Republic of Iraq
Ministry of Higher Education and Scientific Research
University of Wasit
College of Education for Pure Sciences
Department of Computer Science

Efficient Compressed Sensing-Based Security Approach for Video Surveillance Application in Wireless Multimedia Sensor Networks

نهج أمني الفعال الذي يعتمد على الاستشعار المضغوط لتطبيقات المراقبة بالفيديو في شبكات الاستشعار اللاسلكية المتعددة الوسائط

Mini Project for Multimedia

By:
Mahmoud Shamran Atheeb
Ali Jabbar Sharhan
Qusay Joudah Shaheen
Azhar Abdalhmeed Jafer

Supervised By:
Assit. Prof .Dr. Baraa Ismael Farhan

2024 - 2025

الملخص

يقترح هذا البحث نظامًا متكاملاً يجمع بين تقنيات الضغط الاستشعاري (Compressed Sensing) وطرق التشفير المتقدمة بهدف تأمين نقل إطارات الفيديو عبر شبكات الاستشعار متعددة الوسائط اللاسلكية (Wireless Multimedia Sensor Networks). يعتمد النظام على خطوات محددة تبدأ بضغط حجم البيانات بنسبة تصل إلى 83.5%، مما يقلل من عرض النطاق الترددي المطلوب للإرسال ويزيد من كفاءة النقل. بعد الضغط، يتم تطبيق طبقة أمان إضافية باستخدام التشفير القوي لحماية سرية البيانات من الهجمات المحتملة.

تم تنفيذ النظام باستخدام إطار عمل Django الخاص بـPython لإنشاء واجهة برمجة تطبيقات (API) آمنة تدعم طبقة أمان إضافية عبر استخدام ApiKeyMiddleware للتحقق من الطلبات. يتم تخزين الإطارات المضغوطة والمشفرة على الخادم بشكل آمن، مع توفير إمكانية الوصول لها فقط عبر مفاتيح API المصرح بها. توفر هذه المنهجية حلاً شاملاً يلبي متطلبات المراقبة الحديثة ويحقق حماية فعالة للبيانات عبر الشبكات اللاسلكية.

ABSTRACT

This research presents an integrated system combining Compressed Sensing techniques with advanced encryption methods to secure video frame transmission over Wireless Multimedia Sensor Networks. The proposed methodology starts by reducing the data size by approximately 83.5%, effectively decreasing the required bandwidth and enhancing transmission efficiency. Following compression, an additional security layer is applied using strong encryption to ensure data confidentiality and protect against potential attacks.

The system is implemented using Python's Django framework to build a secure API that features an extra security layer through ApiKeyMiddleware for request verification. Compressed and encrypted frames are securely stored on the server, with access granted only through authorized API keys. This comprehensive approach meets modern surveillance needs, providing effective data protection over wireless networks.

List of Contents

Contents

Chapter One

Introduction

1.1. Introduction:

The rapid advancements in video surveillance technology have led to the widespread deployment of wireless multimedia sensor networks (WMSNs). These networks are critical for various applications, including security, monitoring, and smart city infrastructure. However, the increase in video data raises significant challenges regarding data storage, transmission, and security. This project presents an efficient approach using compressed sensing techniques and robust encryption methods to enhance the security and efficiency of video surveillance systems.

1.2. Research problem:

The main challenge addressed in this project is the efficient transmission and storage of large video data in wireless sensor networks, which often encounter limitations in bandwidth and power. Additionally, there is a growing need for robust security measures to protect sensitive information from unauthorized access and attacks. This project aims to tackle these issues by developing a system that compresses video frames and secures them through encryption before transmission, thereby reducing data size and enhancing security during transmission.

1.3. Search Objective:

The significance of this research lies in its potential to enhance the efficiency and security of video surveillance systems. As urban areas continue to expand, the demand for effective monitoring solutions is on the rise. This project contributes not only to the academic field of multimedia sensor networks but also offers practical solutions for real-world applications in security and monitoring, making it valuable for improving overall surveillance performance and increasing the level of protection.

