BAB II
TINJAUAN PUSTAKA

2.1 Privasi

Privasi adalah kata serapan dalam Bahasa Inggris yaitu "*privacy*" merujuk pada kemampuan individu atau kelompok individu untuk menjaga kehidupan pribadi dan urusan personal mereka dari publik, serta untuk mengendalikan aliran informasi tentang diri mereka. Privasi merupakan konsep abstrak yang memiliki banyak makna. Dengan kata lain privasi adalah hak individu untuk menentukan sejauh mana seseorang bersedia membagikan informasi tentang dirinya kepada orang lain, atau sebagai hak untuk tidak diganggu (Yuwinanto, 2020).

File source code dapat menjadi privasi yang harus dijaga. *File source code* seringkali menjadi aset berharga bagi perusahaan. Informasi sensitif seperti algoritma, struktur *database*, logika bisnis, dan strategi pengembangan terkandung di dalamnya. Jika *file source code* sampai bocor dapat meningkatkan risiko terhadap serangan keamanan. Pihak yang tidak berwenang dapat menemukan celah keamanan yang tidak terdeteksi dalam *file source code*, dan menyebabkan kerusakan atau pencurian data.

2.1.1 Urgensi Pengamanan Data Privasi

Setiap individu memiliki data privasi yang melekat padanya. Data privasi memiliki sifat sensitif dan perlu dilindungi karena merupakan hak privasi setiap individu. Data privasi merupakan hak privasi yang harus dilindungi dari berbagai aspek kehidupan. Karena memiliki sifat yang sensitif, data privasi menjadi objek yang menarik bagi pihak lain karena banyaknya kebutuhan kegiatan seseorang yang berkaitan dengannya. Data pribadi juga dapat dianggap sebagai aset atau komoditas dengan nilai ekonomi yang tinggi (Kusnadi & Wijaya, 2021).

Mengamankan *file source code* termasuk upaya mengamankan data privasi. Pengamanan *file source code* merupakan hal yang sangat penting, terutama dalam konteks pengembangan perangkat lunak dan keamanan informasi. Ada beberapa

alasan mengapa pengamanan *file source code* itu penting diantaranya, *file source code* seringkali berisi informasi rahasia tentang logika bisnis, algoritma, dan desain sistem, dimana ini perlu diamankan dalam upaya menjaga hak cipta. Kemudian melindungi *file source code* dari modifikasi yang tidak sah untuk memastikan bahwa aplikasi bekerja sesuai dengan fungsionalitas dan tidak rentan terhadap manipulasi yang merugikan.

2.1.2 Potensi Penyalahgunaan Data Privasi

Perkembangan teknologi informasi dan komunikasi memfasilitasi pengumpulan, penyimpanan, dan penggunaan data privasi dengan cepat dan mudah. Namun, hal ini juga menimbulkan kekhawatiran akan potensi penyalahgunaan data privasi yang dapat merugikan individu. Penyalahgunaan data privasi dapat mencakup pencurian identitas, penyadapan komunikasi, pelanggaran keamanan data, dan pelanggaran privasi (Mahuli, 2023).

File source code bisa memiliki potensi penyalahgunaan yang dapat merugikan pengembang aplikasi, perusahaan, atau bahkan masyarakat pengguna aplikasi tersebut secara keseluruhan. Beberapa contoh diantaranya, *file source code* berisi hal sensitif seperti kode untuk algoritma, keamanan, dan teknologi paten, dapat dicuri dan digunakan oleh pihak yang tidak berwenang dan menyebabkan pelanggaran hak cipta. Kemudian penjahat *cyber* bisa memanfaatkan *file source code* untuk mengeksploitasi kerentanan dalam sistem, dengan menyisipkan kode berbahaya, atau serangan lainnya.

2.2 Kriptografi

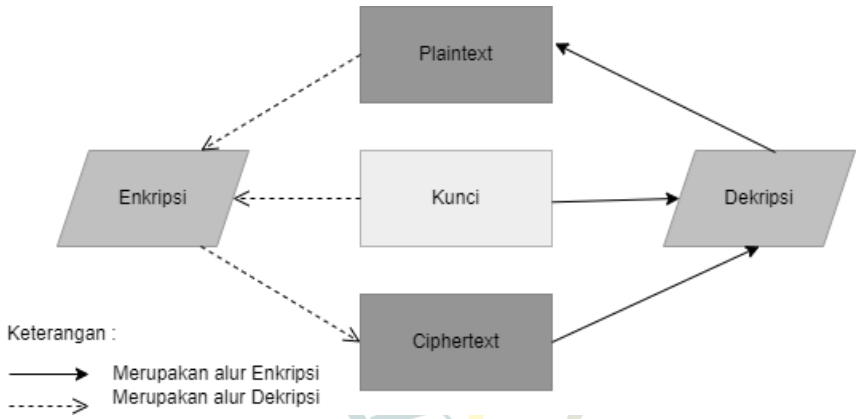
Kata kriptografi (*cryptography*) bersumber dari bahasa Yunani yang merupakan gabungan dari dua kata, *cryptos* dan *graphia*. *Cryptos* yang berarti tersembunyi atau rahasia, dan *graphia* yang berarti tulisan (Megantara & Rafrastara, 2020). Jadi kriptografi merupakan ilmu dan praktik yang berkaitan dengan teknik untuk mengamankan komunikasi dan menyembunyikan informasi dari pihak yang tidak berwenang. Pengertian kriptografi yang lebih lengkap adalah ilmu yang mempelajari teknik untuk menjaga kerahasiaan, integritas, dan

autentikasi dari informasi. Ini melibatkan penggunaan algoritma kriptografi dan kunci untuk mengamankan data atau informasi dari pihak yang tidak berwenang. Kriptografi klasik telah dikenal sejak zaman romawi kuno, yang pernah digunakan oleh Julius Caesar dan telah mengalami perkembangan yang signifikan seiring waktu menjadi kriptografi modern (Rantouli & Ikhsan, 2023).

Perlu diketahui terdapat istilah-istilah didalam kriptografi yang digunakan untuk memahami bagaimana konsep dari kriptografi, yang dimana sejatinya kriptografi adalah bidang studi yang memiliki sejumlah aturan dan metodenya sendiri. Berikut beberapa istilah yang umum digunakan dalam kriptografi :

1. *Plaintext* : Teks atau informasi asli yang dapat dibaca dan dimengerti oleh siapa saja, dan harus diamankan.
2. *Ciphertext* : Teks atau informasi yang telah diubah menjadi bentuk yang sulit dimengerti, dan sudah diamankan.
3. Algoritma kriptografi : Serangkaian langkah atau aturan matematis yang digunakan untuk mengamankan teks atau informasi dengan mengubah (*plaintext*) menjadi (*ciphertext*) dengan menggunakan suatu kunci. Algoritma kriptografi terbagi menjadi dua, algoritma kriptografi bersifat simetris dan algoritma kriptografi bersifat asimetris.
4. Kunci : Sebuah nilai atau deretan nilai yang digunakan dalam algoritma kriptografi untuk melakukan enkripsi dan dekripsi. Kunci juga terbagi menjadi dua, bisa bersifat simetris (digunakan untuk keduanya, enkripsi dan dekripsi) atau asimetris (menggunakan kunci publik untuk enkripsi dan kunci privat untuk dekripsi).
5. Enkripsi : Proses mengubah (*plaintext*) menjadi (*ciphertext*) menggunakan algoritma kriptografi dan kunci tertentu. Hanya penerima yang memiliki kunci yang benar yang dapat mengembalikan pesan ke bentuk aslinya.
6. Dekripsi : Proses mengembalikan pesan yang telah dienkripsi (*ciphertext*) ke dalam bentuk aslinya (*plaintext*) menggunakan kunci yang sesuai.
7. Kriptanalis : Proses menganalisis algoritma kriptografi untuk mencari kelemahan atau celah dari suatu algoritma kriptografi.

