Pengamanan dan Verifikasi Keaslian File Audio Menggunakan Enkripsi AES, Steganografi LSB, dan Verifikasi Kriptografi Visual

Irfan Musthofa - 18222056
Program Studi Sistem dan Teknologi Informasi
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: im@irfanmusthofa.com , 18222056@std.stei.itb.ac.id

***Abstrak*—Dalam era digital yang semakin berkembang, kebutuhan akan metode perlindungan dan penyembunyian data menjadi semakin krusial, terutama dalam konteks komunikasi rahasia dan perlindungan hak cipta. Proyek ini mengembangkan sistem steganografi audio berbasis metode Least Significant Bit (LSB), dikombinasikan dengan enkripsi AES-256 dan verifikasi integritas melalui kode QR. File rahasia terlebih dahulu dienkripsi menggunakan algoritma Advanced Encryption Standard (AES) dengan panjang kunci 256-bit, lalu disisipkan ke dalam file audio WAV menggunakan manipulasi bit paling tidak signifikan dari setiap sampel audio. Untuk menjamin keaslian file audio setelah proses embedding, sistem menghasilkan QR Code yang memuat nilai hash SHA-256 dari audio yang telah dimodifikasi. Verifikasi dilakukan dengan mencocokkan hash dari audio yang diunggah dengan isi QR. Seluruh proses dirancang agar berjalan in-memory tanpa menyimpan file ke disk, dengan antarmuka web berbasis Next.js dan backend FastAPI. Hasil pengujian menunjukkan bahwa sistem mampu menyisipkan, mengenkripsi, mengekstrak, serta memverifikasi file secara efektif tanpa mengurangi kualitas audio secara signifikan. Sistem ini berpotensi diterapkan dalam berbagai skenario nyata seperti perlindungan hak cipta digital, komunikasi rahasia, dan audit forensik digital.**

Kata kunci*— *Steganografi, LSB, AES-256, QR Code Keamanan Data

I. INTRODUCTION (*HEADING 1*)

Seiring dengan pesatnya pertumbuhan teknologi informasi terutama di penggunaan file audio, pertukaran pesan, dan keamanan pesan, kebutuhan akan keamanan data menjadi semakin penting. Pengiriman informasi secara daring berisiko terhadap penyadapan, modifikasi, dan penyalahgunaan oleh pihak yang tidak bertanggung jawab. Salah satu pendekatan yang dapat digunakan untuk meningkatkan keamanan komunikasi adalah dengan menyembunyikan pesan rahasia dalam media lain, teknik ini dikenal sebagai steganografi.

Steganografi audio adalah teknik menyisipkan data rahasia ke dalam file audio digital sedemikian rupa sehingga tidak terdengar perbedaan yang signifikan oleh telinga manusia. Salah satu metode paling umum yang digunakan adalah metode *Least Significant Bit* (LSB), yang memodifikasi bit terkecil dari setiap sampel audio untuk menyisipkan data biner [1]. Bit yang diubah ini akan mengubah audio, tetapi tidak akan terdengar secara signifikan perbedaannya oleh manusia. Meski sederhana dan efisien, metode LSB memiliki kelemahan dari segi keamanan karena mudah dianalisis dan dimodifikasi jika tidak dilindungi [2].

Untuk mengatasi keterbatasan tersebut, kriptografi digunakan sebagai lapisan keamanan tambahan. Dalam proyek ini, metode *Advanced Encryption Standard* (AES) dengan kunci simetris digunakan untuk mengenkripsi file yang akan disisipkan, sehingga data tetap terlindungi meskipun proses penyisipan berhasil dianalisis oleh pihak luar [3]. Lebih lanjut, proyek ini juga mengimplementasikan verifikasi keaslian menggunakan *QR code* yang dihasilkan dari *hash* file audio yang telah disisipkan. Dengan pendekatan ini, pengguna dapat memverifikasi apakah sebuah file audio telah dimodifikasi atau tetap otentik.

Dokumen makalah ini akan memperlihatkan bagaimana penggunaan enkripsi AES dengan kunci simetris dan verifikasi berbasis visual seperti *QR code* dapat membantu mengamankan informasi rahasia di balik file audio dengan serta memudahkan proses verifikasi dengan hanya memindai citra *QR code*.

II. PEMBAHASAN

A. Metode Least Significant Bit pada Steganografi Audio

Steganografi adalah teknik penyembunyian informasi dalam media digital sehingga keberadaan data tersebut tidak terdeteksi oleh pihak luar. Salah satu metode paling sederhana dan umum digunakan dalam steganografi digital adalah *Least Significant Bit* (LSB). Metode ini bekerja dengan mengganti bit terkecil (*least significant*) dari setiap unit data digital. Misalkan pada sampel audio, bit terkecil untuk setiap potongan atau block audio dapat diubah tanpa mengubah karakteristik suara utamanya secara signifikan.