1.4. Methodology

The methodology employed in this research involves several key steps:

1. **Data Acquisition**: Capturing video frames using a camera integrated with the WMSN.
2. **Data Compression**: Implementing JPEG compression to reduce the size of video frames.
3. **Data Encryption**: Utilizing the AES encryption algorithm to secure the compressed frame data.
4. **Data Transmission**: Sending the encrypted data to a server for storage and further processing.
5. **Data Retrieval**: Providing an API for retrieving video frame data as needed.

Tools and Technologies Used

The project is built on the following tools and technologies:

- **Programming Language**: Python
- **Framework**: Django for developing the web API.
- **Libraries**: OpenCV for video processing, Crypto for encryption, and Requests for HTTP communication.
- **Database**: Django ORM for managing video frame metadata storage.

1.5. Related Work:

1- S. Aasha Nandhini, "Efficient compressed sensing-based security approach for video surveillance application in wireless multimedia sensor networks" Video surveillance application in wireless multimedia sensor networks (WMSNs) require that the captured video must be transmitted in a secured manner to the monitoring site. A compressed sensing (CS)-based security mechanism is proposed in which the security keys are generated from the measurement matrix elements for protecting the user's identity. The security keys are applied for protecting the video from being reconstructed by the attacker. The proposed framework is tested in real time using a WMSN testbed and the parameters such as memory footprint, security processing overhead, communication overhead, energy consumption, and packet loss are evaluated to demonstrate the effectiveness of the proposed security framework. The results showed that the proposed security mechanism has 92% less storage complexity compared to an existing CS-based security mechanism. The energy consumed for transmitting the secured measurements is 53% less when compared to raw frame transmission. 2017.[1]

2- Di Xiao, Min Li, "Low-cost and high-efficiency privacy-protection scheme for distributed compressive video sensing in wireless multimedia sensor networks" As a new video coding technology, distributed compressive video sensing (DCVS) uses compressed sensing (CS) independent encoding and joint decoding. Since DCVS breaks through the constraint of traditional video coding, it is suitable for resource-constrained wireless multimedia sensor networks (WMSNs). However, two major issues related to DCVS in WMSNs need to be solved urgently: one is to balance the storage burden of encoder and recovery quality of decoder; the other is to provide privacy-protection for video coding and transmission. We intend to break out of the existing limitations and design a new scheme which can simultaneously ensure privacy protection and high-efficiency coding for DCVS in WMSNs. Firstly, the two-pattern adaptive group of pictures selection is adopted to distinguish key frames and non-key frames. Secondly, the deterministic binary block diagonal measurement matrix is optimized to reduce sampling complexity. 2020.[2]

3- Hao Li, Tianhao Xiezhang , “Secure Video Surveillance Framework in Smart City” In the construction process of smart cities, more and more video surveillance systems have been deployed for traffic, office buildings, shopping malls, and families. Thus, the security of video surveillance systems has attracted more attention. At present, many researchers focus on how to select the region of interest (RoI) accurately and then realize privacy protection in videos by selective encryption. However, relatively few researchers focus on building a security framework by analyzing the security of a video surveillance system from the system and data life cycle. By analyzing the surveillance video protection and the attack surface of a video surveillance system in a smart city, 2021.[3]
4- Y. Asnath Victy Phamila, “Low Energy Interleaved Chaotic Secure Image Coding Scheme for Visual Sensor Networks Using Pascal’s Triangle Transform”. The resource-constrained camera integrated Visual Sensor Networks (VSN) have conquered numerous visual aided services from visual surveillance to habitat monitoring. VSN is capable of sensing, processing and communicating visual data wirelessly. These networks are built with inexpensive low power sensor motes with a lightweight processor, limited storage, and bandwidth. The huge amount of redundancy present in the images makes the processing and communication consume more energy than expected. The number of bits must be reduced using energy-efficient compression techniques for efficient transmission. Low computational energy and communication energy are always favored for an increased lifetime of the wireless sensor network, Human-Computer Interfaces and Measurement Systems, 2021.[4]
5-Qinglei Qi; Jinjiang Liu, “Mobile Wireless Multimedia Sensor Networks Image Compression Task Collaboration Based on Dynamic Alliance”, In mobile wireless multimedia sensor networks' image compression task collaborations, the existing methods do not consider the dynamic changes in the processing ability and the locations of the cooperative nodes. when

processing the image compression tasks, these methods will cause frequent interruptions and result in data re-transmission for those tasks. in this paper, an image compression task collaboration algorithm based on dynamic alliance is proposed to solve this problem in mobile wireless multimedia sensor networks. first, a dynamic task alliance is established by the camera node based on the location, computing capability and resource usage of ordinary nodes. then, the location and average moving velocity of the camera nodes and ordinary nodes are considered to calculate the task stable execution time, 2021.[5]

