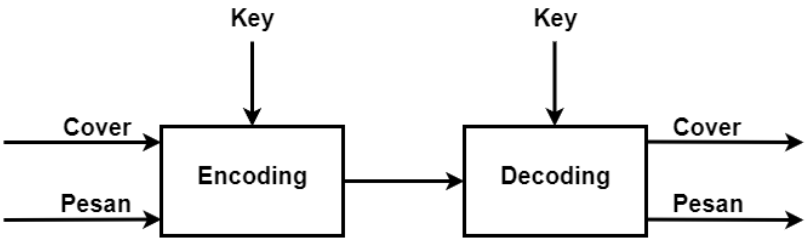
BAB II

TINJAUAN PUSTAKA

2.1. Landasan Teori

2.1.1 Steganografi

Steganografi berasal dari bahasa Yunani. Istilah ini terdiri dari dua bagian: *steganos*, yang berarti tersembunyi atau tertutup, dan *graphein*, yang berarti menulis. Jadi, secara sederhana, steganografi berarti menulis sesuatu yang tampak tersembunyi atau tidak terlihat (Sitio, 2020). Dalam bidang keamanan komputer, steganografi merupakan cara untuk menyimpan informasi dalam sebuah media sehingga sulit untuk dideteksi keberadaannya (Martawireja dkk., 2021). Steganografi yang baik memiliki kriteria – kriteria diantaranya yaitu pesan atau informasi yang disisipkan tidak dapat dikenali oleh orang (*Imperceptibility*), media penampung yang mengandung pesan tidak mengalami perubahan dari *file* aslinya (*fidelity*), media penampung harus kuat terhadap operasi yang mungkin dilakukan pada media penampung (*robustness*), dan pesan harus dapat diperoleh kembali (*recovery*) (Ikram dkk., 2023).



Gambar 2. 1 Cara Kerja Steganografi

Gambar 2.1 menjelaskan cara kerja steganografi. Proses *encoding* dilakukan dengan menyisipkan pesan kedalam *cover* dengan *key* tertentu. Proses *decoding* bertugas mengubah mengeluarkan kembali pesan yang sebelumnya telah disisipkan kedalam *cover*.

2.1.2 End Of File

Metode steganografi berbasis *End Of File* memungkinkan penyisipan pesan di bagian akhir berkas, dengan kapasitas penyisipan yang tidak memiliki batasan jumlah (Eka Putri, Kartikadewi, dan Rosyid, 2020). Pesan tersembunyi ditempatkan pada citra digital diubah menjadi nilai desimal sesuai tabel *American Standard Code for Information Interchange* (ASCII). ASCII adalah sistem pengkodean numerik untuk berbagai karakter termasuk huruf kecil a-z, huruf besar A-Z, angka 0-9, serta simbol-simbol standar yang terdapat pada *keyboard* (Basim dan Painem, 2020).

Contohnya pada citra *grayscale* berukuran 6x6 *pixel* disisipkan pesan "pesan". Untuk menandai berakhirnya pesan digunakan karakter yang jarang dipakai, seperti simbol #. Sehingga pesan yang dimaksud adalah “#pesan”. Kode ASCII dari “#pesan” diberikan sebagai berikut:

113 101 115 97 110 35

Misalkan sebuah citra grayscale dengan kode warna :

83 158 69 160 60 181

197 85 155 60 60 87

102	**91**	**87**	**197**	**77**	**60**
175	**100**	**160**	**95**	**92**	**69**
189	**81**	**71**	**122**	**68**	**127**
186	**77**	**151**	**141**	**164**	**131**

Nilai desimal pesan rahasia berdasarkan tabel ASCII disisipkan pada akhir citra gambar, sehingga citra gambar berubah menjadi :

83	**158**	**69**	**160**	**60**	**181**
197	**85**	**155**	**60**	**60**	**87**
102	**91**	**87**	**197**	**77**	**60**
175	**100**	**160**	**95**	**92**	**69**
189	**81**	**71**	**122**	**68**	**127**
186	**77**	**151**	**141**	**164**	**131**
113	**101**	**115**	**97**	**110**	**35**