Pada audio digital, sekarang umumnya data direpresentasikan dalam bentuk sampel 16-bit. Dengan memodifikasi 1 bit paling kanan dari setiap sampel, pesan biner dapat disisipkan secara bertahap dan berurutan. Misalnya, jika

data audio direpresentasikan sebagai 10101010 11110000, maka bit paling kanan dapat digantikan dengan bit dari pesan rahasia seperti 1 menjadi 10101010 11110001 [1]. Pendekatan ini memanfaatkan ketidakterlihatan perbedaan bit oleh pendengaran manusia, sehingga perubahan tidak terdengar secara signifikan dalam file audio hasil steganografi.

Namun, karena LSB tidak menyediakan enkripsi bawaan, pesan yang disisipkan sangat rentan terhadap serangan jika diketahui oleh pihak ketiga. Pihak ketiga dapat dengan mudah membandingkan audio asli dengan audio yang disisipkan pesan menggunakan perangkat lunak untuk audio seperti Logic Pro, Ableton Live, dan FL Studio. Salah satu metode untuk mendeteksi keberadaan steganografi pada audio adalah dengan teknik inversi fase. Ketika dua file audio yang identik dimainkan bersamaan, namun salah satunya dibalik fasenya (*inverted phase*), maka keduanya akan saling membatalkan dan menghasilkan keheningan (*silence*). Akan tetapi, jika salah satu file telah dimodifikasi secara halus, misalnya karena penyisipan bit melalui metode LSB, maka gelombang audio tidak lagi identik. Akibatnya, pembatalan tidak akan sempurna dan akan terdengar residu *noise* sebagai hasil perbedaan kecil antara dua sinyal tersebut. Teknik ini secara praktis bisa digunakan untuk mendeteksi indikasi adanya steganografi tersembunyi dalam audio. Sehingga, pihak ketiga dapat mengambil semua bit yang menyebabkan residu *noise* untuk dibaca kembali pesan rahasinya. Oleh karena itu, dibutuhkan mekanisme tambahan untuk mengamankan isi pesan.

B. Enkripsi AES-256

Advanced Encryption Standard (AES) merupakan algoritma kriptografi dengan kunci simetris yang banyak digunakan dalam berbagai sistem keamanan informasi modern. AES dikembangkan sebagai pengganti *Data Encryption Standard* (DES) dan disahkan oleh NIST melalui hasil lomba pembuatan standar algoritma kriptografi yang baru [4]. AES bekerja dengan blok data berukuran 128 bit dan mendukung tiga panjang kunci yaitu 128, 192, dan 256 bit. Dalam proyek ini, digunakan AES-256, yaitu AES dengan panjang kunci 256 bit, yang dianggap sangat kuat dan aman terhadap serangan *brute-force.*

AES adalah algoritma *block cipher* simetris, yang berarti proses enkripsi dan dekripsi menggunakan kunci yang sama. AES-256 bekerja dalam 14 putaran (*rounds*) untuk mengenkripsi setiap blok 128-bit data. Setiap putaran melibatkan beberapa tahapan utama, yaitu:

1. AddRoundKey: Setiap *byte* pada blok data dikombinasikan dengan bagian dari kunci yang diperluas menggunakan operasi XOR sebagai tahap inisialisasi awal.
2. Putaran sebanyak 13 kali:
    a. *SubBytes*: Setiap *byte* pada blok data digantikan dengan *byte* lain menggunakan tabel substitusi (S-box) non-linear yang telah ditentukan.
    b. *ShiftRows*: Baris-baris dalam matriks data dirotasi secara melingkar atau siklik untuk memperkuat difusi.
    c. *MixColumns*: Setiap kolom dari matriks data diubah dengan operasi linear yang meningkatkan penyebaran antar *byte*.
    d. *AddRoundKey*: Setiap *byte* pada blok data dikombinasikan dengan bagian dari kunci yang diperluas menggunakan operasi XOR.
3. Putaran terakhir (putaran ke-14):

    Proses yang dilakukan pada putaran terakhir ini adalah hanya *SubBytes, ShiftRows,* dan *AddRoundKey*.

Karena AES bersifat *block cipher* dengan besar setiap blok 128-bit, sehingga data yang panjangnya tidak genap kelipata 16 akan diberi *padding* atau penambahan *byte* khusus agar panjangnya sesuai. *Padding* ini akan dihapus kembali saat proses dekripsi.