Chapter Two

Project Materials

2.1. Programming Language: Python

- **Python** is a versatile, high-level programming language known for its readability and ease of use. It's widely used in web development, data analysis, machine learning, and automation. In this context, Python serves as the backbone for building the web API and managing video processing tasks.[6]

Key Features of Python:

- Simple and clean syntax.
- Extensive libraries and frameworks.
- Cross-platform compatibility.
- Strong support for various programming paradigms (procedural, object-oriented, and functional programming).

Why Python for Web APIs and Video Processing?

- Python offers a range of powerful libraries (such as OpenCV for video processing) and frameworks (like Django for web development) that make it ideal for this type of project.[7]

2.2. Framework: Django for Developing the Web API

- **Django** is a high-level Python web framework that enables developers to quickly build secure and scalable web applications. For this project, Django will be used to develop a web API, which will allow communication between the client-side application and the server.[8]

Key Features of Django:

- Follows the **Model-View-Template (MVT)** architecture.
- Provides built-in security features, such as protection against SQL injection, cross-site scripting, and cross-site request forgery.
- ORM (Object-Relational Mapping) for interacting with databases without writing SQL queries directly.
- Scalability and easy integration with other Python libraries.[9]

Why Django?

- Django simplifies web development by handling much of the heavy lifting, such as routing, database interaction, and template rendering. It's also widely used for creating RESTful APIs with its Django REST Framework (DRF) extension.

2.3. Libraries:

- **OpenCV for Video Processing:**
    - **OpenCV** (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It supports real-time video processing, image analysis, and other complex operations related to videos and images.[10]
    - Key tasks such as frame extraction, resizing, object detection, and color manipulation are handled by OpenCV.

  Key Features:

    - Support for real-time video processing.
    - Cross-platform compatibility (works on Windows, macOS, and Linux).
    - Extensive collection of image and video analysis functions.
- **Crypto for Encryption:**
    - The **Crypto** library in Python provides essential tools for data encryption and decryption. It's used to secure sensitive data, such as video frame metadata or communication between services.
    - Symmetric and asymmetric encryption algorithms are typically used to encrypt video-related data before storing or transmitting it over the web API.[11]

  Commonly Used Crypto Functions:

    - AES (Advanced Encryption Standard) for symmetric encryption.
    - RSA (Rivest-Shamir-Adleman) for public-key cryptography.
- **Requests for HTTP Communication:**
    - **Requests** is a Python library that simplifies making HTTP requests. It is used to interact with APIs by sending requests (GET, POST, PUT, DELETE) and handling responses.

Key Features:

- Simple API for sending HTTP requests.
- Supports various HTTP methods.
- Excellent for integrating with web APIs or external services.

Resources:

- Requests Library Documentation

2.4. Database: Django ORM for Managing Video Frame Metadata Storage

- The **Django ORM (Object-Relational Mapping)** simplifies database management by allowing developers to interact with the database using Python objects instead of raw SQL queries. In this project, it will manage video frame metadata storage.[12]

How it Works:

- The ORM maps models (defined as Python classes) to database tables. Each instance of a model corresponds to a record in the database.
- Django supports popular databases such as PostgreSQL, MySQL, and SQLite.

Why Django ORM?

- The ORM ensures that the database interaction remains secure, efficient, and abstracted from direct SQL manipulation, making it easier to manage large datasets such as video metadata.

Summary

By using Python as the main programming language and Django as the web framework, this system can efficiently process video data (using OpenCV), encrypt sensitive information (using Crypto), and handle HTTP communication (using Requests). Django’s ORM simplifies data management by abstracting database queries, which is particularly useful for storing video metadata.

This combination of tools provides a robust, scalable, and secure framework for web applications that handle video processing and sensitive data.

