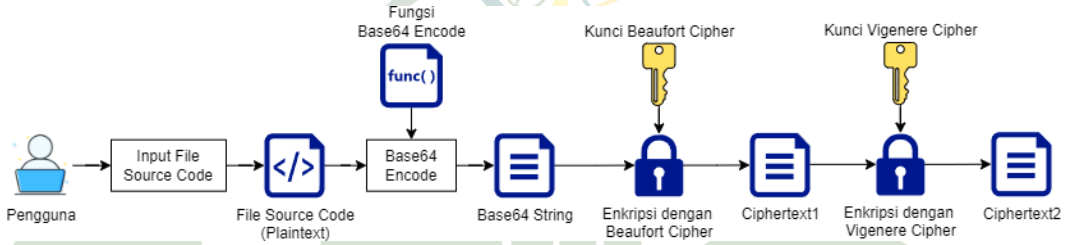
BAB IV
HASIL DAN PEMBAHASAN

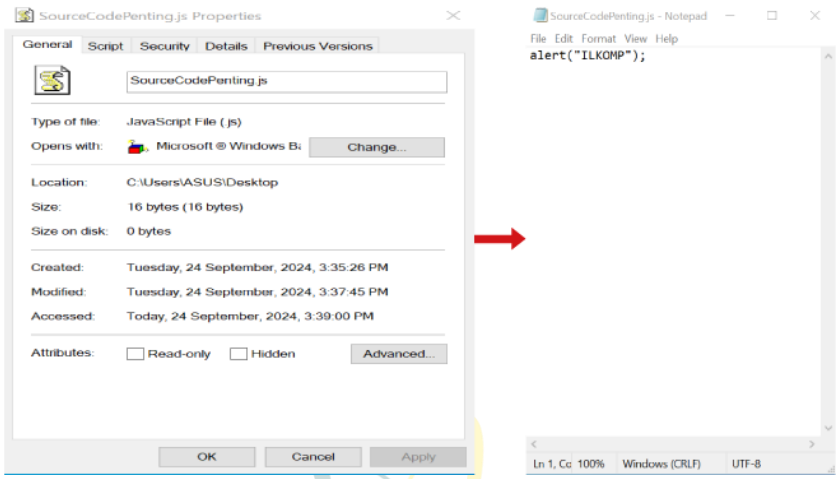
4.1 Analisis Proses Enkripsi

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 4.1, memberikan gambaran umum tentang proses enkripsi yang diterapkan dalam penelitian ini.



Gambar 4.1 Alur Proses Enkripsi *File Source Code*

Berdasarkan Gambar 4.1 terdapat tiga proses utama yang dilakukan dalam mengamankan isi *file source code*. Proses awal yaitu melakukan transformasi isi *file source code* menggunakan *Base64 encode* sehingga menghasilkan *Base64 string*. Kemudian *Base64 string* tersebut akan di enkripsi menggunakan kunci algoritma *Beaufort Cipher* sehingga menghasilkan *ciphertext*1, lalu di enkripsi lagi menggunakan kunci algoritma *Vigenere Cipher* sehingga menghasilkan *ciphertext*2 sebagai hasil akhir dari proses enkripsi *file source code*. *Ciphertext*2 akan berisi teks dalam format *Base64 string* yang telah terenkripsi dan tidak akan mudah didekripsi hanya dengan melakukan *Base64 decode* saja tanpa melewati dua algoritma kriptografi tersebut yaitu algoritma *Beaufort Cipher* dan *Vigenere Cipher*. Pada algoritma *Beaufort Cipher* dan *Vigenere Cipher* sedikit dimodifikasi pada rumusnya yaitu menggunakan *modulo* 64 agar dapat disandingkan dengan algoritma *Base64*, yang memiliki jumlah 64 karakter, dan hanya menggunakan kunci dengan karakter *Base64 alphabet*. Berikut ini ditampilkan contoh sampel *file source code* dengan format .js (JavaScript).



Gambar 4.2 Contoh Sampel *File Source Code*

Pada Gambar 4.2 contoh sampel *file source code* yang digunakan memiliki *size* sebesar 16 *bytes* dengan jumlah 16 karakter agar mempermudah perhitungan secara manual dalam proses enkripsi. 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 4.1 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
“	22	00100010
I	49	01001001
L	4C	01001100
K	4B	01001011
O	4F	01001111
M	4D	01001101
P	50	01010000

Karakter	ASCII (Heksadesimal)	Biner 8-Bit
“	22	00100010
)	29	00101001
;	3B	00111011

Kemudian di *mapping* lagi menjadi biner 6-bit. Apabila di akhir *mapping* mengalami kurang dari biner 6-bit, maka ditambahkan *padding* (nol 8-Bit).

Tabel 4.2 *Mapping* Biner 6-Bit Dan Diberi *Padding*

***Mapping* 6-Bit**							
011000	010110	110001	100101	011100	100111	010000	101000
001000	100100	100101	001100	010010	110100	111101	001101
010100	000010	001000	101001	001110	110000	000000	000000

Setelah di *mapping* biner 6-bit, lalu dikonversi ke dalam bilangan desimal, dengan menggunakan rumus (2.2).

Tabel 4.3 *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	$(1\times2^5) + (0\times2^4) + (1\times2^3) + (0\times2^2) + (0\times2^1) + (0\times2^0)$	$40_{(10)}$
001000	$(0\times2^5) + (0\times2^4) + (1\times2^3) + (0\times2^2) + (0\times2^1) + (0\times2^0)$	$8_{(10)}$
100100	$(1\times2^5) + (0\times2^4) + (0\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0)$	$36_{(10)}$
100101	$(1\times2^5) + (0\times2^4) + (0\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0)$	$37_{(10)}$
001100	$(0\times2^5) + (0\times2^4) + (1\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0)$	$12_{(10)}$
010010	$(0\times2^5) + (1\times2^4) + (0\times2^3) + (0\times2^2) + (1\times2^1) + (0\times2^0)$	$18_{(10)}$
110100	$(1\times2^5) + (1\times2^4) + (0\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0)$	$52_{(10)}$
111101	$(1\times2^5) + (1\times2^4) + (1\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0)$	$61_{(10)}$

Biner 6-Bit	Konversi Bilangan Desimal	Desimal
001101	$(0\times2^5) + (0\times2^4) + (1\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0)$	$13_{(10)}$
010100	$(0\times2^5) + (1\times2^4) + (0\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0)$	$20_{(10)}$
000010	$(0\times2^5) + (0\times2^4) + (0\times2^3) + (0\times2^2) + (1\times2^1) + (0\times2^0)$	$2_{(10)}$
001000	$(0\times2^5) + (0\times2^4) + (1\times2^3) + (0\times2^2) + (0\times2^1) + (0\times2^0)$	$8_{(10)}$
101001	$(1\times2^5) + (0\times2^4) + (1\times2^3) + (0\times2^2) + (0\times2^1) + (1\times2^0)$	$41_{(10)}$
001110	$(0\times2^5) + (0\times2^4) + (1\times2^3) + (1\times2^2) + (1\times2^1) + (0\times2^0)$	$14_{(10)}$
110000	$(1\times2^5) + (1\times2^4) + (0\times2^3) + (0\times2^2) + (0\times2^1) + (0\times2^0)$	$48_{(10)}$

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

Tabel 4.4 Konversi Bilangan Desimal Ke *Base64*

Desimal	Base64	Desimal	Base64
$24_{(10)}$	Y	$12_{(10)}$	M
$22_{(10)}$	W	$18_{(10)}$	S
$49_{(10)}$	x	$52_{(10)}$	0
$37_{(10)}$	l	$61_{(10)}$	9
$28_{(10)}$	c	$13_{(10)}$	N
$39_{(10)}$	n	$20_{(10)}$	U
$16_{(10)}$	Q	$2_{(10)}$	C
$40_{(10)}$	o	$8_{(10)}$	I
$8_{(10)}$	I	$41_{(10)}$	p
$36_{(10)}$	k	$14_{(10)}$	O
$37_{(10)}$	l	$48_{(10)}$	w

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

Plaintext : alert(“ILKOMP”);

Base64 string : YWxlcnQoIklMS09NUCIpOw

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*, dan diketahui bahwa panjang karakter kunci tidak sama dengan panjang karakter dari *plaintext*, maka kunci akan diulang secara periodik, maka akan terlihat seperti berikut:

Tabel 4.5 Substitusi Angka *Beaufort Cipher* Berdasarkan Tabel *Base64*

Plaintext	Y	W	x	l	c	n	Q	o	I	k	l	M	S	0	9	N	U	C	I	p	o	w
Substitusi	24	22	49	37	28	39	16	40	8	36	37	12	18	52	61	13	20	2	8	41	14	48
Kunci	U	I	N	S	U	U	I	N	S	U	U	I	N	S	U	U	I	N	S	U	U	I
Substitusi	20	8	13	18	20	20	8	13	18	20	20	8	13	18	20	20	8	13	18	20	20	8

Selanjutnya, masuk ke proses enkripsi *Beaufort Cipher* menggunakan rumus (2.5), dimana proses perhitungannya sebagai berikut :

$C_1 = (k_1 - P_1)\ mod\ 64$
$= (20 - 24)\ mod\ 64$
$= (-4 + 64)\ mod\ 64$
$= 60 \rightarrow 8$

$C_2 = (k_2 - P_2)\ mod\ 64$
$= (8 - 22)\ mod\ 64$
$= (-14 + 64)\ mod\ 26$
$= 50 \rightarrow y$

$C_3 = (k_3 - P_3)\ mod\ 64$
$= (13 - 49)\ mod\ 64$
$= (-36 + 64)\ mod\ 64$
$= 28 \rightarrow c$

$C_4 = (k_4 - P_4)\ mod\ 64$
$= (18 - 37)\ mod\ 64$
$= (-19 + 64)\ mod\ 64$
$= 45 \rightarrow t$

$C_5 = (k_5 - P_5)\ mod\ 64$
$= (20 - 28)\ mod\ 64$
$= (-8 + 64)\ mod\ 64$
$= 56 \rightarrow 4$

$C_6 = (k_6 - P_6)\ mod\ 64$
$= (20 - 39)\ mod\ 64$
$= (-19 + 64)\ mod\ 64$
$= 45 \rightarrow t$

$C_7 = (k_7 - P_7)\ mod\ 64$
$= (8 - 16)\ mod\ 64$
$= (-8 + 64)\ mod\ 64$
$= 56 \rightarrow 4$

$C_8 = (k_8 - P_8)\ mod\ 64$
$= (13 - 40)\ mod\ 64$
$= (-27 + 64)\ mod\ 64$
$= 37 \rightarrow l$

$C_9 = (k_9 - P_9)\ mod\ 64$
$= (18 - 8)\ mod\ 64$
$= 10 \rightarrow K$

$C_{10} = (k_{10} - P_{10})\ mod\ 64$
$= (20 - 36)\ mod\ 64$
$= (-16 + 64)\ mod\ 64$

$C_{11} = (k_{11} - P_{11})\ mod\ 64$
$= (20 - 37)\ mod\ 64$
$= (-17 + 64)\ mod\ 64$
$= 47 \rightarrow v$

$C_{12} = (k_{12} - P_{12})\ mod\ 64$
$= (8 - 12)\ mod\ 64$
$= (-4 + 64)\ mod\ 64$
$= 60 \rightarrow 8$

$C_{13} = (k_{13} - P_{13})\ mod\ 64$
$= (13 - 18)\ mod\ 64$
$= (-5 + 64)\ mod\ 64$
$= 59 \rightarrow 7$

$C_{14} = (k_{14} - P_{14})\ mod\ 64$
$= (18 - 52)\ mod\ 64$
$= (-34 + 64)\ mod\ 64$
$= 30 \rightarrow e$

$C_{15} = (k_{15} - P_{15})\ mod\ 64$
$= (20 - 61)\ mod\ 64$
$= (-41 + 64)\ mod\ 64$
$= 23 \rightarrow X$

$C_{16} = (k_{16} - P_{16})\ mod\ 64$
$= (20 - 13)\ mod\ 64$
$= 7 \rightarrow H$

$C_{17} = (k_{17} - P_{17})\ mod\ 64$
$= (8 - 20)\ mod\ 64$
$= (-12 + 64)\ mod\ 64$
$= 52 \rightarrow 0$

