Implementasi Steganografi Berbasis *Motion Vector* pada Video H.264 (MP4)

Bryan Cornelius Lauwrence - 13522033
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
bryan.cornelius.lauw@gmail.com, 13522033@std.stei.itb.ac.id

***Abstrak*—Makalah ini membahas implementasi steganografi berbasis *motion vector* pada video H.264 (MP4) sebagai metode untuk menyembunyikan pesan rahasia. Dengan memanfaatkan *motion vector* yang merupakan komponen kompresi H.264, pesan dapat disisipkan tanpa perbedaan signifikan secara visual. Proses *embedding* dilakukan dengan memodifikasi *library* x264 pada bahasa C, sementara ekstraksi menggunakan Python dengan library PyAV dan FFmpeg. Implementasi ini berhasil menyisipkan dan mengekstraksi pesan pendek dengan baik, tetapi memiliki keterbatasan pada kapasitas penyimpanan dan stabilitas data untuk pesan yang lebih panjang. Jadi, steganografi berbasis *motion vector* layak diterapkan, tetapi memerlukan integrasi dengan strategi lain untuk meningkatkan keandalan steganografi.**

Kata kunci—steganografi; motion vector; H.264; MP4

I. Latar Belakang

Di era digital ini, keamanan data menjadi sesuatu yang penting, terutama untuk informasi yang bersifat rahasia. Oleh karena itu, muncul ilmu untuk menjaga keamanan pesan yang disebut dengan kriptografi. Kriptografi umumnya digunakan untuk mengenkripsi pesan. Tujuan utamanya adalah menyembunyikan makna asli dari suatu pesan dengan menjadikannya pesan rahasia.

Seringkali proses enkripsi tidak cukup untuk menyembunyikan pesan karena dapat menimbulkan kecurigaan pada pihak ketiga, seperti yang disebutkan pada *the prisioner's problem*. Meskipun pesan dienkripsi, pengirim akan mencurigai pesan yang dienkripsi tersebut. Oleh karena itu, dalam kriptografi diperlukan juga steganografi. Berbeda dengan kriptografi, yaitu ilmu untuk menyembunyikan pesan, steganografi adalah ilmu untuk menyembunyikan pesan rahasia sehingga tidak ada seorang pun yang mencurigai keberadaan pesan tersebut.

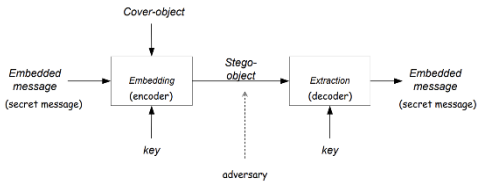
Salah satu teknik steganografi di dunia modern ini adalah menyembunyikan pesan pada suatu media yang berupa *file*. *File* dapat berupa apapun, tetapi salah satu yang dapat digunakan adalah *file* video (MP4). Video dapat menjadi media steganografi yang baik karena ukurannya besar sehingga dapat menyimpan banyak pesan dan sudah umum digunakan, khususnya video H.264/MP4.

Masalah umum untuk steganografi adalah teknik-teknik yang umum belum tentu dapat diterapkan secara langsung pada seluruh jenis berkas. Sebagai contoh teknik steganografi *least significant bit* (LSB) yang paling sederhana, yaitu mengganti bit paling kanan (*least significant*) dengan bit pesan, tidak cocok diterapkan secara langsung pada berkas video H.264/MP4. Teknik yang lebih cocok Adalah penggunaan LSB dengan *motion vector* karena *motion vector* merupakan bagian dari proses kompresi video yang menyimpan informasi pergerakan antar *frame*, sehingga modifikasi lebih sulit dideteksi dan tidak terlalu mempengaruhi kualitas visual.