Kunci AES-256 dihasilkan dari kata sandi pengguna yang kemudian di-hash menggunakan algoritma SHA-256 untuk memastikan kunci memiliki panjang 256-bit tetap. Sebelum proses penyisipan ke dalam audio, file rahasia akan terlebih dahulu dienkripsi menggunakan AES-256 dalam mode CBC (Cipher Block Chaining), dan hasilnya akan disisipkan ke dalam audio dengan metode LSB.

AES-256 yang dipilih ini memiliki tingkat keamanan yang sangat tinggi karena panjang kuncinya sebesar 256-bit, yang berarti membutuhkan 2^{256} kemungkinan tebakan kunci untuk *brute-force*. Selain itu struktur internal non-linear dan banyaknya putaran juga menyulitkan serangan analitik. Dengan integrasi AES-256 pada data sebelum disisipkan ke dalam audio, sistem ini memastikan bahwa sekalipun isi pesan berhasil diekstrak oleh pihak ketiga, pesan tersebut tetap tidak dapat dibaca tanpa kunci yang benar.

C. Verifikasi Keaslian dengan QR Code

QR Code (*Quick Response Code*) merupakan jenis *barcode* dua dimensi yang dikembangkan oleh perusahaan Jepang, Denso Wave, pada tahun 1994. *QR Code* dirancang untuk menyimpan data dalam jumlah besar serta dapat dipindai dengan cepat dari berbagai sudut oleh kamera atau pemindai digital. Format ini sangat populer dalam berbagai aplikasi, mulai dari pelabelan produk hingga autentikasi digital [5].

Untuk memastikan keaslian dan integritas file audio hasil steganografi, digunakan mekanisme verifikasi berbasis *QR code*. Setelah proses penyisipan selesai, sistem akan menghitung *hash* SHA-256 dari file audio yang sudah disisipkan. *Hash* ini kemudian di-*encode* dalam bentuk *QR code* dan dikirimkan bersamaan dengan file audio ke pengguna.

Pengguna dapat menyimpan QR code tersebut, lalu menggunakannya dalam proses verifikasi. Saat ingin memverifikasi keaslian file audio, sistem akan kembali menghitung *hash* file audio yang diunggah, kemudian membandingkannya dengan hash hasil *decoding* QR code.

Jika hash dari audio cocok dengan *hash* yang terdapat pada QR code, maka file dianggap valid dan tidak mengalami modifikasi sejak proses *embedding*. Ini memberi lapisan kepercayaan tambahan terhadap keaslian file.

III. IMPLEMENTASI

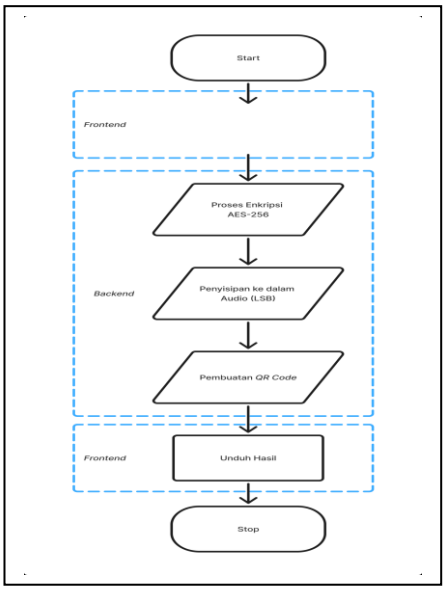
Implementasi sistem ini dilakukan dengan mengembangkan aplikasi web berbasis arsitektur *client-server*. Sistem memungkinkan pengguna untuk menyisipkan file ke dalam audio menggunakan metode Least Significant Bit (LSB), mengenkripsinya dengan AES-256, dan menghasilkan QR code sebagai sarana verifikasi. Proses juga mencakup ekstraksi dan verifikasi data secara aman. Semua proses perhitungan dilakukan di *backend* menggunakan bahasa pemrograman *Python* dan bantuan *FastAPI*. Sedangkan tampilan *frontend* akan menggunakan *framework Next.js* (*TypeScript*). Sistem ini akan di-*deploy* agar dapat dicoba secara publik menggunakan *Railway*.