Berikut merupakan gambaran *flowchart* untuk memahami bagaimana konsep dari kriptografi, seperti yang terlihat pada Gambar 2.1 :



Gambar 2.1 *Flowchart* Dari Konsep Kriptografi

(Sumber : Karima et al., 2024)

Pada ranah kriptografi, setiap individu memiliki kebebasan untuk memilih algoritma yang diinginkan untuk menjaga kerahasiaan pesan. Berbagai algoritma tersebut dapat bervariasi pada setiap praktisi kriptografi, yang menghasilkan estetika dalam penulisan pesan rahasia. Ini membuat kriptografi disebut sebagai seni menyandikan pesan. Seiring perkembangannya, kriptografi kemudian juga menggunakan teknik matematika untuk mengamankan data atau informasi. Dalam menerapkan algoritma kriptografi, terdapat aspek yang harus dicapai, antara lain :

1. *Authentication* : penerima informasi dapat memastikan bahwa pesan tersebut berasal dari sumber yang diminta, dengan kata lain, informasi tersebut benar-benar dikirim oleh pihak yang dikehendaki.
2. *Integrity* : keaslian dari informasi yang dikirim melalui jaringan dapat dijamin, serta dipastikan bahwa informasi tersebut tidak dimodifikasi oleh pihak yang tidak berwenang.
3. *Non-repudiation* : menyatakan bahwa informasi dikirim oleh pengirim yang sebenarnya, artinya pengirim tidak bisa menyangkal bahwa dialah yang mengirimkan informasi tersebut.
4. *Authority* : Informasi dalam sistem jaringan tidak dapat dimodifikasi oleh pihak yang tidak memiliki wewenang untuk mengaksesnya.

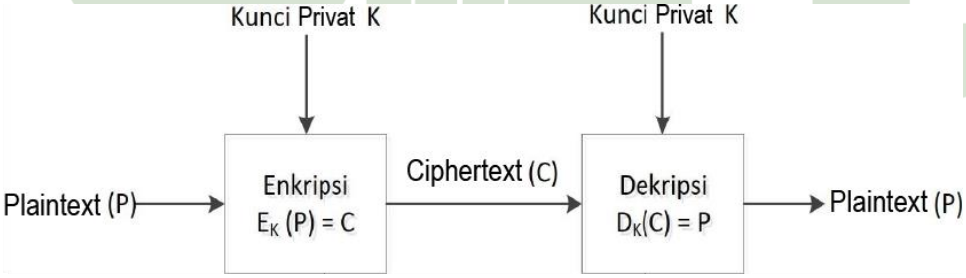
5. *Confidentiality* : upaya untuk melindungi informasi dari pihak yang tidak berwenang untuk mengaksesnya.
6. *Privacy* : merupakan data-data yang bersifat rahasia dan tidak boleh diakses oleh pihak yang tidak berwenang.
7. *Availability* : aspek yang mengacu pada tersedianya informasi saat diperlukan.
8. *Access Control* : berkaitan dengan pengaturan siapa saja yang berhak mengakses sistem dan memahami keamanan sistem tersebut (Suhardi, 2021).

2.2.1 Jenis Algoritma Kriptografi

Secara umum terdapat dua jenis algoritma kriptografi berdasarkan kunci yang digunakan, yaitu algoritma kriptografi simetris dan algoritma kriptografi asimetris.

1. Algoritma Kriptografi Simetris

   Algoritma kriptografi simetris adalah jenis algoritma kriptografi yang menggunakan kunci yang sama untuk proses enkripsi dan dekripsi. Algoritma simetris dapat dibagi menjadi dua kategori, yaitu *stream cipher* dan *block cipher*. *Stream cipher* adalah jenis algoritma kriptografi yang beroperasi pada level bit tunggal, sedangkan *block cipher* adalah jenis algoritma kriptografi yang beroperasi pada level *block* bit (Meko, 2020). Adapun skema proses dari algoritma kriptografi simetris, seperti yang terlihat pada Gambar 2.2 :



Gambar 2.2 Skema Proses Algoritma Kriptografi Simetris

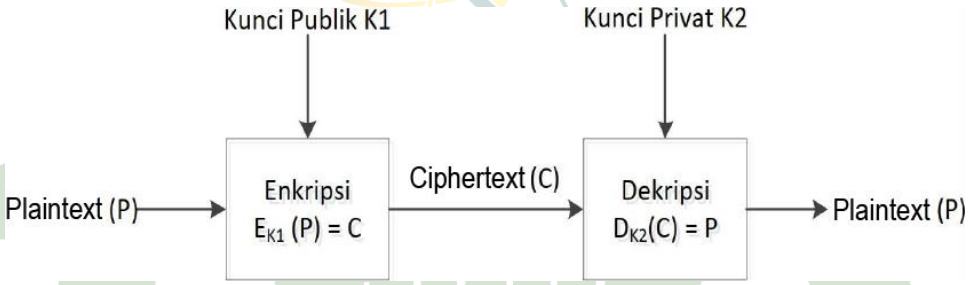
(Sumber : Suhardi, 2021)

Salah satu tantangan dalam algoritma kriptografi simetris adalah keamanan kuncinya. Karena kunci yang digunakan adalah sama untuk enkripsi dan dekripsi, keamanan sistem bergantung pada kerahasiaan kunci tersebut, maka algoritma ini disebut juga dengan algoritma kunci rahasia (*private key*). Adapun contoh

algoritma yang dikategorikan ke dalam algoritma kriptografi simetris adalah AES, DES, RC4, *Blowfish*, dan lain-lain. Kemudian contoh algoritma klasik yang dikategorikan ke dalam algoritma kriptografi simetris adalah *Caesar Cipher*, *Beaufort Cipher*, *Vigenere Cipher*, *Rail Fence Cipher*, dan lain-lain.

2. Algoritma Kriptografi Asimetris

   Algoritma kriptografi asimetris adalah jenis algoritma yang menggunakan kunci yang berbeda untuk proses enkripsi dan dekripsi. Kunci enkripsi, yang disebut sebagai kunci publik (*public key*), dapat disebarkan kepada umum, sementara kunci dekripsi, yang disebut kunci pribadi (*private key*), hanya untuk digunakan oleh penerima pesan (Purba, 2020). Adapun skema proses dari algoritma kriptografi asimetris, seperti yang terlihat pada Gambar 2.3 :



Gambar 2.3 Skema Proses Algoritma Kriptografi Asimetris

(Sumber : Suhardi, 2021)

Kunci publik digunakan untuk enkripsi pesan, sementara kunci privat digunakan untuk dekripsi pesan. Kedua kunci ini terkait matematis, namun sangat sulit untuk menghitung kembali kunci privat dari kunci publik. Salah satu keuntungan utama dari kriptografi asimetris adalah bahwa kunci publik dapat dipublikasikan secara terbuka tanpa mengurangi keamanan sistem. Hal ini memungkinkan siapa pun untuk mengirim pesan terenkripsi kepada penerima tanpa perlu pertukaran kunci rahasia terlebih dahulu. Adapun contoh algoritma yang dikategorikan ke dalam algoritma kriptografi asimetris adalah RSA, ElGamal, ECC, Diffie-Hellman, dan lain-lain.

2.2.2 Algoritma *Base64*

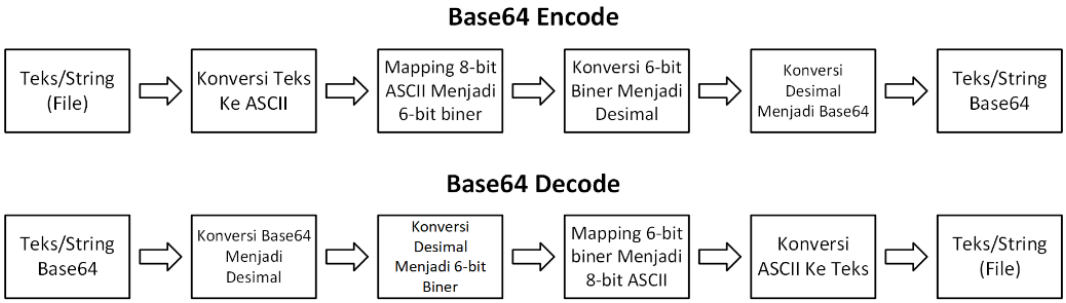
Algoritma *Base64* adalah sebuah algoritma pengkodean yang mentransformasi data biner menjadi format teks ASCII. Algoritma *Base64* menggunakan tabel karakter yang terdiri dari 64 karakter, biasanya terdiri dari huruf besar dan kecil (A-Z, a-z), angka (0-9), dan dua karakter tambahan yaitu karakter "+", "/", serta satu karakter "=" terletak diakhir yang digunakan dalam pengisian *padding* atau penyesuaian dan menggenapkan data biner (Rifki et al., 2023).

