📹 Panduan: Encrypted Video Streaming — Local Development → Deploy ke cPanel

> Alur kerja panduan ini: **bangun dan test dulu di komputer lokal**, baru setelah berjalan sempurna — deploy ke cPanel hosting. Ini cara yang benar agar kamu tidak debugging di server production.

📋 Daftar Isi

FASE 1 — Development Lokal

---

1. Persiapan Environment Lokal

Kamu butuh ini di komputer sebelum mulai:

Software	Fungsi	Download
XAMPP atau **Laragon**	Web server lokal (Apache + PHP + MySQL)	xampp / laragon
Composer	Package manager PHP untuk install CodeIgniter	getcomposer.org
PHP 8.1+	Sudah termasuk dalam XAMPP/Laragon	—
VS Code atau editor lain	Untuk edit kode	code.visualstudio.com

> 💡 **Rekomendasi:** Gunakan **Laragon** — lebih ringan dari XAMPP dan sudah include Composer.

Verifikasi Composer

Buka terminal / Command Prompt, jalankan:

```
composer --version
# Output: Composer version 2.x.x
```

Verifikasi PHP dan Ekstensi OpenSSL

```
php --version
# Output: PHP 8.1.x atau lebih baru

php -m | grep openssl
# Output: openssl  (harus muncul)
```

---

2. Instalasi CodeIgniter 4

Buka terminal, navigasi ke folder htdocs (XAMPP) atau www (Laragon):

```
# XAMPP
cd C:/xampp/htdocs

# Laragon
cd C:/laragon/www

# Buat project baru
composer create-project codeigniter4/appstarter videostreaming
cd videostreaming
```

Setelah selesai, coba akses `http://localhost/videostreaming/public` di browser. Jika muncul halaman welcome CodeIgniter 4, instalasi berhasil.

---

3. Struktur File & Folder

Berikut struktur project yang akan kita buat:

```
videostreaming/
├── app/
│   ├── Controllers/
│   │   └── VideoStream1.php         ← Controller utama
│   ├── Views/
│   │   └── video/
│   │       ├── upload1.php          ← Halaman upload
```

```
│   │       └── embed_video.php          ← Halaman player
│   └── Config/
│       └── Routes.php                   ← Routing
├── public/                              ← Entry point (index.php + .htaccess)
├── writable/
│   └── uploads/
│       ├── original/                    ← Video asli (sementara)
│       └── encrypted/                   ← Video terenkripsi (.enc)
└── .env                                 ← Konfigurasi environment
```

Buat folder upload secara manual (atau akan dibuat otomatis saat pertama upload):

```
mkdir -p writable/uploads/original
mkdir -p writable/uploads/encrypted
```

---

4. Buat File Aplikasi

4.1 Controller — `app/Controllers/VideoStream1.php`

Buat file baru dengan isi berikut:

```php
<?php

namespace App\Controllers;

use CodeIgniter\Controller;

class VideoStream1 extends Controller
{
    protected string $encryptionKey;
    protected $chunkSize = 1024 * 256; // 256KB per chunk

    public function __construct()
    {
        // Ambil kunci dari .env, bukan hardcoded
        $this->encryptionKey = env('VIDEO_ENCRYPTION_KEY', '');
    }

    /** Halaman upload - GET /video/upload */
    public function index()
    {
        return view('video/upload1');
    }

    /** Halaman player - GET /video/player */
    public function player()
    {
        return view('video/embed_video');
```

```
    }

    /** Stream video terenkripsi - GET /api/videos/stream/{videoId} */
    public function stream($videoId = null)
    {
        if (!$videoId) {
            return $this->response->setJSON(['success' => false, 'message' =>
'Video not found'])->setStatusCode(404);
        }

        $videoPath = WRITEPATH . 'uploads/encrypted/' . $videoId . '.enc';

        if (!file_exists($videoPath)) {
            return $this->response->setJSON(['success' => false, 'message' =>
'Video file not found'])->setStatusCode(404);
        }

        $fileSize = filesize($videoPath);
        $handle = fopen($videoPath, 'rb');

        if (!$handle) {
            return $this->response->setJSON(['success' => false, 'message' =>
'Cannot open video file'])->setStatusCode(500);
        }

        header('Content-Type: application/octet-stream');
        header('Content-Length: ' . $fileSize);
        header('Cache-Control: no-cache');
        header('X-Accel-Buffering: no');

        while (!feof($handle)) {
            $chunk = fread($handle, $this->chunkSize);
            echo $chunk;
            flush();
            if (connection_aborted()) break;
        }

        fclose($handle);
        exit;
    }

    /** Info metadata video - GET /api/videos/info/{videoId} */
    public function info($videoId = null)
    {
        if (!$videoId) {
            return $this->response->setJSON(['success' => false, 'message' =>
'Video not found'])->setStatusCode(404);
        }

        $videoPath = WRITEPATH . 'uploads/encrypted/' . $videoId . '.enc';

        if (!file_exists($videoPath)) {
            return $this->response->setJSON(['success' => false, 'message' =>
'Video file not found'])->setStatusCode(404);
```

```
        }

        return $this->response->setJSON([
            'success' => true,
            'data' => [
                'id'           => $videoId,
                'size'         => filesize($videoPath),
                'chunk_size'   => $this->chunkSize,
                'total_chunks' => ceil(filesize($videoPath) / $this->chunkSize),
            ]
        ]);
    }

    /** Ambil kunci enkripsi - GET /api/videos/key */
    public function getKey()
    {
        // TODO: Tambahkan autentikasi sebelum production!
        // if (!session()->get('user_id')) {
        //     return $this->response->setStatusCode(401)->setJSON(['success' =>
false, 'message' => 'Unauthorized']);
        // }

        return $this->response->setJSON([
            'success' => true,
            'key'     => base64_encode($this->encryptionKey)
        ]);
    }

    /** Upload & enkripsi video - POST /video/upload */
    public function doUpload()
    {
        $validation = \Config\Services::validation();
        $validation->setRules([
            'video' =>
'uploaded[video]|max_size[video,102400]|ext_in[video,mp4,avi,mkv,mov]'
        ]);

        if (!$validation->withRequest($this->request)->run()) {
            return $this->response->setJSON([
                'success' => false,
                'errors'  => $validation->getErrors()
            ])->setStatusCode(400);
        }

        $video = $this->request->getFile('video');

        if (!$video->isValid()) {
            return $this->response->setJSON(['success' => false, 'message' =>
'Invalid video file'])->setStatusCode(400);
        }

        $videoId      = uniqid('vid_', true);
        $uploadPath   = WRITEPATH . 'uploads/original/';
        $encryptedPath = WRITEPATH . 'uploads/encrypted/';
```

```
        if (!is_dir($uploadPath)) mkdir($uploadPath, 0755, true);
        if (!is_dir($encryptedPath)) mkdir($encryptedPath, 0755, true);

        $originalFile  = $uploadPath . $videoId . '.' . $video->getExtension();
        $encryptedFile = $encryptedPath . $videoId . '.enc';

        $video->move($uploadPath, $videoId . '.' . $video->getExtension());
        $this->encryptFile($originalFile, $encryptedFile);

        // Hapus file asli setelah dienkripsi
        unlink($originalFile);

        return $this->response->setJSON([
            'success' => true,
            'message' => 'Video uploaded and encrypted',
            'data'    => [
                'video_id' => $videoId,
                'size'     => filesize($encryptedFile)
            ]
        ]);
    }

    /** Enkripsi file dengan AES-256-CBC */
    private function encryptFile($source, $destination)
    {
        $iv        = openssl_random_pseudo_bytes(16);
        $plaintext = file_get_contents($source);
        $encrypted = openssl_encrypt($plaintext, 'aes-256-cbc', $this-
>encryptionKey, OPENSSL_RAW_DATA, $iv);

        $destHandle = fopen($destination, 'wb');
        fwrite($destHandle, $iv);        // 16 byte IV di awal
        fwrite($destHandle, $encrypted); // lalu data terenkripsi
        fclose($destHandle);
    }
}
```

