sulit diprediksi. *Entropy* dalam konsep kriptografi untuk memperkirakan jumlah bit rata-rata dalam pengkodean pesan. Nilai *entropy*-nya dinyatakan dalam satuan bit [12]. Adapun rumus *entropy* Shannon sebagai berikut :

$$H(X) = -\sum_{i=0}^{n} p(x) \cdot \log_2(p(x)) \quad (8)$$

HASIL DAN PEMBAHASAN

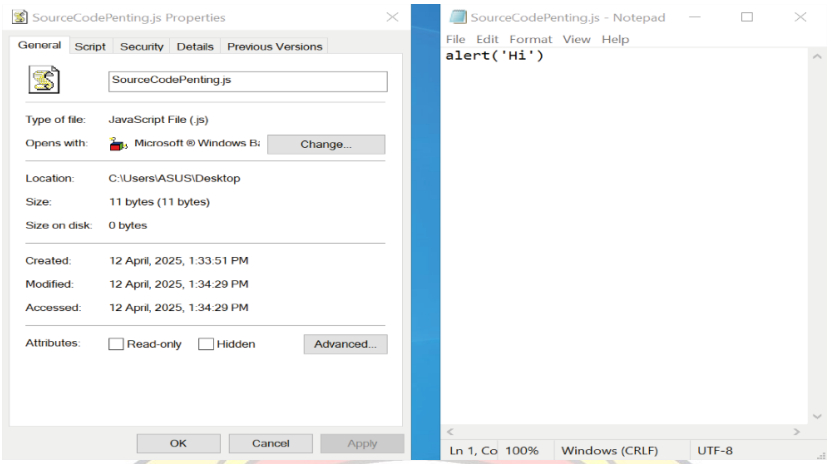
Analisis Proses Enkripsi

Proses enkripsi adalah proses mengubah data menjadi bentuk yang tidak dapat dipahami oleh banyak orang. Dalam penelitian ini, data yang akan dienkripsi berupa *file source code* dengan mengimplementasikan algoritma *Base64*, *Beaufort Cipher*, dan *Vigenere Cipher*. Pada Gambar 7, memberikan gambaran tentang proses enkripsi yang diterapkan dalam penelitian ini :



Gambar 7 Alur Proses Enkripsi *File Source Code*

Berikut ditampilkan contoh sampel *file source code* dengan format JavaScript. *File source code* yang digunakan memiliki size sebesar 11 bytes dengan jumlah 11 karakter.



Gambar 8 Contoh Sampel *File Source Code*

Proses awal yaitu melakukan transformasi isi *file source code* menggunakan *Base64 encode* sehingga menghasilkan *Base64 string*. Masuk ke proses *Base64 encode*, langkah awal yang harus dilakukan yaitu mengkonversi setiap karakter dari teks sesuai pada tabel ASCII, dalam biner yang di mapping 8-bit.

Tabel 1 Konversi ASCII Dan *Mapping* Biner 8-Bit

Karakter	ASCII (Heksadesimal)	Biner 8-Bit
a	61	01100001
l	6C	01101100
e	65	01100101
r	72	01110010
t	74	01110100
(	28	00101000
‘	27	00100111
H	48	01001000
i	69	01101001
‘	27	00100111
)	29	00101001

Kemudian di *mapping* lagi menjadi biner 6-bit. Apabila di akhir *mapping* mengalami kurang dari biner 6-bit, maka ditambahkan padding (nol 8-Bit). Setelah di *mapping* biner 6-bit, lalu dikonversi ke dalam bilangan desimal.