$C_{18} = (k_{18} - P_{18})\ mod\ 64$
$= (13 - 2)\ mod\ 64$
$= 11 \rightarrow L$

$C_{19} = (k_{19} - P_{19})\ mod\ 64$
$= (18 - 8)\ mod\ 64$
$= 10 \rightarrow K$

$C_{20} = (k_{20} - P_{20})\ mod\ 64$
$= (20 - 41)\ mod\ 64$
$= (-21 + 64)\ mod\ 64$
$= 43 \rightarrow r$

$C_{21} = (k_{21} - P_{21})\ mod\ 64$
$= (20 - 14)\ mod\ 64$
$= 6 \rightarrow G$

$C_{22} = (k_{22} - P_{22})\ mod\ 64$
$= (8 - 48)\ mod\ 64$
$= (-40 + 64)\ mod\ 64$
$= 24 \rightarrow Y$

Berdasarkan hasil perhitungan diatas maka diperoleh hasil enkripsi pertama (*ciphertext*1) dengan menggunakan algoritma *Beaufort Cipher* yaitu :

Plaintext : YWxlcnQoIklMS09NUCIpow
Kunci : UINSUUINSUUINSUUINSUUI
*Ciphertext*1 : 8yct4t4lKwv87eXH0LKrGY

Kemudian *ciphertext*1 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*, dan diketahui bahwa panjang karakter kunci tidak sama dengan panjang karakter dari *plaintext*, maka kunci akan diulang secara periodik, maka akan terlihat seperti berikut:

Tabel 4.6 Substitusi Angka *Vigenere Cipher* Berdasarkan Tabel *Base64*

Plaintext	8	y	c	t	4	t	4	l	K	w	v	8	7	e	X	H	0	L	K	r	G	Y
Substitusi	60	50	28	45	56	45	56	37	10	48	47	60	59	30	23	7	52	11	10	43	6	24
Kunci	U	I	N	S	U	U	I	N	S	U	U	I	N	S	U	U	I	N	S	U	U	I
Substitusi	20	8	13	18	20	20	8	13	18	20	20	8	13	18	20	20	8	13	18	20	20	8

Selanjutnya, masuk ke proses enkripsi *Vigenere Cipher* menggunakan rumus (2.7), dimana proses perhitungannya sebagai berikut :

$C_1 = (P_1 + k_1)\ mod\ 64$
$= (60 + 20)\ mod\ 64$
$= 16 \rightarrow Q$

$C_2 = (P_2 + k_2)\ mod\ 64$
$= (50 + 8)\ mod\ 64$
$= 58 \rightarrow 6$

$C_3 = (P_3 + k_3)\ mod\ 64$
$= (28 + 13)\ mod\ 64$
$= 41 \rightarrow p$

$C_4 = (P_4 + k_4)\ mod\ 64$
$= (45 + 18)\ mod\ 64$
$= 63 \rightarrow /$

$C_5 = (P_5 + k_5)\ mod\ 64$
$= (56 + 20)\ mod\ 64$
$= 12 \rightarrow M$

$C_6 = (P_6 + k_6)\ mod\ 64$
$= (45 + 20)\ mod\ 64$
$= 1 \rightarrow B$

$C_7 = (P_7 + k_7)\ mod\ 64$
$= (56 + 8)\ mod\ 64$
$= 0 \rightarrow A$

$C_8 = (P_8 + k_8)\ mod\ 64$
$= (37 + 13)\ mod\ 64$
$= 50 \rightarrow y$

$C_9 = (P_9 + k_9)\ mod\ 64$
$= (10 + 18)\ mod\ 64$
$= 28 \rightarrow c$

$C_{10} = (P_{10} + k_{10})\ mod\ 64$
$= (48 + 20)\ mod\ 64$
$= 4 \rightarrow E$

$C_{11} = (P_{11} + k_{11})\ mod\ 64$
$= (47 + 20)\ mod\ 64$
$= 3 \rightarrow D$

$C_{12} = (P_{12} + k_{12})\ mod\ 64$
$= (60 + 8)\ mod\ 64$
$= 4 \rightarrow E$

$C_{13} = (P_{13} + k_{13}) \; mod \; 64$
$= (59 + 13) \; mod \; 64$
$= 8 \rightarrow I$

$C_{14} = (P_{14} + k_{14}) \; mod \; 64$
$= (30 + 18) \; mod \; 64$
$= 48 \rightarrow w$

$C_{15} = (P_{15} + k_{15}) \; mod \; 64$
$= (23 + 20) \; mod \; 64$
$= 43 \rightarrow r$

$C_{16} = (P_{16} + k_{16}) \; mod \; 64$
$= (7 + 20) \; mod \; 64$
$= 27 \rightarrow b$

$C_{17} = (P_{17} + k_{17}) \; mod \; 64$
$= (52 + 8) \; mod \; 64$
$= 60 \rightarrow 8$

$C_{18} = (P_{18} + k_{18}) \; mod \; 64$
$= (11 + 13) \; mod \; 64$
$= 24 \rightarrow Y$

$C_{19} = (P_{19} + k_{19}) \; mod \; 64$
$= (10 + 18) \; mod \; 64$
$= 28 \rightarrow c$

$C_{20} = (P_{20} + k_{20}) \; mod \; 64$
$= (43 + 20) \; mod \; 64$
$= 63 \rightarrow /$

$C_{21} = (P_{21} + k_{21}) \; mod \; 64$
$= (6 + 20) \; mod \; 64$
$= 26 \rightarrow a$

$C_{22} = (P_{22} + k_{22}) \; mod \; 64$
$= (24 + 8) \; mod \; 64$
$= 32 \rightarrow g$

Berdasarkan hasil perhitungan diatas maka diperoleh hasil enkripsi kedua (*ciphertext*2) dengan menggunakan algoritma *Vigenere Cipher* yaitu "Q6p/MBAyc EDEIwrb8Yc/ag". Adapun perbandingan antara *source code* asli dengan *source code* yang telah terenkripsi dengan menggunakan algoritma Base64, *Beaufort Cipher* dan *Vigenere Cipher* dapat dilihat pada Tabel 4.7 berikut :

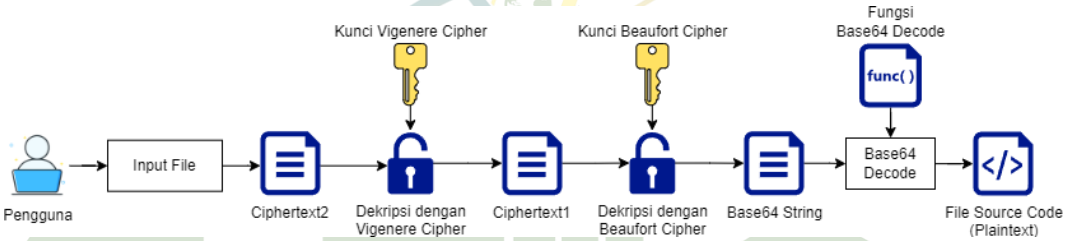
Tabel 4.7 Perbandingan Hasil Enkripsi

***Source Code* Asli**	*Plaintext*	alert("ILKOMP");
	Size	16 *bytes*
***Source Code* (Format *Base64*)**	*Plaintext*	YWxlcnQoIklMS09NUCIpOw
	Size	22 *bytes*
Hasil Enkripsi *Beaufort Cipher*	*Ciphertext*	8yct4t4lKwv87eXH0LKrGY
	Size	22 *bytes*
Hasil Enkripsi *Vigenere Cipher*	*Ciphertext*	Q6p/MBAycEDEIwrb8Yc/ag
	Size	22 *bytes*

Dapat disimpulkan bahwa setelah melalui dua tahap enkripsi, *ciphertext* yang dihasilkan menjadi lebih acak dan tidak menunjukkan hubungan dengan *plaintext*, karena hasil enkripsinya dalam format *Base64 string*, namun akan menghasilkan *ciphertext* yang ukurannya lebih besar dari *plaintext* aslinya.

4.2 Analisis Proses Dekripsi

Setelah *ciphertext* diperoleh, maka selanjutnya melakukan proses dekripsi yang bertujuan untuk mengembalikan isi *file source code* yang telah dienkripsi kembali ke bentuk semula, menjadi kode program yang dapat dieksekusi. Berikut Gambar 4.3, memberikan gambaran tentang proses dekripsi pada penelitian ini.



Gambar 4.3 Alur Proses Dekripsi *File Source Code*

Berdasarkan Gambar 4.3 terdapat tiga proses utama yang dilakukan dalam melakukan proses dekripsi isi *file source code*. Proses pertama yaitu melakukan dekripsi menggunakan kunci algoritma *Vigenere Cipher* sehingga menghasilkan *ciphertext*1, lalu di dekripsi lagi menggunakan kunci algoritma *Beaufort Cipher* sehingga menghasilkan *Base64 string*. Terakhir melakukan transformasi kembali menggunakan *Base64 decode* sehingga mengembalikan isi *file source code* semula (*plaintext*), menjadi kode program yang dapat dieksekusi oleh mesin komputer.

Berdasarkan *ciphertext*2 dari proses enkripsi sebelumnya akan dilakukan proses dekripsi menggunakan kunci algoritma *Vigenere Cipher*. Langkah awal yang dilakukan yaitu mensubstitusikan setiap karakter *plaintext* dan kunci berdasarkan tabel *Base64*, dan kunci akan diulang secara periodik, maka akan terlihat seperti berikut :

Tabel 4.8 Substitusi Angka *Vigenere Cipher* Berdasarkan Tabel *Base64*

Plaintext	Q	6	p	/	M	B	A	y	c	E	D	E	I	w	r	b	8	Y	c	/	a	g
Substitusi	16	58	41	63	12	1	0	50	28	4	3	4	8	48	43	27	60	24	28	63	26	32
Kunci	U	I	N	S	U	U	I	N	S	U	U	I	N	S	U	U	I	N	S	U	U	I
Substitusi	20	8	13	18	20	20	8	13	18	20	20	8	13	18	20	20	8	13	18	20	20	8

Selanjutnya, masuk ke proses dekripsi *Vigenere Cipher* menggunakan rumus (2.8), dimana proses perhitungannya sebagai berikut :

$C_1 = (P_1 - k_1)\ mod\ 64$
$= (16 - 20)\ mod\ 64$
$= (-4 + 64)\ mod\ 64$
$= 60 \rightarrow 8$

$C_2 = (P_2 - k_2)\ mod\ 64$
$= (58 - 8)\ mod\ 64$
$= 50 \rightarrow y$

$C_3 = (P_3 - k_3)\ mod\ 64$
$= (41 - 13)\ mod\ 64$
$= 28 \rightarrow c$

$C_4 = (P_4 - k_4)\ mod\ 64$
$= (63 - 18)\ mod\ 64$
$= 45 \rightarrow t$

$C_5 = (P_5 - k_5)\ mod\ 64$
$= (12 - 20)\ mod\ 64$
$= (-8 + 64)\ mod\ 64$
$= 56 \rightarrow 4$

$C_6 = (P_6 - k_6)\ mod\ 64$
$= (1 - 20)\ mod\ 64$
$= (-19 + 64)\ mod\ 64$
$= 45 \rightarrow t$

$C_7 = (P_7 - k_7)\ mod\ 64$
$= (0 - 8)\ mod\ 64$
$= (-8 + 64)\ mod\ 64$
$= 56 \rightarrow 4$

$C_8 = (P_8 - k_8)\ mod\ 64$
$= (50 - 13)\ mod\ 64$
$= 37 \rightarrow l$

$C_9 = (P_9 - k_9)\ mod\ 64$
$= (28 - 18)\ mod\ 64$
$= 10 \rightarrow K$

$C_{10} = (P_{10} - k_{10})\ mod\ 64$
$= (4 - 20)\ mod\ 64$
$= (-16 + 64)\ mod\ 64$
$= 48 \rightarrow w$

$C_{11} = (P_{11} - k_{11})\ mod\ 64$
$= (3 - 20)\ mod\ 64$
$= (-17 + 64)\ mod\ 64$
$= 47 \rightarrow v$

$C_{12} = (P_{12} - k_{12})\ mod\ 64$
$= (4 - 8)\ mod\ 64$
$= (-4 + 64)\ mod\ 64$
$= 60 \rightarrow 8$