Penjelasan kode secara rinci

```
    protected string $encryptionKey;
```

- `protected` → properti ini hanya bisa diakses dari dalam class ini sendiri dan class turunannya (tidak bisa diakses dari luar)
- `string` → tipe data, artinya variabel ini harus berisi teks
- `$encryptionKey` → nama variabel. Ini akan menyimpan kunci rahasia enkripsi video
- Nilainya tidak diset di sini — akan diisi di `__construct()` nanti

```
    protected $chunkSize = 1024 * 256;
```

- `$chunkSize` → ukuran potongan data saat streaming file
- `1024 * 256` → PHP otomatis menghitung ini: hasilnya `262144` byte = **256 KB**
- Artinya: setiap kali streaming, file dikirim 256KB dulu, lalu 256KB lagi, dst. Ini mencegah PHP kehabisan memori saat streaming file besar

---

```
    public function __construct()
    {
```

`__construct()` adalah **method spesial** yang otomatis dijalankan pertama kali setiap kali class ini dipakai. Namanya selalu `__construct` (dua underscore di depan). Cocok untuk inisialisasi — menyiapkan sesuatu sebelum method lain berjalan.

```
        $this->encryptionKey = env('VIDEO_ENCRYPTION_KEY', '');
```

- `$this->encryptionKey` → mengisi properti `$encryptionKey` yang dideklarasikan di atas
- `$this->` → cara mengakses properti/method dalam class yang sama. Bayangkan `$this` = "diri sendiri"
- `env('VIDEO_ENCRYPTION_KEY', '')` → fungsi bawaan CI4 untuk membaca nilai dari file `.env`
  - Argumen pertama `'VIDEO_ENCRYPTION_KEY'` = nama variabel yang dicari di `.env`
  - Argumen kedua `''` = nilai default jika variabel tidak ditemukan (string kosong)
- Jadi baris ini artinya: "*isi `$encryptionKey` dengan nilai `VIDEO_ENCRYPTION_KEY` dari file `.env`. Kalau tidak ada, pakai string kosong.*"

```
    public function index()
    {
        return view('video/upload1');
    }
```

- `public` → method ini bisa dipanggil dari luar class (termasuk oleh sistem routing CI4)
- `function index()` → nama method. CI4 secara konvensi menggunakan `index` sebagai halaman utama
- `return view('video/upload1')` → fungsi `view()` bawaan CI4 yang memuat file HTML dari folder `app/Views/`. Argumennya `'video/upload1'` artinya muat file `app/Views/video/upload1.php`
- `return` = kembalikan hasil ke pemanggil (dalam hal ini ke browser)

```
    public function player()
    {
```

```
        return view('video/embed_video');
    }
```

Sama persis dengan `index()`, bedanya memuat file `app/Views/video/embed_video.php`. Dipanggil saat user membuka halaman `/video/player`.

```
    public function stream($videoId = null)
    {
```

- `$videoId = null` → parameter dengan nilai default `null`. Artinya jika tidak ada ID yang dikirim, nilainya otomatis `null` daripada error

---

```
        if (!$videoId) {
            return $this->response->setJSON(['success' => false, 'message' =>
'Video not found'])->setStatusCode(404);
        }
```

- `if (!$videoId)` → jika `$videoId` kosong/null/false, jalankan blok ini
- `!` = operator NOT / kebalikan. `!$videoId` artinya "jika `$videoId` tidak ada nilainya"
- `$this->response` → objek response bawaan CI4 untuk mengatur apa yang dikirim ke browser
- `->setJSON([...])` → kirim data dalam format JSON. Array PHP otomatis dikonversi ke teks JSON
- `->setStatusCode(404)` → set HTTP status code 404 (Not Found). Browser/frontend bisa membaca kode ini untuk tahu apakah request berhasil atau gagal
- Perhatikan **method chaining** → `setJSON(...)->setStatusCode(...)` — memanggil dua method berturut-turut dalam satu baris

---

```
        $videoPath = WRITEPATH . 'uploads/encrypted/' . $videoId . '.enc';
```

- `WRITEPATH` → konstanta bawaan CI4 yang berisi path absolut ke folder `writable/`. Contoh: `/home/user/public_html/writable/`
- `.` → operator penggabungan string di PHP (bukan `+` seperti JavaScript)
- Hasil akhirnya: `/home/user/public_html/writable/uploads/encrypted/vid_xxx.enc`

```
        if (!file_exists($videoPath)) {
            return $this->response->setJSON(['success' => false, 'message' =>
'Video file not found'])->setStatusCode(404);
        }
```

- `file_exists($videoPath)` → fungsi PHP bawaan untuk mengecek apakah file benar-benar ada di server
- `!file_exists(...)` → jika file TIDAK ada, kirim error 404

```
        $fileSize = filesize($videoPath);
```

- `filesize()` → fungsi PHP untuk mendapatkan ukuran file dalam satuan **byte**
- Ini perlu dikirim ke browser lewat header agar browser tahu berapa besar data yang akan diterima

---

```
        $handle = fopen($videoPath, 'rb');
```

- `fopen()` → membuka file, seperti membuka buku sebelum dibaca
- Argumen pertama = path file yang dibuka
- `'rb'` = mode buka: `r` = read (baca saja), `b` = binary (file bukan teks biasa, tapi data biner seperti video)
- `$handle` = "pegangan" file yang terbuka, dipakai untuk operasi baca selanjutnya

---

```
        if (!$handle) {
            return $this->response->setJSON(['success' => false, 'message' =>
'Cannot open video file'])->setStatusCode(500);
        }
```

Jika `fopen()` gagal (misalnya permission folder salah), `$handle` bernilai `false`. Baris ini menangkap kondisi itu dan mengirim error 500 (Internal Server Error).

```
        header('Content-Type: application/octet-stream');
```

- `header()` → fungsi PHP untuk mengirim HTTP header ke browser
- `Content-Type: application/octet-stream` → memberitahu browser: "data yang kamu terima adalah file biner mentah, bukan HTML, bukan gambar, bukan JSON". Browser pun tidak mencoba menampilkan, tapi memperlakukannya sebagai data yang perlu diproses oleh JavaScript

---

```
        header('Content-Length: ' . $fileSize);
```

Memberitahu browser ukuran total file yang akan diterima (dalam byte). Berguna agar browser bisa menghitung progress download.

```
        header('Cache-Control: no-cache');
```

Memerintahkan browser untuk **tidak menyimpan** file ini di cache. Penting untuk file terenkripsi — kita tidak mau browser menyimpan data sensitif.

```
        header('X-Accel-Buffering: no');
```

Header khusus untuk server Nginx (juga berpengaruh di beberapa konfigurasi hosting). Memerintahkan server untuk **tidak menahan data** sebelum dikirim ke browser — data harus langsung diteruskan saat tersedia.

```
        while (!feof($handle)) {
```

- `while` → perulangan yang terus berjalan selama kondisinya true
- `feof()` → singkatan dari "file end of file". Mengembalikan `true` jika sudah sampai akhir file
- `!feof($handle)` → "selama belum sampai akhir file, terus ulangi"
- Jadi perulangan ini akan terus berjalan sampai seluruh file habis dibaca