Tabel 2.1 Tabel *Base64*

Nilai	Karakter	Nilai	Karakter	Nilai	Karakter	Nilai	Karakter
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

(Sumber : Saefudin et al., 2020)

Keunggulan dari algoritma *Base64* adalah kemampuannya untuk menyandikan data biner menjadi teks yang dapat ditransmisikan dengan aman dan transportabel melalui protokol teks biasa, seperti *email*, yang hanya mendukung teks ASCII. Adapun proses *encode* dan *decode* dari algoritma *Base64*, terlihat pada Gambar 2.4 :



Gambar 2.4 Proses Dari Algoritma *Base64*

(Sumber : Rifki et al., 2023)

Untuk memperjelas bagaimana proses *encode* dan *decode* dari algoritma *Base64*, diberikan contoh teks 'UINSU' yang akan dikonversi ke *Base64*. 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, dengan rumus (2.1) berikut :

$B_{(2)} = H_{(16)\,digit(1)}\ mod\ 2\ ;\ H_{(16)\,digit(2)}\ mod\ 2 \cdots$ ……..……..……..…(2.1)

Tabel 2.2 Konversi ASCII Dan *Mapping* Biner 8-Bit

Karakter	ASCII (Heksadesimal)	Konversi Bilangan Biner		Biner 8-Bit
U	55	$2\frac{5}{2\frac{2}{2\frac{1}{0}1}1}1$	$2\frac{5}{2\frac{2}{2\frac{1}{0}1}1}1$	01010101
I	49	$2\frac{4}{2\frac{1}{0}1}0$	$2\frac{9}{2\frac{4}{2\frac{2}{2\frac{1}{0}1}0}0}1$	01001001
N	4E	$2\frac{4}{2\frac{1}{0}1}0$	$2\frac{14}{2\frac{7}{2\frac{3}{2\frac{1}{0}1}1}1}0$	01001110
S	53	$2\frac{5}{2\frac{2}{2\frac{1}{0}1}1}1$	$2\frac{3}{2\frac{1}{0}1}1$	01010011

Karakter	ASCII (Heksadesimal)	Konversi Bilangan Biner		Biner 8-Bit
U	55	$2\frac{5}{2}1$ $2\frac{2}{1}0$ $2\frac{1}{0}1$	$2\frac{5}{2}1$ $2\frac{2}{1}0$ $2\frac{1}{0}1$	01010101

Kemudian di *mapping* lagi menjadi biner 6-bit. Apabila di akhir *mapping* mengalami kurang dari biner 6-bit, maka ditambahkan *padding* (nol 8-Bit), seperti terlihat pada Tabel 2.3 :

Tabel 2.3 *Mapping* Biner 6-Bit Dan Diberi *Padding*

***Mapping* 6-Bit**							
010101	010100	100101	001110	010100	110101	010100	000000

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

$$D_{(10)} = \left(B_{(2)\ digit(n)} \times 2^0\right) + \left(B_{(2)\ digit(n-1)} \times 2^1\right) + \cdots \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (2.2)$$

Tabel 2.4 *Mapping* Biner 6-Bit Dan Konversi Ke Bilangan Desimal

Biner 6-Bit	Konversi Bilangan Desimal	Desimal
010101	$(0\times2^5) + (1\times2^4) + (0\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0)$	$21_{(10)}$
010100	$(0\times2^5) + (1\times2^4) + (0\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0)$	$20_{(10)}$
100101	$(1\times2^5) + (0\times2^4) + (0\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0)$	$37_{(10)}$
001110	$(0\times2^5) + (0\times2^4) + (1\times2^3) + (1\times2^2) + (1\times2^1) + (0\times2^0)$	$14_{(10)}$
010100	$(0\times2^5) + (1\times2^4) + (0\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0)$	$20_{(10)}$
110101	$(1\times2^5) + (1\times2^4) + (0\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0)$	$53_{(10)}$
010100	$(0\times2^5) + (1\times2^4) + (0\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0)$	$20_{(10)}$

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

Tabel 2.5 Konversi Bilangan Desimal Ke *Base64*

Desimal	Base64
$21_{(10)}$	V
$20_{(10)}$	U
$37_{(10)}$	l
$14_{(10)}$	O

Desimal	Base64
$20_{(10)}$	U
$53_{(10)}$	1
$20_{(10)}$	U

Dari proses perhitungan yang telah dilakukan, maka teks *Base64* yang dihasilkan adalah 'VUlOU1U'. Masuk ke proses *Base64 decode*, konversikan setiap karakter *Base64* sesuai dengan nilaiya pada tabel *Base64*, kemudian konversikan bilangan desimal ke dalam biner 6-bit, dengan rumus (2.3) berikut :

$B_{(2)} = D_{(10)}\ mod\ 2$……...……(2.3)

Tabel 2.6 Konversi Ke Bilangan Biner Dan *Mapping* Biner 6-Bit

Base64	Desimal	Konversi Bilangan Biner	Biner 6-Bit
V	$21_{(10)}$	$2\overline{)21}\,1$ $2\overline{)10}\,0$ $2\overline{)5}\,1$ $2\overline{)2}\,0$ $2\overline{)1}\,1$ 0	010101
U	$20_{(10)}$	$2\overline{)20}\,0$ $2\overline{)10}\,0$ $2\overline{)5}\,1$ $2\overline{)2}\,0$ $2\overline{)1}\,1$ 0	010100
l	$37_{(10)}$	$2\overline{)37}\,1$ $2\overline{)18}\,0$ $2\overline{)9}\,1$ $2\overline{)4}\,0$ $2\overline{)2}\,0$ 1	100101
O	$14_{(10)}$	$2\overline{)14}\,0$ $2\overline{)7}\,1$ $2\overline{)3}\,1$ $2\overline{)1}\,1$ 0	001110
U	$20_{(10)}$	$2\overline{)20}\,0$ $2\overline{)10}\,0$ $2\overline{)5}\,1$ $2\overline{)2}\,0$ $2\overline{)1}\,1$ 0	010100

Base64	Desimal	Konversi Bilangan Biner	Biner 6-Bit
1	$53_{(10)}$	$2\overline{)53}\,1$ $2\overline{)26}\,0$ $2\overline{)13}\,1$ $2\overline{)6}\,0$ $2\overline{)3}\,1$ 1	110101
U	$20_{(10)}$	$2\overline{)20}\,0$ $2\overline{)10}\,0$ $2\overline{)5}\,1$ $2\overline{)2}\,0$ $2\overline{)1}\,1$ 0	010100

Selanjutnya di *mapping* lagi menjadi biner 8-bit dan dikonversi ke dalam bilangan heksadesimal, dengan rumus (2.4) berikut :

$$H_{(16)} = \begin{cases} \left(B_{(2)\,digit(4)} \times 2^0\right) + \left(B_{(2)\,digit(4-1)} \times 2^1\right) + \cdots \\ \left(B_{(2)\,digit(8)} \times 2^0\right) + \left(B_{(2)\,digit(8-1)} \times 2^1\right) + \cdots \end{cases} \text{.....(2.4)}$$

Tabel 2.7 *Mapping* Biner 8-Bit Dan Konversi Ke Bilangan Heksadesimal

Biner 8-Bit	Konversi Bilangan Heksadesimal	ASCII (Heksadesimal)
01010101	$0101 \rightarrow (0\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0) = 5$ $0101 \rightarrow (0\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0) = 5$	55
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
01001110	$0100 \rightarrow (0\times2^3) + (1\times2^2) + (0\times2^1) + (0\times2^0) = 4$ $1110 \rightarrow (1\times2^3) + (1\times2^2) + (1\times2^1) + (0\times2^0) = 14$	4E
01010011	$0101 \rightarrow (0\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0) = 5$ $0011 \rightarrow (0\times2^3) + (0\times2^2) + (1\times2^1) + (1\times2^0) = 3$	53
01010101	$0101 \rightarrow (0\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0) = 5$ $0101 \rightarrow (0\times2^3) + (1\times2^2) + (0\times2^1) + (1\times2^0) = 5$	55

Terakhir, konversikan setiap nilai bilangan heksadesimal yang didapatkan sesuai dengan karakter yang ada pada tabel ASCII. Dari proses perhitungan yang dilakukan, maka teks kembali lagi didapatkan yaitu ‘UINSU’.