Tabel 2 *Mapping* Biner 6-Bit Dan Konversi Ke Bilangan Desimal

Biner 6-Bit	Konversi Bilangan Desimal	Desimal
011000	$(0\times2^5) + (1\times2^4) + (1\times2^3) + (0\times2^2) + (0\times2^1) + (0\times2^0)$	$24_{(10)}$
010110	$(0\times2^5) + (1\times2^4) + (0\times2^3) + (1\times2^2) + (1\times2^1) + (0\times2^0)$	$22_{(10)}$
110001	$(1\times2^5) + (1\times2^4) + (0\times2^3) + (0\times2^2) + (0\times2^1) + (1\times2^0)$	$49_{(10)}$
100101	$(1\times2^5) + (0\times2^4) + (0\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0)$	$37_{(10)}$
011100	$(0\times2^5) + (1\times2^4) + (1\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0)$	$28_{(10)}$
100111	$(1\times2^5) + (0\times2^4) + (0\times2^3) + (1\times2^2) + (1\times2^1) + (1\times2^0)$	$39_{(10)}$
010000	$(0\times2^5) + (1\times2^4) + (0\times2^3) + (0\times2^2) + (0\times2^1) + (0\times2^0)$	$16_{(10)}$
101000	$(1\times2^5) + (0\times2^4) + (1\times2^3) + (0\times2^2) + (0\times2^1) + (0\times2^0)$	$40_{(10)}$
001001	$(0\times2^5) + (0\times2^4) + (1\times2^3) + (0\times2^2) + (0\times2^1) + (1\times2^0)$	$9_{(10)}$
110100	$(1\times2^5) + (1\times2^4) + (0\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0)$	$52_{(10)}$
100001	$(1\times2^5) + (0\times2^4) + (0\times2^3) + (0\times2^2) + (0\times2^1) + (1\times2^0)$	$33_{(10)}$
101001	$(1\times2^5) + (0\times2^4) + (1\times2^3) + (0\times2^2) + (0\times2^1) + (1\times2^0)$	$41_{(10)}$
001001	$(0\times2^5) + (0\times2^4) + (1\times2^3) + (0\times2^2) + (0\times2^1) + (1\times2^0)$	$9_{(10)}$
110010	$(1\times2^5) + (1\times2^4) + (0\times2^3) + (0\times2^2) + (1\times2^1) + (0\times2^0)$	$50_{(10)}$
100100	$(1\times2^5) + (0\times2^4) + (0\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0)$	$36_{(10)}$

Terakhir, konversikan setiap nilai bilangan desimal yang didapatkan sesuai dengan nilai dan karakter yang ada pada tabel *Base64* :

Tabel 3 Konversi Bilangan Desimal Ke *Base64*

Desimal	Base64	Desimal	Base64	Desimal	Base64
$24_{(10)}$	Y	$39_{(10)}$	n	$33_{(10)}$	h
$22_{(10)}$	W	$16_{(10)}$	Q	$41_{(10)}$	p
$49_{(10)}$	x	$40_{(10)}$	o	$9_{(10)}$	J
$37_{(10)}$	l	$9_{(10)}$	J	$50_{(10)}$	y
$28_{(10)}$	c	$52_{(10)}$	0	$36_{(10)}$	k

Berdasarkan hasil perhitungan pada proses *Base64 encode* maka *Base64 string* yang diperoleh dari isi *file source code* (*plaintext*) yaitu, YWxlcnQoJ0hpJyk

Kemudian *Base64 string* yang didapat akan dienkripsi dengan menggunakan algoritma *Beaufort Cipher* dengan kunci "UINSU". Langkah yang harus dilakukan yaitu mensubstitusikan setiap karakter *plaintext* dan kunci berdasarkan tabel *Base64*. Masuk ke proses enkripsi *Beaufort Cipher* menggunakan Persamaan (1) dengan modulus 64, maka akan terlihat seperti berikut :

Tabel 4 Substitusi Angka Beaufort Cipher Berdasarkan Tabel Base64

Base64 String	Y	W	x	l	c	n	Q	o	J	0	h	p	J	y	k
Substitusi	24	22	49	37	28	39	16	40	9	52	33	41	9	50	36
Kunci	U	I	N	S	U	U	I	N	S	U	U	I	N	S	U
Substitusi	20	8	13	18	20	20	8	13	18	20	20	8	13	18	20
Ciphertext1	8	y	c	t	4	t	4	l	J	g	z	f	E	g	w

Berdasarkan hasil perhitungan diperoleh hasil enkripsi pertama (*ciphertext1*) dengan menggunakan algoritma *Beaufort Cipher* yaitu, 8yct4t4lJgzfEgw