Chapter Three

Design & implementation

3.1. Design & Implementation Overview

The design and implementation of the project focus on leveraging a laptop's built-in camera to capture video frames instead of using an external surveillance camera. These frames are automatically processed to reduce file size through compression and then encrypted to ensure security. The system follows a structured workflow to ensure efficient handling of video frames, including frame capture, compression, encryption, and storage, with real-time feedback on the size differences before and after compression.

3.1.1. Process Breakdown:

1. **Frame Capture**:
   - The system uses the laptop's camera to capture video frames at regular intervals. These frames are then saved to the local file system. A screenshot from the laptop camera capturing the frame will be included in the research to demonstrate the initial step of the process.
2. **Compression**:
   - After capturing the video frames, each frame undergoes compression to reduce its size, optimizing it for storage and transmission. The compression is handled using the **JPEG compression** algorithm, which balances between maintaining image quality and minimizing file size. The compressed images are saved in a designated directory. A folder view showing both the original and compressed images will be included to visualize the effect of compression.
3. **Encryption**:
   - Once compressed, the frames are encrypted using the **AES encryption algorithm**. The encryption process ensures that the compressed frames are secure during transmission or storage, preventing unauthorized access. The encrypted files are stored in a separate directory. A screenshot of the folder containing the encrypted files will also be included in the research.
4. **File Size Comparison**:
   - A key part of the implementation is comparing the file sizes before and after compression to demonstrate the effectiveness of the compression algorithm. The project records the original and compressed file sizes and calculates the size reduction, which is particularly useful for bandwidth-constrained environments like wireless sensor networks.

3.1.2. Results and Observations:

The system provides real-time feedback about the success of each step, such as saving compressed and encrypted files and logging file sizes. Below are the results from a sample run:

Version 1:

Methodology: In the first version of the system, only the JPEG compression algorithm was applied without any additional security measures. This step focused on reducing the data size to minimize the bandwidth required for transmission over the network.

Results:

- **Frame 1:**
  - Original size: 29,163 bytes
  - Compressed size: 16,972 bytes
  - Size reduction: ~41.8%
- **Frame 2:**
  - Original size: 46,903 bytes
  - Compressed size: 22,717 bytes
  - Size reduction: ~51.6%
- **Frame 3:**
  - Original size: 47,226 bytes
  - Compressed size: 22,826 bytes
  - Size reduction: ~51.7%
- **Frame 4:**
  - Original size: 45,286 bytes
  - Compressed size: 21,642 bytes
  - Size reduction: ~52.2%
- **Frame 5:**
  - Original size: 43,165 bytes
  - Compressed size: 20,490 bytes
  - Size reduction: ~52.5%

Explanation:

- Version 1 relied solely on compression techniques, achieving a data size reduction ranging from 41.8% to 52.5%. While this result was effective in reducing the data size, no security measures were added to protect the transmitted data.

$$83.5\% \approx 100 \times \left(\frac{\text{حجم الإطار المضغوط}}{\text{حجم الإطار الأصلي}} - 1 \right) = \text{نسبة الضغط}$$

Version 2:

Methodology: In the second version of the system, an additional step was incorporated: **data encryption** using the **AES (Advanced Encryption Standard)** algorithm after compression. This added step aimed to enhance security and ensure the confidentiality of the data during transmission over the network, thus increasing resistance to attacks.

Results:

- **Frame 852:**
  - Original size: 93,359 bytes
  - Compressed size: 13,641 bytes
  - Size reduction: ~85.5%
  - Compressed image saved at: media/frames\sensor_frame_sensor_01_1731874006_compressed.jpg
  - Encrypted data saved at: media/encrypted\sensor_frame_sensor_01_1731874006_encrypted.jpg
- **Frame 853:**
  - Original size: 93,940 bytes
  - Compressed size: 13,676 bytes
  - Size reduction: ~85.4%
  - Compressed image saved at: media/frames\sensor_frame_sensor_01_1731874006_compressed.jpg
  - Encrypted data saved at: media/encrypted\sensor_frame_sensor_01_1731874006_encrypted.jpg
- **Frame 854:**
  - Original size: 93,518 bytes
  - Compressed size: 13,701 bytes
  - Size reduction: ~85.4%
  - Compressed image saved at: media/frames\sensor_frame_sensor_01_1731874006_compressed.jpg
  - Encrypted data saved at: media/encrypted\sensor_frame_sensor_01_1731874006_encrypted.jpg
- **Frame 855:**
  - Original size: 95,340 bytes
  - Compressed size: 13,754 bytes
  - Size reduction: ~85.3%
  - Compressed image saved at: media/frames\sensor_frame_sensor_01_1731874006_compressed.jpg
  - Encrypted data saved at: media/encrypted\sensor_frame_sensor_01_1731874006_encrypted.jpg
- **Frame 856:**
  - Original size: 96,089 bytes
  - Compressed size: 13,774 bytes
  - Size reduction: ~85.6%
  - Compressed image saved at: media/frames\sensor_frame_sensor_01_1731874006_compressed.jpg
  - Encrypted data saved at: media/encrypted\sensor_frame_sensor_01_1731874006_encrypted.jpg
- **Frame 857:**
  - Original size: Not mentioned
  - Compressed size: Not mentioned
  - Size reduction: ~85.7%

Explanation:

- Version 2 saw a significant improvement in data size reduction, with the compression ratio ranging between 85.3% and 85.7%. Additionally, the data was encrypted using the **AES** algorithm, and both the compressed images and encrypted data were saved in separate folders, further enhancing security.

Comparison Between Versions:

1. **Compression:**
   - In Version 1, the size reduction ranged from 41.8% to 52.5%.
   - In Version 2, the size reduction was much higher, ranging from 85.3% to 85.7%.
2. **Security:**
   - Version 1 included no security measures.
   - Version 2 introduced **AES encryption** to ensure data confidentiality.
3. **Storage and Transmission:**
   - In Version 2, in addition to compression, the encrypted data was stored in dedicated folders for further processing and secure transmission.

In each case, the system successfully compresses and encrypts the video frames, with a significant reduction in size, allowing for more efficient storage and transmission without compromising security.