$C_{13} = (P_{13} - k_{13})\ mod\ 64$
$= (8 - 13)\ mod\ 64$
$= (-5 + 64)\ mod\ 64$
$= 59 \rightarrow 7$

$C_{14} = (P_{14} - k_{14})\ mod\ 64$
$= (48 - 18)\ mod\ 64$
$= 30 \rightarrow e$

$C_{15} = (P_{15} - k_{15})\ mod\ 64$
$= (43 - 20)\ mod\ 64$
$= 23 \rightarrow X$

$C_{16} = (P_{16} - k_{16})\ mod\ 64$
$= (27 - 20)\ mod\ 64$
$= 7 \rightarrow H$

$C_{17} = (P_{17} - k_{17})\ mod\ 64$
$= (60 - 8)\ mod\ 64$
$= 52 \rightarrow 0$

$C_{18} = (P_{18} - k_{18})\ mod\ 64$
$= (24 - 13)\ mod\ 64$
$= 11 \rightarrow L$

$C_{19} = (P_{19} - k_{19})\ mod\ 64$
$= (28 - 18)\ mod\ 64$
$= 10 \rightarrow K$

$C_{20} = (P_{20} - k_{20})\ mod\ 64$
$= (63 - 20)\ mod\ 64$
$= 43 \rightarrow r$

$C_{21} = (P_{21} - k_{21})\ mod\ 64$
$= (26 - 20)\ mod\ 64$
$= 6 \rightarrow G$

$C_{22} = (P_{22} - k_{22})\ mod\ 64$
$= (32 - 8)\ mod\ 64$
$= 24 \rightarrow Y$

Berdasarkan hasil perhitungan diatas diperoleh hasil dekripsi pertama dengan menggunakan algoritma *Vigenere Cipher* yaitu "8yct4t4lKwv87eXH0LKr GY". Kemudian akan di dekripsi lagi dengan menggunakan kunci algoritma *Beaufort Cipher*. Langkah awal yang dilakukan yaitu mensubstitusikan setiap karakter *plaintext* dan kunci berdasarkan tabel *Base64*, dan kunci akan diulang secara periodik, maka akan terlihat seperti berikut :

Tabel 4.9 Substitusi Angka *Beaufort Cipher* Berdasarkan Tabel *Base64*

Plaintext	8	y	c	t	4	t	4	l	K	w	v	8	7	e	X	H	0	L	K	r	G	Y
Substitusi	60	50	28	45	56	45	56	37	10	48	47	60	59	30	23	7	52	11	10	43	6	24
Kunci	U	I	N	S	U	U	I	N	S	U	U	I	N	S	U	U	I	N	S	U	U	I
Substitusi	20	8	13	18	20	20	8	13	18	20	20	8	13	18	20	20	8	13	18	20	20	8

Selanjutnya, masuk ke proses dekripsi *Beaufort Cipher* menggunakan rumus (2.6), dimana proses perhitungannya sebagai berikut :

$C_1 = (k_1 - P_1)\ mod\ 64$
$= (20 - 60)\ mod\ 64$
$= (-40 + 64)\ mod\ 64$
$= 24 \rightarrow Y$

$C_2 = (k_2 - P_2)\ mod\ 64$
$= (8 - 50)\ mod\ 64$
$= (-42 + 64)\ mod\ 26$
$= 22 \rightarrow W$

$C_3 = (k_3 - P_3)\ mod\ 64$

$C_4 = (k_4 - P_4)\ mod\ 64$

$= (13 - 28)\ mod\ 64$

$= (-15 + 64)\ mod\ 64$

$= 49 \rightarrow x$

$C_5 = (k_5 - P_5)\ mod\ 64$

$= (20 - 56)\ mod\ 64$

$= (-36 + 64)\ mod\ 64$

$= 28 \rightarrow c$

$C_7 = (k_7 - P_7)\ mod\ 64$

$= (8 - 56)\ mod\ 64$

$= (-48 + 64)\ mod\ 64$

$= 16 \rightarrow Q$

$C_9 = (k_9 - P_9)\ mod\ 64$

$= (18 - 10)\ mod\ 64$

$= 8 \rightarrow I$

$C_{11} = (k_{11} - P_{11})\ mod\ 64$

$= (20 - 47)\ mod\ 64$

$= (-27 + 64)\ mod\ 64$

$= 37 \rightarrow l$

$C_{13} = (k_{13} - P_{13})\ mod\ 64$

$= (13 - 59)\ mod\ 64$

$= (-46 + 64)\ mod\ 64$

$= 18 \rightarrow S$

$C_{15} = (k_{15} - P_{15})\ mod\ 64$

$= (20 - 23)\ mod\ 64$

$= (-3 + 64)\ mod\ 64$

$= 61 \rightarrow 9$

$C_{17} = (k_{17} - P_{17})\ mod\ 64$

$= (8 - 52)\ mod\ 64$

$= (-44 + 64)\ mod\ 64$

$= 20 \rightarrow U$

$= (18 - 45)\ mod\ 64$

$= (-27 + 64)\ mod\ 64$

$= 37 \rightarrow l$

$C_6 = (k_6 - P_6)\ mod\ 64$

$= (20 - 45)\ mod\ 64$

$= (-25 + 64)\ mod\ 64$

$= 39 \rightarrow n$

$C_8 = (k_8 - P_8)\ mod\ 64$

$= (13 - 37)\ mod\ 64$

$= (-24 + 64)\ mod\ 64$

$= 40 \rightarrow o$

$C_{10} = (k_{10} - P_{10})\ mod\ 64$

$= (20 - 48)\ mod\ 64$

$= (-28 + 64)\ mod\ 64$

$= 36 \rightarrow k$

$C_{12} = (k_{12} - P_{12})\ mod\ 64$

$= (8 - 60)\ mod\ 64$

$= (-52 + 64)\ mod\ 64$

$= 12 \rightarrow M$

$C_{14} = (k_{14} - P_{14})\ mod\ 64$

$= (18 - 30)\ mod\ 64$

$= (-12 + 64)\ mod\ 64$

$= 52 \rightarrow 0$

$C_{16} = (k_{16} - P_{16})\ mod\ 64$

$= (20 - 7)\ mod\ 64$

$= 13 \rightarrow N$

$C_{18} = (k_{18} - P_{18})\ mod\ 64$

$= (13 - 11)\ mod\ 64$

$= 2 \rightarrow C$

$C_{19} = (k_{19} - P_{19})\ mod\ 64$

$C_{20} = (k_{20} - P_{20}) \; mod \; 64$
$= (20 - 43) \; mod \; 64$
$= (-23 + 64) \; mod \; 64$
$= 41 \rightarrow p$
$C_{22} = (k_{22} - P_{22}) \; mod \; 64$
$= (8 - 24) \; mod \; 64$
$= (-16 + 64) \; mod \; 64$
$= 48 \rightarrow w$
$= (18 - 10) \; mod \; 64$
$= 8 \rightarrow I$
$C_{21} = (k_{21} - P_{21}) \; mod \; 64$
$= (20 - 6) \; mod \; 64$
$= 14 \rightarrow O$

Berdasarkan hasil perhitungan diatas diperoleh hasil dekripsi kedua dengan menggunakan algoritma *Beaufort Cipher* yaitu "YWxlcnQoIklMS09NUCIpOw". Terakhir, melakukan proses *Base64 decode* dengan mengkonversikan setiap karakter *Base64* sesuai dengan nilaiya pada tabel *Base64*, kemudian konversikan bilangan desimal ke dalam biner 6-bit.

Tabel 4.10 Konversi Ke Bilangan Biner Dan *Mapping* Biner 6-Bit

Base64	Desimal	Biner 6-Bit
Y	$24_{(10)}$	011000
W	$22_{(10)}$	010110
x	$49_{(10)}$	110001
l	$37_{(10)}$	100101
c	$28_{(10)}$	011100
n	$39_{(10)}$	100111
Q	$16_{(10)}$	010000
o	$40_{(10)}$	101000
I	$8_{(10)}$	001000
k	$36_{(10)}$	100100
l	$37_{(10)}$	100101
M	$12_{(10)}$	001100
S	$18_{(10)}$	010010
0	$52_{(10)}$	110100
9	$61_{(10)}$	111101

Base64	Desimal	Biner 6-Bit
N	$13_{(10)}$	001101
U	$20_{(10)}$	010100
C	$2_{(10)}$	000010
I	$8_{(10)}$	001000
p	$41_{(10)}$	101001
O	$14_{(10)}$	001110
w	$48_{(10)}$	110000

Selanjutnya di *mapping* lagi menjadi biner 8-bit dan dikonversi ke dalam bilangan heksadesimal, maka akan terlihat seperti berikut :

Tabel 4.11 *Mapping* Biner 8-Bit Dan Konversi Ke Bilangan Heksadesimal

Biner 8-Bit	Konversi Bilangan Heksadesimal	ASCII (Heksadesimal)
01100001	$0110 \rightarrow (0\times2^3) + (1\times2^2) + (1\times2^1) + (0\times2^0) = 6$ $0001 \rightarrow (0\times2^3) + (0\times2^2) + (0\times2^1) + (1\times2^0) = 1$	61
01101100	$0110 \rightarrow (0\times2^3) + (1\times2^2) + (1\times2^1) + (0\times2^0) = 6$ $1100 \rightarrow (1\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0) = 12$	6C
01100101	$0110 \rightarrow (0\times2^3) + (1\times2^2) + (1\times2^1) + (0\times2^0) = 6$ $0101 \rightarrow (0\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0) = 5$	65
01110010	$0111 \rightarrow (0\times2^3) + (1\times2^2) + (1\times2^1) + (1\times2^0) = 7$ $0010 \rightarrow (0\times2^3) + (0\times2^2) + (1\times2^1) + (0\times2^0) = 2$	72
01110100	$0111 \rightarrow (0\times2^3) + (1\times2^2) + (1\times2^1) + (1\times2^0) = 7$ $0100 \rightarrow (0\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0) = 4$	74
00101000	$0010 \rightarrow (0\times2^3) + (0\times2^2) + (1\times2^1) + (0\times2^0) = 2$ $1000 \rightarrow (1\times2^3) + (0\times2^2) + (0\times2^1) + (0\times2^0) = 8$	28
00100010	$0010 \rightarrow (0\times2^3) + (0\times2^2) + (1\times2^1) + (0\times2^0) = 2$ $0010 \rightarrow (0\times2^3) + (0\times2^2) + (1\times2^1) + (0\times2^0) = 2$	22
01001001	$0100 \rightarrow (0\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0) = 4$ $1001 \rightarrow (1\times2^3) + (0\times2^2) + (0\times2^1) + (1\times2^0) = 9$	49
01001100	$0100 \rightarrow (0\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0) = 4$ $1100 \rightarrow (1\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0) = 12$	4C

Biner 8-Bit	Konversi Bilangan Heksadesimal	ASCII (Heksadesimal)
01001011	$0100 \rightarrow (0\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0) = 4$ $1011 \rightarrow (1\times2^3) + (0\times2^2) + (1\times2^1) + (1\times2^0) = 11$	4B
01001111	$0100 \rightarrow (0\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0) = 4$ $1111 \rightarrow (1\times2^3) + (1\times2^2) + (1\times2^1) + (1\times2^0) = 15$	4F
01001101	$0100 \rightarrow (0\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0) = 4$ $1101 \rightarrow (1\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0) = 13$	4D
01010000	$0101 \rightarrow (0\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0) = 5$ $0000 \rightarrow (0\times2^3) + (0\times2^2) + (0\times2^1) + (0\times2^0) = 0$	50
00100010	$0010 \rightarrow (0\times2^3) + (0\times2^2) + (1\times2^1) + (0\times2^0) = 2$ $0010 \rightarrow (0\times2^3) + (0\times2^2) + (1\times2^1) + (0\times2^0) = 2$	22
00101001	$0010 \rightarrow (0\times2^3) + (0\times2^2) + (1\times2^1) + (0\times2^0) = 2$ $1001 \rightarrow (1\times2^3) + (0\times2^2) + (0\times2^1) + (1\times2^0) = 9$	29
00111011	$0011 \rightarrow (0\times2^3) + (0\times2^2) + (1\times2^1) + (1\times2^0) = 3$ $1011 \rightarrow (1\times2^3) + (0\times2^2) + (1\times2^1) + (1\times2^0) = 11$	3B

Konversikan setiap nilai bilangan heksadesimal yang didapatkan sesuai dengan karakter yang ada pada tabel ASCII. Dari proses perhitungan yang dilakukan, maka isi dari *file source code* asli kembali lagi didapatkan yaitu alert(“ILKOMP”);.