A. *Proses Penyisipan (Embedding)*

1. *Input Audio* dan *File*: Pengguna mengunggah file audio berformat .wav serta file data yang ingin disembunyikan (dapat berupa format apa saja). Selain itu, pengguna juga diminta memasukkan *key*/*password* yang akan di-*hash* untuk keperluan enkripsi.
2. Proses Enkripsi AES-256: File data akan dienkripsi menggunakan algoritma AES-256 dengan kunci yang dimasukkan pengguna di *backend*. Proses ini memastikan kerahasiaan data meskipun ditemukan oleh pihak ketiga.
3. Penyisipan ke dalam Audio (LSB): Hasil enkripsi kemudian disisipkan ke dalam file audio menggunakan metode LSB.
4. Pembuatan *QR Code*: Setelah penyisipan selesai, *hash* dari audio hasil penyisipan akan dihitung dan dikonversi menjadi *QR code* sebagai identitas digital. *QR code* ini berfungsi untuk verifikasi keaslian.
5. Unduh Hasil: Sistem akan menghasilkan dua file, audio hasil *embedding* dan gambar QR code. Keduanya dapat diunduh oleh pengguna.

Berikut adalah *flowchart diagram* untuk proses penyisipan

Fig. 1. Proses Penyisipan



Berikut adalah implementasi kode untuk proses penyisipan (*embedding*).

```
# ===== *** Derive AES key from hashed string *** =====
def get_key(key_str: str) -> bytes:
    # Hash the input key string to get a fixed-size AES key (256-bit).
    return hashlib.sha256(key_str.encode()).digest()

# ===== *** Encrypt *** =====
def encrypt(data: bytes, key_str: str) -> bytes:
    key = get_key(key_str)
    cipher = AES.new(key, AES.MODE_CBC)
    ct_bytes = cipher.encrypt(pad(data, BLOCK_SIZE))
    return base64.b64encode(bytes(cipher.iv) + ct_bytes)

# ===== *** Generate SHA-256 hash from raw audio bytes *** =====
def hash_audio_bytes(audio_bytes: bytes) -> str:
    return hashlib.sha256(audio_bytes).hexdigest()

# ===== *** Generate QR code image (as BytesIO) from raw audio bytes *** =====
def generate_qr_from_hash_image(audio_bytes: bytes) -> BytesIO:
    hash_val = hash_audio_bytes(audio_bytes)

    factory = qrcode.image.pil.PilImage
    qr = qrcode.make(hash_val, image_factory=factory)

    output_stream = BytesIO()
    qr.save(output_stream, format="PNG")
    output_stream.seek(0)
    return output_stream

def embed_in_audio_bytesio(audio_stream: BytesIO, data: bytes, output_stream: BytesIO):
    message = base64.b64encode(data).decode() + "###" # end marker
    bin_message = ''.join(format(ord(char), '08b') for char in message)

    with wave.open(audio_stream, 'rb') as audio:
        params = audio.getparams()
        frames = np.frombuffer(audio.readframes(audio.getnframes()), dtype=np.int16).copy()

    if len(bin_message) > len(frames):
        raise ValueError("Audio file is too short to hold the message.")

    for i in range(len(bin_message)):
        frames[i] = (frames[i] & ~1) | int(bin_message[i])

    with wave.open(output_stream, 'wb') as stego_audio:
        stego_audio.setparams(params)
        stego_audio.writeframes(frames.tobytes())
```

1) get_key(key_str: str) -> bytes

Fungsi ini bertanggung jawab untuk menghasilkan kunci enkripsi yang kuat dengan mengubah string kunci yang dimasukkan pengguna menjadi kunci AES 256-bit. Proses ini dilakukan dengan menerapkan algoritma hash SHA-256 pada string tersebut. Hasil akhirnya adalah kunci sepanjang 32 byte yang digunakan dalam proses enkripsi.

2) encrypt(data: bytes, key_str: str) -> bytes

Fungsi ini mengenkripsi data mentah menggunakan algoritma AES dalam mode CBC (*Cipher Block Chaining*). Setelah kunci diperoleh dari fungsi get_key, data di-*pad* agar panjangnya menjadi kelipatan blok AES (16 byte), lalu dienkripsi. IV (*Initialization Vector*) yang dihasilkan secara acak dikombinasikan dengan *ciphertext*, kemudian di-*encode* dalam format base64 agar siap disisipkan ke dalam audio.

3) hash_audio_bytes(audio_bytes: bytes) -> str

Fungsi ini digunakan untuk menghasilkan nilai *hash* dari data audio mentah menggunakan algoritma SHA-256. Hash ini berfungsi sebagai *digital fingerprint* dari audio hasil embedding, dan digunakan untuk keperluan verifikasi keaslian di tahap selanjutnya.

