Analisis Penerapan dan Penentuan Teknik Terbaik untuk Steganographic Watermark Beserta Digial Signature Pada Video

Muhamad Farhan Syakir
NIM: 18221145
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: aswehdebrej@gmail.com

***Abstract*—Masalah keamanan informasi dan pelanggaran hak cipta menjadi semakin kompleks di era digital yang berkembang pesat. Platform seperti YouTube, TikTok, dan layanan streaming lainnya memfasilitasi miliaran tayangan video setiap hari, membuat video yang didistribusikannya menjadi publik. Watermarking diimplementasikan untuk mencegah terjadinya pelanggaran hak cipta. Tanda tangan digital digunakan untuk memastikan keutuhan video yang didistribusikan. Dalam meningkatkan perlindungan hak cipta dan penjaminan keutuhan video tersebut, para pembuat konten memanfaatkan tanda tangan digital dan watermarking. Di makalah ini, teknik watermarking secara steganografis dan tanda tangan digital dibahas untuk mendapatkan teknik terbaik disertakan pertimbangan keamanan dan performa teknik. Hasil uji dalam makalah ini menunjukkan seberapa baik kualitas dari teknik yang diuji.**

I. PENDAHULUAN

A. Latar Belakang

Di era digital yang terus berkembang pesat ini, video digital telah menjadi salah satu media utama yang digunakan di berbagai bidang, mulai dari hiburan, pendidikan, hingga bisnis dan komunikasi. Platform seperti YouTube, TikTok, dan layanan *streaming* lainnya memfasilitasi miliaran tayangan video setiap hari. Bersamaan dengan itu, muncul berbagai tantangan terkait keamanan dan perlindungan hak cipta.

Seiring dengan meningkatnya penggunaan video digital, masalah keamanan informasi dan pelanggaran hak cipta menjadi semakin kompleks. Pelanggaran hak cipta dapat menyebabkan kerugian finansial yang signifikan bagi para pembuat konten, sementara manipulasi dan penyalahgunaan video dapat menimbulkan masalah integritas dan kepercayaan. [1]

B. Tujuan

1. Menentukan teknik *watermarking* dan tanda tangan digital yang paling baik untuk diterapkan pada video sebagai keamanan hak milik dan distribusi video pada media.
2. Menentukan kombinasi algoritma paling baik yang memiliki performa tinggi dalam menciptakan keamanan dan pemrosesan *steganographic watermarking* dan tanda tangan digital.
3. Menilai kualitas algoritma dan teknik melalui pengujian.

II. LANDASAN TEORI

A. Digital Watermarking

Penggunaan teknologi multimedia meningkat dari hari ke hari dan untuk menyediakan data resmi dan melindungi informasi rahasia dari penggunaan yang tidak sah sangatlah sulit dan melibatkan proses yang kompleks. Dengan menggunakan teknik watermark, hanya pengguna yang berwenang yang dapat menggunakan data tersebut. Digital watermark adalah teknologi yang banyak digunakan untuk melindungi data digital. Watermark digital berkaitan dengan penyisipan data rahasia ke dalam informasi aktual. Teknik watermark digital diklasifikasikan menjadi tiga kategori utama, berdasarkan domain, jenis dokumen (teks, gambar, musik atau video) dan persepsi manusia. Kinerja gambar yang diberi watermark dianalisis menggunakan rasio sinyal puncak terhadap *noise*, *mean square error*, dan *bit error rate*. *Watermarking* pada gambar telah diteliti secara mendalam untuk pencapaiannya yang terspesialisasi dan modern di semua aplikasi media seperti perlindungan hak cipta, laporan medis (pemindaian MRI dan sinar-X), anotasi, dan kontrol privasi. [2]

B. Tanda Tangan Digital

Tanda Tangan Digital adalah skema matematis yang menjamin privasi percakapan, integritas data, keaslian pesan/pengirim digital dan tidak ada penolakan dari pengirim. Tanda Tangan Digital tertanam di beberapa perangkat keras atau juga keluar sebagai file di perangkat penyimpanan. Tanda Tangan Digital ditandatangani oleh pihak ketiga, beberapa otoritas sertifikasi. Makalah ini menjelaskan perbedaan faktor kunci tanda tangan digital dengan cara kerja tanda tangan digital, melalui berbagai metode dan prosedur yang terlibat dalam penandatanganan data atau pesan dengan menggunakan tanda tangan digital. Ini memperkenalkan algoritma yang digunakan dalam tanda tangan digital. [3]

C. Steganografi

Steganografi adalah seni menyembunyikan fakta bahwa komunikasi sedang berlangsung, dengan menyembunyikan informasi dalam informasi lain. Banyak format *file* pembawa berbeda yang dapat digunakan, namun gambar digital adalah yang paling populer karena frekuensinya di Internet. Untuk menyembunyikan informasi rahasia dalam gambar, terdapat berbagai macam teknik steganografi, beberapa lebih kompleks dari yang lain dan semuanya memiliki kelebihan dan kekurangan masing-masing. Aplikasi yang berbeda memiliki persyaratan berbeda dari teknik steganografi yang digunakan. Misalnya, beberapa aplikasi mungkin

mengharuskan informasi rahasia tidak terlihat secara mutlak, sementara aplikasi lain memerlukan pesan rahasia yang lebih besar untuk disembunyikan. [4]

D. Steganografi Video

Steganografi video menjadi bidang penelitian penting dalam berbagai teknologi penyembunyian data, yang telah menjadi alat yang menjanjikan karena tidak hanya persyaratan keamanan transmisi pesan rahasia yang semakin ketat tetapi juga video lebih disukai. Dalam tulisan ini, menurut posisi tersematnya pesan rahasia, steganografi video dibagi menjadi tiga kategori: *intra-embedding*, *pre-embedding*, dan *post-embedding*. Metode *intra-embedding* dikategorikan menurut tahapan kompresi video seperti intra-prediksi, vektor gerak, interpolasi piksel, koefisien transformasi. Metode prapenyematan dimanipulasi pada video mentah, yang dapat diklasifikasikan ke dalam domain spasial dan transformasi. Metode pasca-penyematan terutama difokuskan pada aliran bit, yang berarti prosedur penyematan dan ekstraksi steganografi video semuanya dimanipulasi pada aliran bit terkompresi. Kemudian kami memperkenalkan penilaian kinerja untuk steganografi video dan steganografi video populer masa depan termasuk steganografi video H.265, steganografi video yang kuat, dan steganografi video yang dapat dibalik. [5]

E. Algoritma Blowfish

Blowfish adalah cipher blok 64-bit yang panjangnya bervariasi, simetris. Dirancang oleh Bruce Schneier pada tahun 1993 sebagai "algoritme tujuan umum", algoritma ini dimaksudkan untuk memberikan alternatif drop-in yang cepat, gratis, terhadap algoritma enkripsi Standar Enkripsi Data (DES) dan Algoritma Enkripsi Data Internasional (IDEA) yang sudah tua.

Blowfish secara signifikan lebih cepat daripada DES dan IDEA dan tidak dipatenkan serta tersedia gratis untuk semua penggunaan. Namun, ia tidak dapat sepenuhnya menggantikan DES karena ukuran bloknya yang kecil, sehingga dianggap tidak aman.

Twofish, penggantinya, mengatasi masalah keamanan dengan ukuran blok yang lebih besar yaitu 128 bit. Meskipun demikian, enkripsi penuh Blowfish tidak pernah rusak, dan algoritma ini disertakan dalam banyak *cipher suite* dan produk enkripsi yang tersedia saat ini. [6]

III. METODOLOGI PENELITIAN

Dalam penelitian ini, yaitu menganalisis penerapan *steganographic watermarking* dan tanda tangan digital pada video serta menentukan teknik terbaik yang dapat digunakan, penelitian ini memiliki beberapa tahapan. Metodologi yang digunakan mencakup studi literatur, eksperimen, analisis komparatif, dan evaluasi hasil. Berikut adalah penjelasan lebih rinci mengenai tahapan-tahapan tersebut.

A. Studi Literatur

Tahap pertama dari penelitian ini adalah melakukan studi literatur. Studi literatur ini bertujuan untuk memahami konsep dasar, teknik-teknik yang ada, dan perkembangan terbaru dalam bidang *watermarking* dan tanda tangan digital pada video. Beberapa langkah yang akan dilakukan dalam tahap ini adalah

- mengkaji jurnal-jurnal ilmiah, makalah konferensi, dan buku teks yang relevan dengan topik penelitian;
- mengidentifikasi dan mengklasifikasikan berbagai teknik *watermarking* dan tanda tangan digital yang telah digunakan dalam konteks video; dan
- mengumpulkan informasi tentang teknik dan algoritma yang umum digunakan.

B. Eksperimen

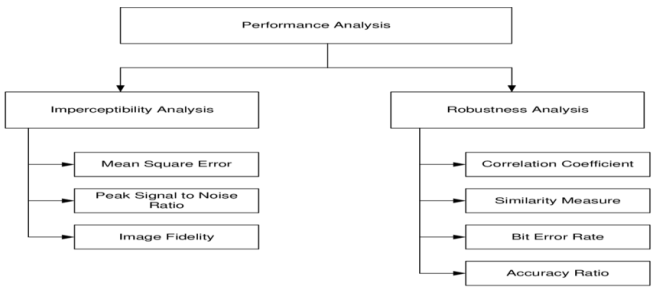
Tahap kedua yang saya ambil adalah eksperimen. Eksperimen ini akan melibatkan penerapan berbagai teknik *steganographic watermarking* dan tanda tangan digital pada video. Langkah-langkah dalam eksperimen ini meliputi:

- Memilih beberapa teknik watermark digital, dan tanda tangan digital yang paling relevan dan menjanjikan berdasarkan hasil studi literatur, ini mencakup algoritma yang saya buat sendiri.
- Mengimplementasikan teknik-teknik terpilih pada video uji, dengan mempertimbangkan parameter-parameter penting seperti kapasitas penyisipan, ketahanan terhadap serangan (misalnya pengeditan), dan dampak pada kualitas video.
- Mencoba beberapa kombinasi algoritma *watermarking* dan tanda tangan digital. Setiap kombinasi dibuat metriknya untuk memperlihatkan perbedaan dan dampak diadakan kombinasi tersebut

IV. HASIL DAN PEMBAHASAN

A. Pendalaman Materi

Dalam mempertimbangkan teknik dan algoritma dari *watermarking* dan tanda tangan digital, beberapa aspek perlu diperhatikan untuk menjadi acuan kualitas dari percobaan. Beberapa metrik dapat digunakan untuk mengukur kualitas dari *watermarking*, lihat Gambar 1.