II. Dasar Teori

A. Steganografi Digital

Steganografi digital adalah penyembunyian pesan digital di dalam dokumen digital lainnya. Dokumen digital yang dimaksud dapat berupa teks, audio, gambar, bahkan video. Pesan yang disembunyikan, disebut *embedded message*, merupakan suatu dokumen digital. Pesan yang digunakan untuk menyimpan *embedded message*, disebut sebagai *cover object*, juga merupakan dokumen digital. Hasil penyembunyian *embedded message* pada *cover object* disebut sebagai *stego object*. Proses penyembunyian dapat memanfaatkan *stego key* untuk menyisipkan dan mengekstrasi pesan. Alur steganografi digital dapat dilihat pada Gbr. 1.



Gbr. 1. Alur Steganografi Digital

Terdapat empat kriteria untuk menilai bagus atau tidaknya suatu steganografi, yaitu *imperceptible*, *fidelity*, *recovery*, dan *capacity*. *Imperceptible* artinya keberadaan pesan rahasia tidak dapat dipersepsi secara visual atau audial, *fidelity* artinya kualitas *cover object* tidak berubah jauh akibat penyisipan, *recovery* artinya pesan yang disembunyikan harus dapat diekstrak kembali, dan *capacity* berarti ukuran pesan yang disembunyikan sedapat mungkin besar. Jadi, tujuan utama steganografi digital adalah menyembunyikan pesan yang dapat diambil kembali dengan proses yang efisien.

Salah satu metode umum yang digunakan untuk

steganografi digital merupakan metode *least significant bit* (LSB). Prinsip dari LSB adalah mengganti bit-bit yang tidak memengaruhi nilai *byte* secara signifikan. Misalnya terdapat satu *byte* bernilai 209, yaitu 11010001. Andaikata bit paling kanan diubah dari 1 menjadi 0, nilai desimal dari *byte* tersebut menjadi 208. Akan tetapi, jika bit paling kiri diubah dari 1 menjadi 0, nilai desimalnya menjadi 81. Perbedaan nilai yang lebih tidak signifikan adalah ketika perubahan dilakukan pada bit paling kanan.

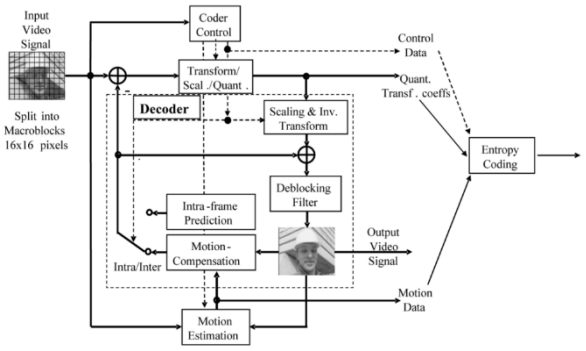
Oleh karena itu, dalam metode LSB, bit-bit di kanan dapat digunakan untuk menyimpan bit-bit dari pesan asli dan tidak terbatas pada 1-bit saja. Tentunya, ukuran *embedded message* yang dapat ditampung jauh lebih kecil daripada ukuran *cover object*. Namun, metode ini cukup sederhana dan dapat diterapkan ke banyak *cover object* karena seluruh dokumen digital pada dasarnya adalah sekumpulan bit-bit.

B. Kompresi H.264

Pada era modern, video menjadi salah satu media informasi yang populer digunakan oleh berbagai kalangan. Video mampu menawarkan fitur-fitur menarik seperti gambar bergerak dan adanya suara. Kini, video sudah menjadi salah satu media yang banyak digunakan untuk berbagi informasi. Ditambah lagi, dengan adanya ponsel pintar yang menyediakan kamera, akses pembuatan video dipermudah bagi seluruh orang.

Dengan banyaknya video yang beredar atau disimpan, perlu suatu cara untuk menyimpan video-video secara efektif. Pada dasarnya, video adalah gambar-gambar yang berurutan yang ditampilkan secara bergantian dalam waktu yang cepat. Tanpa adanya kompresi, suatu video berwarna dengan resolusi 1920×1080 dan kecepatan *frame* 25 *frame per second* akan memerlukan sekitar 1.24 Gb setiap detiknya. Jadi, video sepanjang satu menit saja dapat memakan lebih dari 60 Gb. Jadi, diperlukan metode untuk mengkompresi video supaya efisien memori, tetapi tetap menampilkan video yang mulus.