Tabel 2.8 Konversi Bilangan Heksadesimal Ke Karakter ASCII

ASCII (Heksadesimal)	Karakter
55	U
49	I
4E	N
53	S
55	U

2.2.3 Algoritma *Beaufort Cipher*

Algoritma *Beaufort Cipher* adalah algoritma kriptografi yang digunakan untuk menyandikan pesan, yang menggunakan teknik substitusi dengan kunci simetris (Setiadi et al., 2020). Nama "*Beaufort*" diambil dari nama seorang kapten angkatan laut Inggris, Sir Francis Beaufort, yang dikenal karena kontribusinya pada studi kriptografi (Rachmadsyah et al., 2020). 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. 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* (Ndruru & Zebua, 2022).

Adapun rumus dari algoritma *Beaufort Cipher* yang digunakan dalam tahapan enkripsi dan dekripsi dinyatakan melalui rumus berikut :

$$C_c = (k - P_c)\ mod\ 26 \quad \ldots\ldots(2.5)$$

$$P_c = (k - C_c)\ mod\ 26 \quad \ldots\ldots(2.6)$$

Keterangan :

C_c = Mewakilkan *ciphertext*

P_c = Mewakilkan *plaintext*

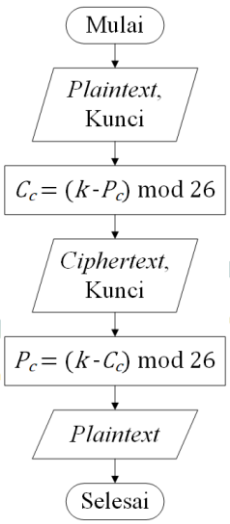
k = Mewakilkan kunci

Catatan :

1. Jika panjang kunci lebih pendek daripada panjang *plaintext*, maka kunci akan diulang secara periodik.

2. Jika hasil perhitungan dari rumus menghasilkan nilai negatif, maka ditambahkan 26 kemudian dimodulus lagi dengan 26.

Adapun *flowchart* tahapan enkripsi dan dekripsi dari algoritma *Beaufort Cipher*, seperti yang terlihat pada Gambar 2.5 :



Gambar 2.5 *Flowchart* Algoritma *Beaufort Cipher*

(Sumber : Abiyuda & Nababan, 2023)

Nilai modulus 26 yang disebutkan pada rumus (2.5) dan (2.6) dapat bervariasi tergantung pada jumlah karakter yang digunakan. Pada awalnya, *Beaufort Cipher* hanya menggunakan 26 karakter *alfabet*. Namun, dengan kemajuan teknologi komputer saat ini, variasi karakter yang digunakan dapat menggunakan karakter *Base64* yang berjumlah 64 karakter, tabel ASCII yang berjumlah 256 karakter, dan lain-lain (Diana & Zebua, 2020).

Untuk memperjelas bagaimana proses enkripsi dan dekripsi dari *Beaufort Cipher* menggunakan rumus (2.5) dan (2.6), diberikan contoh sebagai berikut :

Plaintext : ILKOMP

Kunci : UINSU

Langkah awal yang harus dilakukan yaitu mensubstitusikan setiap karakter *plaintext* dan kunci dengan tabel substitusi yang telah ditentukan, contohnya A=0, B=1, C=2, … , Z=25, seperti yang terlihat pada Tabel 2.9 :

Tabel 2.9 Tabel Substitusi Angka

Alfabet	A	B	C	D	E	F	G	H	I	J	K	L	M
index	0	1	2	3	4	5	6	7	8	9	10	11	12
Alfabet	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
index	13	14	15	16	17	18	19	20	21	22	23	24	25

Diketahui bahwa panjang karakter kunci tidak sama dengan panjang karakter dari *plaintext*, maka kunci akan diulang secara periodik. Karakter *plaintext* dan kunci yang telah disubstitusikan sesuai dengan Tabel 2.9, akan terlihat seperti pada Tabel 2.10 berikut ini :

Tabel 2.10 Proses Substitusi Angka *Beaufort Cipher*

Plaintext	I	L	K	O	M	P
Substitusi	8	11	10	14	12	15
Kunci	U	I	N	S	U	U
Substitusi	20	8	13	18	20	20

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

$$C_1 = (k_1 - P_1)\ mod\ 26$$
$$= (20 - 8)\ mod\ 26$$
$$= 12 \rightarrow M$$

$$C_2 = (k_2 - P_2)\ mod\ 26$$
$$= (8 - 11)\ mod\ 26$$
$$= (-3 + 26)\ mod\ 26$$
$$= 23 \rightarrow X$$

$$C_3 = (k_3 - P_3)\ mod\ 26$$
$$= (13 - 10)\ mod\ 26$$
$$= 3 \rightarrow D$$

$$C_4 = (k_4 - P_4)\ mod\ 26$$
$$= (18 - 14)\ mod\ 26$$
$$= 4 \rightarrow E$$

$$C_5 = (k_5 - P_5)\ mod\ 26$$
$$= (20 - 12)\ mod\ 26$$
$$= 8 \rightarrow I$$

$$C_6 = (k_6 - P_6)\ mod\ 26$$
$$= (20 - 15)\ mod\ 26$$
$$= 5 \rightarrow F$$

Apabila pada proses perhitungan, nilai kunci yang dikurangi dengan nilai *plaintext* menghasilkan nilai negatif, maka akan dijumlahkan 26 (sesuai jumlah karakter yang digunakan), ini dilakukan sampai mendapatkan nilai yang positif,

baru kemudian dapat dimoduluskan 26. Dari proses perhitungan yang dilakukan, maka *ciphertext* yang dihasilkan adalah 'MXDEIF'.

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

$$P_1 = (k_1 - C_1)\ mod\ 26$$
$$= (20 - 12)\ mod\ 26$$
$$= 8 \rightarrow I$$
$$P_2 = (k_2 - C_2)\ mod\ 26$$
$$= (8 - 23)\ mod\ 26$$
$$= (-15 + 26)\ mod\ 26$$
$$= 11 \rightarrow L$$
$$P_3 = (k_3 - C_3)\ mod\ 26$$
$$= (13 - 3)\ mod\ 26$$
$$= 10 \rightarrow K$$
$$P_4 = (k_4 - C_4)\ mod\ 26$$
$$= (18 - 4)\ mod\ 26$$
$$= 14 \rightarrow O$$
$$P_5 = (k_5 - C_5)\ mod\ 26$$
$$= (20 - 8)\ mod\ 26$$
$$= 12 \rightarrow M$$
$$P_6 = (k_6 - C_6)\ mod\ 26$$
$$= (20 - 5)\ mod\ 26$$
$$= 15 \rightarrow P$$

Dari proses perhitungan yang dilakukan, maka *plaintext* kembali didapatkan yaitu 'ILKOMP'. Kelebihan dari algoritma *Beaufort Cipher* adalah relatif mudah untuk dipahami dan diimplementasikan. Meskipun sederhana, Karakteristik algoritma ini membuat teks yang terenkripsi tampak bervariasi. Namun, seperti metode substitusi klasik lainnya, algoritma ini rentan terhadap serangan frekuensi huruf dan analisis statistik lainnya. Oleh karena itu, disarankan untuk menggunakan kunci yang panjang dan kompleks serta mengganti kunci secara teratur.

2.2.4 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, 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 (Afandi & Nurhayati, 2020). Nama "*Vigenere*" diambil dari nama seorang ahli kriptografi Prancis abad ke-16, Blaise de Vigenere. Metode ini digunakan dalam mengamankan pesan rahasia pada masa lalu (Putra et al., 2023).