2.1.3 Kompresi Data

Teknik kompresi data adalah metode untuk memperkecil ukuran data dengan tetap mempertahankan informasi penting di dalamnya (Sihombing dkk., 2024). Kompresi data utamanya bertujuan untuk mengurangi penggunaan ruang penyimpanan dan mengoptimalkan efisiensi ketika data ditransmisikan (Reswan dan Tri Anggoro, 2024). Secara fundamental, proses kompresi data bekerja dalam sebuah sistem yang terdiri dari dua komponen tak terpisahkan: *encoder* (penyandi) dan *decoder* (pembaca sandi). Sebagaimana dijelaskan oleh (Sayood, 2018) dalam literatur standar mengenai kompresi data, *encoder* bertugas mengubah data asli menjadi representasi baru yang lebih ringkas. Representasi ini bukanlah data yang

dapat langsung digunakan, melainkan serangkaian instruksi atau data yang telah dioptimalkan. Oleh karena itu, untuk mengembalikan data ke bentuk aslinya yang dapat dibaca dan digunakan oleh aplikasi, proses kebalikannya yaitu dekompresi melalui *decoder* mutlak diperlukan. Tanpa melalui tahap dekompresi, file yang telah terkompresi tidak akan dapat diinterpretasikan dengan benar oleh aplikasi pada umumnya.

Pada penelitian ini algoritma kompresi data yang gunakan diantaranya *Zstandard* (Zstd), *Lempel-Ziv Markov chain Algorithm* (LZMA), *Brotli*, dan *Huffman Encoding*.

A. Zstandard

Zstandard (zstd) merupakan algoritma kompresi *lossless* yang dikembangkan oleh Yann Collet di Facebook dan diluncurkan sebagai perangkat lunak *open source* pada 31 Agustus 2016 (Poranki, 2020). *Zstandard* (Zstd) merupakan algoritma kompresi yang mengkombinasikan pengkodean entropi melalui algoritma *Finite State Entropy* (FSE) dan Pengkodean *Huffman* (Novanto dkk., 2024). Penelitian (Chen dkk., 2021) menjelaskan alur proses kompresi *Zstandard* sebagai berikut:

1) **Pembagian Data (*Framing & Blocking*)**: Data input pertama-tama dibagi menjadi satu atau lebih frame independen. Setiap frame kemudian dibagi lagi menjadi blok-blok yang lebih kecil. Pembagian ini memungkinkan dekompresi paralel dan pemrosesan data secara streaming.

2) **Pencarian Kecocokan (LZ77 *Stage*)**: Untuk setiap blok, algoritma memindai data untuk menemukan urutan yang berulang. Ini dilakukan dengan

menggunakan hash table untuk dengan cepat menemukan potensi kecocokan. Ketika kecocokan ditemukan, algoritma menghasilkan token yang terdiri dari tiga komponen: literal length (panjang data mentah sebelum kecocokan), match length (panjang urutan yang cocok), dan offset (jarak ke urutan yang cocok sebelumnya).

3) **Pengkodean Entropi (*Entropy Encoding Stage*)**: Literal (data mentah) dan token (informasi kecocokan) yang dihasilkan dari tahap LZ77 kemudian dikompresi lebih lanjut.

   a) **Kompresi Literal**: Blok literal dikompresi menggunakan *Huffman Encoding* atau FSE. Algoritma akan memilih metode yang menghasilkan output terkecil.

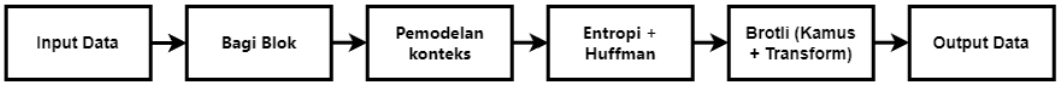
   b) **Kompresi Urutan**: Informasi dari token (*literal length, match length, offset*) juga dikompresi menggunakan FSE

4) **Pembentukan Frame Output**: Hasil kompresi dari setiap blok digabungkan, bersama dengan header yang berisi metadata (seperti ukuran konten, ID kamus, dll.), untuk membentuk *frame* Zstd akhir.

B. Brotli

Brotli adalah algoritma kompresi *lossless* yang dikembangkan oleh Google dan dirilis pada tahun 2015 dirancang untuk mengompres data dengan efisiensi tinggi, terutama untuk data teks dan web (Chýlek, 2020). Kompresi *Brotli* berlangsung dalam dua tahap: tahap pertama menggunakan penguraian berdasarkan kejadian sebelumnya untuk menciptakan kamus LZ77 statis, sedangkan tahap kedua

melakukan pengkodean *Huffman* pada data yang telah diuraikan dengan memilih pengkodean optimal dari bentuk-bentuk *Huffman* kanonik (Narashiman dan Chandrachoodan, 2024). Cara kerja algoritma *Brotli* divisualisasikan pada gambar 2.3.