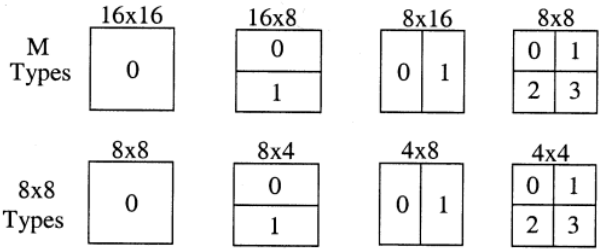
Salah satu metode kompresi video adalah H.264. Metode ini dikembangkan oleh ITU-T VCEG dan ISO/IEC MPEG. Inti dari pemrosesan H.264 adalah pendekatan *hybrid video coding* berbasis blok, setiap gambar dibagi menjadi *macroblock* (biasanya 16x16 piksel) yang kemudian diproses menggunakan prediksi spasial atau temporal.



Gbr. 2. Diagram Blok *Encoder* H.264

Untuk mengurangi redundansi antar-*frame*, H.264 menggunakan metode *inter prediction*. Metode ini memanfaatkan hubungan temporal dengan mencari suatu blok yang mirip dari *frame* sebelum atau sesudahnya untuk memprediksi *frame* saat ini. Proses pencarian blok yang paling cocok ini disebut *motion estimation*. Hasil dari proses ini adalah *motion vector*, yaitu vektor yang menunjukkan pergeseran posisi antara blok saat ini dengan blok referensinya. Dalam konteks steganografi, vektor inilah yang menjadi wadah potensial untuk menyisipkan pesan rahasia karena perubahan kecil pada nilai vektor seringkali tidak terlalu mempengaruhi kualitas visual video secara signifikan.

Salah satu keunggulan H.264 yang relevan dengan kapasitas penyisipan data adalah fleksibilitas ukuran bloknya. H.264 mendukung pembagian *macroblock* menjadi partisi yang lebih kecil, mulai dari 16×16 hingga sekecil 4×4 piksel untuk *motion compensation* yang lebih detail. Selain itu, akurasi *motion vector* dalam H.264 dapat mencapai seperempat piksel yang memberikan presisi lebih tinggi dibandingkan standar lama. Vektor-vektor ini kemudian dikodekan secara diferensial terhadap prediksi vektor tetangganya sebelum ditransmisikan.



Gbr. 3. Partisi *Macroblock*

C. MP4

MP4 yang dikenal sebagai MPEG-4 *Part* 14 adalah suatu wadah yang membungkus berbagai media, yakni video, audio, gambar, beserta dengan metadata. Umumnya MP4 digunakan untuk video, tetapi dapat menyimpan takarir, gambar 3D, dan metadata lainnya yang membuat MP4 serbaguna. MP4 memiliki fungsi yang berbeda dengan H.264. H.264 fungsinya mengompresi video, sedangkan MP4 bertindak sebagai pembungkus untuk menyimpan seluruh data hasil kompresi dan data-data lainnya.

Karena MP4 adalah sebuah wadah (*container*), MP4 tidak memiliki struktur yang khusus. Secara fungsinya, MP4 terbagi menjadi dua bagian, yaitu *media-related data* yang menyimpan audio dan video, serta metadata seperti *timestamp*. Meskipun strukturnya fleksibel, MP4 terbagi menjadi komponen-komponen kecil yang disebut atom. Ukuran minimal atom Adalah 8 *byte*, dengan *byte* pertama menyimpan ukuran atom dan 4 *byte* setelahnya menyimpan tipe atom. Tipe atom yang umum digunakan dirincikan pada Tabel I.