Gbr. 1. Metrik Uji Watermarking. Dikutip dari [7]

Dari beberapa pilihan metrik yang ada tersebut, beberapa metrik akan digunakan. Ada beberapa metrik tambahan yang akan digunakan dalam percobaan. Metrik tersebut antara lain adalah sebagai berikut.

- *Peak Signal-to-Noise* Ratio (PSNR): Mengukur perbedaan sinyal asli dan sinyal yang telah diberi watermark. Nilai yang baik adalah di atas 30dB[8][9]. Semakin besar nilai PSNR, semakin tinggi kualitas ketidakjelasan watermark yang dirasakan penonton.
- *Structural Similarity Index* (SSIM): Mengukur kesamaan struktural antara citra asli dan citra yang diberi watermark. Nilai SSIM memiliki rentang 0 hingga 1. Semakin tinggi nilai SSIM, semakin tinggi kemiripan video yang diberi watermark dengan video asli.

- *Normalized Correlation* (NC): Mengukur seberapa baik watermark dapat diekstrak dari video yang telah diberi watermark. Nilai NC memiliki rentang 0 hingga 1. Nilai NC di atas 0.7 dianggap sangat baik[10].
- *Bit Error Rate* (BER): Mengukur persentase bit yang salah dalam watermark yang diekstrak dengan watermark yang asli. Nilai BER memiliki rentang 0 hingga 1. Nilai BER di bawah 0.2 dianggap sangat baik.
- *Capacity*: Kapasitas pesan/watermark yang dapat disisipkan pada video dalam bit. Tidak ada batan bit terkait kapasitas ini. Nilai kapasitas yang bagus didasarkan pada perbandingan algoritma yang digunakan dari video yang sama.
- *Correlation Coeffitient*: Mengukur nilai korelasi dari video yang telah diberi watermark dan watermark asli yang diterapkan pada video tersebut. Nilai korelasi ini memiliki rentang 0 hingga 1. Nilai korelasi yang bagus bergantung pada perspektif orang tertentu.

Pada tanda tangan digital, kualitas ditentukan oleh algoritma *hashing* dan algoritma enkripsi. Hasil dari tanda tangan digital sendiri tidak disisipkan pada isi video, melainkan dikirimkan dengan cara yang berbeda atau disisipkan pada *header* video. Tanda tangan digital umumnya dilakukan hanya sekali, sehingga performa tidak memengaruhi efisiensi tanda tangan digital secara signifikan

Beberapa aspek yang perlu diperhatikan pada *hashing* adalah adanya *collision*[11] atau seberapa besar panjang pesan hash yang dapat dihasilkan. Beberapa algoritma umum telah diketahui adanya *collision*, sehingga lebih baik penggunaannya dihindarkan. Lihat Gambar 2.

Beberapa aspek yang perlu diperhatikan pada enkripsi nilai *hash* pada tanda tangan digital adalah kematangan dari enkripsi tersebut dan efisiensi enkripsi [12]. Beberapa teknik enkripsi, seperti cipher blok memiliki kematangan yang lebih rendah daripada enkripsi pada umumnya karena pembagian blok enkripsi yang kurang besar, sehingga penggunaan enkripsi yang umum lebih cocok untuk pengiriman atau penyimpanan video secara utuh. Lihat Gambar 3.

hash	year	coll. res.	size (bits)	design	broken?
MD4	1990	64	128	32-bit ARX DM	1995
SHA-0 (SHA)	1993	80	160	32-bit ARX DM	1998
MD5	1993	64	128	32-bit ARX DM	2004
SHA-1	1995	80	160	32-bit ARX DM	2005
SHA-256 (SHA-2)	2002	128	256	32-bit ARX DM	
SHA-384 (SHA-2)	2002	192	384	64-bit ARX DM	
SHA-512 (SHA-2)	2002	256	512	64-bit ARX DM	
SHA-224 (SHA-2)	2008	112	224	32-bit ARX DM	
SHA-512/224	2012	112	224	64-bit ARX DM	
SHA-512/256	2012	128	256	64-bit ARX DM	
SHA3-224	2013	112	224	64-bit Keccak sponge	
SHA3-256	2013	128	256	64-bit Keccak sponge	
SHA3-384	2013	192	384	64-bit Keccak sponge	
SHA3-512	2013	256	512	64-bit Keccak sponge	
SHAKE128	2013	$\leq$128	any	64-bit Keccak sponge	
SHAKE256	2013	$\leq$256	any	64-bit Keccak sponge	

Gbr. 2. Perbandingan Algoritma *Hashing*. Dikutip dari [13]

Name	Maturity	Security	Computing speed	Resource consumption
RSA	High	High	Low	High
DSA	Medium	High	Low	Medium
ECC	Low	High	Medium	Low
RABIN	High	High	Low	High

Gbr. 3. Perbandingan Algoritma Enkripsi. Dikutip dari [12]

Algoritma tanda tangan yang akan diambil dibahas pada subbab-subbab berikutnya.

B. Alternatif Teknik

Beberapa teknik dipertimbangkan untuk mengetahui eksperimen apa yang akan dibutuhkan. Ide-ide teknik dikumpulkan terlebih dahulu lalu dipilih yang paling relevan dan sesuai untuk mencapai tujuan penelitian. Ide dasar dari eksperimen yang dilakukan adalah menerapkan watermark pada seluruh *frame* video, lalu video yang diberi watermark tersebut ditandatangan secara digital. Beberapa ide teknik lain muncul ketika mengetahui masalah dari ide teknik yang sudah ada sebelumnya. Beberapa ide yang dikumpulkan antara lain adalah sebagai berikut.

1) Menerapkan Digital Signature Sebelum Melakukan Watermarking

Pendekatan ini tidak cocok karena *watermarking* sendiri akan mengubah bit dari keseluruhan isi *frame*/video. Dengan perubahan bit dalam video, nilai *hash* dari video akan berubah secara keseluruhan, sehingga hasil tandatangan digital pun akan berubah meskipun watermark telah diekstrak dari video yang telah diberi watermark

2) Menerapkan Digital Signature pada Beberapa Frame

Beberapa *frame* yang tidak diberi watermark ditandatangani. Setiap *hash* dari frame tersebut dienkripsi dan disimpan dalam tempat tertentu seperti *header* dari video. Pendekatan ini cocok untuk kasus sebagian dari video yang dapat dimodifikasi, tetapi membutuhkan tingkat komunikasi yang lebih tinggi karena keharusan dari adanya kesadaran dari penerima dalam mengetahui *frame* mana yang telah ditandatangani. Kasus seperti ini relatif jarang terjadi.

3) Menerapkan Watermarking pada Beberapa Frame

Beberapa *frame* tertentu diberi watermark setiap beberapa *frame* atau interval waktu tertentu. Pendekatan ini dapat menghemat performa dari video dan meningkatkan/menurunkan kepuasan penonton video. Apabila watermark tidak diimplementasikan dengan baik, penonton dapat menyadari dan merasa terganggu oleh watermark yang muncul lalu menghilang setiap beberapa *frame*. Kualitas hasil video terhadap kejelasan watermark yang tampak perlu diperhatikan dalam pendekatan ini.

4) Menerapkan Kunci pada Watermark Sebelum Watermarking

Karena watermark yang diterapkan bersifat steganografis, teknik seperti penerapan watermark berbentuk *noise* memerlukan perlindungan klaim tambahan berupa kunci untuk mengetahui pemilik asli dari video yang telah diberi watermark. Kunci tersebut digunakan sebagai penguat atau *noise* tambahan pada watermark yang akan diterapkan pada video asli.

5) Menggunakan Kunci Simetris pada Watermarking

Pada umumnya, *watermarking* yang membutuhkan kunci menggunakan kunci asimetris. Hal ini bertujuan untuk menyediakan informasi watermark secara publik untuk video yang tersedia secara publik agar penonton dapat melakukan verifikasi tersendiri untuk mengetahui pemilik dari video yang telah diberi watermark tersebut.

Dengan penggunaan kunci simetris, hanya orang tertentu yang memiliki akses dalam mengekstrak atau mendeteksi

keberadaan watermark tersebut dalam video. Pendekatan ini berguna untuk memberikan akses watermark kepada pihak tertentu. Pengiriman kunci dari *watermarking* tersebut juga harus aman, karena akses pada watermark tersebut tidak publik.

C. Pemilihan Teknik

Dari teknik-teknik yang sudah dibahas pada IV.B, beberapa teknik saya pilih untuk diuji. Eksperimen saya lakukan dengan menerapkan kunci pada watermark sebelum *watermarking*. Kunci yang digunakan pada *watermarking* bergantung pada algoritma yang digunakan yang dibahas pada subbab berikutnya.

Watermark akan diterapkan pada semua *frame* dari video. Karena performa *watermarking* memerhatikan waktu *watermarking* per *frame*, untuk menghemat waktu eksperimen, saya membatasi *frame* yang akan diuji sebanyak 100 *frame* per video.

D. Alternatif Algoritma Watermarking

Algoritma yang dipilih dalam eksperimen dibagi menjadi 3 algoritma utama: algoritma *watermarking*, algoritma *hashing*, dan algoritma enkripsi untuk tanda tangan digital. Beberapa algoritma dikombinasikan dengan algoritma yang berbeda. Dalam *watermarking*, ada beberapa jenis algoritma yang dapat digunakan.

1) Spatial Domain Technique

Algoritma *watermarking* yang bekerja di domain spasial biasanya lebih mudah diimplementasikan tetapi kurang tahan terhadap berbagai jenis serangan atau modifikasi. Beberapa contoh teknik ini sebagai berikut.

- *Least Significant Bit* (LSB) *Modification*: Informasi watermark disisipkan ke dalam bit paling tidak signifikan dari piksel gambar atau sampel audio. Metode ini sederhana tetapi sangat rentan terhadap manipulasi.
- *Additive Watermarking*: Informasi watermark ditambahkan langsung ke nilai-nilai piksel gambar. Ini mudah diterapkan tetapi juga mudah terdeteksi dan dihapus.

2) Frequency Domain Techniques

Teknik ini bekerja dengan memodifikasi komponen frekuensi dari media digital. Mereka cenderung lebih tahan terhadap serangan dibandingkan dengan teknik domain spasial.

- *Discrete Fourier Transform* (DFT): Watermark disisipkan ke dalam koefisien Fourier dari gambar atau sinyal.
- *Discrete Cosine Transform* (DCT): Algoritma ini sering digunakan dalam kompresi gambar (seperti JPEG) dan watermark dapat disisipkan ke dalam koefisien DCT.
- *Discrete Wavelet Transform* (DWT): Watermark disisipkan ke dalam koefisien *wavelet* dari gambar, yang memberikan ketahanan yang baik terhadap berbagai jenis serangan dan modifikasi.