Adapun rumus dari algoritma *Vigenere Cipher* yang digunakan dalam tahapan enkripsi dan dekripsi dinyatakan melalui rumus berikut :

$C_i = (P_i + k_i)\ mod\ 26$……..……(2.7)

$P_i = (C_i - k_i)\ mod\ 26$……..…(2.8)

Keterangan :

C_i = Mewakilkan *ciphertext*

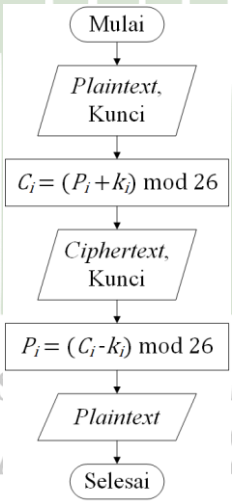
P_i = Mewakilkan *plaintext*

k_i = Mewakilkan kunci

Catatan :

1. Jika panjang kunci lebih pendek daripada panjang *plaintext*, maka kunci akan diulang secara periodik.
2. Jika hasil perhitungan dari rumus menghasilkan nilai negatif, maka ditambahkan 26 kemudian dimodulus lagi dengan 26.

Adapun *flowchart* tahapan enkripsi dan dekripsi dari algoritma *Vigenere Cipher*, seperti yang terlihat pada Gambar 2.6 :



Gambar 2.6 *Flowchart* Algoritma *Vigenere Cipher*

(Sumber : Karima et al., 2024)

Sama seperti *Beaufort* Cipher, nilai modulus 26 disebutkan pada rumus (2.7) dan (2.8) dapat bervariasi tergantung pada jumlah karakter yang digunakan. Untuk memperjelas bagaimana proses enkripsi dan dekripsi dari *Vigenere Cipher* menggunakan rumus (2.7) dan (2.8), menggukan contoh sebelumnya yaitu :

Plaintext : ILKOMP

Kunci : UINSU

Langkah awal yang harus dilakukan yaitu mensubstitusikan setiap karakter *plaintext* dan kunci dengan tabel substitusi yang telah ditentukan, sama seperti pada Tabel 2.9. Kemudian kunci akan diulang secara periodik, maka akan terlihat seperti pada Tabel 2.11 berikut ini :

Tabel 2.11 Proses Substitusi Angka *Vigenere Cipher*

Plaintext	I	L	K	O	M	P
Substitusi	8	11	10	14	12	15
Kunci	U	I	N	S	U	U
Substitusi	20	8	13	18	20	20

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

$C_1 = (P_1 + k_1)\ mod\ 26$
$= (8 + 20)\ mod\ 26$
$= 2 \rightarrow C$

$C_2 = (P_2 + k_2)\ mod\ 26$
$= (11 + 8)\ mod\ 26$
$= 19 \rightarrow T$

$C_3 = (P_3 + k_3)\ mod\ 26$
$= (10 + 13)\ mod\ 26$
$= 23 \rightarrow X$

$C_4 = (P_4 + k_4)\ mod\ 26$
$= (14 + 18)\ mod\ 26$
$= 6 \rightarrow G$

$C_5 = (P_5 + k_5)\ mod\ 26$
$= (12 + 20)\ mod\ 26$
$= 6 \rightarrow G$

$C_6 = (P_6 + k_6)\ mod\ 26$
$= (15 + 20)\ mod\ 26$
$= 9 \rightarrow J$

Maka *ciphertext* yang dihasilkan adalah 'CTXGGJ'. Terakhir, masuk ke proses dekripsi *Vigenere Cipher* menggunakan rumus (2.8), dimana proses perhitungannya sebagai berikut :

$P_1 = (C_1 - k_1)\ mod\ 26$
$= (2 - 20)\ mod\ 26$
$= (-18 + 26)\ mod\ 26$
$= 8 \rightarrow I$

$P_2 = (C_2 - k_2)\ mod\ 26$
$= (19 - 8)\ mod\ 26$
$= 11 \rightarrow L$

$P_4 = (C_4 - k_4)\ mod\ 26$
$= (6 - 18)\ mod\ 26$
$= (-12 + 26)\ mod\ 26$
$= 14 \rightarrow O$

$P_5 = (C_5 - k_5)\ mod\ 26$
$= (6 - 20)\ mod\ 26$
$= (-14 + 26)\ mod\ 26$
$= 12 \rightarrow M$

$$P_3 = (C_3 - k_3)\ mod\ 26$$
$$= (23 - 13)\ mod\ 26$$
$$= 10 \rightarrow K$$

$$P_6 = (C_6 - k_6)\ mod\ 26$$
$$= (9 - 20)\ mod\ 26$$
$$= (-11 + 26)\ mod\ 26$$
$$= 15 \rightarrow P$$

Dari proses perhitungan yang dilakukan, maka *plaintext* kembali didapatkan yaitu 'ILKOMP'. Kelebihan dan kekurangan dari algoritma *Vigenere Cipher* kurang lebih sama dengan *Beaufort Cipher*, karena kedua algoritma ini memiliki karakteristik yang sama, yaitu menggunakan tabel substitusi angka yang sama.

2.3 *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). *Avalanche effect* penting dalam algoritma kriptografi karena memastikan bahwa pesan asli sulit dilacak kembali dari hasil enkripsi. Jika perubahan bit berada pada kisaran 45-60%, maka pengujian *avalanche effect* dianggap baik, dimana 50% sebagai hasil yang dianggap ideal dalam pengujian (Karima et al., 2024). Adapun rumus *avalanche effect* adalah sebagai berikut :

$$AE = \frac{Jumlah\ Perubahan\ Bit}{Jumlah\ Total\ Bit} \times 100\% \quad (2.9)$$

2.4 *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 digunakan untuk menghitung persentase bit dengan membandingkan jumlah bit yang keliru dengan total bit selama proses penyisipan. BER dianggap baik jika nilainya mendekati 0, yang artinya tidak ada perbedaan antara *plaintext* asli dengan hasil dekripsi *ciphertext* (Karima et al., 2024). Adapun rumus *bit error rate* adalah sebagai berikut :

$$BER = \frac{Jumlah\ Bit\ Error}{Jumlah\ Total\ Bit} \times 100\% \quad (2.10)$$

2.5 *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 (Muslih & Handoko, 2022). Adapun rumus *character error rate* adalah sebagai berikut :

$$CER = \frac{Jumlah\ Kesalahan\ Karakter}{Jumlah\ Total\ Karakter} \times 100\% \quad (2.11)$$

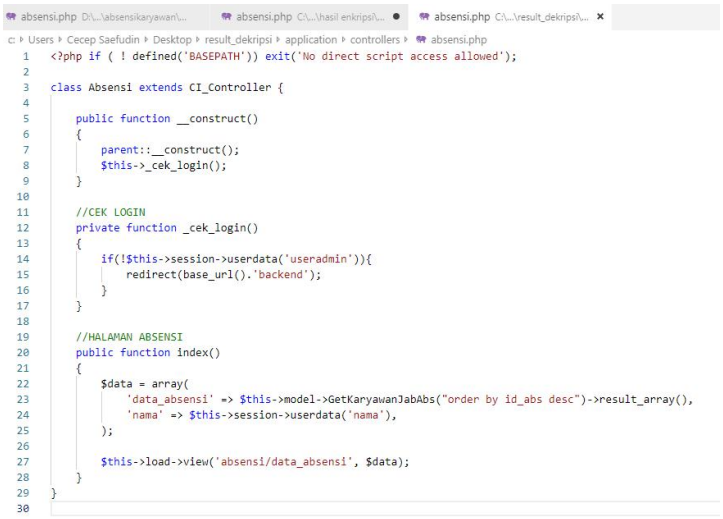
2.6 *Entropy*

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 (Karima et al., 2024). Adapun rumus *entropy* Shannon adalah sebagai berikut :

$$H(X) = -\sum_{i=0}^{n} p(x) \cdot \log_2(p(x)) \quad (2.12)$$

2.7 *Source Code*

Source code adalah kumpulan perintah yang ditulis dalam bahasa pemrograman oleh seorang pengembang untuk membuat sebuah program komputer. Ini adalah bentuk yang dapat dibaca oleh manusia sebelum dikompilasi menjadi bentuk yang dapat dieksekusi oleh mesin komputer. Seorang pengembang dapat berinteraksi dengan komputer dengan perintah yang telah ditentukan melalui *source code*, dan biasanya terdiri dari satu atau lebih berkas teks (Saefudin et al., 2020). Adapun aplikasi untuk membuat dan mengetik *file source code*, disebut sebagai *code editor*. Visual Studio Code merupakan *code editor* yang *open source* dikembangkan oleh Microsoft yang tersedia secara gratis dan mendukung berbagai bahasa pemrograman. Berikut tampilan aplikasi Visual Studio Code, yang terlihat pada Gambar 2.7 :