Tabel I
Tipe Atom MP4

Tipe	Deskripsi
ftyp	Tipe *file*, deskripsi, dan struktur data
pdin	Pengaturan *loading* dan instalasi video
moov	Menyimpan seluruh metadata
moof	Menyimpan konten video
mfra	Menyediakan akses acak ke fragmen video
mdat	Kontainer data media mentah, seperti *bitstream* video dan audio.
ssts/stsc	Memetakan antara sampel dengan waktu (stts) dan antara sampel dengan chunk data (stsc)
stsz	Menyimpan ukuran setiap sampel yang menentukan *framing* video.
meta	Menyimpan metadata tambahan untuk MP4
mvhd	Informasi header video utama.
trak	Kontainer yang merepresentasikan satu *track* individual
udta	Kontainer yang menyimpan informasi pembuat video
iods	Deskriptor file MP4

MP4 sebenarnya tidak memengaruhi H.264, tetapi hanya berperan sebagai wadah yang menyimpan *bitstream* hasil *encoding* H.264. Video yang telah dikompresi oleh H.264 disimpan sebagai *video track* di dalam kontainer MP4, sering kali bersamaan dengan trek audio yang dikompresi menggunakan format AAC (*Advanced Audio Coding*). Steganografi berbasis *motion vector* bekerja pada level *bitstream* H.264 di dalam trek video tersebut, bukan pada struktur kontainer MP4 itu sendiri.

D. Motion Vector

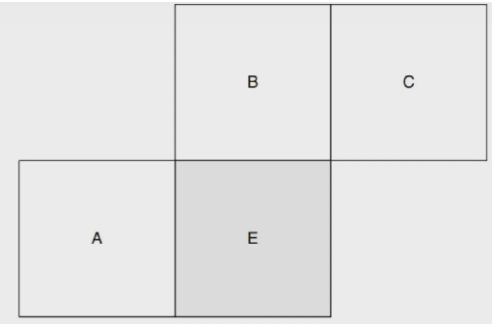
Motion vector (MV) merupakan komponen fundamental dalam teknik *inter prediction* pada standar H.264 yang berfungsi untuk mengeksploitasi redundansi temporal antar *frame* video. Dalam proses kompresi, MV merepresentasikan pergeseran posisi (*displacement*) dari sebuah blok piksel pada *frame* saat ini relatif terhadap posisi blok yang cocok pada frame referensi yang telah dikodekan sebelumnya. Secara sederhana, jika sebuah objek bergerak 2 piksel ke kanan dan 6 piksel ke bawah dibandingkan *frame* sebelumnya, MV akan menyimpan informasi koordinat relatif tersebut, misalnya sebagai vektor (2,6).

Keunggulan utama H.264 dalam penggunaan *motion vector* adalah fleksibilitas ukuran blok dan akurasinya. Standar ini mendukung partisi *macroblock*, yaitu satu *macroblock* berukuran 16×16 piksel dapat dipecah menjadi partisi yang lebih kecil, yakni 16×8, 8×16, hingga sekecil 4×4 piksel seperti pada Gbr. 3. Setiap partisi kecil ini memiliki *motion vector* independen, memungkinkan *encoder* untuk menangkap gerakan yang kompleks dan detail dengan lebih presisi. Selain itu, H.264 meningkatkan presisi vektor ini menggunakan akurasi seperempat piksel (*quarter-sample accuracy*), yang memberikan kualitas prediksi visual yang jauh lebih baik dibandingkan standar sebelumnya yang hanya mendukung akurasi setengah piksel.

Peran utama MV adalah memfasilitasi prediksi kompensasi gerak dengan menunjuk pada pergeseran posisi blok relatif terhadap *frame* referensi. H.264 memanfaatkan korelasi spasial yang kuat antarblok karena blok-blok yang bertetangga cenderung memiliki karakteristik gerakan yang serupa. Oleh karena itu, standar ini tidak menyimpan nilai MV secara absolut, melainkan mengodekan selisihnya yang disebut sebagai *motion vector difference* (MVD).