4) generate_qr_from_hash_image(audio_bytes: bytes) -> BytesIOr

Fungsi ini menghasilkan QR Code dari hash audio hasil embedding. QR Code ini kemudian dikembalikan dalam bentuk objek BytesIO agar dapat langsung dikirim melalui jaringan atau diunduh. Fungsi ini penting sebagai metode otentikasi terhadap audio agar bisa diverifikasi keasliannya di sisi penerima.

5) embed_in_audio_bytesio(audio_stream: BytesIO, data: bytes, output_stream: BytesIO)

Fungsi utama ini melakukan proses penyisipan data ke dalam audio menggunakan metode *Least Significant Bit* (LSB). Pertama, data dienkode dalam base64 dan ditambahkan tanda akhir "###" sebagai penanda batas pesan. Selanjutnya, data dikonversi menjadi representasi biner dan disisipkan ke dalam bit paling tidak signifikan dari setiap sampel audio (tipe int16). Hasil akhirnya ditulis ke output_stream sebagai audio hasil *embedding*

B. Proses Ekstraksi

1. *Input Audio* dan *Key*: Pengguna mengunggah audio hasil embedding dan memasukkan *key*/*password* yang digunakan saat proses penyisipan.
2. Ekstraksi Bit LSB: Sistem mengekstrak bit yang disisipkan dari audio menggunakan metode LSB dan membentuk ulang data terenkripsi.
3. Dekripsi AES-256: Data hasil ekstraksi akan didekripsi menggunakan AES-256. Jika *key* benar, maka file asli dapat dipulihkan sepenuhnya.

```
# ===== *** Extract data from audio stream (BytesIO) ***
=====
def extract_from_audio_bytesio(audio_stream: BytesIO) ->
bytes:
    with wave.open(audio_stream, 'rb') as audio:
        frames =
np.frombuffer(audio.readframes(audio.getnframes()),
dtype=np.int16).copy()

    bits = [str(frame & 1) for frame in frames
    chars = []

    for i in range(0, len(bits), 8):
        byte = bits[i:i+8]
        char = chr(int(''.join(byte), 2))
        chars.append(char)
        if ''.join(chars[-3:]) == "###":
            break
```

```
# ===== *** Decrypt *** =====
def decrypt(encrypted_data: bytes, key_str: str) ->
bytes:
    key = get_key(key_str)
    data = base64.b64decode(encrypted_data)
    iv = data[:BLOCK_SIZE]
    ct = data[BLOCK_SIZE:]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    pt = unpad(cipher.decrypt(ct), BLOCK_SIZE)
    return pt
```

1) extract_from_audio_bytesio(audio_stream: BytesIO) -> bytes

Fungsi ini bertugas mengambil bit-bit data tersembunyi dari file audio. Audio yang telah disisipkan pesan dibaca sebagai array bilangan bulat 16-bit. Lalu, setiap bit paling tidak signifikan (LSB) dari sampel audio diambil dan disusun menjadi byte. Setiap delapan bit membentuk satu karakter, yang dikumpulkan dalam sebuah array hingga ditemukan penanda akhir khusus "###". Setelah penanda ditemukan, seluruh pesan base64 yang tersembunyi dipotong dan didekode ulang untuk memperoleh data terenkripsi aslinya.

2) decrypt(encrypted_data: bytes, key_str: str) -> bytes

Setelah data disisipkan berhasil diambil dan diverifikasi, langkah terakhir adalah dekripsi untuk mendapatkan file asli. Fungsi decrypt menggunakan kunci enkripsi yang berasal dari hash SHA-256 string masukan pengguna. Data terenkripsi dipisahkan menjadi IV (*Initialization Vector*) dan ciphertext. Ciphertext didekripsi dengan mode CBC (*Cipher Block Chaining*), lalu diproses dengan metode unpad untuk menghapus padding yang ditambahkan saat enkripsi. Output dari proses ini adalah data asli berupa gabungan nama file (*filename*) dan isi file (*content*), yang kemudian digunakan untuk menghasilkan file unduhan pengguna.

C. Proses Verifikasi

1. *Input Audio dan QR Code*: Pengguna mengunggah file audio hasil embedding dan QR code yang terkait.Ekstraksi Bit LSB: Sistem mengekstrak bit yang disisipkan dari audio menggunakan metode LSB dan membentuk ulang data terenkripsi.
2. Ekstraksi dan Verifikasi Hash: Sistem mengekstrak hash dari file audio dan membandingkannya dengan hash yang dikandung oleh QR code. Jika cocok, maka audio dipastikan valid dan belum dimodifikasi.*Output* file asli: File hasil dekripsi akan ditawarkan untuk diunduh pengguna, dengan nama dan ekstensi file yang sama seperti file asli.