Tabel 4.12 Konversi Bilangan Heksadesimal Ke Karakter ASCII

ASCII (Heksadesimal)	Karakter
61	a
6C	l
65	e
72	r
74	t
28	(
22	“
49	I
4C	L
4B	K

ASCII (Heksadesimal)	Karakter
4F	O
4D	M
50	P
22	“
29	)
3B	;

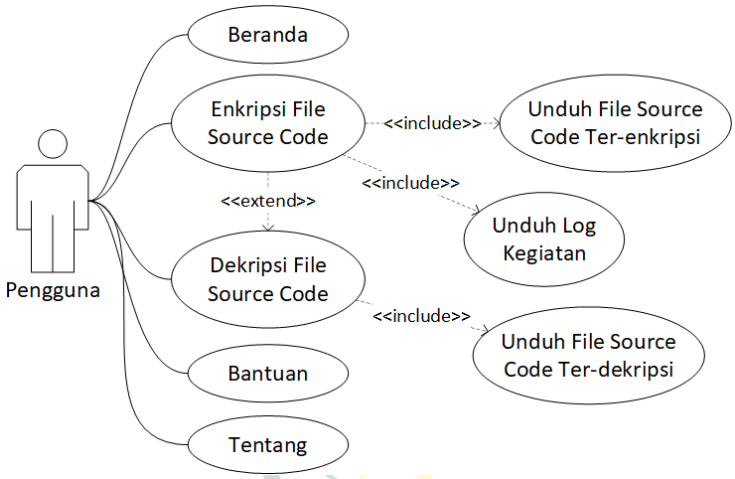
4.3 Perancangan Sistem

Perancangan sistem dilakukan dengan membuat UML (*Unified Modeling Language*) terlebih dahulu agar membantu dalam membuat sistem yang lebih terorganisir, terdokumentasi, dan efisien dalam proses pengembangannya. Dengan menggunakan *use case diagram* yang digunakan untuk merancang struktur dan interaksi sistem dengan pengguna, kemudian *activity diagram* dan *sequence diagram* yang digunakan untuk pemahaman lebih mendalam mengenai bagaimana sistem berfungsi dan bagaimana alur kerjanya. Perancangan tampilan antarmuka pengguna berperan dalam menjembatani komunikasi antara sistem dengan pengguna. Tampilan antarmuka pengguna yang akan dirancang pada sistem ini terdiri dari enam bagian utama, yaitu halaman beranda (utama), halaman enkripsi, halaman dekripsi, halaman bantuan, dan halaman tentang. Sistem ini dikembangkan berbasis *web* dengan menggunakan bahasa pemrograman JavaScript.

4.3.1 Perancangan UML (*Unified Modeling Language*)

UML (*Unified Modeling Language*) menjadi alat penting dalam siklus hidup pengembangan perangkat lunak, mulai dari perencanaan hingga implementasi. Pada UML ada banyak jenis diagram yang digunakan untuk perancangan sistem. Diagram yang dibuat untuk perancangan sistem pada penelitian ini ada tiga yaitu *use case diagram*, *activity diagram*, dan *sequence diagram*.

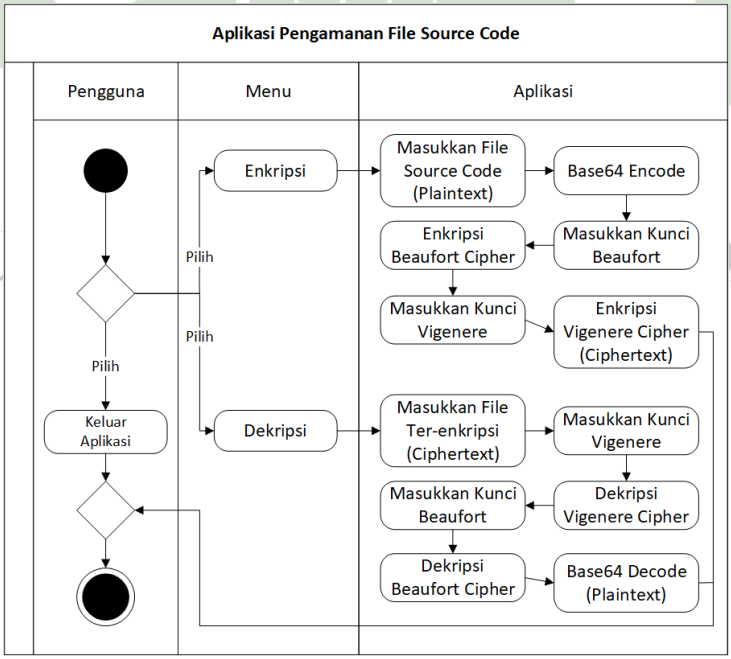
Adapun *use case diagram* yang dibuat untuk perancangan sistem pada penelitian ini, dapat dilihat pada Gambar 4.4 berikut :



Gambar 4.4 *Use Case Diagram* Dari Sistem Yang Dirancang

Use case diagram diatas memberikan gambaran tentang aktor yang dapat berinteraksi dengan kelima menu yang ada pada aplikasi. Menu-menu yang ada diantaranya menu beranda, enkripsi, dekripsi, bantuan, dan tentang. Pada menu enkripsi terdapat fungsi untuk mengunduh *file source code* yang telah ter-enkrpsi dan *log* kegiatan. Pada menu dekripsi juga terdapat fungsi untuk mengunduh *file source code* yang telah ter-dekrpsi. Selanjutnya untuk merepresentasikan bagaimana aktivitas dari proses enkripsi dan dekripsi *file source code*, digunakan *activity diagram*.

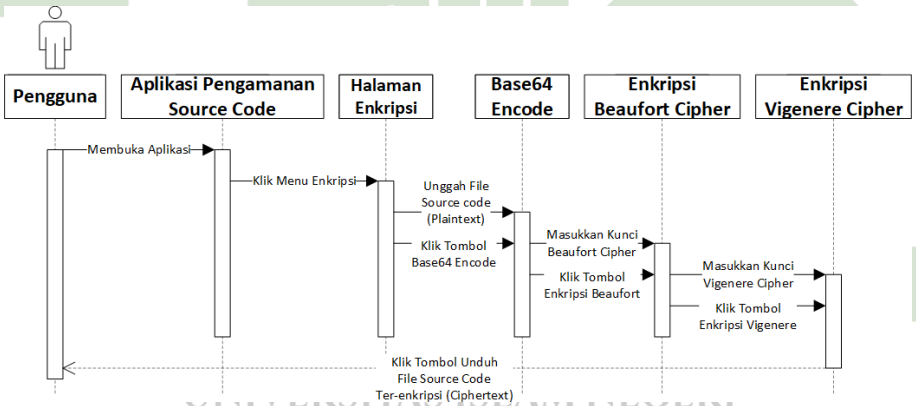
Adapun *activity diagram* yang dibuat untuk perancangan sistem pada penelitian ini, dapat dilihat pada Gambar 4.5 berikut :



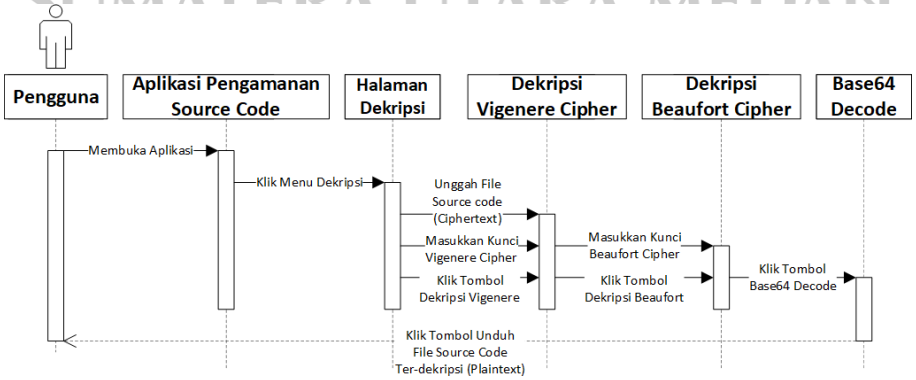
Gambar 4.5 *Activity Diagram* Dari Sistem Yang Dirancang

Activity diagram memberikan gambaran tentang bagaimana aktivitas dari proses enkripsi dan dekripsi *file source code* pada aplikasi. Seorang pengguna atau pengembang yang ingin mengamankan *file source code* mereka, diharuskan pergi ke menu enkripsi, kemudian memasukkan *file source code* (*plaintext*) yang ingin diamankan, beserta kunci yang digunakan untuk mengamankan *file source code* tersebut, lalu aplikasi akan melakukan proses enkripsi sesuai dengan arsitektur sistem dan menghasilkan *file source code* yang telah ter-enkripsi (*ciphertext*). Begitu juga sebaliknya, jika ingin mengembalikan *file source code* mereka, diharuskan pergi ke menu dekripsi, kemudian memasukkan *file source code* yang ter-enkripsi (*ciphertext*), beserta kunci yang digunakan untuk membuka *file source code* tersebut, lalu aplikasi akan melakukan proses dekripsi sesuai dengan arsitektur sistem dan memberikan *file source code* semula (*plaintext*). Berikutnya untuk merepresentasikan interaksi antar objek dalam bentuk pesan dalam urutan waktu tertentu, digunakan *sequence diagram*.

Adapun *sequence diagram* yang dibuat untuk perancangan sistem pada penelitian ini, dapat dilihat pada Gambar 4.6 dan 4.7 berikut :



Gambar 4.6 *Sequence Diagram* Dari Proses Enkripsi *File Source Code*

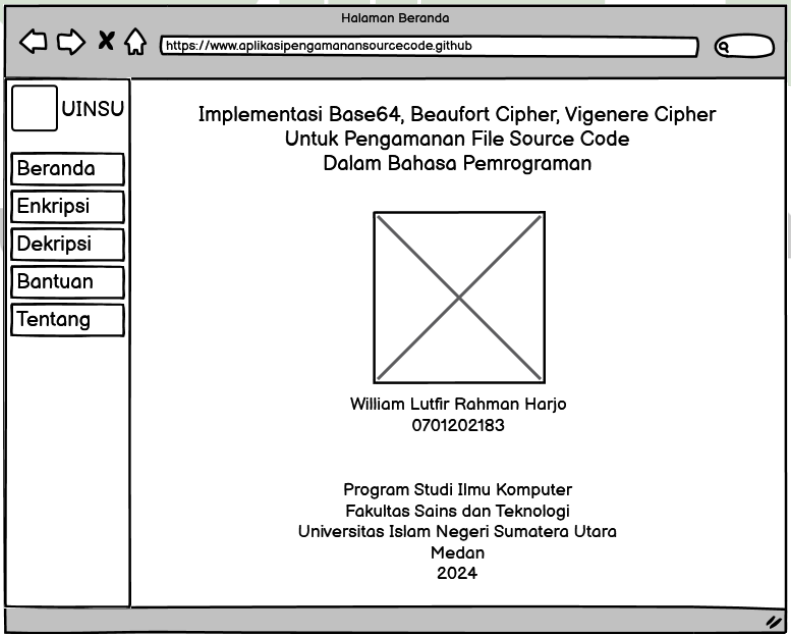


Gambar 4.7 *Sequence Diagram* Dari Proses Dekripsi *File Source Code*

Sequence diagram memberikan gambaran tentang setiap interaksi dari pengguna dengan objek-objek di dalam sistem, dalam proses enkripsi dan dekripsi *file source code*, yang dinotasikan sebagai pesan dalam urutan waktu tertentu. Seorang pengguna atau pengembang yang ingin mengamankan *file source code* mereka, diharuskan klik menu enkripsi, maka akan mengirimkan pesan ke sistem untuk menampilkan halaman enkripsi. Kemudian memasukkan *file source code* (*plaintext*) yang ingin diamankan, beserta kunci yang digunakan untuk mengamankan *file source code* tersebut, maka akan mengirimkan pesan ke sistem untuk melakukan proses enkripsi sesuai dengan arsitektur sistem dan menghasilkan *file source code* yang telah ter-enkripsi (*ciphertext*). Begitu juga sebaliknya, untuk mengembalikan *file source code*, diharuskan klik menu dekripsi, maka akan mengirimkan pesan ke sistem untuk menampilkan halaman dekripsi. Kemudian memasukkan *file source code* yang ter-enkripsi (*ciphertext*), beserta kunci yang digunakan untuk membuka *file source code* tersebut, maka akan mengirimkan pesan ke sistem untuk melakukan proses dekripsi sesuai dengan arsitektur sistem dan memberikan *file source code* semula (*plaintext*).