---

```
            $chunk = fread($handle, $this->chunkSize);
```

- `fread()` → membaca sebagian data dari file yang sudah dibuka
- Argumen kedua `$this->chunkSize` = berapa byte yang dibaca sekali jalan (256KB)
- `$chunk` = potongan data hasil bacaan. Setiap iterasi loop, 256KB data tersimpan di sini

---

```
            echo $chunk;
```

`echo` di sini **langsung mengirim** 256KB data ke browser. Tidak seperti echo HTML biasa yang menampilkan teks, `echo` di dalam streaming context mengirim data biner mentah.

```
            flush();
```

Perintah ke PHP untuk **langsung mengirim** apa yang ada di buffer output ke browser sekarang juga, tidak menunggu. Tanpa `flush()`, PHP mungkin menahan data dulu sebelum mengirim, membuat streaming tidak berjalan mulus.

```
            if (connection_aborted()) break;
```

- `connection_aborted()` → mengecek apakah koneksi dari browser sudah terputus (misalnya user menutup tab atau internet putus)
- `break` → langsung keluar dari loop `while`
- Ini penting untuk efisiensi: kalau browser sudah disconnect, tidak ada gunanya terus membaca file di server

---

```
        fclose($handle);
```

Menutup file yang tadi dibuka dengan `fopen()`. Sama seperti menutup buku setelah selesai dibaca. Penting untuk membebaskan memori server.

```
        exit;
```

Menghentikan eksekusi PHP sepenuhnya. Diperlukan setelah streaming manual agar CI4 tidak mencoba mengirim output tambahan (seperti footer HTML) yang akan merusak data biner.

```
    public function info($videoId = null)
    {
        if (!$videoId) { ... }

        $videoPath = WRITEPATH . 'uploads/encrypted/' . $videoId . '.enc';

        if (!file_exists($videoPath)) { ... }
```

Tiga blok ini sama persis dengan `stream()` — validasi ID dan cek keberadaan file. Tidak diulang penjelasannya.

```
        return $this->response->setJSON([
            'success' => true,
            'data' => [
                'id'           => $videoId,
                'size'         => filesize($videoPath),
                'chunk_size'   => $this->chunkSize,
                'total_chunks' => ceil(filesize($videoPath) / $this->chunkSize),
            ]
        ]);
```

- Mengembalikan JSON berisi informasi file
- `'size' => filesize($videoPath)` → ukuran file dalam byte
- `'chunk_size' => $this->chunkSize` → 262144 (256KB) — berapa besar tiap potongan
- `'total_chunks' => ceil(filesize($videoPath) / $this->chunkSize)` → total potongan yang dibutuhkan
  - Misal file 5MB = 5.242.880 byte, dibagi 262.144 = 20 chunk
  - `ceil()` = pembulatan ke atas. Karena potongan terakhir bisa tidak penuh 256KB

```
    public function getKey()
    {
        return $this->response->setJSON([
            'success' => true,
            'key'     => base64_encode($this->encryptionKey)
        ]);
    }
```

- `base64_encode()` → mengubah string biasa menjadi format **Base64**
- Kenapa Base64? Kunci enkripsi bisa berisi karakter yang tidak aman untuk dikirim lewat JSON (karakter kontrol, spasi, dsb). Base64 mengubahnya jadi karakter-karakter yang aman (A-Z, a-z, 0-9, +, /)
- Di sisi browser, kunci ini akan di-decode kembali ke bentuk aslinya sebelum dipakai

---

```
        $validation = \Config\Services::validation();
```

- `\Config\Services::validation()` → mengambil layanan validasi bawaan CI4
- `\` di depan artinya namespace dimulai dari root (bukan relatif)
- `::` = operator untuk mengakses method/properti static (tanpa perlu membuat objek baru dulu)
- Hasilnya disimpan di `$validation` — objek yang siap dipakai untuk memvalidasi input

---

```
        $validation->setRules([
            'video' =>
'uploaded[video]|max_size[video,102400]|ext_in[video,mp4,avi,mkv,mov]'
        ]);
```

- `setRules()` → mendefinisikan aturan validasi. Menerima array dengan format `'nama_field' => 'aturan'`
- `'video'` → nama field yang divalidasi (harus sama dengan `name` di form HTML)
- Aturan-aturan dipisah dengan `|` (pipe), dibaca kiri ke kanan:
  - `uploaded[video]` → wajib ada file yang diupload di field bernama `video`
  - `max_size[video,102400]` → ukuran maks 102.400 KB = **100MB** (102400 ÷ 1024 = 100)
  - `ext_in[video,mp4,avi,mkv,mov]` → ekstensi file harus salah satu dari daftar ini

```
        if (!$validation->withRequest($this->request)->run()) {
```

- `withRequest($this->request)` → berikan data request (termasuk file yang diupload) ke validator
- `->run()` → jalankan validasi, mengembalikan `true` jika semua aturan lolos, `false` jika ada yang gagal
- `!...->run()` → jika validasi **gagal**, jalankan blok `if`

```
            return $this->response->setJSON([
                'success' => false,
                'errors'  => $validation->getErrors()
            ])->setStatusCode(400);
```

- `$validation->getErrors()` → mengambil semua pesan error validasi dalam bentuk array
- Status code `400` = Bad Request — data yang dikirim user tidak memenuhi syarat

---

```
        $video = $this->request->getFile('video');
```

- `$this->request` → objek request bawaan CI4 berisi semua data yang dikirim user
- `->getFile('video')` → ambil file yang diupload dari field bernama `'video'`
- Hasilnya adalah objek `UploadedFile` yang punya banyak method berguna

```
        if (!$video->isValid()) {
```

- `isValid()` → method dari objek `UploadedFile` untuk mengecek apakah file benar-benar valid (tidak korup, berhasil ter-upload)
- Pengecekan ganda — sudah divalidasi di atas, tapi ini validasi dari sisi file system

---

```
        $videoId = uniqid('vid_', true);
```

- `uniqid()` → fungsi PHP untuk generate ID unik berdasarkan waktu (microsecond)
- `'vid_'` → prefix yang ditambahkan di depan ID. Hasilnya: `vid_67abc123def456`
- `true` → parameter kedua, jika `true` menambahkan angka acak ekstra di belakang agar lebih unik lagi

---

```
        $uploadPath    = WRITEPATH . 'uploads/original/';
        $encryptedPath = WRITEPATH . 'uploads/encrypted/';
```

Mendefinisikan dua path folder: satu untuk menyimpan file asli sementara, satu lagi untuk file terenkripsi.

```
        if (!is_dir($uploadPath)) mkdir($uploadPath, 0755, true);
        if (!is_dir($encryptedPath)) mkdir($encryptedPath, 0755, true);
```

- `is_dir()` → cek apakah folder sudah ada
- `mkdir()` → buat folder jika belum ada
  - Argumen pertama = path folder yang dibuat
  - `0755` = permission folder (angka diawali `0` artinya format oktal). `755` berarti: pemilik bisa baca/tulis/eksekusi, orang lain hanya baca/eksekusi
  - `true` = buat folder secara rekursif (buat folder induk jika belum ada)
- Dengan pola `if (!is_dir(...)) mkdir(...)` dalam satu baris — jika folder belum ada, buat; jika sudah ada, lewati

---

```
        $originalFile  = $uploadPath . $videoId . '.' . $video->getExtension();
        $encryptedFile = $encryptedPath . $videoId . '.enc';
```