3) Spread Spectrum Techniques

Teknik ini bekerja dengan menyebarkan informasi ke dalam rentang frekuensi yang lebih luas daripada yang biasanya dibutuhkan. Teknik ini cenderung lebih tahan terhadap serangan dibandingkan teknik domain frekuensi.

- *Direct Sequence Spread Spectrum* (DSSS): Teknik ini menyebarkan watermark di seluruh spektrum frekuensi dari sinyal host, yang membuatnya lebih tahan terhadap deteksi dan penghapusan.
- *Frequency Hopping Spread Spectrum* (FHSS): Watermark disisipkan dengan mengubah frekuensi sinyal secara berurutan berdasarkan pola hopping yang telah ditentukan.

E. Pemilihan Algoritma Watermarking

Berdasarkan algoritma yang dibahas pada subbab III.D, Algoritma watermarking yang akan diuji pada eksperimen adalah DSSS, DWT, dan DCT. Teknik domain spasial tidak dipilih karena kerentanan watermark terhadap serangan seperti *noise*, rotasi, brightness, dan sebagainya sehingga memberikan nilai *robustness* yang kurang.

Ketahanan serangan pada video yang telah diberi watermark akan dinilai menggunakan metrik SSIM pada video yang telah diserang. Metrik utama yang digunakan untuk video yang belum diserang adalah PSNR dan NC.

F. Pemilihan Algoritma Hashing

Algoritma *hashing* yang akan digunakan adalah algoritma yang tidak memiliki *collision*. Algoritma ini adalah SHA2 dan SHA3. Panjang hasil *hash* tidak diperhatikan dalam eksperimen ini karena kebutuhan panjang hasil *hash* bergantung pada penggunaan video dan bagaimana video tersebut akan didistribusikan.

G. Pemilihan Algoritma Enkripsi Hash

Efisiensi pada algoritma enkripsi tanda tangan digital tidak memberikan dampak yang cukup besar karena panjang pesan yang dienkripsi, yaitu *hash* dari video tersebut. Maka dari itu, lebih baik jika menggunakan algoritma enkripsi yang memiliki kematangan dan keamanan tinggi. Pada eksperimen ini, saya memilih untuk menggunakan RSA.

H. Eksperimen

Eksperimen dilakukan menggunakan Bahasa Python. *Repository* Github saya terkait eksperimen ini dapat dikunjungi pada halaman Github saya[1]. Eksperimen tidak dilakukan dalam lingkungan khusus, dalam artian, tidak dilakukan pada sebuah *virtual machine*.

1) Aset

Watermark dan video yang diproses dalam eksperimen diambil dari sumber luar. Lihat Gambar 4 untuk video dan Gambar 5 untuk watermark.

[1] https://github.com/Asweh-Debrej/digital-signature-and-steganographic-watermark-on-video

Gbr. 4. Tampilan Video yang Digunakan dalam Eksperimen[2]

Gbr. 5. Tampilan Watermark yang Digunakan dalam Eksperimen[3]

2) Tahapan Eksperimen

Eksperimen yang dilakukan memiliki beberapa tahap utama:

a) Mengimpor Aset

Aset diimpor terlebih dahulu untuk mempersiapkan bahan pemrosesan.

b) Penyisipan Watermark

Watermark disisipkan pada video. Teknik *watermarking* seperti DSSS membutuhkan kunci untuk membuat watermark yang berbeda dalam bentuk noise. Dengan ini, watermark eksternal tidak dibutuhkan karena pembuatannya telah dilakukan secara otomatis.

c) Ekstraksi dan Deteksi Watermark

Watermark yang telah disisipkan diekstrak sesuai dengan teknik *watermarking*-nya. Teknik seperti DSSS tidak melakukan ekstraksi, melainkan mendeteksi langsung watermark pada video yang telah diberi watermark.

d) Melakukan Penyerangan

Video yang telah diberi watermark diserang dengan teknik tertentu untuk mendapatkan hasil ekstraksi dan deteksi yang berbeda

e) Mengekspor Hasil

Hasil dari ekstraksi dievaluasi menggunakan metrik-metrik yang telah dibahas sebelumnya. Hasil dari evaluasi termasuk hasil deteksi diekspor dalam bentuk file terpisah.

3) Kode Sumber Algoritma

a) Kelas Hash

```python
from abc import ABC, abstractmethod

class Hash(ABC):
  def __init__(self, name):
    """Inisialisasi objek Hash

    Args:
      name (str): nama hash
    """
    self.name = name

    return self

  @abstractmethod
  def get_hash(self, data):
    """Menghasilkan hash dari data

    Args:
      data (str): data yang akan diambil hashnya

    Returns:
      str: hash dari data
    """
    pass

  @abstractmethod
  def get_file_hash(self, file_path):
    """Menghasilkan hash dari file

    Args:
      file_path (str): path ke file yang akan diambil hashnya

    Returns:
      str: hash dari file
    """
    pass
```

b) Kelas SHA2

```python
# hash/sha2.py

import hashlib
from hash.hash import Hash

"""
Menggunakan SHA-256 untuk membuat hash data
"""
class SHA2(Hash):
  def __init__(self):
    """Inisialisasi objek SHA2
    """
    super().__init__('SHA-256')
    self.hash = hashlib.sha256

  def get_hash(self, data):
    """Menghasilkan hash dari data

    Args:
      data (str): data yang akan diambil hashnya

    Returns:
      str: hash dari data
    """
    return self.hash(data.encode()).hexdigest()

  def get_file_hash(self, file_path):
    """Menghasilkan hash dari file

    Args:
```

[2] Video by Maksim Goncharenok from Pexels: https://www.pexels.com/video/a-woman-walking-in-the-field-of-flowers-5642529/

[3] https://www.pngwing.com/en/free-png-yxnls

```
        file_path (str): path ke file yang akan diambil hashnya

    Returns:
        str: hash dari file
    """
    with open(file_path, 'rb') as file:
        return self.hash(file.read()).hexdigest()
```

c) Kelas SHA3

```python
# hash/sha3.py

import hashlib
from hash.hash import Hash

"""
Menggunakan SHA-3 untuk membuat hash data
"""
class SHA3(Hash):
  def __init__(self):
    """Inisialisasi objek SHA3
    """
    super().__init__('SHA-3')
    self.hash = hashlib.sha3_256

  def get_hash(self, data):
    """Menghasilkan hash dari data

    Args:
      data (str): data yang akan diambil hashnya

    Returns:
      str: hash dari data
    """
    return self.hash(data.encode()).hexdigest()

  def get_file_hash(self, file_path):
    """Menghasilkan hash dari file

    Args:
      file_path (str): path ke file yang akan diambil hashnya

    Returns:
      str: hash dari file
    """
    with open(file_path, 'rb') as file:
      return self.hash(file.read()).hexdigest()
```

d) Kelas Signature

```python
from abc import ABC, abstractmethod

class Signature(ABC):
  def __init__(self, name):
    """Inisialisasi objek Signature

    Args:
      name (str): nama signature
    """
    self.name = name

    return self

  @abstractmethod
  def sign(self, data, private_key):
    """Menandatangani data

    Args:
      data (str): data yang akan ditandatangani
      private_key (str): kunci untuk menandatangani data

    Returns:
      str: tanda tangan digital dari data
    """
    pass

  @abstractmethod
  def verify(self, data, signature, public_key):
    """Memverifikasi tanda tangan digital

    Args:
      data (str): data yang akan diverifikasi
      signature (str): tanda tangan digital
      public_key (str): kunci publik untuk verifikasi

    Returns:
      bool: hasil verifikasi tanda tangan digital
    """
    pass
```

e) Kelas RSA

```python
# signature/sha2.py

"""
Menggunakan RSA-2048 untuk enkripsi digital signature
"""

from Cryptodome.PublicKey import RSA
from Cryptodome.Cipher import PKCS1_OAEP
from signature.signature import Signature

class RSA2048(Signature):
  def __init__(self):
    """Inisialisasi objek RSA2048
    """
    super().__init__('RSA-2048')
    self.key = RSA.generate(2048)
    self.public_key = self.key.export_key()
    self.private_key = self.key.publickey().export_key()

  def sign(self, data, private_key):
    """Mengenkripsi data dengan RSA-2048

    Args:
      data (str): data yang akan dienkripsi
      private_key (str): kunci privat enkripsi, tidak akan digunakan

    Returns:
      bytes: data yang telah dienkripsi
    """
    cipher_rsa = PKCS1_OAEP.new(self.key)
    return cipher_rsa.encrypt(data.encode())

  def verify(self, data):
    """Mendekripsi data dengan RSA-2048

    Args:
      data (bytes): data yang akan didekripsi

    Returns:
      str: data yang telah didekripsi
    """
    cipher_rsa = PKCS1_OAEP.new(self.key)
    return cipher_rsa.decrypt(data).decode()
```

f) Kelas Watermark

```python
from abc import ABC, abstractmethod

class Watermark:
    def __init__(self, name):
        """Inisialisasi objek Watermark

        Args:
            name (str): nama watermark
        """
        self.name = name

        return self

    @abstractmethod
    def embed(self, image, watermark, key):
        """Menyisipkan watermark ke dalam citra

        Args:
            image (numpy.ndarray): citra asli
            watermark (numpy.ndarray): watermark yang akan disisipkan
            key (str): kunci untuk menentukan posisi penyisipan watermark

        Returns:
            numpy.ndarray: citra hasil watermarking
        """
        pass

    @abstractmethod
    def extract(self, watermarked, original, key):
        """Mengekstrak watermark dari citra

        Args:
            watermarked (numpy.ndarray): citra hasil watermarking
            original (numpy.ndarray): citra asli
            key (str): kunci untuk menentukan posisi penyisipan watermark

        Returns:
            numpy.ndarray: watermark hasil ekstraksi
        """
        pass
```