4.3.2 Perancangan Antarmuka Halaman Beranda

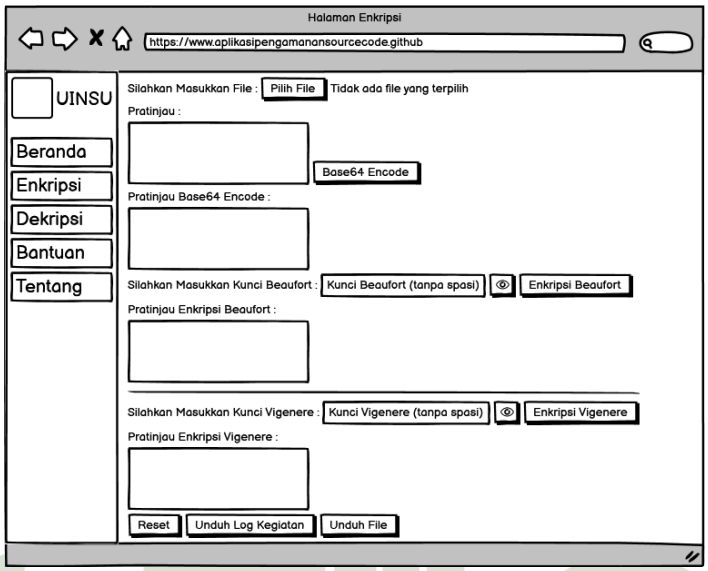
Halaman beranda adalah halaman yang akan pertama kali ditampilkan saat aplikasi dibuka. Rancangan dari halaman beranda dapat dilihat pada Gambar 4.8 :



Gambar 4.8 Rancangan Antarmuka Halaman Beranda

4.3.3 Perancangan Antarmuka Halaman Enkripsi

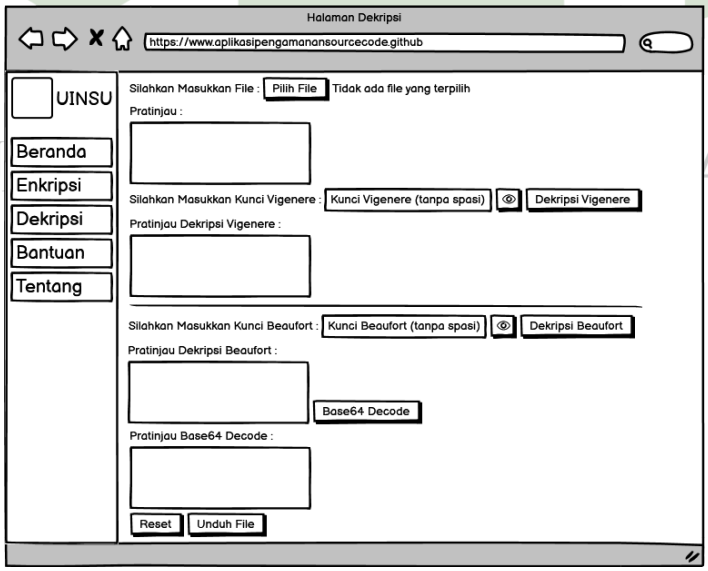
Halaman enkripsi berisi *form* yang dirancang untuk melakukan proses enkripsi isi dari *file source code* (*plaintext*) dengan menggunakan algoritma *Base64*, *Beaufort Cipher*, *Vigenere Cipher*. Rancangan halaman enkripsi dapat dilihat pada Gambar 4.9 :



Gambar 4.9 Rancangan Antarmuka Halaman Enkripsi

4.3.4 Perancangan Antarmuka Halaman Dekripsi

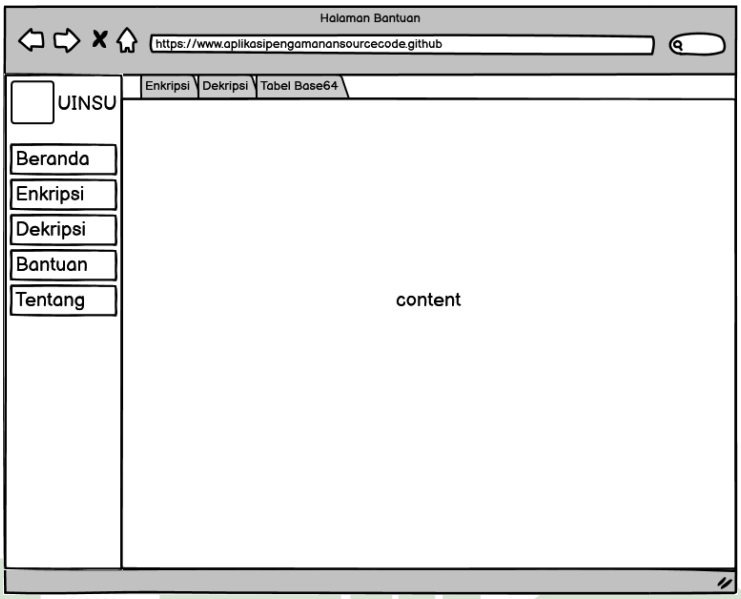
Halaman dekripsi berisi *form* yang dirancang untuk melakukan proses dekripsi *file source code* yang terenkripsi (*ciphertext*), menggunakan algoritma *Base64*, *Beaufort Cipher*, *Vigenere Cipher*. Rancangan halaman dekripsi terlihat pada Gambar 4.10 :



Gambar 4.10 Rancangan Antarmuka Halaman Dekripsi

4.3.5 Perancangan Antarmuka Halaman Bantuan

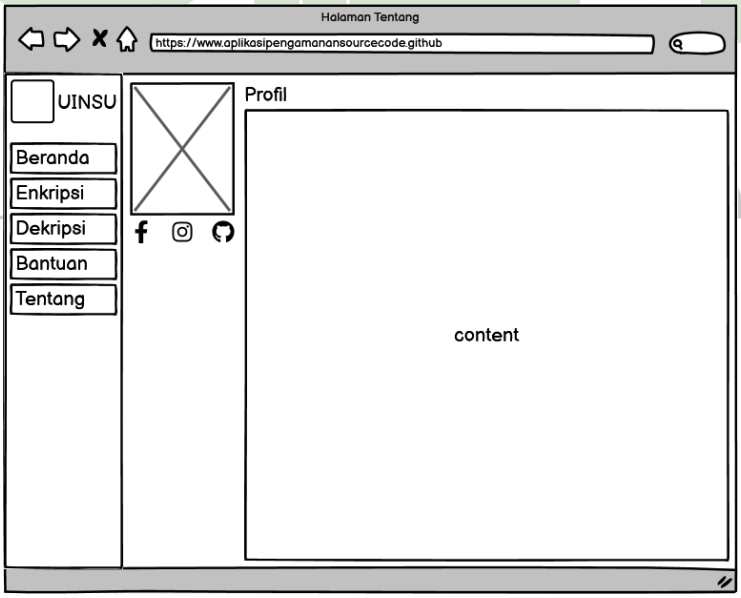
Halaman bantuan dirancang untuk menampilkan informasi tentang cara penggunaan dari aplikasi, bagaimana cara melakukan enkripsi dan dekripsi *file source code* dengan aplikasi tersebut. Rancangan halaman bantuan terlihat pada Gambar 4.11 :



Gambar 4.11 Rancangan Antarmuka Halaman Bantuan

4.3.6 Perancangan Antarmuka Halaman Tentang

Halaman tentang dirancang untuk menampilkan informasi tentang profil dari pembuat aplikasi tersebut. Rancangan halaman tentang terlihat pada Gambar 4.12 :



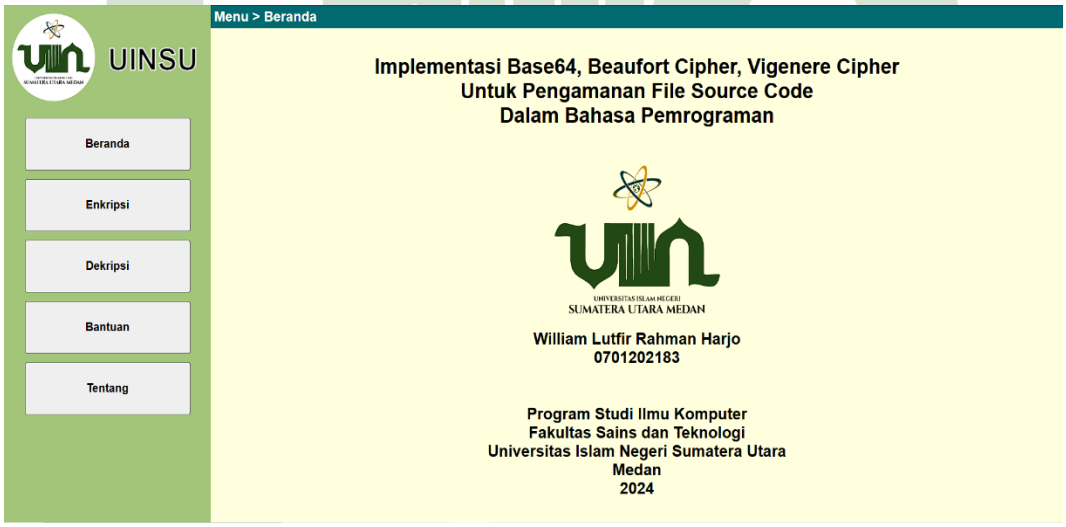
Gambar 4.12 Rancangan Antarmuka Halaman Tentang

4.4 Implementasi Program

Implementasi program adalah tahap yang dilakukan setelah perancangan sistem dan pembuatan *flowchart* sistem. Setelah analisis dan perancangan sistem selesai, langkah berikutnya adalah mengimplementasikan hasil tersebut ke dalam bentuk aplikasi menggunakan bahasa pemrograman. Dalam penelitian ini, implementasi program dibuat berbasis *web* dan terdiri dari lima halaman, yaitu halaman beranda, halaman enkripsi, halaman dekripsi, halaman bantuan, dan halaman tentang. Ruang lingkup spesifikasi kebutuhan perangkat lunak dan perangkat keras yang digunakan dalam pengembangan dan menjalankan aplikasi ini, telah dijelaskan pada bab tiga sebelumnya.