- `$originalFile` → path lengkap file asli, contoh: `.../original/vid_xxx.mp4`
- `$video->getExtension()` → mengambil ekstensi file yang diupload (mp4, avi, dsb)
- `$encryptedFile` → path lengkap file terenkripsi, contoh: `.../encrypted/vid_xxx.enc`
- Ekstensi `.enc` adalah konvensi — bukan format standar, hanya penanda bahwa file ini terenkripsi

---

```
        $video->move($uploadPath, $videoId . '.' . $video->getExtension());
```

- `->move()` → memindahkan file dari lokasi sementara (temp upload PHP) ke folder tujuan
  - Argumen pertama = folder tujuan
  - Argumen kedua = nama file baru di tujuan
- PHP menyimpan file upload di folder temp dulu, baru dipindah ke lokasi permanen dengan `move()`

```
        $this->encryptFile($originalFile, $encryptedFile);
```

Memanggil method private `encryptFile()` yang ada di bawah — mengenkripsi file asli dan menyimpan hasilnya sebagai file `.enc`.

```
        unlink($originalFile);
```

- `unlink()` → menghapus file dari server (seperti delete di file manager)
- File asli dihapus setelah dienkripsi — tidak ada gunanya menyimpan dua versi

---

```
        return $this->response->setJSON([
            'success' => true,
            'message' => 'Video uploaded and encrypted',
            'data'    => [
                'video_id' => $videoId,
                'size'     => filesize($encryptedFile)
            ]
        ]);
```

Mengembalikan JSON sukses ke browser, berisi `video_id` yang akan ditampilkan ke user untuk disimpan.

```
    private function encryptFile($source, $destination)
    {
```

- `private` → method ini hanya bisa dipanggil dari dalam class yang sama, tidak bisa dari luar. Cocok untuk fungsi internal yang tidak perlu diakses dari routing
- `$source` = path file asli (input)
- `$destination` = path file hasil enkripsi (output)

```
        $iv = openssl_random_pseudo_bytes(16);
```

- `openssl_random_pseudo_bytes(16)` → menghasilkan **16 byte data acak** yang sangat aman secara kriptografi
- `$iv` = **Initialization Vector** — nilai acak yang dibutuhkan oleh algoritma AES-CBC
- Kenapa perlu IV? AES-CBC menggunakan IV sebagai "garam" awal enkripsi. Dua file yang sama akan menghasilkan enkripsi berbeda jika IV-nya berbeda. Ini sangat penting untuk keamanan
- IV harus **16 byte** persis untuk AES (AES bekerja dengan blok 128-bit = 16 byte)

```
        $plaintext = file_get_contents($source);
```

- `file_get_contents()` → membaca seluruh isi file sekaligus ke dalam sebuah string/variabel
- `$plaintext` = data mentah video yang belum dienkripsi
- ⚠ Untuk file besar (100MB), ini berarti 100MB data masuk ke memori PHP sekaligus — ini kelemahan implementasi saat ini

---

```
        $encrypted = openssl_encrypt(
            $plaintext,          // data yang akan dienkripsi
            'aes-256-cbc',       // algoritma enkripsi
            $this->encryptionKey, // kunci rahasia
            OPENSSL_RAW_DATA,    // flag output
            $iv                  // initialization vector
        );
```

- `openssl_encrypt()` → fungsi PHP untuk enkripsi menggunakan OpenSSL
- `'aes-256-cbc'` → nama algoritma. AES = Advanced Encryption Standard, 256 = panjang kunci dalam bit, CBC = Cipher Block Chaining (mode operasi)
- `OPENSSL_RAW_DATA` → konstanta yang memberitahu fungsi untuk mengembalikan data biner mentah (bukan Base64). Kita simpan sendiri sebagai binary karena lebih efisien

---

```
        $destHandle = fopen($destination, 'wb');
```

- Membuka file **tujuan** untuk **ditulis** (`w` = write, `b` = binary)
- Mode `w` akan membuat file baru jika belum ada, atau menimpa jika sudah ada

---

```
        fwrite($destHandle, $iv);
```

- `fwrite()` → menulis data ke file yang sudah dibuka
- **IV ditulis pertama** ke file `.enc` — 16 byte pertama
- Ini penting: browser perlu IV yang sama untuk mendekripsi. Dengan menyimpannya di awal file, browser bisa mengambilnya kembali tanpa perlu mekanisme penyimpanan terpisah

---

```
        fwrite($destHandle, $encrypted);
```

Data terenkripsi ditulis setelah IV. Struktur file `.enc` jadi:

```
[ 16 byte IV ] [ ... data terenkripsi ... ]
```

```
        fclose($destHandle);
    }
}
```

- `fclose()` → menutup file, memastikan semua data tertulis ke disk dan memori dibebaskan
- Kurung kurawal `}` pertama menutup method `encryptFile()`, kedua menutup class `VideoStream1`