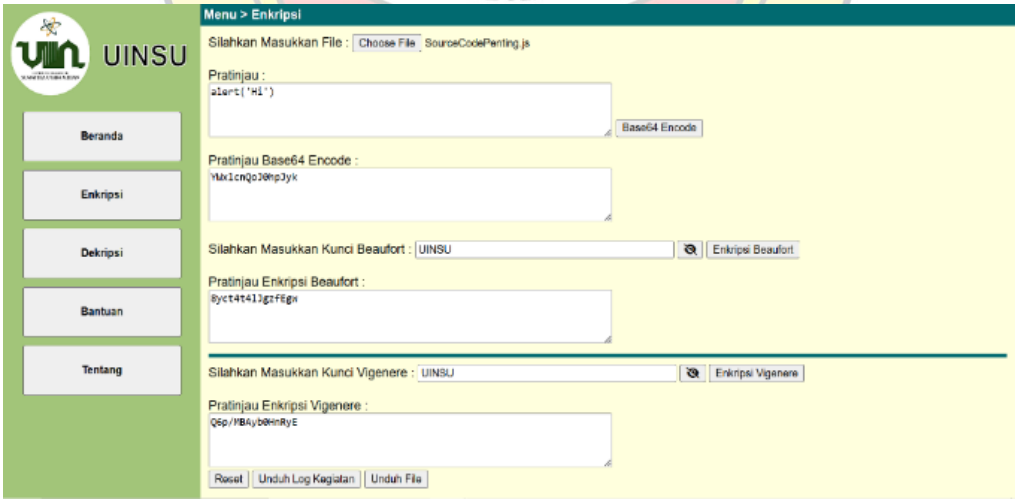
Kemudian *ciphertext1* akan di enkripsi lagi dengan menggunakan algoritma *Vigenere Cipher* dengan kunci "UINSU" juga. Langkah yang harus dilakukan yaitu mensubstitusikan setiap karakter *plaintext* dan kunci berdasarkan tabel *Base64*. Masuk ke proses enkripsi *Vigenere Cipher* menggunakan Persamaan (3) dengan modulus 64, maka akan terlihat seperti berikut :

Tabel 5 Substitusi Angka *Vigenere Cipher* Berdasarkan Tabel *Base64*

Ciphertext1	8	y	c	t	4	t	4	l	J	g	z	f	E	g	w
Substitusi	60	50	28	45	56	45	56	37	9	32	51	31	4	32	48
Kunci	U	I	N	S	U	U	I	N	S	U	U	I	N	S	U
Substitusi	20	8	13	18	20	20	8	13	18	20	20	8	13	18	20
Ciphertext2	Q	6	p	/	M	B	A	y	b	0	H	n	R	y	E

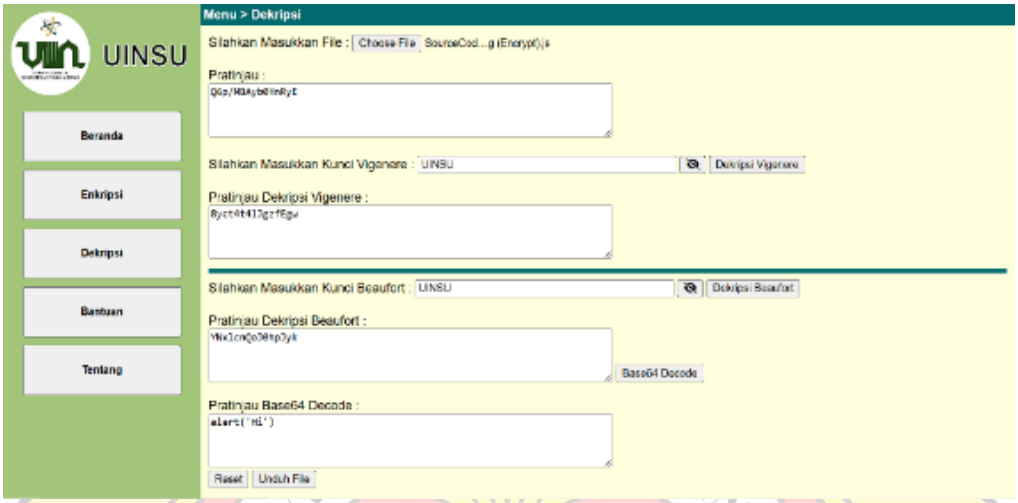
Implementasi Aplikasi

Halaman enkripsi berfungsi untuk melakukan proses enkripsi isi dari *file source code* (*plaintext*) dengan menggunakan algoritma *Base64*, *Beaufort Cipher*, *Vigenere Cipher*. Implementasi halaman enkripsi dapat dilihat pada Gambar 9 :