   Berikut adalah implementasi kode untuk proses verifikasi.

```
# ===== *** Verify audio hash against QR image bytes *** =====
def verify_audio_hash_bytes(audio_bytes: bytes, qr_bytes: bytes) -> bool:
    audio_hash = hash_audio_bytes(audio_bytes)

    qr_img = Image.open(BytesIO(qr_bytes)).convert("RGB")
    qr_array = np.array(qr_img)

    detector = cv2.QRCodeDetector()
    val, _, _ = detector.detectAndDecode(qr_array)

    return audio_hash == val
```

1) verify_audio_hash_bytes(audio_bytes: bytes, qr_bytes: bytes) -> bool

Untuk menjamin integritas audio yang telah disisipkan, sistem memverifikasi kesesuaiannya dengan QR Code. Fungsi ini pertama-tama menghitung hash SHA-256 dari audio asli. Lalu, QR Code yang diunggah oleh pengguna dibuka sebagai gambar dan dipindai menggunakan cv2.QRCodeDetector. Data hash yang dibaca dari QR tersebut dibandingkan dengan hash audio. Jika keduanya cocok, maka file audio dianggap valid dan belum diubah, memungkinkan proses ekstraksi dilanjutkan.

IV. HASIL DAN ANALISIS

Sistem steganografi audio yang telah dibangun menggunakan metode LSB (*Least Significant Bit*), enkripsi AES-256, dan verifikasi QR code berhasil dijalankan dan diuji pada beberapa skenario yang mewakili seluruh alur fungsionalitas sistem. Pengujian dilakukan terhadap lima fitur utama sistem: penyisipan data, pembangkitan *QR code*, verifikasi file, ekstraksi data, dan ketahanan terhadap integritas data. Hasil dan analisis dari setiap fitur dijelaskan sebagai berikut.

1. Penyisipan Data dalam Audio (Embedding)

Fungsi embed_in_audio_bytesio() bekerja dengan baik dalam menyisipkan pesan ke dalam file audio berekstensi .wav. Proses ini dilakukan sepenuhnya di memori (BytesIO), tanpa menyimpan file ke disk, yang mempercepat dan menyederhanakan pipeline sistem. Uji coba dilakukan dengan menyisipkan berbagai jenis file (PDF, gambar, teks, dan ZIP) ke dalam audio berdurasi minimal 2 detik (44.1 kHz). Semua file berhasil disisipkan asalkan ukuran payload tidak melebihi kapasitas audio yang dihitung oleh get_max_capacity_from_bytesio().

Sistem juga memberikan respons yang tepat jika file terlalu besar untuk disisipkan, dengan menampilkan pesan kesalahan dari backend (400: Audio file is too short to hold the message). Ini menunjukkan validasi kapasitas sudah berjalan optimal.

2. Enkripsi AES-256

Fungsi encrypt() dan decrypt() menggunakan algoritma AES-256 dalam mode CBC (Cipher Block Chaining) dengan IV acak. Pengujian menunjukkan bahwa data yang telah disisipkan dalam audio tidak dapat diekstrak tanpa memasukkan kunci enkripsi yang sesuai. Percobaan dengan kunci yang salah menghasilkan pesan kesalahan dari backend, menandakan bahwa mekanisme dekripsi dan padding sudah bekerja sebagaimana mestinya.

Hasil enkripsi dienkode ke format base64 dan disisipkan ke dalam audio melalui LSB. Keutuhan ciphertext dijamin karena ekstraksi berhasil membedakan dengan benar bagian IV dan ciphertext sebelum didekripsi.

3. Pembangkitan QR Code untuk Verifikasi

Fungsi generate_qr_from_hash_image() menghasilkan QR Code dalam format PNG yang menyimpan hasil hash SHA-256 dari file audio yang telah disisipkan. QR Code berhasil dihasilkan dan dapat dipindai menggunakan fungsi verifikasi atau aplikasi pemindai QR eksternal. QR Code ini berperan sebagai checksum eksternal yang merepresentasikan integritas file audio hasil embedding.

4. Verifikasi Audio dan QR Code

Fungsi verify_audio_hash_bytes() mampu mendeteksi apakah QR code cocok dengan file audio. Pengujian dilakukan pada dua skenario:

QR code cocok dengan audio hasil embedding → sistem mengembalikan { "valid": true }.

QR code diambil dari file lain atau audio telah dimodifikasi → sistem mengembalikan { "valid": false }.