Gambar 2. 2 Algoritma *Brotli*

Pada Gambar 2.3 secara garis besar cara kerja algoritma *Brotli* diantaranya:

1) Algoritma akan memecah konten *file* teks menjadi sejumlah segmen atau potongan kecil, kemudian melakukan kompresi pada masing-masing segmen secara independen.

2) Pada fase kedua, algoritma menerapkan teknik pemodelan konteks untuk memaksimalkan efisiensi kompresi. Dalam tahap ini, algoritma menganalisis data untuk menciptakan model yang mengidentifikasi pola statistik pada tiap segmen, sehingga menyempurnakan proses kompresi.

3) Tahap berikutnya melibatkan proses pengkodean yang menggabungkan metode pengkodean entropi dengan pengkodean *Huffman* untuk memadatkan data. Pengkodean entropi berfungsi menyederhanakan representasi data, sedangkan pengkodean *Huffman* mengalokasikan kode-kode pendek untuk pola yang frekuensi kemunculannya tinggi guna mengoptimalkan kompresi.

4) Di tahap akhir, kompresi data dilakukan dengan algoritma *Brotli* yang mengintegrasikan teknik kompresi berbasis kamus dan transformasi untuk menghasilkan tingkat kompresi yang optimal.

C. Lempel-Ziv-Markov chain (LZMA)

Algoritma *Lempel-Ziv-Markov Chain* (LZMA) merupakan metode kompresi data yang dikembangkan sebagai penyempurnaan dari algoritma LZ77 (Aswar dkk., 2023). Algoritma *Lempel-Ziv-Markov chain* (LZMA) menggunakan teknik kompresi berbasis kamus (*dictionary compression*) dengan ukuran kamus yang besar dan dukungan khusus untuk jarak pencocokan yang sering digunakan (Sugirtham dkk., 2024). Kompresi LZMA memiliki dasar prinsip yang serupa dengan teknik *deflate*, namun mengadopsi *range coding* sebagai alternatif dari metode *Huffman Encoding*. Algoritma ini memanfaatkan segmen dari *stream input* sebagai kamus dalam proses kompresi data. *Encoder* memproses input secara sekuensial dari kanan ke kiri bersamaan dengan keseluruhan data yang akan dikompresi. Teknik ini dikenal dengan konsep *sliding windows*. *Window* tersebut terbagi menjadi dua komponen: bagian kiri yang disebut *search buffer* dan bagian kanan yang dinamakan *look-ahead buffer* yang menampung teks yang akan mengalami kompresi. Dalam penerapan praktisnya, algoritma LZMA menghasilkan berkas terkompresi yang tersusun dari sekuensi karakter yang merepresentasikan kombinasi terpadu dari tiga elemen utama: *distance, length, dan next symbol* (Ramadhan, 2016). *Distance* mengacu pada interval antara karakter yang ditemukan dalam *look-ahead buffer* yang identik dengan karakter di *search buffer*. *Length* merujuk pada ukuran rangkaian karakter terpanjang dalam *search*

buffer yang memiliki kecocokan di *look-ahead buffer*. Sedangkan *next symbol* didefinisikan sebagai karakter tunggal yang berada di *look-ahead buffer* yang muncul setelah rangkaian karakter terpanjang yang telah dipilih.

Tahap awal kompresi dimulai dengan pembacaan *input stream* per satuan *bytes* dan merekam sejumlah n *bytes* terakhir yang telah diproses. Ketika sistem mengidentifikasi rangkaian kata yang identik, sistem akan menghasilkan output berupa nilai yang dikenal sebagai token. Token tersebut memiliki struktur dengan format sebagai berikut:

T = (x,y,z)

x = *offset* (2 bit)

y = *length* (2 bit)

z = *next symbol* (1 byte)

Bersamaan dengan proses pengkodean, program juga mencatat peluang kemunculan setiap karakter yang berdampingan dan membangun model Markov dalam bentuk grafik terarah (*directed graph*).

Sebagai contoh, data yang akan dikompresi sebagai berikut : AABBCBBAABC.