Gambar 9 Hasil Pengujian Enkripsi *File Source Code*

Halaman dekripsi berfungsi untuk melakukan proses dekripsi isi teks dari *file source code* yang terenkripsi (*ciphertext*) dengan menggunakan algoritma *Base64*, *Beaufort Cipher*, *Vigenere Cipher*, agar *file source code* asli akan bisa didapatkan kembali. Implementasi halaman dekripsi dapat dilihat pada Gambar 10 :



Gambar 10 Hasil Pengujian Dekripsi *File Source Code*

Pengujian Black Box

Pengujian *black box* adalah metode pengujian perangkat lunak yang berfokus pada fungsionalitas dari program, khususnya pada aspek *input* dan *output*, untuk memastikan kesesuaiannya dengan harapan. Adapun pengujian *black box* yang dilakukan adalah sebagai berikut :

Tabel 6 Pengujian *Black Box* Aplikasi

No.	Kasus Uji	Langkah Uji	Hasil	Keterangan
1.	*Input file source code*	Memilih satu *file source code* dari berbagai ekstensi bahasa pemrogramannya dengan cara mengklik tombol "*Choose File*".	Sistem dapat menerima *file source code* dalam berbagai ekstensi bahasa pemrograman dan dapat menampilkan *preview* isi dari *file source code*.	Valid
2.	*Base64 encode*	Melakukan proses *Base64 encode* pada isi *file source code* dengan cara mengklik tombol "*Base64 Encode*".	Sistem dapat menam-pilkan hasil *Base64 encode* berupa *Base64 string* dari *file source code* tersebut.	Valid
3.	Enkripsi *Beaufort Cipher*	Mengenkripsi *file source code* dengan	Sistem dapat menam-pilkan transformasi *Base64 string* yang	Valid

No.	Kasus Uji	Langkah Uji	Hasil	Keterangan
		cara mengklik tombol "Enkripsi *Beaufort*".	telah di enkripsi dengan kunci *Beaufort Cipher*.	
4.	Enkripsi *Vigenere Cipher*	Mengenkripsi *file source code* dengan cara mengklik tombol "Enkripsi *Vigenere*".	Sistem dapat menam-pilkan transformasi *Base64 string* yang telah di enkripsi dengan kunci *Vigenere Cipher*.	Valid

Pengujian Algoritma Kriptografi

Dalam pengujian ini dilakukan 10 kali percobaan dengan *file source code* berukuran kecil hingga besar. Pada semua percobaan digunakan kunci yang sama yaitu "UINSU", seperti yang terlihat pada Tabel 7 :

Tabel 7 Hasil Pengujian Enkripsi Berbagai *Format File Source Code*

No.	Nama *File*	*Size Plaintext* (KB)	Kunci	*Size Ciphertext* (KB)	Waktu (ms)
1	*File*Uji.js	0.011 KB	UINSU	0.015 KB	3.20 ms
2	*File*Uji2.js	1.56 KB	UINSU	2.09 KB	6.29 ms
3	*File*Uji3.js	1500 KB	UINSU	2010 KB	1230 ms
4	*File*Uji4.js	2010 KB	UINSU	2680 KB	1360 ms
5	*File*Uji5.php	0.023 KB	UINSU	0.031 KB	4.20 ms
6	*File*Uji6.php	2.11 KB	UINSU	2.82 KB	7.50 ms
7	*File*Uji7.php	1210 KB	UINSU	1620 KB	895.5 ms
8	*File*Uji8.py	0.015 KB	UINSU	0.020 KB	2.69 ms
9	*File*Uji9.py	3.10 KB	UINSU	4.13 KB	9.29 ms
10	*File*Uji10.py	1370 KB	UINSU	1830 KB	916.2 ms

Berikut ini adalah salah satu perhitungan nilai *avalanche effect* pada FileUji.js :