Gambar 2.7 Tampilan Aplikasi Visual Studio Code

(Sumber : Saefudin et al., 2020)

2.8 Aplikasi *Web*

Aplikasi *web* adalah jenis aplikasi yang dapat diakses melalui *web browser* menggunakan jaringan internet atau intranet. Aplikasi *web* dapat diakses melalui *web browser* di perangkat seperti komputer, smartphone, atau tablet. Ini berbeda dengan aplikasi *desktop* yang harus di-*install* terlebih dahulu di dalam perangkat. Aplikasi *web* umumnya terdiri dari dua bagian utama yaitu, *frontend* (bagian yang dilihat dan diakses oleh pengguna) dan *backend* (bagian yang berjalan di belakang layar yang mengelola data serta logika aplikasi). Pada awalnya, aplikasi *web* dikembangkan hanya dengan menggunakan bahasa *markup* bernama HTML (*HyperText Markup Language*), yang bersifat statis. Namun, dalam perkembangannya, sejumlah skrip dan bahasa pemrograman seperti PHP dan JavaScript dikembangkan untuk memperluas kemampuan HTML yang dapat bersifat menjadi dinamis (Rosid, 2023).

2.8.1 HTML (*HyperText Markup Language*)

HTML (*HyperText Markup Language*) adalah bahasa standar *web* yang dijaga oleh organisasi bernama W3C (*World Wide Web Consortium*). HTML berperan sebagai peyusun dan merancang struktur halaman *web* yang menempatkan tata letak setiap elemen sesuai yang diinginkan. HTML memberikan struktur dasar untuk konten *web*, seperti teks, gambar, video, dan *hyperlink*, dengan menggunakan serangkaian elemen

atau <*tag*>, *tag-tag* ini yang memberikan petunjuk kepada *browser* tentang bagaimana konten tersebut ditampilkan, yang kemudian disematkan di dalam dokumen HTML. HTML disimpan dalam *file* berekstensi *.html*, dan untuk membuat atau mengetikkan skrip HTML, dapat menggunakan aplikasi *code editor* seperti Visual Studio Code, Notepad++, Sublime Text, dan lain-lain (Sari & Suhendi, 2020).

2.8.2 CSS (*Cascading Style Sheets*)

CSS (*Cascading Style Sheets*) adalah bahasa yang digunakan untuk mendesain tampilan atau gaya dari dokumen HTML. CSS dapat mengontrol tata letak, warna, *font*, ukuran, dan berbagai properti visual lainnya dari elemen-elemen HTML di halaman *web*. Dengan menggunakan CSS, pengembang dapat menciptakan desain yang konsisten dan menarik untuk halaman *web*. CSS bekerja dengan memodifikasi HTML melalui pemilihan elemen HTML yang ingin diatur, kemudian memberikan properti yang sesuai dengan tampilan yang diinginkan. Dalam memberikan aturan pada elemen HTML, skrip CSS terdiri dari tiga bagian yaitu *selector* untuk memilih elemen yang akan diberi aturan, *property* yang merupakan aturan yang diberikan, dan *value* sebagai nilai dari aturan yang diberikan (Sari & Suhendi, 2020).

2.8.3 JavaScript

JavaScript adalah bahasa pemrograman yang sangat penting dalam pengembangan aplikasi *web*. JavaScript diciptakan oleh Brendan Eich pada tahun 1995, JavaScript dirancang untuk membuat halaman *web* lebih interaktif dengan memberikan kemampuan untuk memanipulasi elemen HTML, merespons interaksi dari pengguna, dan mengubah konten secara dinamis. JavaScript berjalan di sisi klien (*browser*) dan memungkinkan interaksi langsung dengan halaman *web*. Ini membuatnya menjadi alat yang kuat untuk pengembangan *front-end*. JavaScript merupakan bahasa *interpreter*, yang berarti skrip di eksekusi tanpa proses kompilasi. JavaScript dapat disisipkan di dalam dokumen HTML, atau dapat dibuat pada *file* terpisah. JavaScript telah mendukung paradigma pemrograman berorientasi objek (Hidayatullah et al., 2020).

2.8.4 UML (*Unified Modelling Language*)

UML (*Unified Modelling Language*) adalah bahasa standar yang digunakan untuk mendokumentasikan, merancang, dan memodelkan suatu sistem perangkat lunak. UML menyediakan seperangkat notasi grafis yang dapat digunakan untuk menggambarkan struktur, perilaku, dan interaksi dari sistem yang akan dikembangkan. UML berfungsi membantu pendeskripsian dan desain sistem perangkat lunak, terutama untuk sistem yang dikembangkan dengan pemrograman berorientasi objek. Pemodelan UML bahkan bisa siap diimplementasikan ke dalam kode bahasa pemrograman. UML terdiri dari berbagai jenis diagram, seperti *use diagram*, *activity diagram*, *sequence diagram*, dan lainnya, yang masing-masing digunakan untuk menggambarkan aspek yang berbeda dari suatu sistem perangkat lunak (Nistrina & Sahidah, 2022).

1. *Use Case Diagram*

   Use Case Diagram adalah salah satu cara untuk menggambarkan interaksi antara sistem dengan para aktor atau pengguna yang akan menggunakan sistem tersebut. *Use Case diagram* juga memberikan gambaran umum tentang bagaimana pengguna akan berinteraksi dengan sistem yang dibangun, serta mengidentifikasi fungsi-fungsi apa saja yang tersedia dalam sistem dan dapat digunakan oleh pengguna aplikasi (Andiko & Cahyono, 2022).

Tabel 2.12 Simbol Dan Keterangan *Use Case Diagram*

No	Simbol	Nama	Keterangan
1		*Actor*	Mewakili peran orang ketika berkomunikasi dengan *use case*.
2		*Use Case*	Fungsionalitas atau layanan yang disediakan oleh sistem, yang dilakukan oleh aktor.
4		*System*	Batasan yang menunjukkan ruang lingkup sistem di mana *use case* terjadi.

No	Simbol	Nama	Keterangan
5		*Association*	Hubungan antara aktor dan *use case* yang menunjukkan interaksi atau komunikasi.
6		*Include*	Hubungan yang menunjukkan bahwa *use case* tertentu selalu mencakup *use case* lain (*use case* yang disertakan).
7		*Extend*	Hubungan yang menunjukkan bahwa *use case* tertentu dapat diperluas dengan fungsionalitas tambahan.

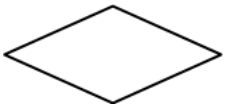
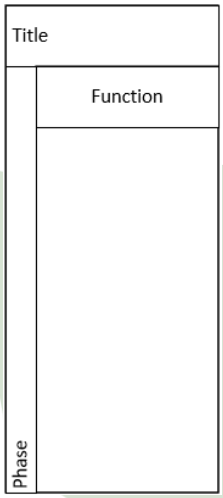
(Sumber : Santoso & Migunani, 2021)

2. *Activity Diagram*

Activity Diagram adalah suatu diagram yang menggambarkan konsep aliran kendali/kontrol, aksi yang terstruktur, dan dirancang dengan baik dalam suatu sistem. *Activity diagram* digunakan menggambarkan alur kerja atau aktivitas yang terjadi dalam suatu proses atau sistem. *Activity diagram* berguna untuk mendokumentasikan, menganalisis, dan memodelkan alur kerja yang melibatkan serangkaian aktivitas, keputusan, dan kondisi yang terjadi dalam suatu proses perangkat lunak aplikasi, atau sistem lainnya (Nazir et al., 2022).

Tabel 2.13 Simbol Dan Keterangan *Activity Diagram*

No	Simbol	Nama	Keterangan
1		*Initial Node*	Titik awal dari diagram aktivitas, yang menandakan dimulainya alur aktivitas.
2		*Transition*	Menunjukkan alur atau perpindahan dari satu aktivitas ke aktivitas lain.