4.4.1 Implementasi Halaman Beranda

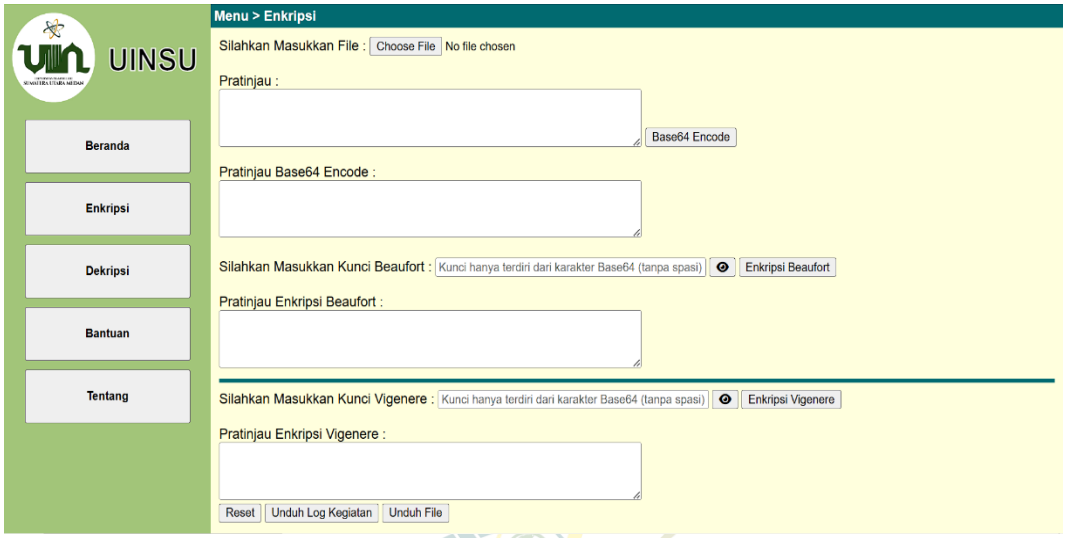
Halaman beranda sebagai halaman utama yang akan pertama kali ditampilkan saat aplikasi dibuka. Pada halaman ini juga akan menampilkan menu yang dapat diakses oleh pengguna, yaitu menu enkripsi, menu dekripsi, menu bantuan, menu tentang. Implementasi halaman beranda dapat dilihat pada Gambar 4.13 :



Gambar 4.13 Tampilan Halaman Beranda

4.4.2 Implementasi Halaman Enkripsi

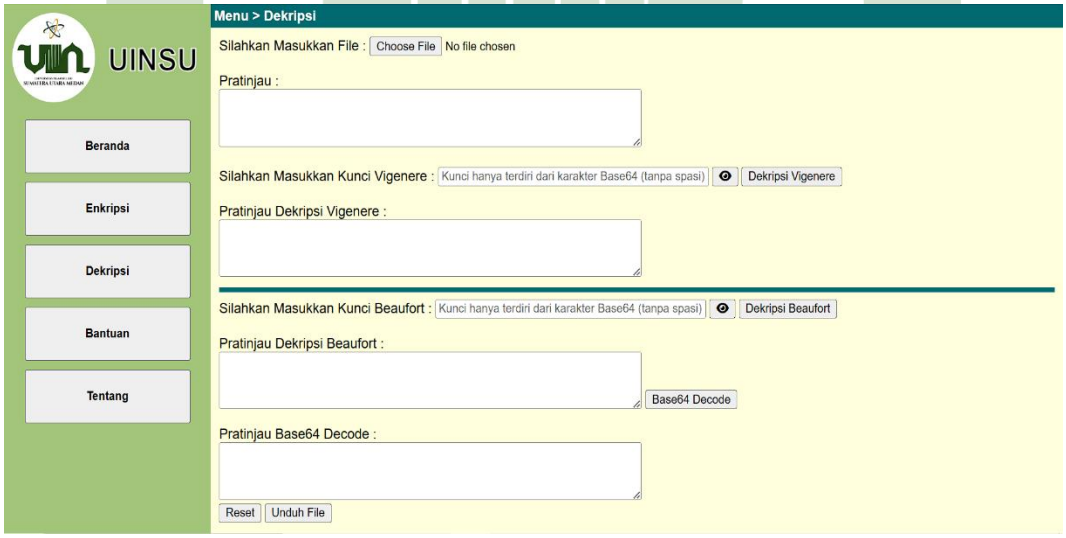
Halaman enkripsi berfungsi untuk melakukan proses enkripsi isi dari *file source code* (*plaintext*) dengan menggunakan algoritma *Base64*, *Beaufort Cipher*, *Vigenere Cipher*. Implementasi halaman enkripsi dapat dilihat pada Gambar 4.14 :



Gambar 4.14 Tampilan Halaman Enkripsi

4.4.3 Implementasi Halaman Dekripsi

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 4.15 :



Gambar 4.15 Tampilan Halaman Dekripsi

4.4.4 Implementasi Halaman Bantuan

Halaman bantuan berfungsi untuk menampilkan informasi yang membantu cara penggunaan dari aplikasi, dan membantu pengguna bagaimana cara untuk melakukan enkripsi dan dekripsi *file source code* dengan aplikasi tersebut. Rancangan halaman bantuan terlihat pada Gambar 4.16 :

Gambar 4.16 Tampilan Halaman Bantuan

4.4.5 Implementasi Halaman Tentang

Halaman tentang berfungsi untuk menampilkan informasi tentang profil dari pembuat aplikasi tersebut. Rancangan halaman tentang terlihat pada Gambar 4.17 :

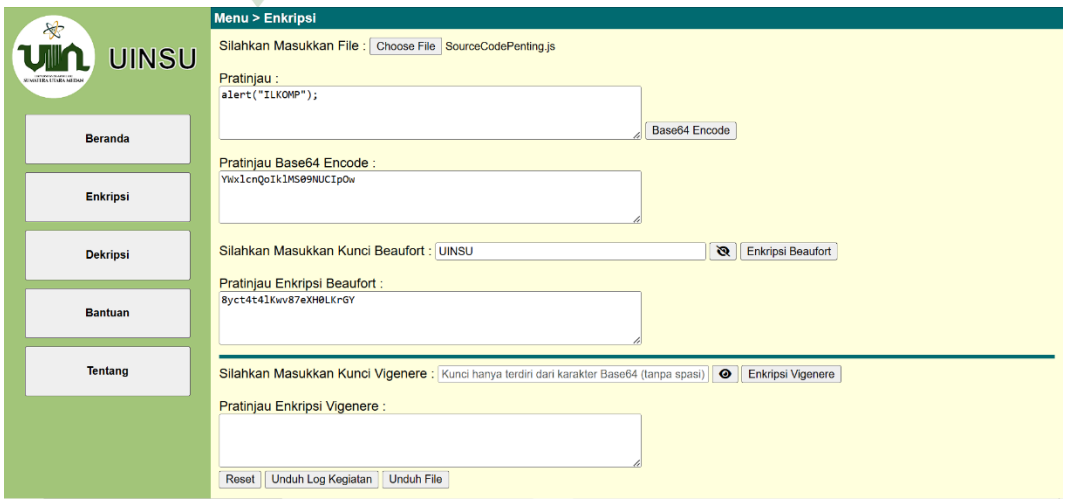
Gambar 4.17 Tampilan Halaman Tentang

4.5 Hasil Pengujian

Hasil pengujian menunjukkan kemampuan atau keakuratan metode yang digunakan dalam menyelesaikan masalah yang sedang dihadapi. Hal ini mengacu pada penerapan algoritma *Base64*, *Beaufort Cipher*, *Vigenere Cipher* pada sistem yang telah dibangun, akan diuji dengan sampel data berupa *file* uji. Data *input* yang digunakan sebagai sampel dalam pengujian ini adalah *file source code* dalam berbagai format dan ukuran yang bervariasi.

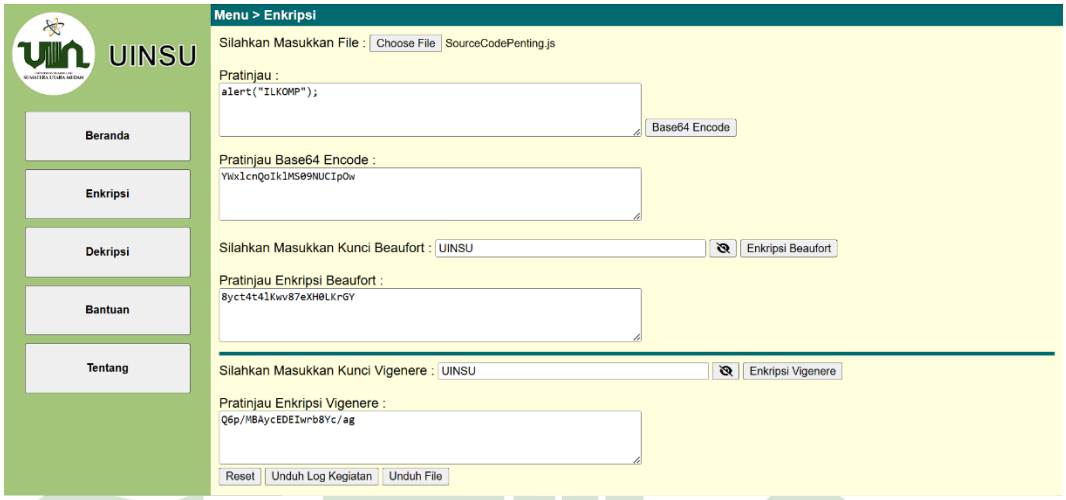
4.5.1 Hasil Pengujian Enkripsi

Pada saat klik menu enkripsi, maka akan dibawa ke halaman enkripsi yang berisi *form*, yang digunakan untuk melakukan proses enkripsi isi dari *file source code* (*plaintext*) dengan menggunakan algoritma *Base64*, *Beaufort Cipher*, *Vigenere Cipher*. Langkah awal yang dilakukan yaitu mengunggah *file source code* yang akan dienkripsi dengan klik tombol "*Choose File*" kemudian pilih *file source code* tersebut, maka sistem akan menampilkan pratinjau isi dari *file source code* tersebut didalam *textbox*. Berikutnya melakukan proses *Base64 encode* dengan klik tombol "*Base64 Encode*", maka sistem akan menampilkan pratinjau *Base64 string* didalam *textbox*. Selanjutnya melakukan proses enkripsi *Beaufort Cipher* dan memasukkan kunci yang digunakan untuk algoritma *Beaufort Cipher*, kemudian klik tombol "Enkripsi *Beaufort*", maka sistem akan menampilkan pratinjau *Base64 string* yang telah dienkripsi didalam *textbox*. seperti yang terlihat pada Gambar 4.18 :



Gambar 4.18 Hasil Pengujian Enkripsi *Beaufort Cipher*

Hasil enkripsi pertama dari algoritma *Beaufort Cipher* akan menghasilkan *ciphertext*1, selanjutnya melakukan enkripsi lagi dengan *Vigenere Cipher* dan memasukkan kunci yang sama untuk algoritma *Vigenere Cipher*, kemudian klik tombol "Enkripsi *Vigenere*", maka sistem akan menampilkan pratinjau *Base64 string* yang telah dienkripsi didalam *textbox*, seperti yang terlihat pada Gambar 4.19 :

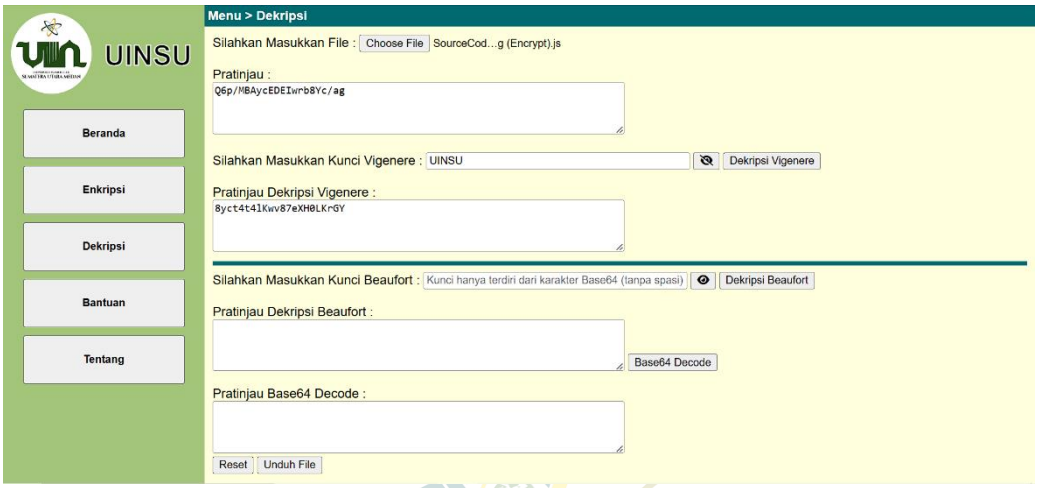


Gambar 4.19 Hasil Pengujian Enkripsi *Vigenere Cipher*

Hasil pengujian enkripsi kedua dari algoritma *Vigenere Cipher* akan menghasilkan *ciphertext*2, dimana ini merupakan *file source* yang telah terenkripsi dan dapat diunduh dengan klik tombol "Unduh *File*". Dapat disimpulkan bahwa setelah melalui dua tahap enkripsi, *ciphertext* yang dihasilkan menjadi lebih acak dan tidak menunjukkan hubungan dengan *plaintext*, karena hasil enkripsinya dalam format *Base64 string*, dimana ini tidak akan mudah didekripsi hanya dengan melakukan *Base64 decode* saja tanpa melewati dua algoritma kriptografi tersebut.