Di bawah ini adalah tabel untuk proses Encoding yang disajikan pada tabel 2.1

Tabel 2. 1 hasil proses encoding algoritma LZMA

Step	Pos	Match	Token
1	1	--	(0,0,”A”)
2	3	A	(1,1,”B”)
3	5	B	(1,1,”C”)
4	6	BBB	(3,2,”A”)
5	11	ABA	(7,2,”C”)

Tabel 2.1 menyajikan penjelasan tentang proses *encoding* yang menggambarkan tahapan pengkodean *input stream*.

Pos : menunjukkan lokasi setiap karakter dalam aliran input.

Step : menguraikan tahapan dalam proses pengkodean.

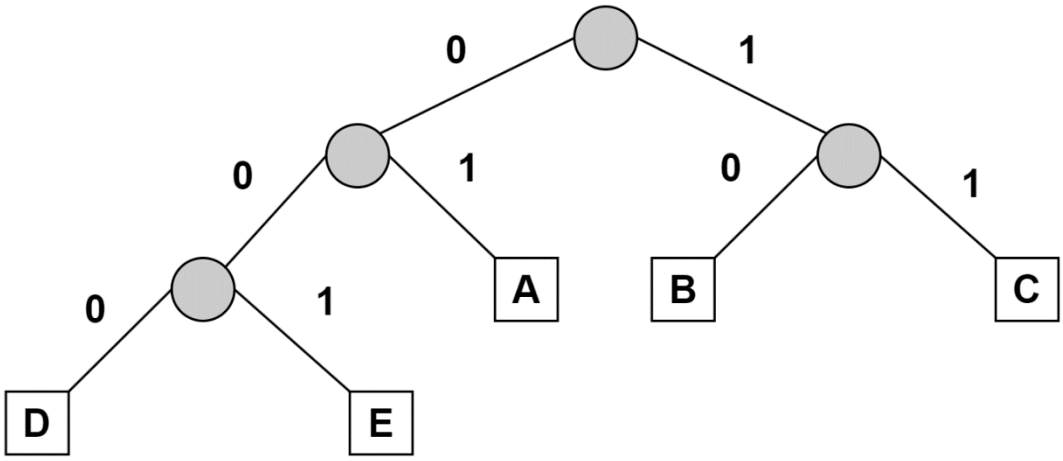
Match : merepresentasikan karakter-karakter identik yang teridentifikasi.

Token : merupakan rangkaian simbol hasil *encoding* yang terdiri dari nilai *offset, length,* serta karakter berikutnya.

D. Huffman Encoding

Algoritma Huffman merupakan algoritma yang diciptakan oleh David A (Satyapratama dan Yunus, 2015). Dalam algoritma *Huffman*, karakter dalam format ASCII dikonversi menjadi rangkaian bit (0 dan 1), dengan sistem pengkodean yang mirip dengan prinsip sandi morse dan menghitung frekuensi kemunculan karakter-karakter yang berulang, baik itu vokal maupun konsonan, dalam suatu data atau berkas (Supiyandi dan Frida, 2018). Sebagai contoh, sebuah dokumen mengandung

karakter A, B, C, D dan E. Pengurutan karakter dilakukan berdasarkan frekuensi kemunculannya, dimulai dari yang paling sering muncul hingga yang paling jarang ditemukan. Struktur pohon *Huffman* selanjutnya dibentuk untuk pengkodean karakter A hingga karakter E seperti pada gambar 2.4.



Gambar 2. 3 Struktur Pohon Huffman

Pohon Huffman pada gambar 2.1 menunjukkan hasil pengkodean sebagai berikut:

Karakter A memperoleh representasi kode biner 01.

Karakter B dikonversi menjadi kode biner 10.

Karakter C ditransformasikan ke dalam kode biner 11.

Karakter D direpresentasikan dengan kode biner 000.

Karakter E diwakili oleh kode biner 001.

2.2. State-of-the-Art

Terdapat beberapa penelitian yang signifikan dalam mengembangkan dan menganalisis berbagai algoritma kompresi guna meningkatkan efisiensi dalam steganografi gambar. Kontribusi penting dari penelitian-penelitian tersebut terletak pada pemahaman karakteristik serta kelebihan dan kekurangan masing-masing algoritma. Penelitian sebelumnya memiliki potensi untuk menjadi dasar penting dalam pengembangan metode kompresi yang lebih optimal guna meningkatkan kinerja metode EOF dalam steganografi gambar. Tabel 2.2 yang memuat penelitian terkait yang menjadi landasan dalam penelitian ini.