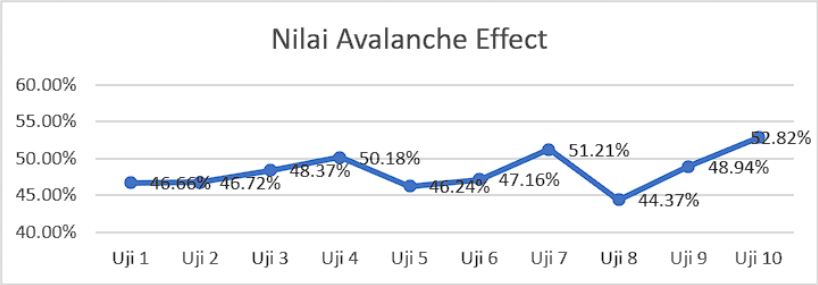
1. Ubah kedua *ciphertext* ke dalam bentuk biner 8-bit berdasarkan pengkodean ASCII, kemudian bandingkan setiap bit antara dua *ciphertext* dan hitung jumlah bit yang berbeda.
2. Hitung persentase dengan rumus pada Persamaan (5).

Hasil perhitungan *avalanche effect* adalah sebagai berikut :

Jumlah bit yang berbeda : 56 $\frac{56}{120} \times 100\% = 46.66\%$

Total bit : 120

Setelah melakukan 10 pengujian terhadap *file source code*, hasil pengujian *avalanche effect* dapat dilihat pada Gambar 11 :



Gambar 11 Grafik Hasil Pengujian *Avalanche Effect*

Nilai *avalanche effect* yang ideal mendekati 50% (hampir setengah dari bit berubah), dimana nilai yang tinggi menunjukkan bahwa kombinasi algoritma kriptografi yang digunakan memiliki sifat jika perubahan kecil dalam *plaintext* (misalnya, satu karakter) menghasilkan perubahan besar dalam *ciphertext*.

Berikut ini adalah salah satu perhitungan nilai *bit error rate* pada FileUji.js :

Langkah perhitungan *bit error rate* :

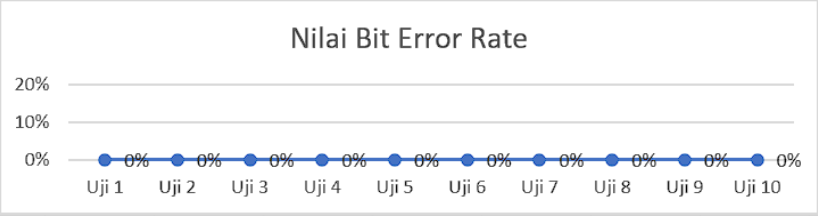
1. Ubah *source code* asli dan hasil dekripsi ke dalam bentuk biner 8-bit berdasarkan pengkodean ASCII.
2. Bandingkan setiap bit dan hitung jumlah bit yang salah.
3. Hitung persentase dengan rumus pada Persamaan (6).

Hasil perhitungan *bit error rate* adalah sebagai berikut :

Jumlah bit yang salah : 0, karena hasil dekripsi sama dengan *source code* asli, jadi tidak ada bit yang salah. $\frac{0}{88} \times 100\% = 0\%$

Total bit : 88

Setelah melakukan 10 pengujian terhadap *file source code* sebelumnya, hasil pengujian *bit error rate* dapat dilihat pada Gambar 12 :



Gambar 12 Grafik Hasil Pengujian *Bit Error Rate*

Nilai *bit error rate* 0% artinya tidak ada kesalahan bit yang terjadi selama proses enkripsi dan dekripsi. Berikut adalah salah satu perhitungan nilai *character error rate* pada *File*Uji.js :

Langkah perhitungan *character error rate* :

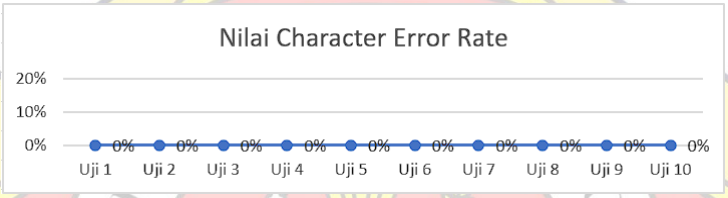
1. Hitung jumlah keseluruhan karakter.
2. Bandingkan setiap karakter dan hitung jumlah karakter yang salah.
3. Hitung persentase dengan rumus pada Persamaan (7).