4.2 View Player — `app/Views/video/embed_video.php`

Buat file baru dengan isi:

```
<!DOCTYPE html>
<html lang="id">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Video Player</title>
    <style>
        * {
            margin: 0;
            padding: 0;
            box-sizing: border-box;
            user-select: none;
            -webkit-user-select: none;
            -moz-user-select: none;
            -ms-user-select: none;
        }

        html,
        body {
            width: 100%;
            height: 100%;
            overflow: hidden;
            background: #000;
        }

        .player-container {
            width: 100%;
            height: 100%;
            position: relative;
        }

        video {
            width: 100%;
            height: 100%;
            display: block;
            object-fit: contain;
            pointer-events: auto;
        }

        video::-webkit-media-controls-download-button {
            display: none !important;
        }
```

```
        video::-webkit-media-controls-enclosure {
            overflow: hidden;
        }

        video::-internal-media-controls-download-button {
            display: none !important;
        }

        /* ==================== LOADING OVERLAY ==================== */

        .loading-overlay {
            position: absolute;
            top: 0;
            left: 0;
            right: 0;
            bottom: 0;
            background: rgba(0, 0, 0, 0.92);
            display: flex;
            flex-direction: column;
            align-items: center;
            justify-content: center;
            z-index: 1000;
            transition: opacity 0.4s ease;
        }

        .loading-overlay.hidden {
            opacity: 0;
            pointer-events: none;
        }

        .progress-circle {
            position: relative;
            width: 100px;
            height: 100px;
        }

        .progress-ring {
            transform: rotate(-90deg);
        }

        .progress-ring-circle {
            stroke-dasharray: 282.743;
            stroke-dashoffset: 282.743;
            transition: stroke-dashoffset 0.3s ease;
            stroke: #667eea;
            stroke-width: 6;
            fill: none;
            stroke-linecap: round;
        }

        .progress-text {
            position: absolute;
            top: 50%;
            left: 50%;
```

```
            transform: translate(-50%, -50%);
            font-size: 20px;
            font-weight: bold;
            color: #667eea;
            font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Arial,
sans-serif;
        }

        .loading-label {
            margin-top: 20px;
            color: rgba(255, 255, 255, 0.75);
            font-size: 13px;
            font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Arial,
sans-serif;
            letter-spacing: 0.5px;
        }

        /* ==================== ERROR OVERLAY ==================== */

        .error-overlay {
            position: absolute;
            top: 50%;
            left: 50%;
            transform: translate(-50%, -50%);
            background: rgba(220, 53, 69, 0.95);
            color: white;
            padding: 20px 30px;
            border-radius: 10px;
            text-align: center;
            display: none;
            z-index: 1001;
            font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Arial,
sans-serif;
            min-width: 220px;
        }

        .error-overlay.show {
            display: block;
        }

        .error-title {
            font-size: 15px;
            font-weight: bold;
            margin-bottom: 8px;
        }

        .error-message {
            font-size: 13px;
            opacity: 0.9;
        }

        /* Block right-click layer */
        .no-context {
            position: absolute;
```

```
            top: 0;
            left: 0;
            right: 0;
            bottom: 0;
            z-index: 999;
            pointer-events: none;
        }
    </style>
</head>

<body>
    <div class="player-container">

        <video id="videoPlayer"
            autoplay
            controls
            playsinline
            webkit-playsinline
            controlsList="nodownload noremoteplayback"
            disablePictureInPicture
            oncontextmenu="return false;">
        </video>

        <!-- Layer pemblokir klik kanan -->
        <div class="no-context"></div>

        <!-- Loading overlay dengan progress lingkaran -->
        <div class="loading-overlay" id="loadingOverlay">
            <div class="progress-circle">
                <svg class="progress-ring" width="100" height="100">
                    <circle class="progress-ring-circle" r="45" cx="50" cy="50"
id="progressCircle" />
                </svg>
                <div class="progress-text" id="progressPercent">0%</div>
            </div>
            <div class="loading-label" id="loadingLabel">Loading...</div>
        </div>

        <!-- Error overlay -->
        <div class="error-overlay" id="errorOverlay">
            <div class="error-title">⚠ Error</div>
            <div class="error-message" id="errorMessage"></div>
        </div>

    </div>

    <script>
        // ================================================================
        // KONFIGURASI
        // Video ID diambil dari URL param ?id=xxx
        // Contoh embed: <iframe src="/video/player?id=vid_xxx"></iframe>
        // ================================================================

        const urlParams  = new URLSearchParams(window.location.search);
```

```
        const videoId    = urlParams.get('id') || '<?= $videoId ?? "" ?>';
        const API_BASE   = '<?= base_url("api/videos") ?>';

        // Referensi elemen DOM
        const video          = document.getElementById('videoPlayer');
        const loadingOverlay = document.getElementById('loadingOverlay');
        const progressCircle = document.getElementById('progressCircle');
        const progressPercent = document.getElementById('progressPercent');
        const loadingLabel   = document.getElementById('loadingLabel');
        const errorOverlay   = document.getElementById('errorOverlay');
        const errorMessage   = document.getElementById('errorMessage');

        // State
        let decryptionKey = null;  // CryptoKey object (Web Crypto API)

        // ================================================================
        // DISABLE RIGHT-CLICK, DOWNLOAD, SHORTCUT KEYBOARD
        // ================================================================

        // Blokir klik kanan di seluruh halaman
        document.addEventListener('contextmenu', function(e) {
            e.preventDefault();
            return false;
        }, false);

        // Blokir klik kanan khusus di elemen video
        video.addEventListener('contextmenu', function(e) {
            e.preventDefault();
            return false;
        });

        // Blokir shortcut keyboard yang umum dipakai untuk inspect/download
        document.addEventListener('keydown', function(e) {
            if (
                e.keyCode === 123 ||                                  // F12
                (e.ctrlKey && e.shiftKey && e.keyCode === 73) ||  // Ctrl+Shift+I
                (e.ctrlKey && e.shiftKey && e.keyCode === 74) ||  // Ctrl+Shift+J
                (e.ctrlKey && e.keyCode === 85) ||                 // Ctrl+U
                (e.ctrlKey && e.keyCode === 83)                    // Ctrl+S
            ) {
                e.preventDefault();
                return false;
            }
        });

        // Blokir seleksi teks
        document.onselectstart = function() { return false; };

        // Blokir drag pada elemen video
        video.addEventListener('dragstart', function(e) {
            e.preventDefault();
            return false;
        });
```

```
        // Pastikan tidak ada atribut download
        video.removeAttribute('download');

        // ================================================================
        // PROGRESS & ERROR HELPERS
        // ================================================================

        /**
         * Memperbarui tampilan progress lingkaran SVG.
         * @param {number} percent  - Nilai 0–100
         * @param {string} label    - Teks label di bawah lingkaran (opsional)
         */
        function updateProgress(percent, label = '') {
            const circumference = 2 * Math.PI * 45; // 282.743
            const offset = circumference - (percent / 100) * circumference;
            progressCircle.style.strokeDashoffset = offset;
            progressPercent.textContent = Math.round(percent) + '%';
            if (label) loadingLabel.textContent = label;
        }

        /**
         * Menampilkan overlay error dan menyembunyikan loading.
         * @param {string} msg - Pesan error untuk ditampilkan ke user
         */
        function showError(msg) {
            loadingOverlay.classList.add('hidden');
            errorMessage.textContent = msg;
            errorOverlay.classList.add('show');
        }

        // ================================================================
        // HELPER: Base64 → ArrayBuffer
        // Dibutuhkan karena Web Crypto API bekerja dengan ArrayBuffer,
        // bukan string. Server mengirim kunci dalam format Base64.
        // ================================================================

        function base64ToArrayBuffer(base64) {
            const binary = atob(base64);
            const bytes  = new Uint8Array(binary.length);
            for (let i = 0; i < binary.length; i++) {
                bytes[i] = binary.charCodeAt(i);
            }
            return bytes.buffer;
        }

        // ================================================================
        // STEP 1 — AMBIL KUNCI ENKRIPSI DARI SERVER
        // Endpoint: GET /api/videos/key
        // Response: { success: true, key: "<base64 string>" }
        //
        // Catatan: Kunci dari server dalam format Base64.
        // Kita import ke Web Crypto API sebagai CryptoKey AES-CBC
        // agar bisa dipakai untuk dekripsi di browser.
        // ================================================================
```

```
        async function getLicense() {
            try {
                updateProgress(5, 'Securing connection...');

                const response = await fetch(`${API_BASE}/key`);
                const data     = await response.json();

                if (!data.success) throw new Error(data.message || 'Key request
failed');

                // Konversi Base64 → ArrayBuffer
                const keyData = base64ToArrayBuffer(data.key);

                // Import kunci ke Web Crypto API
                // 'raw'    = format kunci mentah (ArrayBuffer)
                // AES-CBC  = algoritma yang sama dengan yang dipakai PHP
(openssl_encrypt)
                // false    = kunci tidak bisa diekspor kembali
                // decrypt  = kunci hanya boleh dipakai untuk dekripsi
                decryptionKey = await crypto.subtle.importKey(
                    'raw',
                    keyData,
                    { name: 'AES-CBC' },
                    false,
                    ['decrypt']
                );

                return true;

            } catch (error) {
                console.error('License error:', error);
                showError('Failed to get license. Please try again.');
                return false;
            }
        }

        // ================================================================
        // STEP 2 — AMBIL METADATA VIDEO
        // Endpoint: GET /api/videos/info/{videoId}
        // Response: { success: true, data: { id, size, chunk_size, total_chunks }
}
        //
        // Dipakai untuk menampilkan info dan mengupdate progress bar
        // dengan akurat berdasarkan ukuran file.
        // ================================================================

        async function getMetadata() {
            try {
                updateProgress(10, 'Loading info...');

                const response = await fetch(`${API_BASE}/info/${videoId}`);
                const result   = await response.json();
```

```
                if (!result.success) throw new Error(result.message || 'Metadata
request failed');

                return result.data;

            } catch (error) {
                console.error('Metadata error:', error);
                showError('Video not found or unavailable.');
                return null;
            }
        }

        // ================================================================
        // STEP 3 — DOWNLOAD FILE .enc & DEKRIPSI DI BROWSER
        // Endpoint: GET /api/videos/stream/{videoId}
        //
        // Alur:
        // 1. Fetch seluruh file .enc sebagai ArrayBuffer
        // 2. Ambil 16 byte pertama sebagai IV
        //    (IV disimpan di awal file oleh PHP saat enkripsi)
        // 3. Sisa byte = data terenkripsi
        // 4. Dekripsi dengan AES-CBC menggunakan kunci + IV
        // ================================================================

        async function downloadAndDecrypt(metadata) {
            try {
                updateProgress(15, 'Downloading...');

                const response = await fetch(`${API_BASE}/stream/${videoId}`);

                if (!response.ok) throw new Error(`Stream request failed:
${response.status}`);

                // Baca seluruh response sebagai data biner (ArrayBuffer)
                // Lalu bungkus dalam Uint8Array agar bisa diakses per-byte
                const buffer        = await response.arrayBuffer();
                const encryptedData = new Uint8Array(buffer);

                updateProgress(70, 'Decrypting...');

                // Struktur file .enc: [16 byte IV][data terenkripsi]
                // slice(0, 16) → ambil 16 byte pertama sebagai IV
                // slice(16)    → ambil semua byte setelah byte ke-16 sebagai data
                const iv               = encryptedData.slice(0, 16);
                const encryptedContent  = encryptedData.slice(16);

                // Dekripsi menggunakan Web Crypto API
                // AES-CBC membutuhkan IV yang sama dengan yang dipakai saat
enkripsi
                const decrypted = await crypto.subtle.decrypt(
                    {
                        name: 'AES-CBC',
                        iv: iv          // IV yang diambil dari 16 byte pertama
file
```

```
                    },
                    decryptionKey,      // CryptoKey yang sudah di-import di
getLicense()
                    encryptedContent    // Data yang akan didekripsi
                );

                return decrypted;

            } catch (error) {
                console.error('Download/decrypt error:', error);
                showError('Failed to decrypt video. Please contact support.');
                return null;
            }
        }

        // ================================================================
        // STEP 4 — BUAT BLOB URL & PUTAR VIDEO
        // Blob URL adalah URL sementara (blob:https://...) yang menunjuk
        // langsung ke data biner di memori browser.
        // URL ini hanya valid selama tab masih terbuka.
        // ================================================================

        function loadVideo(decryptedData) {
            updateProgress(95, 'Preparing...');

            // Bungkus data yang sudah didekripsi dalam objek Blob
            // type: 'video/mp4' memberitahu browser cara memproses data ini
            const blob     = new Blob([decryptedData], { type: 'video/mp4' });
            const videoUrl = URL.createObjectURL(blob);

            // Set Blob URL sebagai sumber elemen <video>
            video.src = videoUrl;
            video.load();

            // Ketika video sudah siap dimuat
            video.addEventListener('loadeddata', () => {
                updateProgress(100, 'Ready!');

                // Coba sembunyikan tombol download dari shadow DOM browser
                setTimeout(() => {
                    const downloadBtn =
video.shadowRoot?.querySelector('[part="download-button"]');
                    if (downloadBtn) downloadBtn.style.display = 'none';
                }, 100);

                // Sembunyikan loading overlay dengan sedikit jeda
                // agar user sempat melihat "100%"
                setTimeout(() => {
                    loadingOverlay.classList.add('hidden');
                    video.play().catch(() => {
                        // Autoplay mungkin diblokir browser — tidak apa-apa,
                        // user bisa klik play secara manual
                    });
                }, 300);
```

```
            }, { once: true }); // { once: true } = event listener ini hanya
dijalankan sekali
        }

        // ================================================================
        // INISIALISASI — Dijalankan saat halaman selesai dimuat
        // Urutan: getLicense → getMetadata → downloadAndDecrypt → loadVideo
        // ================================================================

        async function init() {
            // Validasi: Video ID harus ada
            if (!videoId) {
                showError('Video ID required. Use: ?id=video_xxx');
                return;
            }

            try {
                // Step 1: Ambil dan import kunci enkripsi
                if (!await getLicense()) return;

                // Step 2: Ambil metadata (untuk info dan progress)
                const metadata = await getMetadata();
                if (!metadata) return;

                // Step 3: Download file .enc dan dekripsi di browser
                const decryptedData = await downloadAndDecrypt(metadata);
                if (!decryptedData) return;

                // Step 4: Buat Blob URL dan putar
                loadVideo(decryptedData);

            } catch (error) {
                console.error('Init error:', error);
                showError(error.message || 'Failed to load video.');
            }
        }

        // Jalankan init saat DOM siap
        window.addEventListener('DOMContentLoaded', init);

        // ================================================================
        // KOMUNIKASI IFRAME ↔ PARENT (untuk embed)
        // Saat video siap diputar, kirim pesan ke halaman induk
        // yang meng-embed player ini via <iframe>.
        //
        // video.addEventListener('canplaythrough', () => {
        //    parent.postMessage({ type: 'VIDEO_READY' }, '*');
        // });
        //
        // Contoh penggunaan di halaman induk:
        // window.addEventListener('message', (e) => {
        //     if (e.data.type === 'VIDEO_READY') console.log('Player siap!');
        // });
```

```
        // ================================================================

    </script>
</body>

</html>
```