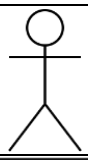
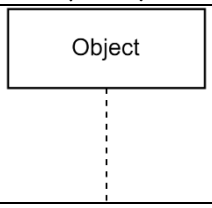
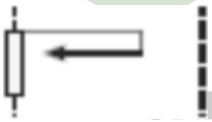
No	Simbol	Nama	Keterangan
3		*Activity / Action*	Langkah-langkah yang mewakili tindakan yang dilakukan dalam sistem, yang biasanya diawali dengan kata kerja.
4		*Decision*	Titik keputusan yang membagi alur ke dalam beberapa cabang berdasarkan kondisi tertentu.
5		*Final Node*	Menunjukkan akhir dari aktivitas atau alur proses.
6		*Swimlane*	Mengelompokkan aktivitas berdasarkan aktor atau *unit* dalam sistem. Setiap *swimlane* di identifikasikan dengan peran atau entitas tertentu.

(Sumber : Santoso & Migunani, 2021)

3. *Sequence Diagram*

Sequence Diagram adalah suatu diagram yang menggambarkan interaksi antara objek-objek atau entitas dalam suatu sistem perangkat lunak atau sistem, yang berbasis pesan/*message*, dalam urutan waktu tertentu. *Sequence diagram* menggambarkan kelakuan objek pada *use case diagram* dengan mendeskripsikan *life time* dari objek dan pesan yang dikirimkan dan diterima antar objek. Dengan begitu dalam membuat *sequence diagram* harus diketahui terlebih dahulu objek-objek yang terlibat dalam suatu *use case diagram* beserta metode yang dimiliki kelas yang diinstansisasi menjadi objek tersebut (Hariansyah et al., 2021).

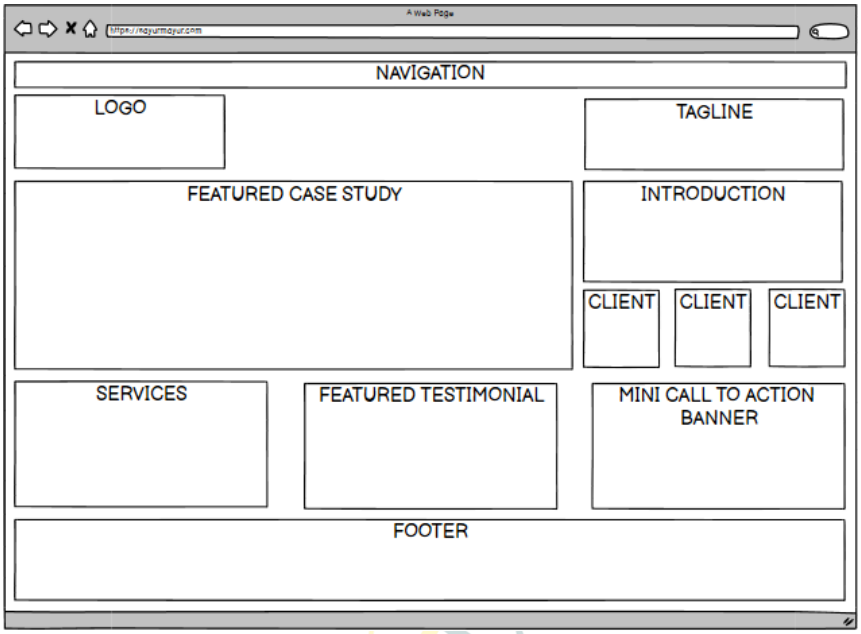
Tabel 2.14 Simbol Dan Keterangan *Sequence Diagram*

No	Simbol	Nama	Keterangan
1		*Actor*	Menggambarkan seseorang yang berinteraksi dengan sistem.
2		*Object / Entity*	Objek atau entitas sistem yang berinteraksi dalam sebuah skenario.
3		*Activation*	Menunjukkan bahwa objek atau aktor sedang menjalankan proses atau fungsi.
4		*Message*	Panah yang menunjukkan komunikasi antara objek, seperti pengiriman pesan atau panggilan metode.
5		*Return Message*	Panah putus-putus yang menunjukkan pengembalian nilai dari objek setelah selesai eksekusi metode.
6		*Self Message*	Panah yang kembali ke objeknya sendiri, dimana menggambarkan pemanggilan metode oleh objek pada dirinya sendiri.

(Sumber : Santoso & Migunani, 2021)

2.8.5 Desain Antarmuka

Desain antarmuka adalah proses merancang tampilan visual dan fungsional suatu aplikasi agar mudah dipahami dan digunakan oleh pengguna. Ini melibatkan pemikiran tentang bagaimana pengguna akan berinteraksi dengan aplikasi, mulai dari navigasi hingga tata letak elemen-elemennya (Fauzi, 2022). Berikut contoh desain antarmuka suatu aplikasi, yang terlihat pada Gambar 2.8 :



Gambar 2.8 Contoh Desain Antarmuka Suatu Aplikasi

(Sumber : Niagahoster.co.id – *Wireframe*, 2022)

2.8.6 Pengujian Aplikasi

Suatu aplikasi membutuhkan pengujian untuk memverifikasi bahwa aplikasi yang sedang dibuat atau sudah selesai dapat beroperasi sesuai dengan fungsionalitas yang diharapkan. Pengembang dari aplikasi perlu menyusun serangkaian uji coba untuk menguji program dari aplikasi yang telah selesai dibuat, sehingga kelemahan atau kesalahan dapat terdeteksi sejak awal, dan perbaikan dapat dilakukan pada siklus pengembangan perangkat lunak berikutnya (Rifki et al., 2023). Ada beberapa jenis pengujian perangkat lunak, di antaranya :

1. Pengujian *White Box*

   Pengujian *white box*, juga dikenal sebagai pengujian struktural atau pengujian berbasis kode, adalah metode pengujian perangkat lunak yang melibatkan analisis internal dari *source code* aplikasi. Dalam pengujian *white box*, penguji memiliki akses ke struktur internal kode, termasuk alur kontrol dan algoritma pemrograman. Tujuannya adalah untuk memeriksa keakuratan dan keefektifan logika internal aplikasi. Secara umum, kesimpulan yang dapat diambil dari pengujian *white box* adalah petunjuk untuk mendapatkan program yang benar (Utomo et al., 2020).

2. Pengujian *Black Box*

   Pengujian *black box* adalah pengujian perangkat lunak di mana pengujian dilakukan tanpa pengetahuan internal tentang struktur atau implementasi kode dari aplikasi. Dalam pengujian *black box*, penguji hanya memiliki akses ke tampilan antarmuka pengguna dari sistem yang diuji. Tujuannya adalah untuk mengevaluasi perilaku aplikasi dari perspektif pengguna. Pengujian dapat di definisikan dengan kumpulan kondisi *input* dari pengguna berikan ke sistem, dan respons sistem terhadap situasi tersebut, agar memastikan bahwa aplikasi berperilaku dengan benar dalam berbagai situasi yang dilakukan oleh pengguna (Utomo et al., 2020).

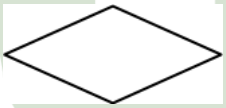
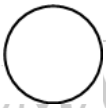
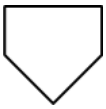
2.8.7 *Flowchart*

Flowchart atau diagram alir adalah representasi visual dari alur proses atau serangkaian langkah-langkah yang digunakan untuk menyelesaikan suatu tugas atau mencapai suatu tujuan. *Flowchart* menggunakan simbol-simbol grafis seperti kotak, panah, dan garis-garis untuk menunjukkan langkah-langkah, keputusan, percabangan, dan aliran informasi dalam proses tersebut. Ini memungkinkan seseorang agar dapat memahami secara intuitif bagaimana proses tersebut bekerja dan membantu dalam menganalisis dan merancang suatu sistem (Listyoningrum et al., 2023). Terdapat beberapa jenis *flowchart* diantaranya :

1. *Flowchart* sistem merupakan *flowchart* yang menunjukkan seluruh arus pekerjaan dari suatu sistem. diagram ini menggambarkan urutan prosedur-prosedur dalam sistem. Diagram alir sistem mengilustrasikan aktivitas yang dilakukan di dalam sistem.
2. *Flowchart* dokumen atau formulir merupakan *flowchart* yang menunjukkan arus dari laporan dan formulir termasuk tembusan-tembusannya.
3. *Flowchart* skematik merupakan *flowchart* yang menggambarkan prosedur di dalam sistem dengan menggunakan simbol-simbol *flowchart* sistem dan gambar peranti komputer serta peralatan lainnya yang digunakan oleh sistem.
4. *Flowchart* program merupakan *flowchart* yang menjelaskan secara rinci langkah-langkah dari proses program.