Nilai MVD diperoleh dengan terlebih dahulu menghitung *motion vector predictor* (MVP) berdasarkan MV dari blok-blok tetangga yang telah dikodekan sebelumnya, yaitu blok kiri (A), atas (B), dan kanan-atas (C). Prediktor ini umumnya ditentukan dengan mengambil nilai median dari komponen X dan Y ketiga vektor tetangga tersebut. Setelah nilai prediksi (MVP) didapatkan, *encoder* menghitung selisih antara vektor gerak aktual MV_{cur} yang ditemukan dari proses estimasi gerak dengan vektor prediksi tersebut menggunakan rumus

$$MVD = MV_{cur} - MVP \tag{1}$$



Gbr. 4, Ilustrasi Partisi Blok

Sebagai contoh pada Gbr. 4, misalnya blok A memiliki MV (3,2), blok B memiliki MV (4,2), blok C memiliki MV (3,10), dan blok E memiliki MV (5,6). Dengan demikian, MVP hasil median adalah (3, 2) sehingga nilai MVD E (2,4). Namun, penggunaan MVP dengan median hanya berlaku jika partisi E berbentuk persegi. Jika partisi E berukuran 16×8, MVP bagian atas menggunakan partisi B dan bagian bawah menggunakan partisi A. Jika partisi E berukuran 8×16, MVP partisi kanan menggunakan A dan MVP partisi kiri menggunakan C. Jika *macroblock* di-*skip* (datanya kosong), *macroblock* dianggap berukuran 16×16 dan menggunakan median.

Nilai MVD inilah yang kemudian dikompresi dan disimpan dalam *bitstream*. Dalam konteks steganografi, manipulasi data biasanya dilakukan pada nilai MVD ini, karena perubahan kecil pada nilai selisih akan terintegrasi kembali dengan nilai prediksi saat proses *decoding*, menghasilkan distorsi visual yang minim. Namun, tetap mampu membawa informasi tersembunyi.

III. DESAIN DAN IMPLEMENTASI

Sebagaimana yang telah disebutkan, steganografi akan dilakukan pada berkas video atau MP4. Teknik yang akan digunakan adalah LSB pada *motion vector*. Implementasi proses *embedding* akan menggunakan *library* x264 dari pustaka bahasa pemrograman C untuk memudahkan penyisipan pada *motion vector* selama proses *encoding*. Sementara itu, proses ekstraksi akan menggunakan bahasa pemrograman Python memanfaatkan *library* PyAV dan FFmpeg yang memungkinkan akses terhadap informasi *motion vector* melalui metadata dan *side data* yang tersedia pada proses *decoding*. Untuk menyederhanakan proses implementasi, data yang disisipkan adalah data teks.

A. Embedding

Proses penyisipan *embedding* dilakukan dengan memodifikasi *library* x264 C. Pertama diperlukan suatu tipe data untuk mendefinisikan pesan rahasia dapat dilihat pada Gbr. 5.

```
typedef struct {
  uint8_t *data;
  long total_bits;
  long current_bits;
} StegoContext;
```

Gbr. 5, Tipe Data Pesan Rahasia

Data berisi seluruh *byte* dari pesan, total_bits merupakan panjang pesan dalam satuan bit, dan current_bits digunakan sebagai *counter* bit yang di-*embed* selama proses *embedding*.

Logika utama *embedding* ditambahkan pada file "analyse.c". *Embedding* hanay dilakukan pada *frame motion vector* yang berukuran 16×16. Jika nilai LSB *motion vector* sama dengan *current bit*, *motion vector* dibiarkan begitu saja. Jika nilai LSB *motion vector* berbeda, nilainya akan ditambah atau dikurang dengan satu supaya hanya satu bit yang berubah. Dengan demikian, perubahan pada video tidak terdeteksi, tetapi pesan masih bisa disimpan secara tersembunyi. Proses dapat dilihat pada Gbr. 6.

```
int secret_bit = stego_get_next_bit();
  if (secret_bit != -1)
    {
      // Simpan index
      embedded_map[mb_index] = 1;
      global_bit_counter++;

      // Core embedding
      int mv_x = a->l0.me16x16.mv[0];
      if ((mv_x & 1) != secret_bit)
        {
          if (mv_x < h->mb.mv_max_spel[0])
            mv_x++;
          else
            mv_x--;
          a->l0.me16x16.mv[0] = (int16_t)mv_x;
    }
```