g) Kelas DSSS

```python
# watermark/dsss.py

import numpy as np
import cv2
import os

from watermark.watermark import Watermark
from scipy.ndimage import gaussian_filter
from scipy.signal import fftconvolve

class DSSS(Watermark):
    def __init__(self, k=7):
        """Inisialisasi objek DSSS

        Args:
            k (float, optional): scaling factor untuk watermark. Defaults to 0.005.
        """
        Watermark.__init__(self, 'DSSS')
        self.k = k # Penguat watermark

    def _enhance(self, image):
        """Meningkatkan citra menggunakan gaussian filter

        Args:
            image (numpy.ndarray): citra yang akan ditingkatkan

        Returns:
            numpy.ndarray: citra yang telah ditingkatkan
        """
        # Membuat kernel filter
        kernel = np.array([[-1, -1, -1],
                           [-1,  8, -1],
                           [-1, -1, -1]])

        # Menerapkan filter
        filtered = cv2.filter2D(image.astype(np.int16), -1, kernel)
        return filtered

    def generate_pseudo_random_sequence(self, shape, key):
        """Menghasilkan urutan pseudorandom

        Args:
            shape (tuple): bentuk urutan yang dihasilkan
            key (str): kunci untuk menentukan urutan pseudorandom

        Returns:
            numpy.ndarray: urutan pseudorandom
        """

        np.random.seed(int.from_bytes(key.encode(), 'little'))
        pseudo_random_sequence = np.random.choice([1, -1], size=shape)
        return pseudo_random_sequence.astype(np.int16)

    def _detect_single_channel(self, watermarked, key):
        """Mendeteksi watermark pada citra pada satu channel

        Args:
            watermarked (numpy.ndarray): citra hasil watermarking
            key (str): kunci untuk menentukan urutan pseudorandom

        Returns:
            int: nilai korelasi antara watermark yang diekstrak dan watermark asli
        """
        # Generate pseudorandom sequence
        pseudo_random_sequence = self.generate_pseudo_random_sequence(
            watermarked.shape, key)

        # Enhance using gaussian filter
        watermarked = self._enhance(watermarked)

        result = cv2.matchTemplate(watermarked.astype(np.float32), pseudo_random_sequence.astype(np.float32), cv2.TM_CCORR_NORMED)

        _, _, _, max_loc = cv2.minMaxLoc(result)

```

```
        correlation = result[max_loc[1], max_loc[0]]

        return correlation

    def _embed_single_channel(self, image, key):
        """Menyisipkan watermark ke dalam citra pada satu channel

        Args:
          image (numpy.ndarray): citra asli
          watermark (numpy.ndarray): watermark yang akan disisipkan

        Returns:
          numpy.ndarray: citra hasil watermarking
        """

        # Generate pseudorandom sequence
        pseudo_random_sequence = self.generate_pseudo_random_sequence(
            image.shape, key)

        # Embed the watermark using DSSS
        watermarked_image = cv2.add(image.astype(np.int16), self.k * pseudo_random_sequence)

        # Convert below 0 to 0 and above 255 to 255
        watermarked_image = np.clip(watermarked_image, 0, 255).astype(np.uint8)

        return watermarked_image

    def embed(self, image, watermark, key):
        """Menyisipkan watermark ke dalam citra

        Metode ini akan menyisipkan watermark ke dalam citra menggunakan LSB

        Args:
          image (numpy.ndarray): citra asli
          watermark (numpy.ndarray): watermark yang akan disisipkan
          key (str): kunci untuk menentukan posisi penyisipan watermark

        Returns:
          numpy.ndarray: citra hasil watermarking
        """
        # Check if the image is grayscale
        if len(image.shape) == 2:
            return self._embed_single_channel(image, key)

        # Embed watermark in each channel
        watermarked_image = np.zeros_like(image)
        for i in range(image.shape[-1]):
            watermarked_image[..., i] = self._embed_single_channel(
                image[..., i], key)

        return watermarked_image

    def detect(self, watermarked, key):
        """Mendeteksi watermark pada citra

        Metode ini akan mendeteksi watermark pada citra menggunakan DSSS

        Args:
          watermarked (numpy.ndarray): citra hasil watermarking
          key (str): kunci untuk menentukan urutan pseudorandom

        Returns:
          int: nilai korelasi antara watermark yang diekstrak dan watermark asli
        """
        # Check if the image is grayscale
        if len(watermarked.shape) == 2:
            return self._detect_single_channel(watermarked, key)

        # Detect watermark in each channel
        correlations = []
        for i in range(watermarked.shape[-1]):
            correlations.append(self._detect_single_channel(
                watermarked[..., i], key))

        return correlations
```

h) Kelas DCT

```
# watermark/dct.py

import numpy as np
import cv2
from watermark.watermark import Watermark
from scipy.ndimage import gaussian_filter
from scipy.fftpack import dct, idct

class DCT(Watermark):
    def __init__(self, k=0.006):
        """Inisialisasi objek DCT

        Args:
          k (int, optional): scaling factor untuk watermark. Defaults to 2.
        """
        Watermark.__init__(self, 'DCT')
        self.k = k

    def _embed_single_channel(self, image, watermark):
        """Menyisipkan watermark ke dalam citra pada satu channel

        Args:
          image (numpy.ndarray): citra asli
          watermark (numpy.ndarray): watermark yang akan disisipkan

        Returns:
          numpy.ndarray: citra hasil watermarking
        """
        # Apply DCT to the image
        image_dct = dct(dct(image.T, norm='ortho').T, norm='ortho')
        watermark_dct = dct(dct(watermark.T, norm='ortho').T, norm='ortho')

        # Embed the watermark in the DCT coefficients
        image_dct += self.k * watermark_dct
```

```

        watermarked_image = idct(
            idct(image_dct.T, norm='ortho').T,
norm='ortho')

        return watermarked_image

    def _extract_single_channel(self,
watermarked, original):
        """Mengekstrak watermark dari citra
pada satu channel

        Args:
          watermarked (numpy.ndarray): citra
hasil watermarking
          original (numpy.ndarray): citra asli

        Returns:
          numpy.ndarray: watermark yang
diekstrak
        """
        # Apply DCT to the images
        watermarked_dct =
dct(dct(watermarked.T, norm='ortho').T,
norm='ortho')
        original_dct = dct(dct(original.T,
norm='ortho').T, norm='ortho')

        # Extract the watermark from the DCT
coefficients
        watermark_dct = (watermarked_dct -
original_dct) / self.k

        watermark = idct(idct(watermark_dct.T,
norm='ortho').T, norm='ortho')

        return watermark

    def embed(self, image, watermark,
key=None):
        """Menyisipkan watermark ke dalam citra

        Args:
          image (numpy.ndarray): citra asli
          watermark (numpy.ndarray): watermark
yang akan disisipkan

        Returns:
          numpy.ndarray: citra hasil
watermarking
        """
        # Check if the image is grayscale
        if len(image.shape) == 2:
            return
self._embed_single_channel(image, watermark)

        # Embed watermark in each channel
        watermarked_image =
np.zeros_like(image)
        for i in range(image.shape[-1]):
            watermarked_image[..., i] =
self._embed_single_channel(image[..., i],
watermark[..., i])

        return watermarked_image

    def extract(self, watermarked, original,
key=None):
        """Mengekstrak watermark dari citra

        Args:
          watermarked (numpy.ndarray): citra
hasil watermarking
          original (numpy.ndarray): citra asli

        Returns:
          numpy.ndarray: watermark yang
diekstrak
        """
        # Check if the image is grayscale
        if len(watermarked.shape) == 2:
            return
self._extract_single_channel(watermarked, original)

        # Extract watermark from each channel
        watermark = np.zeros_like(watermarked)
        for i in range(watermarked.shape[-1]):
            watermark[..., i] =
self._extract_single_channel(watermarked[..., i],
original[..., i])

        return watermark
```

i) Kelas DWT

```
# watermark/dwt.py

import numpy as np
import pywt
import cv2
from watermark.watermark import Watermark
from scipy.ndimage import gaussian_filter

"""Kelas untuk melakukan Discrete Wavelet
Transform
Kelas memiliki beberapa metode untuk melakukan
DWT pada citra
Watermark disisipkan dalam sub-band tertentu
menggunakan kunci untuk menentukan posisi yang tepat
"""

class DWT(Watermark):
    def __init__(self, wavelet='haar',
mode='symmetric', k = 0.03):
        """Inisialisasi objek DWT

        Args:
          wavelet (str, optional): nama
wavelet. Defaults to 'haar'.
          mode (str, optional): mode. Defaults
to 'symmetric'.
        """
        Watermark.__init__(self, 'DWT')
        self.wavelet = wavelet
        self.mode = mode
        self.k = k # alpha

    def _embed_single_channel(self, image,
watermark):
        """Menyisipkan watermark ke dalam citra
pada satu channel

        Args:
          image (numpy.ndarray): citra asli
          watermark (numpy.ndarray): watermark
yang akan disisipkan
          key (str): kunci untuk menentukan
posisi penyisipan watermark

        Returns:
          numpy.ndarray: citra hasil
watermarking
        """
```

```
        # Apply DWT to the image
        coeffs_image = pywt.dwt2(image, self.wavelet, mode=self.mode)
        LL_image, (LH_image, HL_image, HH_image) = coeffs_image

        coeffs_watermark = pywt.dwt2(watermark, self.wavelet, mode=self.mode)
        LL_watermark, (LH_watermark, HL_watermark, HH_watermark) = coeffs_watermark

        # Embed the watermark in the DWT coefficients
        LL_watermarked = LL_image + self.k * LL_watermark
        LH_watermarked = LH_image + self.k * LH_watermark
        HL_watermarked = HL_image + self.k * HL_watermark
        HH_watermarked = HH_image + self.k * HH_watermark

        coeffs_watermarked = LL_watermarked, (LH_watermarked, HL_watermarked, HH_watermarked)

        watermarked_image = pywt.idwt2(coeffs_watermarked, self.wavelet, mode=self.mode)

        return watermarked_image

    def _extract_single_channel(self, watermarked, original):
        """Mengekstrak watermark dari citra pada satu channel

        Args:
          image (numpy.ndarray): citra hasil watermarking

        Returns:
          numpy.ndarray: watermark yang diekstrak
        """
        # Apply DWT to the watermarked image
        coeffs_watermarked = pywt.dwt2(watermarked, self.wavelet, mode=self.mode)
        LL_watermarked, (LH_watermarked, HL_watermarked, HH_watermarked) = coeffs_watermarked

        coeffs_original = pywt.dwt2(original, self.wavelet, mode=self.mode)
        LL_original, (LH_original, HL_original, HH_original) = coeffs_original

        # Extract the watermark from the DWT coefficients
        LL_extracted = (LL_watermarked - LL_original) / self.k
        LH_extracted = (LH_watermarked - LH_original) / self.k
        HL_extracted = (HL_watermarked - HL_original) / self.k
        HH_extracted = (HH_watermarked - HH_original) / self.k

        coeffs_extracted = LL_extracted, (LH_extracted, HL_extracted, HH_extracted)

        extracted_watermark = pywt.idwt2(coeffs_extracted, self.wavelet, mode=self.mode)

        return extracted_watermark

    def embed(self, image, watermark, key):
        """Menyisipkan watermark ke dalam citra
        Metode ini akan menyisipkan watermark ke dalam citra menggunakan DWT

        Args:
          image (numpy.ndarray): citra asli
          watermark (numpy.ndarray): watermark yang akan disisipkan
          key (str): kunci untuk menentukan posisi penyisipan watermark

        Returns:
          numpy.ndarray: citra hasil watermarking
        """
        # Check if the image is grayscale
        if len(image.shape) == 2:
            return self._embed_single_channel(image, watermark)

        # Embed watermark in each channel
        watermarked_image = np.zeros_like(image)
        for i in range(image.shape[-1]):
            watermarked_image[..., i] = self._embed_single_channel(image[..., i], watermark[..., i])

        return watermarked_image

    def extract(self, watermarked, original, key):
        """Mengekstrak watermark dari citra
        Metode ini akan mengekstrak watermark dari citra menggunakan DWT

        Args:
          watermarked_image (numpy.ndarray): citra hasil watermarking
          key (str): kunci untuk menentukan posisi penyisipan watermark

        Returns:
          numpy.ndarray: watermark hasil ekstraksi
        """
        # Check if the image is grayscale
        if len(watermarked.shape) == 2:
            return self._extract_single_channel(watermarked, original)

        # Extract watermark from each channel
        extracted_watermark = np.zeros_like(watermarked)
        for i in range(watermarked.shape[-1]):
            extracted_watermark[..., i] = self._extract_single_channel(watermarked[..., i], original[..., i])

        return extracted_watermark
```