5. *Flowchart* proses merupakan *flowchart* yang digunakan di teknik industri. Diagram alir ini berguna untuk analisis sistem untuk menggambarkan proses dalam suatu prosedur (Budiman et al., 2021).

Tabel 2.15 Simbol Dan Keterangan *Flowchart*

No	Simbol	Nama	Keterangan
1		*Terminator*	Permulaan atau akhir program.
2		*Process*	Proses penghitung atau proses pengolahan data.
3		*Predefine Process*	Permulaan sub program.
4		*Preparation*	Proses inisialisasi atau pemberian harga awal.
5		*Input/Output*	Proses *input* atau *output* data, parameter, informasi.
6		*Decision*	Perbandingan, pernyataan, penyeleksian data yang memberikan pilihan untuk langkah selanjutnya.
7		*Flow Line*	Arah aliran program
8		*On-Page Reference*	Penghubung bagian-bagian *flowchart* yang berada pada satu halaman.
9		*Off-Page Reference*	Penghubung bagian-bagian *flowchart* yang berada pada halaman berbeda.

(Sumber : Budiman et al., 2021)

2.9 Penelitian Terdahulu

Penelitian terdahulu menjadi dasar dalam melakukan penelitian ini, yang membantu memperkaya teori yang digunakan untuk mengkaji penelitian yang sedang dilakukan dengan merujuk pada penelitian sebelumnya. Penelitian ini mengacu pada beberapa penelitian terdahulu yang relevan, kemudian digunakan sebagai referensi untuk memperkuat landasan teoritis dalam penelitian. Berikut adalah beberapa jurnal-jurnal yang berkaitan dengan penelitian ini, yang terlihat pada Tabel 2.16 :

Tabel 2.16 Penelitian Terdahulu

No	Penulis	Judul Penelitian	Hasil Penelitian
1	Rachmadsyah, Perdana dan Budiman (2020)	Kombinasi Algoritma *Beaufort Cipher* dan *Vigenere Cipher* untuk Pengamanan Pesan Teks Berbasis *Mobile Application* (Rachmad syah et al., 2020)	Hasil penelitian menunjukkan kombinasi dari kedua algoritma tersebut berhasil digunakan untuk mengamankan pesan teks dalam sebuah aplikasi *mobile* berbasis Android. Implementasi tersebut memungkinkan pesan teks dienkripsi dan didekripsi dengan baik, sehingga pesan yang disampaikan menjadi aman dan rahasia. Hasil uji coba menunjukkan bahwa aplikasi mampu mengenkripsi dan mendekripsi pesan sesuai dengan yang diharapkan.
2	Setiadi, Jatmoko, Rachmawanto dan Sari (2020)	Kombinasi *Cipher* Substitusi (*Beaufort* Dan *Vigenere*) Pada Citra *Grayscale* Digital (Setiadi et al., 2020)	Hasil penelitian menunjukkan kombinasi kedua algoritma tersebut pada citra digital menunjukkan bahwa kualitas enkripsi meningkat dibandingkan dengan menggunakannya secara

No	Penulis	Judul Penelitian	Hasil Penelitian
			terpisah. Kombinasi kedua algoritma berhasil diterapkan untuk enkripsi citra, 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.
3	Darmeli Nasution, Sulistianingsih dan Arie Candra (2023)	Kombinasi *Vigenere* dan *Beaufort Cipher* Konsep Simulasi *Three-pass Protocol* pada Data Penerbangan (D. Nasution et al., 2023)	Hasil penelitian menunjukkan kombinasi kedua algoritma tersebut yang diterapkan dalam skema *Three-pass Protocol* meningkatkan kualitas enkripsi data penerbangan. Dengan skema ini, mengurangi risiko kebocoran kunci dan meningkatkan keamanan data. Hasil uji coba pada data penerbangan, membuktikan bahwa data yang dikirimkan tetap terjaga kerahasiaannya selama proses pengiriman.
4	Yusuf, Hasugian, Mahyudi (2023)	Penerapan Teknik Super Enkripsi Untuk Keamanan *File* Teks Menggunakan Gabungan Algoritma *Beaufort Cipher* dan	Hasil penelitian menunjukkan kombinasi algoritma *Beaufort Cipher* dan RSA dalam teknik super enkripsi dapat meningkatkan keamanan *file* teks. Proses enkripsi dilakukan dalam dua tahap, pertama

No	Penulis	Judul Penelitian	Hasil Penelitian
		Algoritma RSA (Y. R. Nasution et al., 2023)	dengan menggunakan algoritma *Beaufort Cipher*, dilanjutkan dengan enkripsi menggunakan kunci publik RSA. Penelitian ini menghasilkan aplikasi berbasis *web* yang diuji untuk keamanan dan efisiensinya, dengan estimasi waktu enkripsi sekitar 0.579 ms dan dekripsi 0.433 ms.
5	Ikhsan (2021)	Pengamanan *File* MP3 Berbasis Android Dengan Menggunakan Metode *Vigenere Cipher* (Ikhsan, 2021)	Hasil penelitian menunjukkan algoritma *Vigenere Cipher* berhasil digunakan untuk mengamankan *file* MP3 dengan aplikasi *mobile* berbasis Android. Implementasi tersebut memungkinkan *file* MP3 dienkripsi dan didekripsi dengan baik, sehingga *file* MP3 menjadi aman dan rahasia. Hasil uji coba menunjukkan bahwa aplikasi mampu mengenkripsi dan mendekripsi *file* MP3 dengan komputasi yang cepat dan sesuai dengan yang diharapkan.
6	Rifki, Raditya dan Hasugian (2023)	Aplikasi Keamanan Data Teks Menggunakan Algoritma *Base64* Berbasis *Mobile* (Rifki et al., 2023)	Hasil penelitian menunjukkan algoritma *Base64* dapat digunakan untuk mengamankan pesan teks yang dikembangkan untuk perangkat *mobile*. Penelitian ini memberikan opsi tambahan dalam melindungi data pribadi saat bertransaksi melalui

No	Penulis	Judul Penelitian	Hasil Penelitian
			jaringan komunikasi. Algoritma *Base64* dapat mengacak struktur informasi sehingga berbeda jauh dari bentuk aslinya. Aplikasi ini dirancang dengan pengembangan sistem berbasis UML dan diuji menggunakan metode "*black box*" yang menunjukkan hasil yang sukses dalam proses mengamankan data teks.

Pada penelitian ini, yang mengangkat topik tugas akhir dengan judul penelitian "Implementasi *Base64*, *Beaufort Cipher*, *Vigenere Cipher* Untuk Pengamanan *File Source Code* Dalam Bahasa Pemrograman", akan menerapkan ketiga algoritma yaitu *Base64*, *Beaufort Cipher*, *Vigenere Cipher* untuk mengamankan isi dari suatu *file source code* yang berbasis teks (*string*), dan dalam berbagai ekstensi bahasa pemrograman. Tujuan dari pengamanan *file source* adalah untuk mencegah modifikasi yang tidak sah dan menjaga integritas dari *file source*, kemudian penelitian ini diimplemenkan menjadi aplikasi berbasis *web*.

Nilai kebaruan dari penelitian ini yaitu dapat mengimplementasikan ketiga algoritma yaitu *Base64*, *Beaufort Cipher*, *Vigenere Cipher* dalam mengamankan *file source code*, dan melakukan sedikit modifikasi pada rumus algoritma *Beaufort Cipher* dan *Vigenere Cipher* agar dapat disandingkan dengan algoritma *Base64*, yang memiliki jumlah 64 karakter, agar terhindar dari karakter spesial yang tidak dapat terikut dalam proses enkripsi. Pembeda aplikasi pengamanan *file source code* ini dengan aplikasi *encryptor online* seperti pengkaburan (*obfuscator*) *file source code*, yaitu adanya keterlibatan menggunakan kunci untuk mengenkripsi dan mendekripsi *file source code*, kemudian proses enkripsi menggunakan metode *real-time encription* yang tidak menyimpan *record* didalam *database* dari *file source code* asli dan kunci yang digunakan untuk mengamankan *file source code* tersebut.