Tabel 2. 2 *State-of-the-Art*

No	**Nama Peneliti/Journal**		**Hasil Penelitian**
1	**Peneliti (tahun)**	(Maulana dan Rahmatulloh, 2025)	Penelitian ini menganalisis kinerja kompresi *Huffman Encoding* dan *Burrows-Wheeler Transform* (BWT) pada steganografi berbasis *End of File* (EOF) untuk mereduksi ukuran gambar. Hasil pengujian
	Judul	Analisis Kinerja Kompresi *Huffman* dan BWT pada	

No	Nama Peneliti/Journal		Hasil Penelitian
		Steganografi dalam Reduksi Ukuran Gambar	menunjukkan bahwa metode *Huffman* berhasil mengurangi ukuran *file* rata-rata sebesar 1,39%
	Algoritma/Metode	*End of File, Huffman Encoding,* BWT	
2	**Peneliti (tahun)**	(Hlayel dkk., 2024)	Penelitian ini membandingkan beberapa algoritma kompresi *lossless* pada paket aset 3D. Hasil menunjukkan bahwa LZMA mencapai rasio kompresi tertinggi (45% pengurangan) sambil tetap menjaga waktu muat yang dapat diterima untuk AR seluler.
	Judul	*Enhancing unity-based AR with optimal lossless compression for digital twin assets*	
	Algoritma/Metode	LZMA, LZ4, G-Zip, 7Zip, Zip	
3	**Peneliti (tahun)**	(Prayogo dkk., 2024)	

No	Nama Peneliti/Journal		Hasil Penelitian
	Judul	*Enhancing Least Significant Bit Steganography Image Fidelity Using Brotli Compression*	Penelitian ini membahas peningkatan kualitas gambar steganografi menggunakan metode Least Significant Bit (LSB) yang dikombinasikan dengan kompresi *Brotli*. Hasil kompresi menunjukkan rasio rata-rata sebesar 64% (dihitung dari lima sampel: 58,67%, 64,10%, 64,84%, 65,70%, dan 66,55%). Artinya, data teks setelah kompresi rata-rata menjadi 64% dari ukuran aslinya.
	Algoritma/Metode	*Least Significant Bit* (LSB) dan *Brotli*	
4	**Peneliti (tahun)**	(Yiko Satriyawibawa dkk., 2024)	Penelitian ini berfokus pada peningkatan kapasitas penyisipan data dalam steganografi menggunakan metode *Least Significant Bit* (LSB-2) yang dikombinasikan dengan algoritma kompresi *Brotli* dan encoding base64. Dari segi efisiensi kompresi, algoritma *Brotli* menunjukkan kinerja yang sangat baik, dengan rata-rata reduksi ukuran data sebesar 47,13% untuk data kecil (1.000-5.000 byte) dan
	Judul	*LSB-2 Steganography with Brotli Compression and base64 Encoding for Improving Data Embedding Capacity*	

No	Nama Peneliti/Journal		Hasil Penelitian
	Algoritma/Metode	*Least Significant Bit* (LSB), *Brotli*, dan base64	71,34% untuk data besar (10.000-20.000 byte). Semakin besar ukuran data, semakin tinggi pula tingkat kompresi yang dicapai.
5	**Peneliti (tahun)**	(Yiko Satriyawibawa dkk., 2024)	Penelitian ini membahas optimasi steganografi gambar digital dengan menggabungkan metode *Least Significant Bit* (LSB) dan *Zstandard* (Zstd). Dari segi efisiensi kompresi, hasil eksperimen menunjukkan bahwa metode Zstd berhasil mengurangi ukuran teks sebelum penyisipan dengan rata-rata 43,99% hingga 56,48%, tergantung pada ukuran data awal. Ini berarti semakin besar ukuran data yang dikompresi, semakin tinggi pula efisiensi kompresi yang dicapai.
	Judul	*Optimizing Digital Image Steganography through Hybridization of LSB and Zstandard Compression*	
	Algoritma/Metode	*Least Significant Bit* (LSB) dan *Zstandard.*	
6	**Peneliti (tahun)**	(Ahlberg dan Bondesson, 2024)	