j) Kelas Program Utama

```
# main.py

```

```
  3. """Program utama
  4. Program utama untuk melakukan serangkaian
proses pada video.
  5.
  6. Program menjalankan fungsi-fungsi yang
terdefinisi dari beberapa algoritma watermarking dan
digital signature.
  7. Urutan proses utama yang dilakukan adalah:
  8. 1. Menerapkan watermarking pada video,
watermark secara opsional dapat dibuat oleh
algoritma terpilih
  9. 2. Menghasilkan digital signature dari video
yang telah di-watermark
 10. 3. Mengenkripsi digital signature dengan kunci
tertentu
 11. 4. Menyimpan hasil watermarking dan digital
signature ke dalam file terpisah
 12. 5. Membaca hasil watermarking dan digital
signature dari file
 13. 6. Mendekripsi digital signature
 14. 7. Memverifikasi digital signature dengan video
yang telah di-watermark
 15. 8. Memberikan evaluasi hasil verifikasi
 16. 9. Menampilkan hasil evaluasi
 17. """
 18.
 19. from skimage.metrics import
peak_signal_noise_ratio as psnr,
structural_similarity as ssim
 20. import time
 21. import numpy as np
 22. import cv2
 23. import json
 24. import os
 25. from watermark.watermark import Watermark
 26. from watermark.dwt import DWT
 27. from watermark.dsss import DSSS
 28. from watermark.dct import DCT
 29. from signature.signature import Signature
 30. from signature.rsa import RSA2048
 31. from hash.hash import Hash
 32. from hash.sha2 import SHA2
 33. from hash.sha3 import SHA3
 34.
 35. watermarks = [
 36.     DSSS(),
 37.     DWT(),
 38.     DCT()
 39. ]
 40.
 41. signatures = [
 42.     RSA2048(),
 43. ]
 44.
 45. hashes = [
 46.     SHA2(),
 47.     SHA3()
 48. ]
 49.
 50. FRAME_COUNT = 100
 51.
 52. def calculate_psnr(original, watermarked):
 53.     """Menghitung nilai PSNR antara dua citra
 54.
 55.     Args:
 56.       original (numpy.ndarray): citra asli
 57.       watermarked (numpy.ndarray): citra hasil
watermarking
 58.
 59.     Returns:
 60.       float: nilai PSNR
 61.     """
 62.     return psnr(original, watermarked)
 63.
 64. def calculate_ssim(original, watermarked):
 65.     """Menghitung nilai SSIM antara dua citra
 66.
 67.     Args:
 68.       original (numpy.ndarray): citra asli
 69.       watermarked (numpy.ndarray): citra hasil
watermarking
 70.
 71.     Returns:
 72.       float: nilai SSIM
 73.     """
 74.     multichannel = len(original.shape) == 3
 75.     channel_axis = 2 if multichannel else None
 76.     return ssim(original, watermarked,
multichannel=multichannel,
channel_axis=channel_axis,
data_range=watermarked.max() - watermarked.min())
 77.
 78. def calculate_nc(original_watermark,
extracted_watermark):
 79.     """Menghitung nilai NC antara dua watermark
 80.
 81.     Args:
 82.       original_watermark (numpy.ndarray):
watermark asli
 83.       extracted_watermark (numpy.ndarray):
watermark hasil ekstraksi
 84.
 85.     Returns:
 86.       float: nilai NC
 87.     """
 88.     return
np.corrcoef(original_watermark.flatten(),
extracted_watermark.flatten())[0, 1]
 89.
 90. def calculate_ber(original_watermark,
extracted_watermark):
 91.     """Menghitung nilai BER antara dua
watermark
 92.
 93.     Args:
 94.       original_watermark (numpy.ndarray):
watermark asli
 95.       extracted_watermark (numpy.ndarray):
watermark hasil ekstraksi
 96.
 97.     Returns:
 98.       float: nilai BER
 99.     """
100.     # menggunakan numpy unpackbits untuk
mendapatkan nilai bit dari watermark
101.     original_bits =
np.unpackbits(original_watermark.flatten())
102.     extracted_bits =
np.unpackbits(extracted_watermark.flatten())
103.
104.     # menghitung jumlah bit yang berbeda
105.     return np.sum(original_bits !=
extracted_bits) / len(original_bits)
106.
107. def calculate_capacity(video_path):
108.     """Menghitung kapasitas watermarking pada
video
109.
110.     Args:
111.       video_path (str): path ke video yang akan
dihitung kapasitasnya
112.
```

```
    Returns:
      int: kapasitas watermarking
    """
    video = cv2.VideoCapture(video_path)
    frame_count =
int(video.get(cv2.CAP_PROP_FRAME_COUNT))
    width =
int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
    height =
int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
    return frame_count * width * height

def apply_attack(video_path, attack_type,
output_path):
    """Menerapkan serangan pada video

    Args:
      video_path (str): path ke video yang akan
diserang
      attack_type (str): tipe serangan yang
akan diterapkan
      output_path (str): path ke video hasil
serangan

    Returns:
      None
    """
    cap = cv2.VideoCapture(video_path)
    fourcc = cv2.VideoWriter_fourcc('m', 'p',
'4', 'v')

    frame_count =
int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    width =
int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height =
int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = int(cap.get(cv2.CAP_PROP_FPS))

    out = cv2.VideoWriter(output_path, fourcc,
fps, (width, height))

    for i in range(frame_count):
        ret, frame = cap.read()
        if not ret:
            break

        if attack_type == 'noise':
            noise = np.random.normal(0, 10,
frame.shape).astype(np.uint8)
            frame = cv2.add(frame, noise,
dtype=cv2.CV_8UC3)
        elif attack_type == 'brightness':
            frame = cv2.convertScaleAbs(frame,
alpha=1.5, beta=0)
        elif attack_type == 'contrast':
            frame = cv2.convertScaleAbs(frame,
alpha=1, beta=50)
        elif attack_type == 'rotation':
            rows, cols, _ = frame.shape
            M = cv2.getRotationMatrix2D((cols /
2, rows / 2), 1, 1)
            frame = cv2.warpAffine(frame, M,
(cols, rows))
        elif attack_type == 'compression':
            encode_param =
[int(cv2.IMWRITE_JPEG_QUALITY), 50]
            _, buffer = cv2.imencode('.jpg',
frame, encode_param)
            frame = cv2.imdecode(buffer,
cv2.IMREAD_COLOR)

        # printout first frame as thumbnail
        if i == 0:
            cv2.imwrite(f'{output_path[:-
4]}.png', frame)

        out.write(frame)

    cap.release()
    out.release()

def evaluate_watermarking(watermarking:
Watermark, frames, watermark, watermarking_key,
frame_count=FRAME_COUNT, width=0, height=0, fps=0,
folder=''):
    """Mengevaluasi hasil watermarking pada
video

    Args:
      Watermark_class (object): kelas algoritma
watermarking
      video_path (str): path ke video yang akan
di-watermark
      watermark_path (str): path ke watermark
yang akan di-embed
      watermarking_key (str): kunci
watermarking

    Returns:
      dict: hasil evaluasi dengan format
{'psnr': float, 'ssim': float, 'nc': float, 'ber':
float, 'capacity': int, 'time': float, 'attack':
{{'psnr': float, 'ssim': float, 'nc': float, 'ber':
float}}
    """
    print(f'Evaluating {watermarking.name}')

    # output frames dan watermark original
    original_frame_path =
f'{folder}/original_frames.mp4'
    os.makedirs(os.path.dirname(original_frame_
path), exist_ok=True)
    fourcc = cv2.VideoWriter_fourcc('m', 'p',
'4', 'v')
    out = cv2.VideoWriter(original_frame_path,
fourcc, fps, (width, height))
    for frame in frames:
        out.write(frame)
    out.release()

    cv2.imwrite(f'{folder}/original_thumbnail.p
ng', frames[0])

    original_watermark_path =
f'{folder}/original_watermark.png'
    os.makedirs(os.path.dirname(original_waterm
ark_path), exist_ok=True)
    cv2.imwrite(original_watermark_path,
watermark)

    # Menerapkan watermarking
    watermarked_frames = []
    start_time = time.time()

    print('Embedding watermark')
    print('_'*FRAME_COUNT)
    for i in range(FRAME_COUNT):
        watermarked_frame = watermarking.embed(
            frames[i], watermark,
watermarking_key)
```

```
            watermarked_frames.append(watermarked_f
rame)
            print('', end='', flush=True)

        print('\tDone!')
        print('¯'*FRAME_COUNT)

        # Menghitung waktu watermarking
        watermarking_time = time.time() -
start_time

        # Membuat video hasil watermarking
        watermarking_path =
f'{folder}/watermarked_video.mp4'
        os.makedirs(os.path.dirname(watermarking_pa
th), exist_ok=True)
        fourcc = cv2.VideoWriter_fourcc('m', 'p',
'4', 'v')
        out = cv2.VideoWriter(watermarking_path,
fourcc, fps, (width, height))
        for frame in watermarked_frames:
            out.write(frame)
        out.release()

        # printout first frame as thumbnail
        cv2.imwrite(f'{folder}/watermarked_thumbnai
l.png', watermarked_frames[0])

        # Menghitung kapasitas watermarking
        # capacity =
watermarking.calculate_capacity(video_path)

        # ekstrak watermark
        extracted_watermarks = []

        print('Extracting watermark')
        print('_'*FRAME_COUNT)
        for i in range(FRAME_COUNT):
            extracted_watermark =
watermarking.extract(
                watermarked_frames[i], frames[i],
watermarking_key)
            extracted_watermarks.append(extracted_w
atermark)
            print('', end='', flush=True)

        print('\tDone!')
        print('¯'*FRAME_COUNT)

        extracted_watermark_path =
f'{folder}/extracted_watermark.png'
        os.makedirs(os.path.dirname(extracted_water
mark_path), exist_ok=True)
        cv2.imwrite(extracted_watermark_path,
extracted_watermarks[0])

        # Menghitung PSNR dan SSIM
        psnr_values = []
        ssim_values = []

        print('Calculating PSNR and SSIM')
        print('_'*FRAME_COUNT)
        for i in range(FRAME_COUNT):
            psnr_value = calculate_psnr(frames[i],
watermarked_frames[i])
            ssim_value = calculate_ssim(frames[i],
watermarked_frames[i])
            psnr_values.append(psnr_value)
            ssim_values.append(ssim_value)
            print('', end='', flush=True)

        print('\tDone!')
        print('¯'*FRAME_COUNT)

        nc_values = []
        ber_values = []

        print('Calculating NC and BER')
        print('_'*FRAME_COUNT)
        for i in range(FRAME_COUNT):
            nc_value = calculate_nc(watermark,
extracted_watermarks[i])
            ber_value = calculate_ber(watermark,
extracted_watermarks[i])
            nc_values.append(nc_value)
            ber_values.append(ber_value)
            print('', end='', flush=True)

        print('\tDone!')
        print('¯'*FRAME_COUNT)

        # Menerapkan serangan pada video hasil
watermarking
        print('Applying attacks...')
        attacks = [
            'noise',
            'brightness',
            'contrast',
            'rotation',
            'compression'
        ]
        attack_results = {}
        for attack in attacks:
            print(f'Applying {attack} attack...')
            attacked_video_path =
f'{folder}/{attack}_attack.mp4'
            os.makedirs(os.path.dirname(attacked_vi
deo_path), exist_ok=True)
            apply_attack(watermarking_path, attack,
attacked_video_path)

            # Membaca video hasil serangan
            attacked_video =
cv2.VideoCapture(attacked_video_path)
            attacked_frames = []

            print('Reading attacked video')
            print('_'*FRAME_COUNT)
            for _ in range(FRAME_COUNT):
                ret, frame = attacked_video.read()
                if not ret:
                    break

                attacked_frames.append(frame)
                print('', end='', flush=True)

            print('\tDone!')
            print('¯'*FRAME_COUNT)

            # Menghitung PSNR dan SSIM
            attacked_psnr_values = []
            attacked_ssim_values = []

            print('Calculating PSNR and SSIM')
            print('_'*FRAME_COUNT)
            for i in range(FRAME_COUNT):
                psnr_value = calculate_psnr(
                    attacked_frames[i],
watermarked_frames[i])
                ssim_value = calculate_ssim(
                    attacked_frames[i],
watermarked_frames[i])
```

```
            attacked_psnr_values.append(psnr_value)
            attacked_ssim_values.append(ssim_value)
            print('▉', end='', flush=True)

        print('\tDone!')
        print('‾'*FRAME_COUNT)

        # Menghitung NC dan BER
        extracted_watermarks = []

        print('Extracting watermark')
        print('_'*FRAME_COUNT)
        for i in range(FRAME_COUNT):
            extracted_watermark = watermarking.extract(
                attacked_frames[i], frames[i], watermarking_key)