Hasil perhitungan *character error rate* adalah sebagai berikut :

Jumlah karakter yang salah : 0, karena hasil dekripsi sama dengan *source code* asli, jadi tidak ada karakter yang salah. $\frac{0}{11} \times 100\% = 0\%$

Total karakter : 11

Setelah melakukan 10 pengujian terhadap *file source code* sebelumnya, hasil pengujian *character error rate* dapat dilihat pada Gambar 13 :



Gambar 13 Grafik Hasil Pengujian *Character Error Rate*

Nilai *character error rate* 0% artinya tidak ada kesalahan karakter yang terjadi pada hasil dekripsi dibandingkan dengan *source code* asli. Berikut adalah salah satu perhitungan nilai *entropy* pada *File*Uji.js :

Langkah perhitungan *entropy* :

1. Hitung frekuensi setiap huruf dalam *ciphertext*.

Q=1; 6=1; p=1; /=1; M=1; B=1; A=1; y=2; b=1; 0=1; H=1; n=1; R=1; E=1

2. Hitung probabilitas masing-masing huruf.

Q=1/15; 6=1/15; p=1/15; /=1/15; M=1/15; B=1/15; A=1/15; y=2/15; b=1/15; 0=1/15; H=1/15; n=1/15; R=1/15; E=1/15

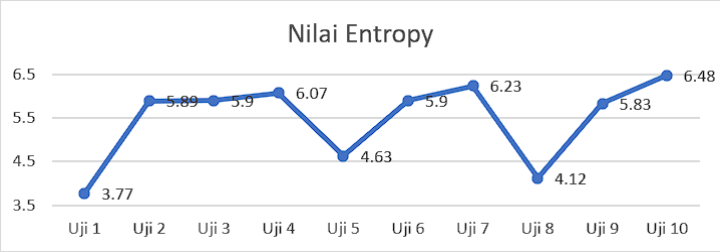
3. Hitung nilai *entropy* dengan rumus pada Persamaan (8).

Hasil perhitungan nilai *entropy* adalah sebagai berikut :

$$H(X) = -\left(\left(13 \times \left(\frac{1}{15}\right) \times \log_2 \frac{1}{15}\right) + \left(1 \times \left(\frac{2}{15}\right) \times \log_2 \frac{2}{15}\right)\right)$$

$$H(X) = -(-3{,}38 + (-0{,}39)) \rightarrow 3{,}77 \text{ bit}$$

Setelah melakukan 10 pengujian terhadap *file source code* sebelumnya, hasil pengujian *entropy* dapat dilihat pada Gambar 14 :



Gambar 14 Grafik Hasil Pengujian *Entropy*

Nilai *entropy* yang dikatakan baik mendekati 7 bit per karakter, maka rata-rata setiap karakter mengandung 7 bit informasi. Nilai *entropy* yang diperoleh dari *ciphertext* memberikan indikasi tentang tingkat keacakan atau ketidakpastian data dalam *ciphertext* tersebut. Hasil dari 10 pengujian yang dilakukan, untuk mendapatkan nilai *entropy* mendekati 7 harus mengenkripsi *file source code* dengan ukuran *file* yang besar dan mengandung karakter yang banyak.

SIMPULAN

Pengamanan isi *file source code* dilakukan dengan cara mentransformasi isi *file source code* menjadi format *Base64* sehingga menghasilkan *Base64 string*, kemudian dienkripsi menggunakan kunci algoritma *Beaufort Cipher*, selanjutnya dienkripsi lagi menggunakan kunci algoritma *Vigenere Cipher*, untuk menghasilkan *file source code* dengan berbagai ekstensi bahasa pemrogramannya yang isinya telah terenkripsi. Penelitian ini menghasilkan aplikasi berbasis *web* yang dapat digunakan untuk mengamankan isi *file source code*. Pengujian yang telah dilakukan yaitu *black box*, *avalanche effect*, *bit error rate*, *character error rate*, *entropy* terhadap algoritma kriptografi, mengindikasikan bahwa sistem yang telah dibangun memiliki 100% tingkat akurasi, yang menandakan keberhasilan aplikasi dalam melakukan enkripsi dan dekripsi *file source code*.