No	Nama Peneliti/Journal		Hasil Penelitian
	Judul	*Comparing compression algorithms for transaction logs in graph native databases*	Penelitian ini membandingkan algoritma kompresi GZIP, ZSTD, dan LZ4 untuk mengompresi transaction logs dalam database graf Neo4j. Hasil penelitian menunjukkan bahwa LZ4 adalah algoritma tercepat dengan hampir tidak ada dampak pada waktu eksekusi, tetapi rasio kompresinya berkisar antara 97% hingga 21% dari ukuran asli. GZIP dan ZSTD memiliki rasio kompresi yang lebih baik, berkisar antara 75% hingga 10%, tetapi memiliki dampak negatif terhadap waktu eksekusi, bahkan hingga 8 kali lebih lambat dari *baseline*.
	Algoritma/Metode	GZIP, *Zstandard* dan LZ4	
7	**Peneliti (tahun)**	(Resdiansyah dkk., 2021)	Penelitian ini membahas kompresi citra biner menggunakan algoritma Freeman Chain Code dan LZMA. Hasil menunjukkan bahwa kombinasi menghasilkan pengurangan ukuran rata-rata 56.9%.
	Judul	*Comparing Freeman Chain Code 4 Adjacency Algorithm*	

No	Nama Peneliti/Journal		Hasil Penelitian
		and LZMA Algorithm in Binary Image Compression	
	Algoritma/Metode	LZMA	
8	**Peneliti (tahun)**	(Dan dan Bimantoro, 2020)	Penelitian ini menerapkan algoritma Run-Length Encoding (RLE) yang telah dimodifikasi untuk meningkatkan efektivitas kompresi data dalam teknik steganografi. Modifikasi tersebut menghasilkan peningkatan rasio kompresi rata-rata dari -98,2% (RLE standar untuk JPG/PNG) menjadi 0,17%, serta dari -34,7% (RLE standar untuk BMP) menjadi 18,8%. Penggabungan teknik steganografi *End of File* (EOF), *First of File* (FOF), dan kombinasi keduanya berhasil menanamkan pesan terkompresi ke dalam *file cover*.
	Judul	Implementasi Modifikasi Kompresi *Run-Length Encoding* pada Steganografi	
	Algoritma/Metode	*End Of File* dan *Run-Length Encoding*	

No	Nama Peneliti/Journal		Hasil Penelitian
9	**Peneliti (tahun)**	(Nasution dan Johar, 2020)	Penelitian ini menerapkan algoritma kompresi *Huffman Encoding* pada steganografi *End Of File* untuk mengurangi ukuran *stego-file*. Ukuran *file cover* yang rata – rata 300KB – 500KB disisipkan *file* .tif ukuran 1.399KB dan video .3gp 4.284KB. Tanpa menggunakan algoritma kompresi *Huffman Encoding*, ukuran *stego-file* sebesar 1.896 KB dan 4.474 KB. Terbukti adanya *overhead* ukuran *file* setelah disisipkan.
	Judul	Aplikasi Penyembunyian Multimedia Menggunakan Metode End Of File (Eof) Dan Huffman Coding	
	Algoritma/Metode	*End Of File*, *Huffman Encoding*, AES-128	

Secara keseluruhan, penelitian-penelitian ini menunjukkan bahwa algoritma kompresi seperti *Huffman Encoding*, LZMA, *Brotli*, dan *Zstandard* memiliki potensi besar dalam mengoptimalkan algoritma steganografi berbasis EOF dikarenakan masih terdapat kekurangan pada steganografi berbasis EOF, seperti *overhead* membuat ukuran *file* lebih besar serta minimnya perbandingan komprehensif antara algoritma kompresi berdasarkan rasio kompresi, durasi pemrosesan, dan penggunaan memori dalam konteks steganografi berbasis EOF. Oleh karena itu, penelitian ini bertujuan untuk mengatasi kekurangan tersebut dengan menganalisis kinerja komparasi algoritma kompresi (*Zstandard*, LZMA, *Brotli*, dan *Huffman Encoding*) pada steganografi berbasis EOF, dengan fokus pada rasio kompresi, durasi pemrosesan, penggunaan memori. Pendekatan ini diharapkan dapat memberikan rekomendasi algoritma kompresi yang paling optimal dalam rasio kompresi, durasi pemrosesan dan penggunaan memori pada steganografi berbasis EOF