            extracted_watermarks.append(extracted_watermark)
            print('▉', end='', flush=True)

        print('\tDone!')
        print('‾'*FRAME_COUNT)

        extracted_attacked_watermark_path = f'{folder}/{attack}_extracted_watermark.png'
        os.makedirs(os.path.dirname(extracted_attacked_watermark_path), exist_ok=True)
        cv2.imwrite(extracted_attacked_watermark_path, extracted_watermarks[0])

        attacked_nc_values = []
        attacked_ber_values = []

        print('Calculating NC and BER')
        print('_'*FRAME_COUNT)
        for i in range(FRAME_COUNT):
            nc_value = calculate_nc(watermark, extracted_watermarks[i])
            ber_value = calculate_ber(watermark, extracted_watermarks[i])
            attacked_nc_values.append(nc_value)
            attacked_ber_values.append(ber_value)
            print('▉', end='', flush=True)

        print('\tDone!')
        print('‾'*FRAME_COUNT)

        attack_results[attack] = {
            'psnr': sum(attacked_psnr_values) / len(attacked_psnr_values),
            'ssim': sum(attacked_ssim_values) / len(attacked_ssim_values),
            'nc': sum(attacked_nc_values) / len(attacked_nc_values),
            'ber': sum(attacked_ber_values) / len(attacked_ber_values),
        }

        print(f'{attack} done')

    # Menyimpan hasil watermarking
    results = {
        'name': f'{watermarking.name}',
        'time': watermarking_time,
        'time per frame': watermarking_time / FRAME_COUNT,
        'psnr': sum(psnr_values) / len(psnr_values),
        'ssim': sum(ssim_values) / len(ssim_values),
        'nc': sum(nc_values) / len(nc_values),
        'ber': sum(ber_values) / len(ber_values),
        # 'capacity': capacity,
        'attack': attack_results,
    }

    return results

def evaluate_dsss(frames, watermarking: DSSS, watermarking_key, frame_count=FRAME_COUNT, width=0, height=0, fps=0, folder=''):
    """Mengevaluasi hasil watermarking pada video, khusus untuk DSSS

    Args:
        frames (list): list of frames
        watermark (numpy.ndarray): watermark
        watermarking_key (str): kunci watermarking
        frame_count (int): jumlah frame yang akan di-watermark
        width (int): lebar frame
        height (int): tinggi frame
        fps (int): frame per second
        folder (str): folder untuk menyimpan hasil evaluasi

    Returns:
        dict: hasil evaluasi
    """
    print(f'Evaluating DSSS')

    # output frames dan watermark original
    original_frame_path = f'{folder}/original_frames.mp4'
    os.makedirs(os.path.dirname(original_frame_path), exist_ok=True)
    fourcc = cv2.VideoWriter_fourcc('m', 'p', '4', 'v')
    out = cv2.VideoWriter(original_frame_path, fourcc, fps, (width, height))
    for frame in frames:
        out.write(frame)
    out.release()

    # printout first frame as thumbnail
    cv2.imwrite(f'{folder}/original_thumbnail.png', frames[0])

    watermark = np.zeros_like(frames[0])
    for i in range(frames[0].shape[-1]):
        watermark[..., i] = watermarking.generate_pseudo_random_sequence(
            (height, width), watermarking_key)

    watermark_path = f'{folder}/watermark.png'
    os.makedirs(os.path.dirname(watermark_path), exist_ok=True)
    cv2.imwrite(watermark_path, watermark)

    # Menerapkan watermarking
    watermarked_frames = []
    start_time = time.time()

    print('Embedding watermark')
    print('_'*FRAME_COUNT)
```

```
    for i in range(FRAME_COUNT):
        watermarked_frame = watermarking.embed(
            frames[i], watermark,
watermarking_key)
        watermarked_frames.append(watermarked_f
rame)
        print('', end='', flush=True)

    print('\tDone!')
    print('¯'*FRAME_COUNT)

    # Menghitung waktu watermarking
    watermarking_time = time.time() -
start_time

    # Membuat video hasil watermarking
    watermarking_path =
f'{folder}/watermarked_video.mp4'
    os.makedirs(os.path.dirname(watermarking_pa
th), exist_ok=True)
    fourcc = cv2.VideoWriter_fourcc('m', 'p',
'4', 'v')
    out = cv2.VideoWriter(watermarking_path,
fourcc, fps, (width, height))
    for frame in watermarked_frames:
        out.write(frame)
    out.release()

    # printout first frame as thumbnail
    cv2.imwrite(f'{folder}/watermarked_thumbnai
l.png', watermarked_frames[0])

    print('Calculating PSNR and SSIM')
    print('_'*FRAME_COUNT)
    psnr_values = []
    ssim_values = []
    for i in range(FRAME_COUNT):
        psnr_value = calculate_psnr(frames[i],
watermarked_frames[i])
        ssim_value = calculate_ssim(frames[i],
watermarked_frames[i])
        psnr_values.append(psnr_value)
        ssim_values.append(ssim_value)
        print('', end='', flush=True)

    print('\tDone!')
    print('¯'*FRAME_COUNT)

    # Menghitung kapasitas watermarking
    # capacity =
watermarking.calculate_capacity(video_path)

    # Ambil nilai korelasi watermark dengan
video
    print('Detecting watermark')
    print('_'*FRAME_COUNT)
    correlations = []
    for i in range(FRAME_COUNT):
        correlation = watermarking.detect(
            watermarked_frames[i],
watermarking_key)
        correlation_value = sum(correlation) /
len(correlation)
        correlations.append(correlation_value)
        print('', end='', flush=True)

    print('\tDone!')
    print('¯'*FRAME_COUNT)

    # Menerapkan serangan pada video hasil
watermarking
    print('Applying attacks...')
    attacks = [
        'noise',
        'brightness',
        'contrast',
        'rotation',
        'compression'
    ]
    attack_results = {}
    for attack in attacks:
        print(f'Applying {attack} attack...')
        attacked_video_path =
f'{folder}/{attack}_attack.mp4'
        os.makedirs(os.path.dirname(attacked_vi
deo_path), exist_ok=True)
        apply_attack(watermarking_path, attack,
attacked_video_path)

        # Membaca video hasil serangan
        attacked_video =
cv2.VideoCapture(attacked_video_path)
        attacked_frames = []

        print('Reading attacked video')
        print('_'*FRAME_COUNT)
        for _ in range(FRAME_COUNT):
            ret, frame = attacked_video.read()
            if not ret:
                break

            attacked_frames.append(frame)
            print('', end='', flush=True)

        print('\tDone!')
        print('¯'*FRAME_COUNT)

        print('Calculating PSNR and SSIM')
        print('_'*FRAME_COUNT)
        attacked_psnr_values = []
        attacked_ssim_values = []
        for i in range(FRAME_COUNT):
            psnr_value = calculate_psnr(
                attacked_frames[i],
watermarked_frames[i])
            ssim_value = calculate_ssim(
                attacked_frames[i],
watermarked_frames[i])
            attacked_psnr_values.append(psnr_va
lue)
            attacked_ssim_values.append(ssim_va
lue)
            print('', end='', flush=True)

        print('\tDone!')
        print('¯'*FRAME_COUNT)

        print('Detecting watermark')
        print('_'*FRAME_COUNT)
        attacked_correlations = []
        for i in range(FRAME_COUNT):
            correlation = watermarking.detect(
                attacked_frames[i],
watermarking_key)
            correlation_value =
sum(correlation) / len(correlation)
            attacked_correlations.append(correl
ation_value)
            print('', end='', flush=True)

        print('\tDone!')
        print('¯'*FRAME_COUNT)
```

```
553.
554.         attack_results[attack] = {
555.             'psnr': sum(attacked_psnr_values) /
len(attacked_psnr_values),
556.             'ssim': sum(attacked_ssim_values) /
len(attacked_ssim_values),
557.             'correlation':
sum(attacked_correlations) /
len(attacked_correlations),
558.         }
559.
560.         print(f'{attack} done')
561.
562.     # Menyimpan hasil watermarking
563.     results = {
564.         'name': f'DSSS',
565.         'time': watermarking_time,
566.         'time per frame': watermarking_time /
FRAME_COUNT,
567.         'psnr': sum(psnr_values) /
len(psnr_values),
568.         'ssim': sum(ssim_values) /
len(ssim_values),
569.         'correlation': sum(correlations) /
len(correlations),
570.         'attack': attack_results,
571.     }
572.
573.     return results
574.
575. def sign(video_path, signature: Signature,
hash: Hash, key):
576.     """Membuat digital signature dari video
577.
578.     Args:
579.       video_path (str): path ke video yang akan
di-sign
580.       signature (object): kelas algoritma
signature
581.       hash (object): kelas algoritma hash
582.       key (str): kunci untuk menandatangani
video
583.
584.     Returns:
585.       str: digital signature
586.     """
587.     video = cv2.VideoCapture(video_path)
588.     frame_count =
int(video.get(cv2.CAP_PROP_FRAME_COUNT))
589.     width =
int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
590.     height =
int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
591.     fps = int(video.get(cv2.CAP_PROP_FPS))
592.
593.     start_time = time.time()
594.
595.     video_hash = hash.get_file_hash(video_path)
596.     signature_value =
signature.sign(video_hash, key)
597.
598.     signature_time = time.time() - start_time
599.
600.     return (signature_value, video_hash,
signature_time)
601.
602. def generate_random_watermark(shape):
603.     """Membuat watermark acak dengan ukuran
tertentu
604.
605.     Args:
606.       shape (tuple): ukuran watermark
607.
608.     Returns:
609.       numpy.ndarray: watermark acak dengan
nilai -255 atau 255
610.     """
611.     watermark = np.random.randint(0, 2,
shape).astype(np.uint8)
612.     return watermark * 255
613.
614. def main():
615.     video_path = 'video.mp4'
616.     watermark_path = 'watermark.png'
617.     watermarking_key = 'key'
618.     key = 'key'
619.
620.     # import grayscale
621.     video = cv2.VideoCapture(video_path)
622.
623.     frame_count =
int(video.get(cv2.CAP_PROP_FRAME_COUNT))
624.     width =
int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
625.     height =
int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
626.     fps = int(video.get(cv2.CAP_PROP_FPS))
627.     video_shape = (height, width, 3)
628.
629.     print('Reading, there are', frame_count,
'frames. Only',
630.           FRAME_COUNT, 'frames will be
processed')
631.     print('_'*FRAME_COUNT)
632.     frames = []
633.     for i in range(FRAME_COUNT):
634.         ret, frame = video.read()
635.         if not ret:
636.             break
637.
638.         # frame = cv2.cvtColor(frame,
cv2.COLOR_BGR2GRAY)
639.
640.         frames.append(frame)
641.         print('▉', end='', flush=True)
642.     print('\tDone!')
643.     print('¯'*FRAME_COUNT)
644.     # # Membuat watermark acak sebagai image
645.     # watermark =
generate_random_watermark(video_shape)
646.
647.     # import watermark 3 channel
648.     watermark = cv2.imread(watermark_path,
cv2.IMREAD_COLOR)
649.     watermark = cv2.resize(watermark, (width,
height))
650.
651.     results = []
652.     for wm in watermarks:
653.         if wm.name == 'DSSS':
654.             result = evaluate_dsss(frames, wm,
watermarking_key,
655.                                    frame_count=
FRAME_COUNT, width=width, height=height, fps=fps,
folder=f'./{wm.name}')
656.         else:
657.             result = evaluate_watermarking(wm,
frames, watermark, watermarking_key,
658.                                            fram
e_count=FRAME_COUNT, width=width, height=height,
fps=fps, folder=f'./{wm.name}')
659.
```

```
660.         result['signature'] = []
661.         for signature in signatures:
662.             for hash in hashes:
663.                 signature_value, hash_value,
signature_time = sign(
664.                     f'./{wm.name}/watermarked_v
ideo.mp4', signature, hash, key)
665.
666.                 result['signature'].append({
667.                     'name': f'{signature.name}
- {hash.name}',
668.                     'value':
str(signature_value),
669.                     'hash': hash_value,
670.                     'time': signature_time,
671.                 })
672.
673.                 print(f'{wm.name} -
{signature.name} - {hash.name} done')
674.
675.         results.append(result)
676.
677.         with open(f'./{wm.name}/result.json',
'w') as f:
678.             json.dump(result, f, indent=2)
679.
680.     with open('results.json', 'w') as f:
681.         json.dump(results, f, indent=2)
682.
683. if __name__ == '__main__':
684.     main()
```