Gbr. 6, Potongan Proses *Embedding* LSB

Yang menjadi masalah adalah proses *embedding* dapat tidak selau berurutan, misalnya dari *frame* pertama ke *frame* kedua. Hal ini disebabkan karena *library* x264 melakukan kondisional berdasarkan bentuk *frame*. Oleh karena itu, selain menyimpan hasil MP4, program pun akan menyimpan *file* pemetaan, berupa TXT, yang mendefinisikan urutan *embedding* dilakukan. Lokasi yang disimpan meliputi indeks *frame*, koordinat X pada *macroblock*, dan koordinat Y pada *mocroblock*. Gbr. 7 menunjukkan cuplikan dari berkas TXT tersebut.

```
1 3 1
1 7 2
1 1 3
…
```

Gbr. 7, Potongan Berkas Pemetaan

Setiap baris mendifisinikan lokasi sebuah bit. Kolom pertama adalah indeks *frame* dan kolom kedua serta ketiga adalah koordinat X dan Y.

Sementara itu, alur *embedding* dibuat sebagai berikut.

1. Pengguna *input* nama berkas MP4.
2. Pengguna *input* pesan rahasia.
3. Terakhir, pengguna *input* nama berkas *stego object*.

Untuk proses pembacaan *file input*, program akan menggunakan FFmpeg untuk mengubah video menjadi YUV mentah. Hasil YUV akan di-*encode* ulang menjadi H.264. Dengan demikian, proses pembacaan *motion vector* menjadi lebih mudah dan sederhana. Setelah *embedding* selesai, berkas H.264 dikembalikan menjadi berkas MP4.

Untuk mempermudah proses ekstraksi, bit yang disimpan tidak hanya bit-bit pesan rahasia, tetapi juga panjang pesan. Dengan demikian, ekstraksi dapat mengetahui sampai mana pesan harus dibaca. Nantinya, bit yang disimpan formatnya seperti pada Gbr. 8.

```
[32 bit panjang pesan] [bit-bit karakter]
```

Gbr. 8, Ururtan Bit untuk *Embedding*

B. Extraction

Proses ekstraksi tidak menggunakan *library* x264 dari C karena *library* tersebut hanya digunakan untuk *encoding* H.264 sehingga sulit membaca bit-bit *motion vector*. Oleh karena itu, digunakan Python dengan PyAV yang mampu membuka video, bahkan secara khusus mengambil *frame* yang berisi *motion vector*.

Proses *extraction* memerlukan dua berkas, yaitu *file* MP4 *stego object* dan *file* TXT pemetaan lokasi *embedding*. Pemetaan diperlukan di sini karena PyAV membaca seluruh *motion vector* yang tersedia pada video secara berurutan, sedangkan proses *embedding*, seperti yang telah dijelaskan, tidak meng-*embed* secara urut. Pengacakan urutan disebabkan *threading* dalam proses *encoding* dari x264.

Program akan meminta pengguna untuk *input* nama *file* MP4 *stego object*. Kemudian, program akan mencari *file* tersebut beserta *file* pemetaan dengan asumsi kedua nama *file* sama. PyAV akan membuka MP4 dan mengambil data-data *motion vector* di video. *File* pemetaan akan dibaca

baris demi baris dan mencari *frame* serta koordinat yang sesuai pada *motion vector*. Dari situ, diambil bit *motion vector* satu per satu sampai pesan kembali ditemukan. Gbr. 9 menunjukkan cuplikan kode Python untuk proses ektraksi.

```
for frame in packet.decode():
  if coding_frame_idx in target_map:
    frame_targets =
      target_map[coding_frame_idx]
    mvs= frame.side_data.get('MOTION_VECTORS')
      if mvs:
        mv_lookup = {}
        for mv in mvs:
          if mv.w == 16 and mv.h == 16:
            # Floor 16 untuk mencari koordinat
            mb_x = int(mv.dst_x / 16)
            mb_y = int(mv.dst_y / 16)
            mv_lookup[(mb_x, mb_y)] =
              mv.motion_x

        for (target_x, target_y) in
          frame_targets:

          if (target_x, target_y) in
            mv_lookup:

            mx = mv_lookup[(target_x,
                            target_y)]
            # LSB
            bit = int(mx) & 1
            extracted_bits.append(bit)
          else:
            extracted_bits.append(0)

  coding_frame_idx += 1
```