4.3 View Upload — `app/Views/video/upload1.php`

Buat file baru (bebas layout dan style) dengan isi:

> Note: Pahami dulu kode dibawah untuk meminimalisir error, kalau sudah paham bisa implementasi ke kode html atau php kamu

```
<!DOCTYPE html>
<html lang="id">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Upload Video</title>
    <style>
        body { font-family: sans-serif; max-width: 600px; margin: 40px auto;
padding: 20px; }
        input[type=file] { display: block; margin: 10px 0; }
        button { padding: 10px 20px; background: #667eea; color: white; border:
none;
                 border-radius: 6px; cursor: pointer; font-size: 14px; }
        button:disabled { opacity: 0.5; cursor: not-allowed; }
        #result { margin-top: 20px; padding: 15px; border-radius: 8px; background:
#f0f0f0; }
        #progress { display: none; margin-top: 10px; }
        progress { width: 100%; height: 20px; }
    </style>
</head>
<body>
    <h1>📤 Upload & Enkripsi Video</h1>
    <p>Format: MP4, AVI, MKV, MOV | Maks: 100MB</p>

    <input type="file" id="videoFile" accept=".mp4,.avi,.mkv,.mov">
    <button onclick="uploadVideo()" id="uploadBtn">Upload & Enkripsi</button>

    <div id="progress">
        <p id="progressText">Mengupload...</p>
        <progress id="progressBar" value="0" max="100"></progress>
    </div>

    <div id="result" style="display:none;"></div>

    <script>
    async function uploadVideo() {
        const file = document.getElementById('videoFile').files[0];
```

```
        if (!file) { alert('Pilih file video dulu!'); return; }

        const btn      = document.getElementById('uploadBtn');
        const progress = document.getElementById('progress');
        const result   = document.getElementById('result');

        btn.disabled           = true;
        progress.style.display = 'block';
        result.style.display   = 'none';

        const formData = new FormData();
        formData.append('video', file);

        const xhr = new XMLHttpRequest();
        xhr.open('POST', '<?= base_url("video/upload") ?>');

        xhr.upload.onprogress = function(e) {
            if (e.lengthComputable) {
                const pct = Math.round((e.loaded / e.total) * 100);
                document.getElementById('progressBar').value = pct;
                document.getElementById('progressText').textContent =
                    pct < 100 ? `Mengupload: ${pct}%` : 'Mengenkripsi... harap
tunggu';
            }
        };

        xhr.onload = function() {
            btn.disabled           = false;
            progress.style.display = 'none';
            result.style.display   = 'block';
            try {
                const data = JSON.parse(xhr.responseText);
                if (data.success) {
                    result.style.background = '#e8f5e9';
                    result.innerHTML = `
                        <h3>☑ Berhasil!</h3>
                        <p><strong>Video ID:</strong></p>
                        <code style="font-
size:16px;background:#fff;padding:8px;display:block;border-
radius:4px;">${data.data.video_id}</code>
                        <p style="margin-top:10px;">Simpan ID ini untuk memutar
video.</p>
                        <p><a href="<?= base_url('video/player') ?>">🎬 Buka
Video Player</a></p>`;
                } else {
                    result.style.background = '#ffebee';
                    result.innerHTML = '<h3>❌ Gagal Upload</h3><pre>' +
                        JSON.stringify(data.errors || data.message, null, 2) +
'</pre>';
                }
            } catch(e) {
                result.style.background = '#ffebee';
                result.innerHTML = '<h3>❌ Error parsing response</h3><pre>' +
xhr.responseText + '</pre>';
```

```
            }
        };

        xhr.onerror = function() {
            btn.disabled           = false;
            progress.style.display = 'none';
            result.style.display   = 'block';
            result.style.background = '#ffebee';
            result.innerHTML = '<h3>❌ Network Error</h3><p>Koneksi terputus saat
upload.</p>';
        };

        xhr.send(formData);
    }
    </script>
</body>
</html>
```