4) Hasil dan Pembahasan

Parameter yang digunakan dalam teknik telah dicoba beberapa kali hingga menemukan nilai parameter yang menghasilkan hasil yang optimum. Video yang diuji memilki ukuran *full HD*, yaitu 1920 x 1080 piksel. *Channel* yang diuji pada eksperimen merupakan 3D *multichannel*, yang berarti memiliki komponen RGB (*red, green, blue*). Jumlah *frame* dalam uji dibatasi hingga 100 *frames*.

a) Penyisipan Watermark

Penyisipan berhasil dilakukan pada beberapa teknik watermark dengan hasil gambar di bawah. Bandingkan dengan Gambar 6.

Hasil dari watermark DSSS terlihat *noisy*, hasil dari DCT terlihat lebih halus, dan hasil dari DWT terlihat ada sedikit watermark. Kualitas dari ketidakjelasan watermark merupakan hasil dari argumen parameter yang digunakan pada sumber kode.

Gbr. 6. Potongan *Frame* dari Video Orisinal

Gbr. 7. Potongan *Frame* dari Hasil Watermark DSSS

Gbr. 8. Potongan *Frame* dari Hasil Watermark DCT

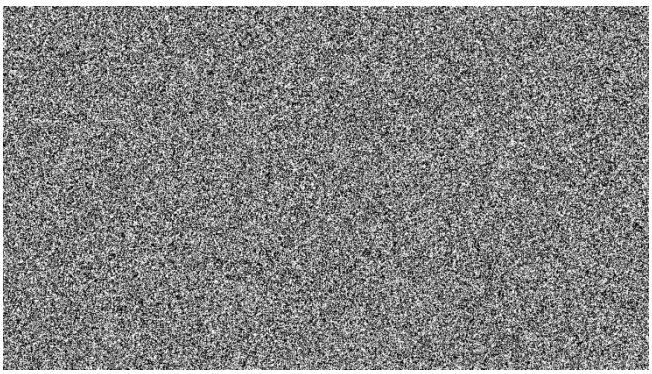
Gbr. 9. Potongan *Frame* dari Hasil Watermark DWT

b) Ekstraksi dan Deteksi Watermark

Watermark diekstrak dari setiap *frame* yang telah diberi watermark lalu dibandingkan dengan watermark asli. Metode ekstraksi berbeda untuk setiap algoritma. Pada DSSS, ekstraksi tidak dilakukan, melainkan deteksi dilakukan dengan menghitung koefisien korelasi dari watermark asli dan *frame* yang diberi watermark. Pada DWT, video hasil *watermarking* dipecah menjadi 4 subband dan dihitung kembali dengan menggunakan *frame* orisinal untuk mendapatkan watermark hasil ekstraksi

Perhitungan matematis dari ekstraksi watermark merupakan perhitungan terbalik (*invers*) dari perhitungan matematis penyisipan watermark. Berikut adalah gambar dari watermark hasil ekstraksi masing-masing algoritma, bandingkan dengan Gambar 10.

Gbr. 10. Potongan Watermark Orisinal



Gbr. 11. Potongan Watermark DSSS

Gbr. 12. Potongan Watermark dari Hasil Ekstraksi DCT

Gbr. 13. Potongan Watermark dari Hasil Ekstraksi DWT

c) Penyerangan Terhadap Video yang Diberi Watermark

Penyerangan dilakukan pada video yang telah diberi watermark untuk melihat seberapa tahan (*robust*) teknik yang digunakan terhadap serangan visual. Penyerangan yang dilakukan:

- ***Brightness***: Meningkatkan pencahayaan sebesar nilai tertentu
- ***Compression***: Mengompresi *frame* untuk menguji kasus pada kualitas video yang berbeda
- ***Contrast***: Meningkatkan kontras *frame* sebesar nilai tertentu
- ***Rotation***: Memutar video sebesar satu derajat
- ***Noise***: Menerapkan *noise* random pada *frame*

Berikut adalah hasil dari beberapa serangan pada gambar yang telah diberi watermark, bandingkan dengan Gambar 14.

Gbr. 14. Potongan *Frame* dari Video Orisinal

Gbr. 15. Penerapan Serangan *Brightness* pada Hasil Watermarking DWT

Gbr. 16. Penerapan Serangan *Compression* pada Hasil Watermarking DWT

Gbr. 17. Penerapan Serangan *Contrast* pada Hasil Watermarking DWT

Gbr. 18. Penerapan Serangan *Rotation* pada Hasil Watermarking DWT

Gbr. 19. Penerapan Serangan *Noise* pada Hasil Watermarking DWT

d) Hasil Evaluasi

Pengambilan hasil dilakukan pada seluruh 100 *frame* yang diuji. Nilai-nilai tersebut kemudian dirata-ratakan untuk mendapatkan hasil yang lebih umum, berikut hasilnya.