Gbr. 9, Potongan Proses *Extraction*

IV. PENGUJIAN

Pengujian dilakukan pada sebuah video MP4 berdurasi 4 detik. Video tersebut tidak memiliki audio dan tidak banyak pergerakkan. Proses pengujian akan dibagi menjadi dua, yaitu pengujian normal dan pengujian pesan panjang.

Pengujian normal dilakukan dengan *input* pesan yang relatif pendek, yaitu kata “testing”.

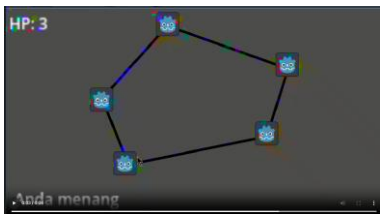
```
Penyisipan Pesan
Pastikan file input ada di folder 'input/'
Input nama file (misal: video.mp4): input.mp4
Masukkan pesan rahasia: testing
Nama file output (misal: stego.mp4): output-1.mp4
```

Gbr. 10, *Input* Kasus Pertama

```
Masukkan nama file stego (pastikan file ada di folder 'output/'): output-1.mp4
Ekstraksi pesan rahasia dimulai
Pencarian pesan rahasia...
Pencarian selesai. Total bit ditemukan: 88
Panjang pesan rahasia: 7 karakter

Pesan Rahasia:
testing
```

Gbr. 11, *Ouput* Kasus Pertama



Gbr. 12, *Screenshot File* output-1.mp4

Dari Gbr. 10 dan Gbr. 11, pesan berhasil disisipkan dan diekstraksi dengan baik, tetapi dari Gbr. 12 menunjukkan bahwa terjadi kerusakan pada video. Jika dilihat pada pranala GitHub, video awalnya baik-baik saja, tetapi seiring berjalannya waktu video akan semakin rusak.

Pengujian kedua dilakukan dengan *input* pesan yang lebih panjang, lebih dari lima belas karakter, yaitu kalimat “testing nomor dua”.

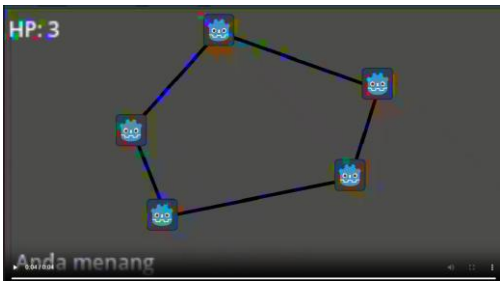
```
Penyisipan Pesan
Pastikan file input ada di folder 'input/'
Input nama file (misal: video.mp4): input.mp4
Masukkan pesan rahasia: testing nomor dua
Nama file output (misal: stego.mp4): output-2.mp4
```

Gbr. 13, *Input* Kasus Kedua

```
Masukkan nama file stego (pastikan file ada di folder 'output/'): output-2.mp4
Ekstraksi pesan rahasia dimulai
Pencarian pesan rahasia...
Pencarian selesai. Total bit ditemukan: 168
Panjang pesan rahasia: 17 karakter

Pesan Rahasia:
testing nomoR `u1
```

Gbr. 14, *Output* Kasus Kedua



Gbr. 15, *Screenshot File* output-2.mp4

Jika pesan terlalu panjang, seperti pada Gbr. 13, pesan akan rusak. Dari Gbr. 14, pesan yang keluar malahan “testing nomoR `u1”. Kerusakan ini disebabkan karena *library* x264 masih memproses *frame* setelah *embedding* dilakukan. *Frame* tersebut akan menghilang sehingga tidak terdeteksi oleh PyAV dan nilai LSB-nya dianggap 0. Kejadian ini hanya terjadi ketika pesan terlalu panjang.