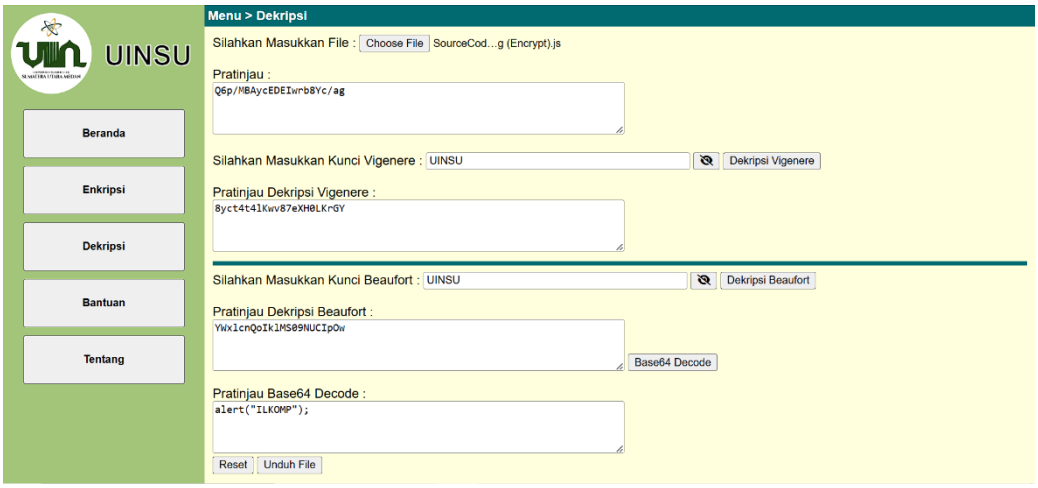
4.5.2 Hasil Pengujian Dekripsi

Pada saat klik menu dekripsi, maka akan dibawa ke halaman dekripsi yang berisi *form*, yang digunakan untuk melakukan proses dekripsi dari *file source code* yang terenkripsi (*ciphertext*) dengan menggunakan algoritma *Base64*, *Beaufort Cipher*, *Vigenere Cipher*. Langkah awal yang dilakukan yaitu mengunggah *file source code* yang terenkripsi dengan klik tombol "*Choose File*" kemudian pilih *file source code* tersebut, maka sistem akan menampilkan pratinjau isi dari *file source code* yang terenkripsi tersebut didalam *textbox*, seperti terlihat pada Gambar 4.20 :



Gambar 4.20 Hasil Pengujian Dekripsi *Vigenere Cipher*

Berikutnya melakukan proses dekripsi *Vigenere Cipher* dan memasukkan kunci yang digunakan untuk algoritma *Vigenere Cipher*, kemudian klik tombol "Dekripsi *Vigenere*", maka sistem akan menampilkan pratinjau hasil dekripsi *Vigenere Cipher* didalam *textbox*. Selanjutnya melakukan dekripsi lagi dengan *Beaufort Cipher* dan memasukkan kunci yang sama untuk algoritma *Beaufort Cipher*, kemudian klik tombol "Dekripsi *Beaufort*", maka sistem akan menampilkan pratinjau hasil dekripsi *Beaufort Cipher* didalam *textbox*. Terakhir melakukan transformasi kembali menggunakan *Base64 decode* dengan klik tombol "*Base64 Decode*" sehingga akan mengembalikan *file source code* asli (*plaintext*) dan dapat diunduh dengan klik tombol "Unduh *File*". Dapat disimpulkan, *file source code* asli kembali didapatkan dan menjadi kode program yang dapat dieksekusi oleh mesin komputer, seperti terlihat pada Gambar 4.21 :



Gambar 4.21 Hasil Pengujian Dekripsi *Beaufort Cipher*

4.5.3 Hasil Pengujian *Avalanche Effect*

Avalanche effect digunakan untuk 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). *Avalanche effect* penting dalam algoritma kriptografi karena memastikan bahwa pesan asli sulit dilacak kembali dari hasil enkripsi. 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 4.13 :

Tabel 4.13 Hasil Pengujian Enkripsi Berbagai Format *File Source Code*

No.	Nama *File*	*Size Plaintext* (KB)	Kunci	*Size Ciphertext* (KB)	Waktu (ms)
1	*File*Uji.js	0.016 KB	UINSU	0.022 KB	3.40 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 *File*Uji.js :

Plaintext : alert("ILKOMP"); Kunci : UINSU

Enkripsi *Beaufort* (*Ciphertext*1) : 8yct4t4lKwv87eXH0LKrGY

Enkripsi *Vigenere* (*Ciphertext*2) : Q6p/MBAycEDEIwrb8Yc/ag

Langkah perhitungan *avalanche effect* :

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.

Tabel 4.14 Membandingkan Setiap Bit Antara Dua *Ciphertext*

<table>
<tr><td colspan="48">Enkripsi Beaufort (Ciphertext1)</td></tr>
<tr><td colspan="8">8</td><td colspan="8">y</td><td colspan="8">c</td><td colspan="8">t</td><td colspan="8">4</td><td colspan="8">t</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td colspan="8">Q</td><td colspan="8">6</td><td colspan="8">p</td><td colspan="8">/</td><td colspan="8">M</td><td colspan="8">B</td></tr>
<tr><td colspan="48">Enkripsi Vigenere (Ciphertext2)</td></tr>
</table>

<table>
<tr><td colspan="48">Enkripsi Beaufort (Ciphertext1)</td></tr>
<tr><td colspan="8">4</td><td colspan="8">l</td><td colspan="8">K</td><td colspan="8">w</td><td colspan="8">v</td><td colspan="8">8</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td colspan="8">A</td><td colspan="8">y</td><td colspan="8">c</td><td colspan="8">E</td><td colspan="8">D</td><td colspan="8">E</td></tr>
<tr><td colspan="48">Enkripsi Vigenere (Ciphertext2)</td></tr>
</table>

<table>
<tr><td colspan="48">Enkripsi Beaufort (Ciphertext1)</td></tr>
<tr><td colspan="8">7</td><td colspan="8">e</td><td colspan="8">X</td><td colspan="8">H</td><td colspan="8">0</td><td colspan="8">L</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td colspan="8">I</td><td colspan="8">w</td><td colspan="8">r</td><td colspan="8">b</td><td colspan="8">8</td><td colspan="8">Y</td></tr>
<tr><td colspan="48">Enkripsi Vigenere (Ciphertext2)</td></tr>
</table>

<table>
<tr><td colspan="32">Enkripsi Beaufort (Ciphertext1)</td></tr>
<tr><td colspan="8">K</td><td colspan="8">r</td><td colspan="8">G</td><td colspan="8">Y</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td colspan="8">c</td><td colspan="8">/</td><td colspan="8">a</td><td colspan="8">g</td></tr>
<tr><td colspan="32">Enkripsi Vigenere (Ciphertext2)</td></tr>
</table>

2. Hitung persentase dengan rumus pada Persamaan (2.9).

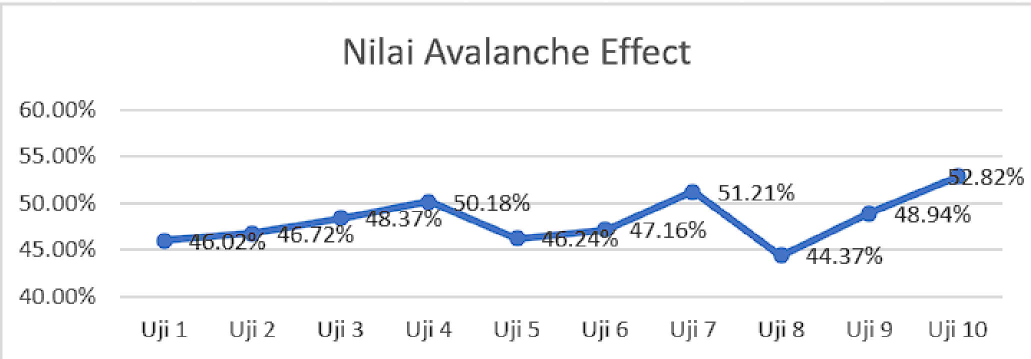
Hasil perhitungan *avalanche effect* adalah sebagai berikut :

Jumlah bit yang berbeda : 81

Total bit : 176

$$\frac{81}{176} \times 100\% = 46.02\%$$

Setelah melakukan 10 pengujian terhadap *file source code*, hasil pengujian *avalanche effect* dapat dilihat pada Gambar 4.22 :



Gambar 4.22 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*. Hasil dari 10 pengujian yang dilakukan, untuk mendapatkan nilai *avalanche effect* 50% harus mengenkripsi *file source code* dengan ukuran diatas 1 MB.

4.5.4 Hasil Pengujian *Bit Error Rate*

Bit error rate digunakan untuk mengevaluasi performa dari sistem yang menerapkan algoritma kriptografi. BER mengukur frekuensi kesalahan bit dalam transmisi data dan menilai kualitas enkripsi dan dekripsi data. Berikut ini adalah salah satu perhitungan nilai *bit error rate* pada *File*Uji.js :

Source code asli : alert("ILKOMP");

Hasil dekripsi : alert("ILKOMP");

Langkah perhitungan *bit error rate* :

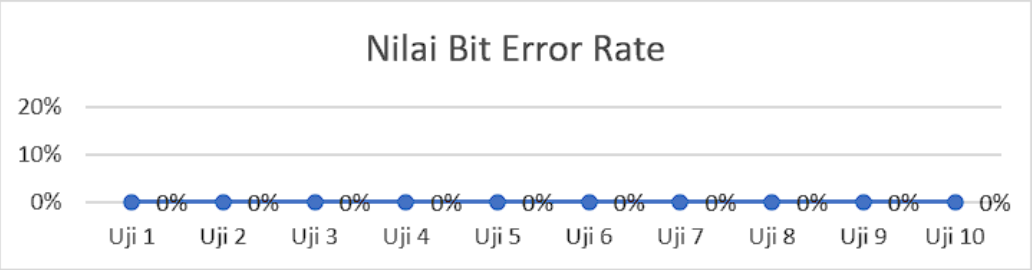
1. Ubah *source code* asli dan hasil dekripsi ke dalam bentuk biner 8-bit berdasarkan pengkodean ASCII.
2. Bandingkan setiap bit dan hitung jumlah bit yang salah.
3. Hitung persentase dengan rumus pada Persamaan (2.10).

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. $\frac{0}{128} \times 100\% = 0\%$

Total bit : 128

Setelah melakukan 10 pengujian terhadap *file source code* sebelumnya, hasil pengujian *bit error rate* dapat dilihat pada Gambar 4.23 :



Gambar 4.23 Grafik Hasil Pengujian *Bit Error Rate*

Nilai *bit error rate* 0% artinya tidak ada kesalahan bit yang terjadi selama proses enkripsi dan dekripsi, dimana dalam konteks kriptografi, ini menunjukkan bahwa kombinasi algoritma kriptografi yang digunakan bekerja dengan baik tanpa adanya gangguan. Nilai *bit error rate* yang diperoleh memiliki makna penting dalam menilai keandalan suatu sistem yang menerapkan algoritma kriptografi.