Validasi ini berjalan real-time dan mendeteksi perubahan sekecil apa pun dalam struktur audio. Ini memperkuat integritas sistem dan mencegah manipulasi data tersembunyi.

5. Ekstraksi File dan Dekripsi

Fungsi extract_from_audio_bytesio() berhasil mengambil bit-bit tersembunyi dan menggabungkannya kembali menjadi pesan base64. Kemudian fungsi decrypt() mengembalikan data asli, yang terdiri dari nama file dan isi file. Setelah didekripsi, pengguna langsung menerima file unduhan dengan nama dan ekstensi asli seperti sebelum disisipkan.

Pengujian terhadap berbagai tipe file (TXT, PNG, MP3, PDF) berhasil dilakukan, dan file hasil ekstraksi tetap utuh serta dapat dibuka dengan aplikasi standar. Ini menandakan bahwa encoding, enkripsi, dan decoding berjalan lossless.

6. Ketahanan dan Deteksi Modifikasi

Salah satu fitur keamanan penting dari sistem ini adalah kemampuannya mendeteksi modifikasi audio. Jika file audio yang telah diembedding dimodifikasi (misalnya dikompresi ulang, dipotong, atau diubah LSB-nya), maka hasil verifikasi terhadap QR Code akan gagal. Ini mengindikasikan bahwa sistem tidak hanya menyembunyikan data, tetapi juga menjaga integritas data melalui QR.

Selain itu, dilakukan uji pendengaran (aural testing) menggunakan teknik inversi fase. Dua file audio identik yang dibalik fase dan dijumlahkan menghasilkan silence.

Namun, ketika salah satu file mengandung data steganografi, muncul noise residu akibat bit yang berubah. Ini membuktikan bahwa LSB embedding memang menghasilkan perbedaan yang bisa dideteksi secara akustik jika dibandingkan dengan audio murni.

V. KESIMPULAN

Sistem yang dikembangkan dalam proyek ini berhasil mengimplementasikan teknik steganografi audio menggunakan metode Least Significant Bit (LSB) yang dikombinasikan dengan enkripsi AES-256 dan verifikasi integritas melalui QR Code. Sistem ini dibangun secara *in-memory*, tanpa menyimpan file di server, sehingga sangat efisien, ringan, dan cocok untuk deployment cloud seperti Railway. Berdasarkan hasil implementasi dan pengujian sistem, berikut adalah kesimpulan yang dapat diambil:

1. Keberhasilan Implementasi Teknik Steganografi

Sistem ini mampu menyisipkan file rahasia ke dalam audio WAV tanpa menghasilkan perubahan yang dapat didengar. Proses embedding dan ekstraksi dilakukan langsung melalui memori menggunakan objek BytesIO, yang menjadikan sistem ini cepat, efisien, dan hemat sumber daya. Penyisipan pesan dilakukan dengan menyisipkan bit-bit data ke bit terakhir setiap sample audio (LSB).

2. Perlindungan File Menggunakan Enkripsi AES-256

Sebelum data disisipkan, sistem mengenkripsi kontennya menggunakan algoritma AES-256 dalam mode CBC. Dengan panjang kunci 256 bit yang diperoleh dari hash kunci pengguna (SHA-256), data menjadi sangat sulit untuk dibuka tanpa kunci yang tepat. Ini menambah lapisan keamanan penting jika file audio jatuh ke tangan pihak yang tidak berwenang.

3. Verifikasi Keaslian Menggunakan QR Code

Sistem menghasilkan QR Code yang berisi nilai hash dari file audio hasil steganografi. Saat proses verifikasi, sistem menghitung ulang hash dari file audio dan mencocokkannya dengan nilai dalam QR. Ini memungkinkan pengguna untuk memverifikasi apakah audio telah diubah sejak embedding dilakukan, dan mendeteksi manipulasi sekecil apa pun.

4. Konsistensi Antarmuka dan Kinerja Responsif

Frontend yang dikembangkan menggunakan Next.js, React, TailwindCSS, dan Framer Motion memberikan pengalaman pengguna yang mulus dan profesional. Halaman /embed, /verify, dan /extract didesain dengan tampilan yang konsisten dan responsif. File hasil embedding maupun hasil ekstraksi dapat langsung diunduh dengan nama asli, berkat konfigurasi header Content-Disposition di backend FastAPI.