---

5. Konfigurasi Lokal (.env)

Salin file `.env.example` menjadi `.env`:

```
cp env .env
```

Edit file `.env`:

```
# Mode development untuk lokal
CI_ENVIRONMENT = development

# Kunci enkripsi video — buat sendiri, 32 karakter
VIDEO_ENCRYPTION_KEY = KunciRahasiaLokal32KarakterDisin
```

Generate kunci video:

```
php -r "echo bin2hex(random_bytes(16));"
# Copy hasilnya (32 karakter) → paste sebagai nilai VIDEO_ENCRYPTION_KEY
```

---

6. Pengaturan Routing

Edit `app/Config/Routes.php`, tambahkan sebelum baris `$routes->get('/', 'Home::index');`:

```
// Halaman UI
$routes->get('video/upload', 'VideoStream1::index');
```

```
$routes->post('video/upload', 'VideoStream1::doUpload');
$routes->get('video/player', 'VideoStream1::player');

// API Endpoints
$routes->get('api/videos/stream/(:segment)', 'VideoStream1::stream/$1');
$routes->get('api/videos/info/(:segment)',   'VideoStream1::info/$1');
$routes->get('api/videos/key',               'VideoStream1::getKey');
```

---

7. Test di Lokal

Jalankan server lokal:

```
# Cara 1: PHP built-in server (paling simpel)
php spark serve
# Akses: http://localhost:8080

# Cara 2: lewat XAMPP/Laragon
# Akses: http://localhost/videostreaming/public
```

Checklist test lokal — pastikan semua ☑ sebelum lanjut ke deploy:

- ☐ Buka `http://localhost:8080` → muncul halaman CodeIgniter
- ☐ Buka `http://localhost:8080/video/upload` → muncul form upload
- ☐ Upload video kecil (misal 5MB MP4) → muncul Video ID
- ☐ Buka `http://localhost:8080/video/player` → masukkan Video ID → video bisa diputar
- ☐ Buka `http://localhost:8080/api/videos/key` → muncul JSON dengan key
- ☐ Cek folder `writable/uploads/encrypted/` → ada file `.enc`
- ☐ Cek folder `writable/uploads/original/` → **kosong** (file asli terhapus setelah enkripsi)

> 🔴 **Jangan lanjut ke deploy jika ada checklist yang belum ☑.** Debug dulu di lokal — jauh lebih mudah daripada debugging di server.

FASE 2 — Deploy ke cPanel

> Asumsi: website CodeIgniter 4 kamu **sudah berjalan** di cPanel. Ini hanya penambahan fitur baru, bukan instalasi dari nol. Karena itu tidak ada langkah setup server dari awal — fokusnya hanya pada file dan konfigurasi yang perlu ditambahkan/diubah.

---

8. Persiapan Sebelum Upload

Lakukan ini di komputer lokal sebelum mengupload file ke server.

8.1 Bersihkan File Sementara

Hapus file-file ini agar tidak ikut ter-upload ke server:

```
# Hapus video hasil test lokal
rm -rf writable/uploads/original/*
rm -rf writable/uploads/encrypted/*

# Hapus cache CI4
rm -rf writable/cache/*
rm -rf writable/logs/*
rm -rf writable/session/*
```

8.2 Buat File `.htaccess` Proteksi Folder Encrypted

Buat file baru di `writable/uploads/encrypted/.htaccess`:

```
Options -Indexes
Deny from all
```

File ini akan ikut ter-upload dan memastikan file `.enc` tidak bisa diakses langsung lewat browser.

8.3 Identifikasi File yang Perlu Diupload

Karena ini **penambahan fitur** (bukan deploy pertama), kamu tidak perlu upload ulang seluruh project. Hanya file-file baru dan yang berubah saja:

File	Status	Keterangan
`app/Controllers/VideoStream1.php`	🆕 Baru	Controller utama fitur ini
`app/Views/video/upload1.php`	🆕 Baru	Halaman upload
`app/Views/video/embed_video.php`	🆕 Baru	Halaman player
`app/Config/Routes.php`	✏️ Diubah	Ditambahkan route baru
`writable/uploads/original/`	🆕 Folder baru	Buat di server jika belum ada
`writable/uploads/encrypted/`	🆕 Folder baru	Buat di server jika belum ada
`writable/uploads/encrypted/.htaccess`	🆕 Baru	Proteksi folder .enc

> 💡 **Jangan** upload ulang `vendor/`, `system/`, `index.php`, atau file CI4 lainnya yang tidak berubah — itu hanya membuang waktu dan berisiko menimpa konfigurasi yang sudah ada di server.

---

9. Upload File ke cPanel

Bagian ini sudah kamu ketahui — upload file-file di tabel 8.3 ke lokasi yang sesuai via **File Manager** cPanel.

---

10. Yang Harus Dilakukan di cPanel

Ini semua yang perlu dikonfigurasi di cPanel setelah file terupload.

☑ Langkah 1: Pastikan Ekstensi `openssl` Aktif

Fitur enkripsi video ini bergantung pada ekstensi `openssl`. Cek apakah sudah aktif:

cPanel → Select PHP Version → tab Extensions

Cari `openssl` → pastikan **tercentang** ☑ → klik **Save** jika baru dicentang.

> Jika menu Extensions tidak tersedia, buat file `phpinfo.php` sementara di `public_html`, akses lewat browser, cari bagian **openssl**. Hapus file setelah selesai.

☑ Langkah 2: Tambahkan Ekstensi `fileinfo`

Ekstensi ini dibutuhkan CI4 untuk validasi file upload. Sama seperti langkah 1:

cPanel → Select PHP Version → tab Extensions → centang `fileinfo` → Save

---

☑ Langkah 3: Naikkan Limit Upload via MultiPHP INI Editor

Ini langkah terpenting — tanpa ini, upload video akan selalu gagal karena limit bawaan hosting biasanya hanya 2–8MB.

cPanel → MultiPHP INI Editor → tab Editor Mode → pilih domain kamu

Gunakan **Ctrl+F** untuk mencari tiap nama opsi, lalu ubah nilainya:

Opsi	Nilai	Keterangan
`upload_max_filesize`	`512M`	Batas ukuran file upload
`post_max_size`	`512M`	Harus ≥ `upload_max_filesize`
`max_execution_time`	`300`	Maks waktu enkripsi di server (detik)
`max_input_time`	`300`	Maks waktu terima data dari browser
`memory_limit`	`512M`	Memori untuk proses enkripsi

Klik **Save** — perubahan langsung aktif tanpa restart.