TABEL I. HASIL UJI WATERMARKING SEBELUM PENERAPAN SERANGAN

	DWT	DCT	DSSS
Waktu watermarking (detik/frame)	**0.3411**	**0.2593**	**0.0714**
PSNR (dB)	**34.008**	**49.777**	**31.232**
SSIM	**0.995**	**0.999**	**0.761**
NC	**0.996**	**0.892**	-
BER	**0.277**	**0.346**	-
Correlation Coefficient	-	-	**0.814**
Waktu tanda tangan RSA - SHA2 - 256 (detik)	**0.016**	**0.017**	**0.068**
Waktu tanda tangan RSA - SHA3 - 256 (detik)	**0.030**	**0.027**	**0.129**

Dari Tabel I, waktu *watermarking* DWT memakan waktu sekitar 4 hingga 5 kali lebih lama dibandingkan DSSS, disusul oleh DCT yang memakan waktu 3 hingga 4 kali lebih lama dibandingkan DSSS.

Hasil PSNR DCT menunjukkan yang paling tinggi dari teknik lain, yang berarti video hasil *watermarking* DCT lebih dapat dinikmati penonton. Meskipun begitu, hasil PSNR dari teknik lain juga menunjukkan nilai yang sangat bagus (setidaknya 30 dB).

Hasil SSIM dari DWT dan DCT memiliki nilai yang sangat tinggi. Sementara DSSS memiliki nilai cukup baik yang lebih rendah dari yang lainnya, menandakan perbedaan struktural video asli dengan video yang telah diberi watermark lebih tinggi.

Hasil NC dari ekstraksi DWT dan DCT menunjukkan nilai yang sangat bagus. Ini berarti watermark dapat diekstrak dengan kualitas yang cukup tinggi.

Hasil BER dari ekstraksi DWT dan DCT menunjukkan nilai yang cukup rendah. Ini berarti perbandingan bit error dari ekstraksi memiliki nilai yang cukup bagus.

Hasil dari koefisien korelasi DSSS memiliki nilai yang sangat tinggi. Ini berarti nilai penguat watermark (k) dan penerapannya sudah cukup baik.

Tanda tangan menggunakan SHA3 pada panjang *hash* yang sama cenderung memakan waktu tanda tangan mendekati 2 kali lebih lama dibandingkan dengan SHA2. Meskipun begitu, tanda tangan ini dilakukan sekali per video sehingga dampaknya pada performa tanda tangan digital tidak signifikan.

TABEL II. HASIL UJI WATERMARKING UNTUK SERANGAN *NOISE*

	DWT	DCT	DSSS
PSNR (dB)	**9.807**	**9.615**	**9.570**
SSIM	**0.160**	**0.153**	**0.1400**
NC	**0.007**	**0.009**	-
BER	**0.505**	**0.500**	-
Correlation Coefficient	-	-	**0.043**

Pada uji serangan menggunakan *noise*, nilai SSIM dan PSNR dari semua teknik menurun secara drastis. Ini dapat disebabkan oleh lemahnya teknik tersebut terhadap serangan *noise* atau *noise* yang diterapkan sebagai serangan terlalu tinggi/kuat.

TABEL III. HASIL UJI WATERMARKING UNTUK SERANGAN *BRIGHTNESS*

	DWT	DCT	DSSS
PSNR (dB)	**13.901**	**14.069**	**14.073**
SSIM	**0.862**	**0.866**	**0.800**
NC	**0.022**	**-0.011**	-
BER	**0.499**	**0.503**	-

Correlation Coefficient	-	-	**0.621**

Pada uji serangan menggunakan penaikan pencahayaan, nilai SSIM untuk semua teknik masih cenderung tinggi atau cukup baik. Ini menandakan bahwa semua teknik tersebut tidak rentan terhadap serangan pencahayaan. Nilai koefisien korelasi DSSS menjadi cukup baik, terjadi penurunan yang sedang terhadap nilai ini.

TABEL IV. HASIL UJI WATERMARKING UNTUK SERANGAN *CONTRAST*

	DWT	DCT	DSSS
PSNR (dB)	**15.134**	**15.117**	**15.081**
SSIM	**0.874**	**0.868**	**0.816**
NC	**-0.062**	**-0.018**	-
BER	**0.519**	**0.501**	-
Correlation Coefficient	-	-	**0.639**

Mirip dengan serangan *brightness.* Nilai SSIM dan nilai koefisien korelasi dari semua teknik masih cenderung baik. Begitu juga dengan penurunan nilai koefisien korelasi dari DSSS yang sedang.

TABEL V. HASIL UJI WATERMARKING UNTUK SERANGAN *ROTATION*

	DWT	DCT	DSSS
PSNR (dB)	**17.815**	**17.885**	**17.640**
SSIM	**0.445**	**0.443**	**0.277**
NC	**-0.010**	**0.008**	-
BER	**0.507**	**0.498**	-
Correlation Coefficient	-	-	**0.001**

Berbeda dengan serangan yang lain, serangan rotasi membuat nilai SSIM menurun dan koefisien korelasi menurun secara drastis. Teknik DSSS sangat rentan terhadap serangan rotasi meskipun besar rotasi yang diberikan adalah satu derajat.

TABEL VI. HASIL UJI WATERMARKING UNTUK SERANGAN *COMPRESSION*

	DWT	DCT	DSSS
PSNR (dB)	**32.421**	**32.468**	**29.257**
SSIM	**0.948**	**0.947**	**0.779**
NC	**-0.088**	**-0.034**	-
BER	**0.536**	**0.510**	-
Correlation Coefficient	-	-	**0.211**

Serangan kompresi tidak memengaruhi nilai SSIM secara signifikan, tetapi teknik DSSS sangat rentan pada serangan ini. Nilai koefisien korelasi DSSS sangat kecil dan tidak baik untuk serangan ini.

V. KESIMPULAN

Teknik terbaik ditentukan pada kebutuhan performa komputasi. Penerapan watermark pada semua frame lalu ditanda tangan secara digital dapat menjadi pilihan terbaik apabila waktu yang lama tidak menjadi masalah pada durasi video tertentu. Apabila aspek waktu penting, watermark dapat diterapkan pada sebagian *frame* setiap interval waktu tertentu. Waktu pemrosesan akan menjadi lebih teratur dengan pemilihan algoritma yang tepat.

Kombinasi algoritma terbaik bergantung relatif berat pada pemilihan algoritma *watermarking*. Untuk *hashing* dalam tanda tangan digital, penggunaan SHA3 dapat menjadi pilihan yang paling tepat untuk menjamin keamanan yang lebih tinggi. Untuk enkripsi, penggunaan RSA menjadi pilihan yang paling tepat untuk mendapatkan keamanan dan kematangan enkripsi yang tinggi.

Penggunaan teknik DSSS untuk watermarking dapat menjadi pilihan tepat apabila adanya penjaminan keamanan yang lebih tinggi terhadap serangan pada video dan waktu pemrosesan menjadi hal penting. Di sisi lain, teknik DWT dan DCT menyediakan tingkat *robustness* yang tinggi sehinga menjadikannya pilihan yang tepat untuk kebutuhan keamanan watermark yang tinggi.

REFERENSI

[1] A. P. Komuna and A. R. Wirawan, "Pelanggaran Hak Cipta Pada Konten Video Tiktok," *Alauddin Law Development Journal*, vol. 3, no. 3, pp. 483–492, Nov. 2021, doi: 10.24252/ALDEV.V3I3.24762.

[2] A. Mohanarathinam, S. Kamalraj, G. K. D. Prasanna Venkatesan, R. V. Ravi, and C. S. Manikandababu, "Digital watermarking techniques for image security: a review," *J Ambient Intell Humaniz Comput*, vol. 11, no. 8, pp. 3221–3229, Aug. 2020, doi: 10.1007/S12652-019-01500-1/METRICS.

[3] R. Kaur and A. Kaur, "Digital signature," *Proceedings: Turing 100 - International Conference on Computing Sciences, ICCS 2012*, pp. 295–301, 2012, doi: 10.1109/ICCS.2012.25.

[4] T. Morkel, J. Eloff, and M. Olivier, "AN OVERVIEW OF IMAGE STEGANOGRAPHY."

[5] Y. Liu, S. Liu, Y. Wang, H. Zhao, and S. Liu, "Video steganography: A review," *Neurocomputing*, vol. 335, pp. 238–250, Mar. 2019, doi: 10.1016/J.NEUCOM.2018.09.091.

[6] R. Awati, "What is Blowfish and how is it used in cryptography?" Accessed: Jun. 12, 2024. [Online]. Available: https://www.techtarget.com/searchsecurity/definition/Blowfish

[7] A. Dixit and R. Dixit, "A Review on Digital Image Watermarking Techniques," *International Journal of Image, Graphics and Signal Processing*, vol. 9, no. 4, pp. 56–66, Apr. 2017, doi: 10.5815/IJIGSP.2017.04.07.

[8] O. S. Faragallah *et al.*, "A Comprehensive Survey Analysis for Present Solutions of Medical Image Fusion and Future Directions," *IEEE Access*, vol. 9, pp. 11358–11371, 2021, doi: 10.1109/ACCESS.2020.3048315.

[9] N. Chervyakov, P. Lyakhov, and N. Nagornov, "Analysis of the Quantization Noise in Discrete Wavelet Transform Filters for 3D Medical Imaging," *Applied Sciences*, vol. 10, no. 4, p. 1223, Feb. 2020, doi: 10.3390/app10041223.

[10] D. Mindrila and M. E. Phoebe, "Scatterplots and Correlation."

[11] G. Basatwar, "Hashing Algorithms – An In-Depth Guide To Understanding Hash Functions - AppSealing." Accessed: Jun. 15, 2024. [Online]. Available: https://www.appsealing.com/hashing-algorithms/#deterministic

[12] Z. Wang, M. Zuo, S. Yao, and N. Aihemaiti, "Internet of Vehicles Based on TrustZone and Optimized RSA," *IOP Conf Ser Mater*

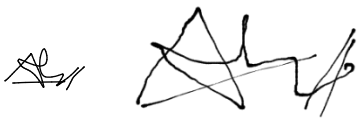
Sci Eng, vol. 782, no. 2, Apr. 2020, doi: 10.1088/1757-899X/782/2/022073.

[13] LORY, “Comparison of all hashing Algorithms | by LORY | Medium.” Accessed: Jun. 15, 2024. [Online]. Available: https://iorilan.medium.com/comparison-of-all-hashing-algorithms-88eda61f064a

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, 16 Juni 2024



Muhamad Farhan Syakir
18221145