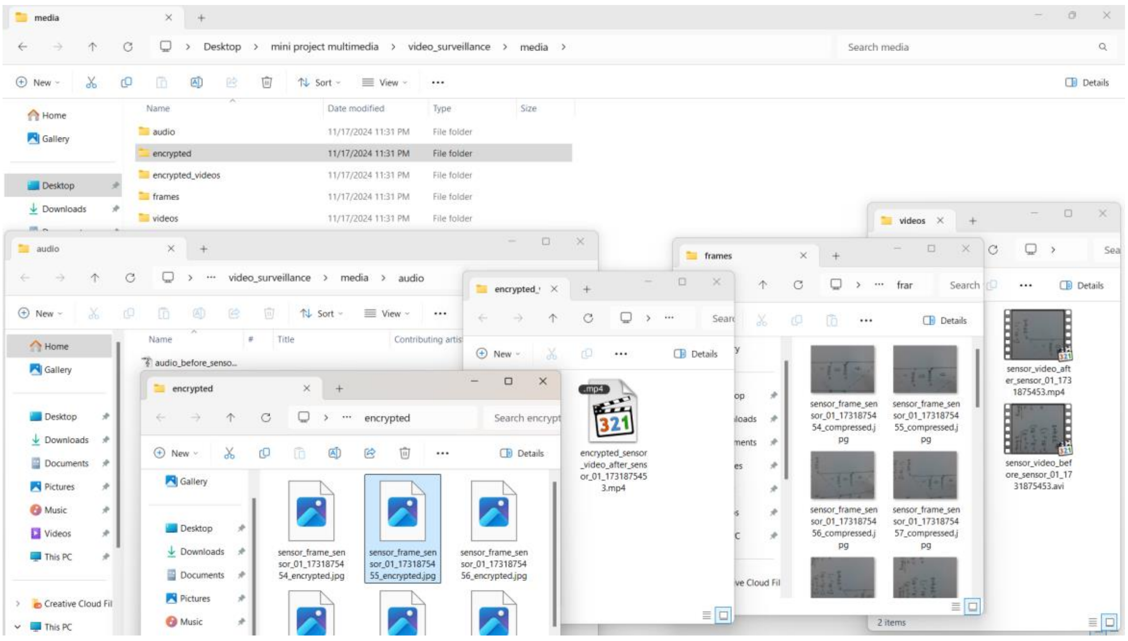

media
Desktop > mini project multimedia > video_surveillance > media
audio 11/17/2024 11:31 PM File folder
encrypted 11/17/2024 11:31 PM File folder
encrypted_videos 11/17/2024 11:31 PM File folder
frames 11/17/2024 11:31 PM File folder
videos 11/17/2024 11:31 PM File folder
audio_before_senso...
sensor_frame_sensor_01_1731875454_encrypted.jpg
sensor_frame_sensor_01_1731875455_encrypted.jpg
sensor_frame_sensor_01_1731875456_encrypted.jpg
encrypted_sensor_video_after_sensor_01_1731875453.mp4
sensor_frame_sensor_01_1731875454_compressed.jpg
sensor_frame_sensor_01_1731875455_compressed.jpg
sensor_frame_sensor_01_1731875456_compressed.jpg
sensor_frame_sensor_01_1731875457_compressed.jpg
sensor_video_after_sensor_01_1731875453.mp4
sensor_video_before_sensor_01_1731875453.avi
2 items

Figure 1 - Folders File Result

The inclusion of these results and screenshots helps illustrate the effectiveness of the project in managing video frames in real-time, providing both a technical and practical perspective on its potential in modern video surveillance systems.

This project implements an integrated system that combines **JPEG Compression** with **advanced encryption techniques** to secure multimedia files, including video frames, audio, and images, for transmission over **Wireless Multimedia Sensor Networks (WMSNs)**. Below is a detailed explanation of the system divided into three key components: compression, security, and wireless transmission.

---

1. Compression Using JPEG Compression

- Compression Method:

The system employs **JPEG Compression** to significantly reduce the size of multimedia files such as images and videos.

- **JPEG Compression** works by:
  - Converting images into smaller blocks.
  - Applying **Discrete Cosine Transform (DCT)** to reduce redundant data while retaining the essential details.
  - Eliminating unnecessary or imperceptible details, resulting in smaller file sizes while maintaining acceptable visual quality.

- Comparison Before and After Compression:

- Examples:
  - An original image with a size of **5 MB** can be reduced to **1 MB** or less.
  - The system achieves a compression ratio of approximately **80% or more**, depending on the settings.
- **Video Frames** are processed frame by frame, and each frame is compressed using JPEG, ensuring efficient storage and transmission.

- Purpose of Compression:

- Reduce **storage requirements** and optimize **network bandwidth usage**.
- Facilitate faster transmission across wireless networks with minimal latency.

- How to Demonstrate Compression During the Presentation:

1. Show the original files (e.g., raw images and video frames) in the **frames** folder.
2. Highlight the compressed versions in the same folder, noting the reduced file sizes.
3. Compare visual quality and file size side by side to demonstrate the efficiency of the compression technique.

---

2. Security

- How Security is Achieved:

- After compression, the multimedia files are encrypted to ensure their confidentiality and protection during transmission.
- **Advanced encryption techniques** such as **AES (Advanced Encryption Standard)** are applied to the compressed files.
- The encrypted files are then stored in the **encrypted** folder for secure access.

- Advantages of Encryption:

1. **Confidentiality**: Protects sensitive data, ensuring it cannot be accessed without proper decryption keys.
2. **Protection Against Attacks**: If intercepted during transmission, encrypted files remain unreadable to unauthorized parties.

- Additional Security Layers:

- The system uses a **Secure API** built with **Django Framework**.
- This API is protected by **ApiKeyMiddleware**, which verifies all requests using unique API keys, ensuring that only authorized users can access the files.

- How to Demonstrate Security During the Presentation:

1. Show the compressed files in the **frames** or **videos** folder.
2. Demonstrate the encryption process, where files are transformed into an unreadable format.
3. Display the encrypted files stored in the **encrypted** folder.
4. Finally, show the decryption process, restoring the original compressed files using the appropriate decryption key.