> ⚠ `post_max_size` harus **sama atau lebih besar** dari `upload_max_filesize`, jika tidak upload tetap gagal meski `upload_max_filesize` sudah besar.

☑ Langkah 4: Buat Folder Upload & Set Permission

cPanel → File Manager

Jika folder belum ada, buat dulu:

- Navigasi ke `public_html/writable/`
- Buat folder `uploads` → masuk ke dalamnya → buat `original` dan `encrypted`

Lalu set permission dengan klik kanan tiap folder → **Change Permissions**:

Folder	Permission
`writable/uploads/`	`755`
`writable/uploads/original/`	`755`
`writable/uploads/encrypted/`	`755`

> ⚠ Jangan gunakan `777`.

☑ Langkah 5: Tambahkan Baris Proteksi ke `.htaccess`

cPanel → File Manager → buka `public_html/.htaccess` → klik kanan → Edit

`.htaccess` yang sudah ada **jangan dihapus atau diganti seluruhnya**. Cukup **tambahkan** dua blok berikut di bagian paling bawah file:

```
# Blokir akses langsung ke file .env
<Files .env>
    Order allow,deny
    Deny from all
</Files>

# Blokir akses langsung ke folder sensitif
RewriteRule ^(app|writable|system|vendor)(/|$) - [F,L]
```

> 💡 Jika baris `RewriteEngine On` belum ada di `.htaccess`-mu, pastikan baris itu ada sebelum `RewriteRule` di atas bisa berfungsi.

☑ Langkah 6: Tambahkan `VIDEO_ENCRYPTION_KEY` ke `.env` Server

cPanel → File Manager → buka `public_html/.env` → klik kanan → Edit

Tambahkan satu baris baru di bagian paling bawah file `.env` yang sudah ada:

```
VIDEO_ENCRYPTION_KEY = KunciRahasiaVideoKamu32Karakter!
```

Cara generate kunci yang aman (tanpa terminal):

Buat file sementara `public_html/genkey.php`:

```php
<?php
// HAPUS FILE INI SETELAH DIGUNAKAN!
echo bin2hex(random_bytes(16)); // menghasilkan 32 karakter hex
```

Akses `https://domainmu.com/genkey.php` di browser → copy hasilnya → paste sebagai nilai `VIDEO_ENCRYPTION_KEY` di `.env` → **hapus `genkey.php` segera!**

> 🔑 **PENTING:** Setelah kunci ini dipakai untuk mengenkripsi video di server, **jangan pernah diubah lagi**. Jika diubah, semua video yang sudah terupload tidak akan bisa diputar.

---

11. Verifikasi Setelah Deploy

Buka browser dan test satu per satu. Semua harus ☑ sebelum fitur dianggap siap digunakan.

- ☐ `https://domainmu.com/video/upload` → muncul form upload (bukan 404 atau error)
- ☐ `https://domainmu.com/video/player` → muncul halaman player
- ☐ `https://domainmu.com/api/videos/key` → muncul JSON `{"success":true,"key":"..."}`
- ☐ Upload video kecil (5MB MP4) → muncul Video ID, tidak timeout
- ☐ Masukkan Video ID di player → video bisa diputar
- ☐ `https://domainmu.com/writable/` → muncul **403 Forbidden**
- ☐ `https://domainmu.com/.env` → muncul **403 Forbidden**

---

12. Verifikasi Setelah Deploy

❌ Route `/video/upload` tidak ditemukan (404)

Penyebab: `Routes.php` belum terupload atau ada konflik route.

Solusi: Pastikan `Routes.php` yang sudah ditambahkan route baru sudah ter-upload ke server. Buka file tersebut via File Manager dan periksa apakah route sudah ada.

❌ Upload video gagal / langsung error

Penyebab: Limit PHP belum berubah.

Solusi: Ulangi Langkah 3. Pastikan kamu sudah klik **Save** di MultiPHP INI Editor. Coba cek nilai saat ini lewat file `phpinfo.php` sementara — cari `upload_max_filesize` dan lihat apakah nilainya sudah `512M`.

❌ Video tidak bisa diputar setelah upload berhasil (layar hitam)

Penyebab & solusi:

Penyebab	Cara Cek	Solusi
`VIDEO_ENCRYPTION_KEY` belum ada di `.env`	Buka `.env` server, cek apakah baris itu ada	Tambahkan sesuai Langkah 6

Penyebab	Cara Cek	Solusi
Kunci kosong atau salah format	Pastikan panjang kunci tepat 32 karakter	Generate ulang dengan `genkey.php`
File `.enc` korup karena timeout	Upload gagal di tengah jalan	Naikkan `max_execution_time`, upload ulang

Cek Console browser (F12 → Console) untuk detail:

- `DOMException: The operation failed` → masalah kunci enkripsi
- `MediaError code 4` → format video tidak didukung browser, konversi ke MP4 H.264

❌ `https://domainmu.com/writable/` bisa diakses (tidak 403)

Penyebab: Baris proteksi belum ditambahkan ke `.htaccess` atau `RewriteEngine` belum aktif.

Solusi:

1. Pastikan dua blok dari Langkah 5 sudah ada di `.htaccess`
2. Pastikan `RewriteEngine On` ada di `.htaccess` sebelum baris `RewriteRule`
3. Jika masih tidak berhasil, buat `.htaccess` terpisah langsung di dalam folder `writable/`:

```
Options -Indexes
Deny from all
```

📋 Ringkasan Checklist Deploy (Penambahan Fitur)

```
PERSIAPAN LOKAL
  [ ] File sampah sudah dibersihkan (cache, log, video test)
  [ ] writable/uploads/encrypted/.htaccess sudah dibuat

UPLOAD KE SERVER (file baru/berubah saja)
  [ ] app/Controllers/VideoStream1.php
  [ ] app/Views/video/upload1.php
  [ ] app/Views/video/embed_video.php
  [ ] app/Config/Routes.php
  [ ] writable/uploads/encrypted/.htaccess

DI cPanel
  [ ] Ekstensi openssl sudah aktif
  [ ] Ekstensi fileinfo sudah aktif
  [ ] MultiPHP INI Editor: upload_max_filesize & post_max_size = 512M
  [ ] MultiPHP INI Editor: max_execution_time & memory_limit = 300 & 512M
  [ ] Folder writable/uploads/original/ dan encrypted/ sudah ada, permission 755
  [ ] .htaccess — dua blok proteksi sudah ditambahkan di bagian bawah
  [ ] .env server — baris VIDEO_ENCRYPTION_KEY sudah ditambahkan
```

```
VERIFIKASI
  [ ] /video/upload bisa diakses
  [ ] /api/videos/key menampilkan JSON
  [ ] Upload video berhasil dan muncul Video ID
  [ ] Video bisa diputar di player
  [ ] /writable/ menampilkan 403 Forbidden
  [ ] /.env menampilkan 403 Forbidden
```

> 📝 **Setelah semua berfungsi:** Tambahkan autentikasi pada `getKey()` dan `doUpload()` di controller sebelum fitur ini dibuka untuk umum. Lihat komentar `// TODO` di `VideoStream1.php`.