5. *Use Case* di Dunia Nyata

Sistem steganografi ini memiliki banyak potensi aplikasi nyata, di antaranya:

- Digital Rights Verification (DRM Support): Seniman, produser, atau perusahaan musik dapat menyisipkan informasi hak cipta dalam file audio sebagai identifikasi tersembunyi. Melalui QR Code, pihak ketiga dapat memverifikasi apakah audio tersebut masih asli atau telah dimodifikasi.
- Keamanan Komunikasi: Sistem ini dapat digunakan untuk mengirimkan file rahasia secara aman lewat file audio (misalnya dalam pesan suara atau rekaman panggilan) yang tidak mencurigakan secara kasat mata atau telinga.
- Pemantauan dan Audit Internal: Organisasi dapat menyisipkan file log atau metadata ke dalam audio untuk mendeteksi dan melacak modifikasi file dalam sistem distribusi internal.
- Penegakan Hukum Digital: Institusi forensik dapat menggunakan teknik ini untuk menyisipkan penanda digital sebagai watermark tak terlihat yang dapat diverifikasi selama investigasi.

6. Rekomendasi Pengembangan Lanjutan

Untuk pengembangan ke depan, sistem ini dapat ditingkatkan melalui:

- Penambahan opsi metode embedding lain seperti Phase Coding atau Spread Spectrum.
- Penambahan algoritma enkripsi alternatif.
- Penghitungan otomatis PSNR/SNR setelah embedding untuk mengevaluasi kualitas.
- Integrasi user authentication dan histori penggunaan untuk aplikasi komersial.

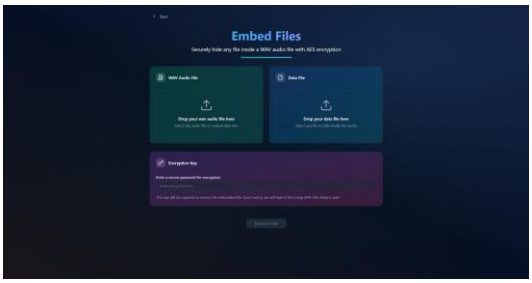
Dengan keberhasilan penggabungan antara teknik steganografi audio, kriptografi AES-256, dan verifikasi visual melalui QR Code, sistem ini terbukti sebagai solusi yang efisien, aman, dan praktis untuk menyimpan dan memverifikasi data tersembunyi. Sistem ini memiliki potensi nyata dalam berbagai bidang, mulai dari keamanan digital, perlindungan hak cipta, hingga forensik multimedia.

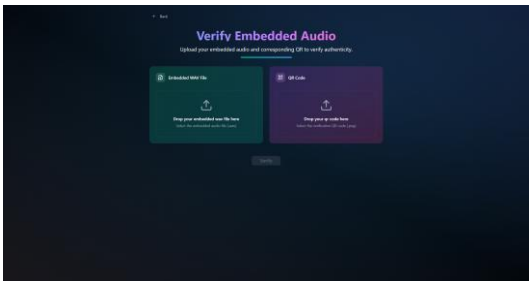
LAMPIRAN

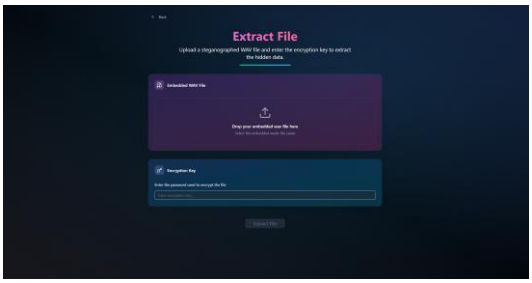
Web: https://audio-curify-web.up.railway.app/

API Documentation: https://audio-curify-backend.up.railway.app/docs

Screenshots:







REFERENCES

[1] N. Provos dan P. Honeyman, "Hide and Seek: An Introduction to Steganography", IEEE Security & Privacy, vol. 1, no. 3, pp. 32–44, 2003.

[2] B. E. Sánchez Rinza, L. G. Munive Morales, and A. Jaramillo Núñez, "LSB Algorithm to Hide Text in an Audio Signal," Computación y Sistemas, vol. 26, no. 1, Mar. 2022, doi: https://doi.org/10.13053/cys-26-1-4150.

[3] J. Daemen dan V. Rijmen, "The Design of Rijndael: AES — The Advanced Encryption Standard", Springer-Verlag, 2002.

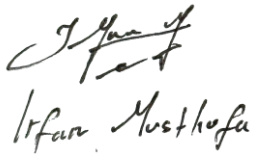
[4] Munir, R. "Beberapa Blok Cipher (Bagian 2)" 2025.

[5] ISO/IEC 18004:2024. Information Technology – Automatic Identification and Data Capture Techniques – QR Code Bar Code Symbology Specification.

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, 1 Juni 2025



Irfan Musthofa 18222056