DAFTAR PUSTAKA

BSSN, “Lanskap Keamanan Siber Indonesia 2024,” *Id-SIRTII /CC*, pp. 1–143, 2024, [Online]. Available: https://idsirtii.or.id/halaman/tentang/laporan-hasil-monitoring.html

Suhardi, “Kombinasi Metode Affine Cipher Dan Exclusive OR (XOR) Dalam Pengamanan Pesan,” *Repos. Univ. Islam Negeri Sumatera Utara*, 2021, [Online]. Available: http://repository.uinsu.ac.id/10731/

R. Tinendung, S. Khairani, and S. Sundari, “Aplikasi Messaging Dengan Algoritma Base64

Untuk Mengamankan Data Pesan Berbasis Web," *Algoritm. J. Ilmu Komput. dan Inform.*, vol. 06, no. 01, pp. 104–111, 2022.

A. Rachmadsyah, A. Perdana, and A. Budiman, "Kombinasi Algoritma Beaufort Cipher dan Vigenere Cipher untuk Pengamanan Pesan Teks Berbasis Mobile Application," *J. Minfo Polgan*, vol. 9, no. 2, pp. 12–17, 2020.

D. Nasution, I. Sulistianingsih, and A. Candra Panjaitan, "Kombinasi Vigenère dan Beaufort Cipher Konsep Simulasi Three-pass Protocol pada Data Penerbangan," *CICES (cyberpreneursh. Innov. Creat. Exact Soc. Sci.*, vol. 9, no. 2, pp. 165–173, 2023.

D. R. I. M. Setiadi, C. Jatmoko, E. H. Rachmawanto, and C. A. Sari, "Kombinasi Cipher Subtitusi (Beaufort Dan Vigenere) Pada Citra Digital," *Pros. SENDI_U*, vol. 7, no. 10, pp. 52–57, 2019.

A. G. Silalahi and Yahfizham, "Manajemen Proyek Sistem Informasi Agenda Kegiatan Pegawai Bkad Provinsi Sumatera Utara," *Device J. Inf. Syst. Comput. Sci. Inf. Technol.*, vol. 5, no. 1, pp. 19–32, 2024.

B. Sanjaya and Yahfizham, "Perancangan Sistem Informasi SPPD Pada DPRD Sumatera Utara Dengan Pendekatan Manajemen Proyek Sistem Informasi," *Device J. Inf. Syst. Comput. Sci. Inf. Technol.*, vol. 5, no. 1, pp. 44–60, 2024.

E. Ndruru and T. Zebua, "Pembangkitan Kunci Beaufort Cipher Dengan Teknik Blum-blum Shub pada Pengamanan Citra Digital," *Bull. Inf. Technol.*, vol. 3, no. 2, pp. 149–154, 2022.

N. B. Putra, B. C. Andika, A. D. P. Bagas, and M. Ridwan, "Implementasi Sandi Vigenere Cipher Dalam Mengenkripsikan Pesan," *JOCITIS-Journal Sci. Infomatica Robot.*, vol. 1, no. 1, pp. 42–50.

M. I. Afandi and Nurhayati, "Implementasi Algoritma Vigenere Cipher Dan Atbash Cipher Untuk Keamanan Teks Pada Aplikasi Catatan Berbasis Android," *It (Informatic Tech. J.*, vol. 8, no. 1, pp. 30–41, 2020.

N. A. Karima, A. N. Aisyah, H. V. Silla, L. B. Handoko, and R. R. Sani, "Kriptografi Teks Berbasis Algoritma Substitusi Vigenere Cipher 8 Bit," *J. Masy. Inform.*, vol. 15, no. 1, pp. 1–13, 2024.

Muslih and L. B. Handoko, "Pengujian Avalanche Effect Pada Kriptografi Teks Menggunakan Autokey Cipher," *Semin. Nas. Teknol. dan Multidisiplin Ilmu*, vol. 2, no. 1, pp. 127–134, 2022.