4.5.5 Hasil Pengujian *Character Error Rate*

Character error rate digunakan untuk mengukur presentase tingkat akurasi sebuah hasil dekripsi dengan cara mencocokan dan membandingkan *character* (huruf, angka, simbol) dengan *plaintext*-nya. CER mengukur seberapa sering karakter yang dihasilkan setelah proses dekripsi berbeda dari karakter aslinya. Berikut adalah salah satu perhitungan nilai *character error rate* pada *File*Uji.js :

Source code asli : alert("ILKOMP");

Hasil dekripsi : alert("ILKOMP");

Langkah perhitungan *character error rate* :

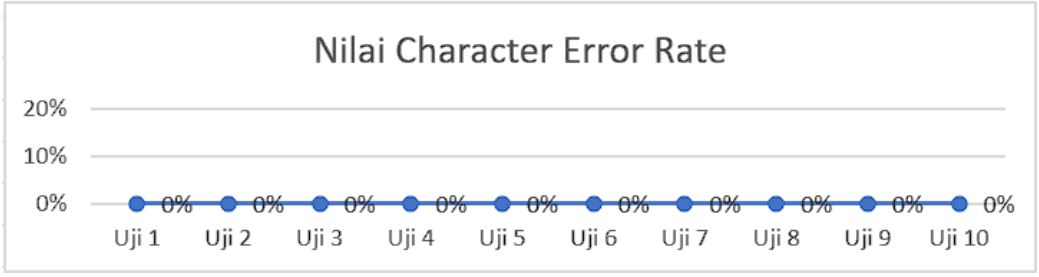
1. Hitung jumlah keseluruhan karakter.
2. Bandingkan setiap karakter dan hitung jumlah karakter yang salah.
3. Hitung persentase dengan rumus pada Persamaan (2.11).

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. $\frac{0}{16} \times 100\% = 0\%$

Total karakter : 16

Setelah melakukan 10 pengujian terhadap *file source code* sebelumnya, hasil pengujian *character error rate* dapat dilihat pada Gambar 4.24 :



Gambar 4.24 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, dimana *file source code* terenkripsi berhasil didekripsi dengan baik tanpa adanya kesalahan karakter. Nilai *character error rate* yang diperoleh mengindikasikan bahwa sistem melakukan proses enkripsi dan dekripsi bekerja dengan baik.

4.5.6 Hasil Pengujian *Entropy*

Entropy dalam konteks kriptografi merujuk pada ukuran keacakan dalam sistem kriptografi. *Entropy* digunakan untuk menunjukkan seberapa acak distribusi karakter dan simbol dalam *ciphertext* dan sulit untuk memprediksi atau menganalisis pola dalam *ciphertext*. Berikut adalah salah satu perhitungan nilai *entropy* pada *File*Uji.js :

*Ciphertext*2 : Q6p/MBAycEDEIwrb8Yc/ag

Langkah perhitungan *entropy* :

1. Hitung frekuensi setiap huruf dalam *ciphertext*.

Q=1; 6=1; p=1; /=2; M=1; B=1; A=1; y=1; c=2; E=2; D=1; I=1; w=1; r=1; b=1; 8=1; Y=1; a=1; g=1

2. Hitung probabilitas masing-masing huruf.

Q=1/22; 6=1/22; p=1/22; /=2/22; M=1/22; B=1/22; A=1/22; y=1/22; c=2/22; E=2/22; D=1/22; I=1/22; w=1/22; r=1/22; b=1/22; 8=1/22; Y=1/22; a=1/22; g=1/22

3. Hitung nilai *entropy* dengan rumus pada Persamaan (2.12).

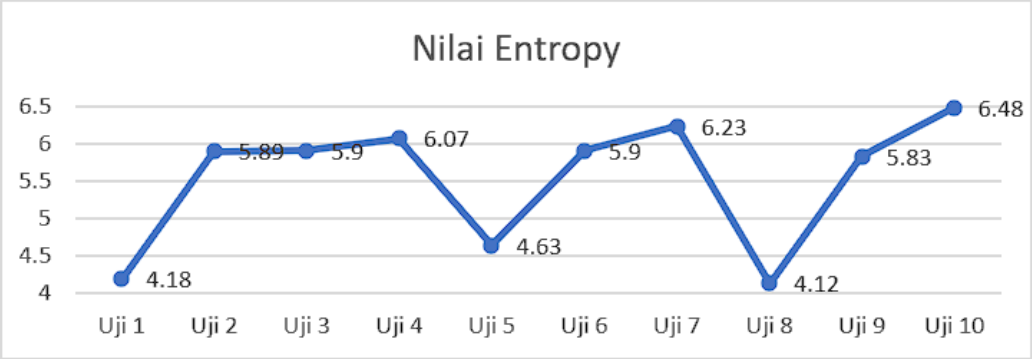
Hasil perhitungan nilai *entropy* adalah sebagai berikut :

$$H(X) = -\left(\left(16 \times \left(\frac{1}{22}\right) \times \log_2 \frac{1}{22}\right) + \left(3 \times \left(\frac{2}{22}\right) \times \log_2 \frac{2}{22}\right)\right)$$

$$H(X) = -(-3{,}24 + (-0{,}94))$$

$$H(X) = 4{,}18 \text{ bit}$$

Setelah melakukan 10 pengujian terhadap *file source code* sebelumnya, hasil pengujian *entropy* dapat dilihat pada Gambar 4.25 :



Gambar 4.25 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.

4.5.7 Hasil 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. Dalam metode ini, sejumlah *input* diberikan ke program, kemudian diproses sesuai kebutuhan fungsional untuk mengevaluasi apakah *output* yang dihasilkan sesuai yang diinginkan dengan fungsi dasar program. Jika *output* yang dihasilkan sesuai dengan kebutuhan fungsional, maka program dianggap benar. Namun, jika *output* tidak sesuai, berarti terdapat kesalahan dalam program, yang kemudian perlu ditelusuri dan harus diperbaiki.

Adapun pengujian *black box* yang dilakukan adalah sebagai berikut :

1. Pengujian *black box*, melakukan enkripsi *file source code*

   Berikut adalah hasil pengujian *black box* pada proses enkripsi *file source code*, dapat disajikan pada Tabel 4.15.

Tabel 4.15 Pengujian *Black Box* Enkripsi *File Source Code*

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.	*Input* kunci *Beaufort* dengan benar	Mengetikkan kunci *Beaufort Cipher* pada *input text* yang hanya terdiri dari kombinasi karakter *Base64 alphabet*.	Sistem hanya dapat menerima kunci yang terdiri dari kombinasi karakter *Base64 alphabet*.	Valid
4.	*Input* kunci *Beaufort* dengan kesalahan	Mengetikkan kunci *Beaufort Cipher* pada *input text* yang terdiri dari kombinasi diluar karakter *Base64 alphabet*.	Sistem akan munculkan pesan notifikasi bahwa kunci terdapat karakter yang diluar dari *Base64 alphabet*.	Tidak Valid
5.	Enkripsi *Beaufort Cipher*	Mengenkripsi *file source code* dengan cara mengklik tombol "Enkripsi *Beaufort*".	Sistem dapat menam-pilkan transformasi *Base64 string* yang telah di enkripsi dengan kunci *Beaufort Cipher*.	Valid
6.	*Input* kunci *Vigenere*	Mengetikkan kunci *Vigenere Cipher* pada *input text* yang hanya	Sistem hanya dapat menerima kunci yang	Valid

No.	Kasus Uji	Langkah Uji	Hasil	Keterangan
	dengan benar	terdiri dari kombinasi karakter *Base64 alphabet*.	terdiri dari kombinasi karakter *Base64 alphabet*.	
7.	*Input* kunci *Vigenere* dengan kesalahan	Mengetikkan kunci *Vigenere Cipher* pada *input text* yang terdiri dari kombinasi diluar karakter *Base64 alphabet*.	Sistem akan munculkan pesan notifikasi bahwa kunci terdapat karakter yang diluar dari *Base64 alphabet*.	Tidak Valid
8.	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
9.	Unduh *ciphertext*	Mengunduh hasil enkripsi *file source code* dengan klik tombol "Unduh *File*".	Sistem dapat menyimpan hasil enkripsi (*ciphertext*) ke dalam *file* baru sesuai dengan ekstensinya dan dapat terunduh ke perangkat pengguna.	Valid
10.	Unduh *log* kegiatan	Mengunduh *file log* kegiatan dengan cara klik tombol "Unduh *Log* Kegiatan".	Sistem dapat membuat *file log* kegiatan yang akan berisi tanggal dan waktu enkripsi, nama *file source code*, kunci yang digunakan, dan dapat terunduh ke perangkat pengguna.	Valid

Tabel 4.15 pengujian *black box* dengan kasus uji untuk enkripsi *file source code*, status hasil pengujian yang berhasil.

2. Pengujian *black box*, melakukan dekripsi *file source code*

   Berikut adalah hasil pengujian *black box* pada proses dekripsi *file source code*, dapat disajikan pada Tabel 4.16.

Tabel 4.16 Pengujian *Black Box* Dekripsi *File Source Code*

No.	Kasus Uji	Langkah Uji	Hasil	Keterangan
1.	*Input file source code*	Mengunggah *file source code* yang terenkripsi (*ciphertext*) dengan klik tombol "*Choose File*".	Sistem dapat menerima *file source code* dalam berbagai ekstensi dan menampilkan isinya.	Valid
2.	*Input* kunci *Vigenere* dengan benar	Mengetikkan kunci *Vigenere Cipher* pada *input text* yang hanya terdiri dari kombinasi karakter *Base64 alphabet*.	Sistem hanya dapat menerima kunci yang terdiri dari kombinasi karakter *Base64 alphabet*.	Valid
3.	*Input* kunci *Vigenere* dengan kesalahan	Mengetikkan kunci *Vigenere Cipher* pada *input text* yang terdiri dari kombinasi diluar karakter *Base64 alphabet*.	Sistem akan munculkan pesan notifikasi bahwa kunci terdapat karakter yang diluar dari *Base64 alphabet*.	Tidak Valid
3.	Dekripsi *Vigenere Cipher*	Mendekripsi *file source code* dengan klik tombol "Dekripsi *Vigenere*".	Sistem dapat menam-pilkan transformasi *Base64 string* yang telah di dekripsi dengan kunci *Vigenere Cipher*.	Valid
3.	*Input* kunci *Beaufort* dengan benar	Mengetikkan kunci *Beaufort Cipher* pada *input text* yang hanya terdiri dari kombinasi karakter *Base64 alphabet*.	Sistem hanya dapat menerima kunci yang terdiri dari kombinasi karakter *Base64 alphabet*.	Valid

No.	Kasus Uji	Langkah Uji	Hasil	Keterangan
4.	*Input* kunci *Beaufort* dengan kesalahan	Mengetikkan kunci *Beaufort Cipher* pada *input text* yang terdiri dari kombinasi diluar karakter *Base64 alphabet*.	Sistem akan munculkan pesan notifikasi bahwa kunci terdapat karakter yang diluar dari *Base64 alphabet*.	Tidak Valid
5.	Dekripsi *Beaufort Cipher*	Mendekripsi *file source code* dengan klik tombol "Dekripsi *Beaufort*".	Sistem dapat menam-pilkan transformasi *Base64 string* yang telah di dekripsi dengan kunci *Beaufort Cipher*.	Valid
6.	*Base64 Decode*	Melakukan proses *Base64 decode* dengan cara mengklik tombol "*Base64 Decode*".	Sistem dapat menam-pilkan hasil *Base64 decode* yang merupakan isi dari *file source code* asli.	Valid
7.	Unduh *plaintext*	Mengunduh hasil dekripsi *file source code* dengan klik tombol "Unduh *File*".	Sistem dapat menyimpan hasil dekripsi yang merupakan isi dari *file source code* asli (*plaintext*) ke dalam *file* baru sesuai dengan ekstensinya dan dapat terunduh ke perangkat pengguna.	Valid

Tabel 4.16 pengujian *black box* dengan kasus uji untuk dekripsi *file source code*, status hasil pengujian yang berhasil.

3. Pengujian *black box*, masuk ke halaman bantuan dan tentang

   Berikut adalah hasil pengujian *black* untuk masuk ke halaman bantuan dan tentang, dapat disajikan pada Tabel 4.17.

Tabel 4.17 Pengujian *Black Box* Masuk Halaman Bantuan Dan Tentang

No.	Kasus Uji	Langkah Uji	Hasil	Status
1.	Masuk ke halaman bantuan	Mengklik pada menu "Bantuan".	Sistem dapat menampilkan halaman informasi panduan tentang cara enkripsi dan dekripsi *file source code* pada aplikasi *web* tersebut.	Valid
2.	Masuk ke halaman tentang	Mengklik pada menu "Tentang".	Sistem dapat menampilkan halaman informasi tentang profil dari pengembang aplikasi *web* tersebut.	Valid

Tabel 4.17 pengujian *black box* dengan kasus uji untuk masuk ke halaman bantuan dan tentang, status hasil pengujian yang berhasil.