Output dari kedua pengujian ini dapat dilihat pada *repository* GitHub. Berkas yang disimpan sesuai dengan nama-nama yang di-*input* pada kasus pengujian di atas.

V. KESIMPULAN DAN SARAN

Dari percobaan dan analisis yang telah dilakukan, steganografi berbasis *motion vector* pada video H.264 memang bisa dilakukan. Namun, strategi ini memiliki kelemahan. Pertama, steganografi hanya memanfaatkan bagian *motion vector* dari video sehingga lokasi yang dapat digunakan untuk *embedding* sangat minim. Kedua, karena

yang umum dilakukan adalah memodifikasi *library* yang ada, cukup sulit untuk menentukan area yang dapat di-*embed* tanpa menghilangkan data. Terakhir, *encoding* H.264 akan melalui proses panjang untuk mengoptimalkan memori sehingga pesan rawan menghilang.

Oleh karena itu, penulis menyarankan untuk menggabungkan metode LSB *motion vector* dengan metode lainnya karena MP4 terdiri dari audio dan data-data lainnya. LSB pun dapat dilakukan pada *frame* berukuran selain 16×16 untuk memperbesar kapasitas penyimpanan. Selain itu, terdapat banyak *library* pemrosesan H.264, seperti JCodec pada Java, yang dapat dilakukan eksperimen untuk melakukan steganografi.

PRANALA GITHUB

Seluruh program yang diimplementasikan dapat dilihat pada pranala berikut.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa oleh karena karunia-Nya sehingga penulis mampu menyelesaikan makalah IF4020 Kriptografi. Penulis mengucapkan terima kasih banyak kepada Bapak Rinaldi Munir sebagai pengajar penulis untuk mata kuliah ini. Terakhir, penulis mengucapkan terima kasih kepada para pembaca yang berkenan membaca makalah ini.

REFERENCES

[1] SecurityInfoWatch, "Understanding H.264 Video Compression," *SecurityInfoWatch*, [Daring]. Tersedia: https://www.securityinfowatch.com/home/article/10558554/understanding-h264-video-compression. Diakses pada: 22 Desember 2025.

[2] Toolstud Blog, "Video Bitrate Explained," *Toolstud*, 3 Mei 2023. [Daring]. Tersedia: https://blog.toolstud.io/video/2023/05/03/video-bitrate/. Diakses pada: 22 Desember 2025.

[3] D. Marpe, T. Wiegand, dan G. J. Sullivan, "The H.264/MPEG-4 Advanced Video Coding Standard and its Applications," *University of British Columbia*. [Daring]. Tersedia: https://www.cs.ubc.ca/~krasic/cpsc538a/papers/h264avc-overview.pdf. Diakses pada: 22 Desember 2025.

[4] G. Liang, "Intra Prediction and Inter Prediction," *gwliang.com*. [Daring]. Tersedia: https://gwliang.com/en/posts/intra-prediction-and-inter-prediction/. Diakses pada: 22 Desember 2025.

[5] G. J. Sullivan, "Overview of the H.264/AVC Video Coding Standard," *ITU-T Workshop on Video Coding*. [Daring]. Tersedia: https://www.itu.int/ITU-T/worksem/vica/docs/presentations/S3_P1_Sullivan.pdf. Diakses pada: 22 Desember 2025.

[6] Cloudinary, "MP4 Format (MPEG-4 Part 14): How It Works, Pros, and Cons," *Cloudinary Guides*. [Daring]. Tersedia: https://cloudinary.com/guides/video-formats/mp4-format-mpeg-4-part-14-how-it-works-pros-and-cons. Diakses pada: 22 Desember 2025.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 26 Desember 2025

Bryan Cornelius Lauwrence 13522033