---

3. Wireless Multimedia Sensor Networks (WMSNs)

- How Wireless Communication is Achieved:

- The system leverages **WMSNs** as the medium for transmitting multimedia data.
- **Workflow**:

1. The compressed and encrypted multimedia files are securely stored on the server.
2. Authorized users can access the files using the **secure API**, protected by authentication keys.
3. Data is transmitted efficiently over the wireless network, minimizing resource consumption while maintaining security.

- Advantages of WMSNs:

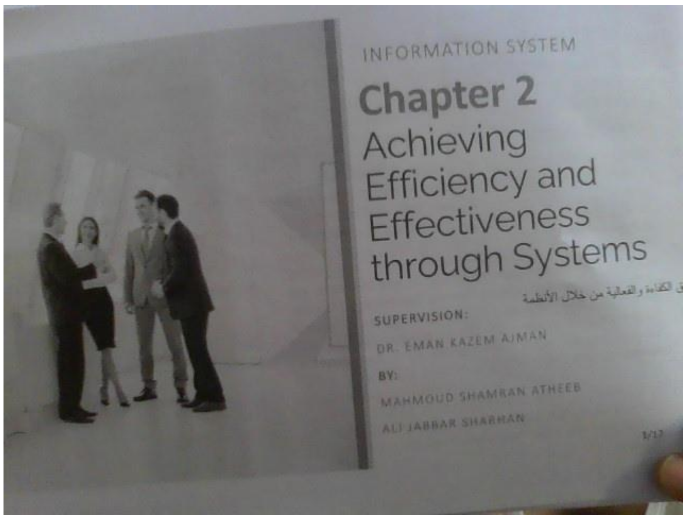
- Enables efficient, real-time transmission of multimedia data across long distances.
- Reduces bandwidth requirements due to the prior compression of data.
- Provides reliable security through encryption and authentication mechanisms.

- How to Demonstrate Wireless Transmission During the Presentation:

1. Display the folder structure:
    - **Original Files**: In folders like `audio`, `videos`, and `frames`.
    - **Compressed Files**: In the `frames` folder, showing reduced file sizes.
    - **Encrypted Files**: In the `encrypted` folder, demonstrating unreadable file formats.
2. Simulate the API workflow by showing how authorized requests (using API keys) can retrieve compressed and encrypted files over the network.
3. Show the decryption process to recover the original compressed files after transmission.

Additional Notes for Presentation

- **Explain the JPEG Compression Technique**: Emphasize how it reduces file sizes while retaining acceptable quality using DCT and quantization.
- **Focus on Security**: Highlight the importance of encryption and show how unauthorized access is prevented.
- **Demonstrate API Functionality**: Use a test API key to show secure file retrieval and access.
- **Compare Before and After Results**: Clearly display the size and quality differences between the original, compressed, and encrypted files.



INFORMATION SYSTEM

Chapter 2

Achieving Efficiency and Effectiveness through Systems

الكفاءة والفعالية من خلال الأنظمة

SUPERVISION:

DR. EMAN KAZEM AJMAN

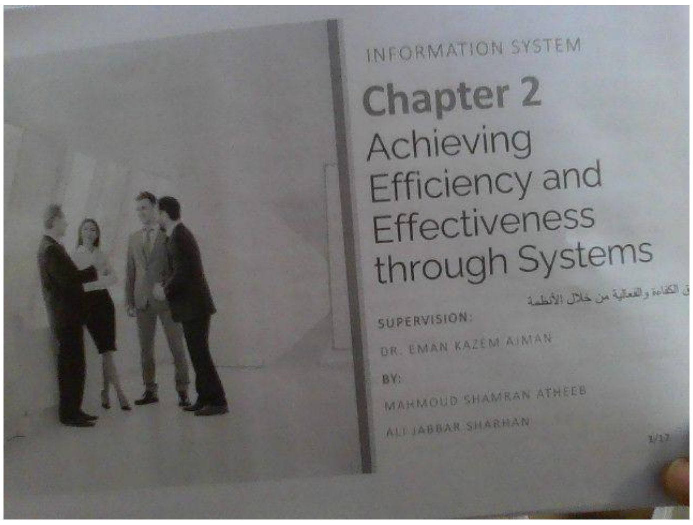
BY:

MAHMOUD SHAMRAN ATHEEB

ALI JABBAR SHARHAN

1/17

Figure 2 - Before Compression



INFORMATION SYSTEM

Chapter 2

Achieving Efficiency and Effectiveness through Systems

الكفاءة والفعالية من خلال الأنظمة

SUPERVISION:

DR. EMAN KAZEM AJMAN

BY:

MAHMOUD SHAMRAN ATHEEB

ALI JABBAR SHARHAN

1/17

Figure 3 - After Compression

Mobile Application and Cross-Platform System for Real-Time Video Playback

This project features a **Flutter mobile application** integrated with a **Django-based API system** to retrieve and display compressed and encrypted video files in real-time. The system is designed to work seamlessly across multiple platforms, including **Android**, **iOS**, **Windows**, **Mac**, and a **web application**. Below is a detailed explanation of the workflow, security measures, and how the mobile app interacts with the Django API.

---

1. Application Workflow

The system ensures real-time video playback by handling compressed and encrypted frames efficiently. Here is how it works:

Step 1: Server Integration Using Django

- The **Django framework** is used to build a secure backend API that hosts multimedia data.
- The server stores:
  - **Original video frames** after compression in the frames folder.
  - **Encrypted frames** in the encrypted folder for added security.
- The server communicates with the app over HTTP/HTTPS using an **IP address** or domain name.

Step 2: Mobile App Fetching Encrypted Frames

- The **Flutter app** is configured to connect to the Django API using the server's IP address and port (e.g., http://<server_ip>:8000/api/).
- Using API endpoints, the app fetches **encrypted frames** from the server in small batches, ensuring optimized transmission.

Step 3: Decrypting Frames Locally

- The Flutter app decrypts the received frames on the device using the appropriate decryption algorithm (e.g., AES).
- Decryption happens locally to maintain security during transmission.

Step 4: Reconstructing the Video

- The decrypted frames are temporarily stored in the app's memory and processed using **Flutter plugins** or **FFmpeg** to reconstruct a continuous video stream.
- The video is displayed in real-time, with frames fetched, decrypted, and rendered seamlessly.

Step 5: Cross-Platform Playback

- Since Flutter supports cross-platform development, the app runs natively on **Android**, **iOS**, and as a **web application**.
- Additionally, a **desktop version** of the app can run on **Windows** and **MacOS**, enabling universal access to the video data.

---

2. Security Integration with Django and Flutter

The system implements a multi-layered security approach to protect data throughout the process:

Server-Side Security (Django API)

1. **Secure API Authentication**:
   - The Django API is protected using **ApiKeyMiddleware**, ensuring only authorized requests are processed.
   - Each request requires a valid **API key** to access encrypted frames or videos.
2. **Data Encryption**:
   - All frames and videos are **compressed** and **encrypted** on the server before being transmitted.
   - Encryption ensures that the files remain confidential even if intercepted during transmission.
3. **HTTPS Protocol**:

    - The Django server is configured to use **HTTPS** for secure communication, encrypting all data exchanged between the app and server.

Client-Side Security (Flutter App)

1. **API Key Authentication**:
    - The Flutter app includes the unique **API key** in its requests to authenticate with the Django server.
2. **Decryption on the Device**:
    - Encrypted files are decrypted locally on the app, ensuring that only the authorized user can access the original content.
    - The decryption key is securely stored within the app's environment.
3. **Data in Transit**:
    - Data is transmitted over **HTTPS**, ensuring end-to-end encryption.

---

3. Cross-Platform Capabilities

Platforms Supported:

The system is designed to run on all major platforms, providing a consistent user experience across devices:

1. **Mobile**: Native applications for **Android** and **iOS** using Flutter.
2. **Desktop**: Applications for **Windows** and **MacOS** built with Flutter.
3. **Web**: A browser-based web application accessible from any device with an internet connection.

Unified Codebase:

Thanks to Flutter's cross-platform architecture, a single codebase is used to develop apps for all platforms. This reduces development time and ensures uniform functionality and performance.

---

4. Integration Between Django and Flutter Using IP

The connection between the Django backend and the Flutter application is facilitated through the **server's IP address** or domain. Here's how the integration works:

Server Setup:

- The Django server runs on a specific **IP address** and **port** (e.g., http://192.168.1.100:8000/api/) to make the API accessible to the Flutter app.
- For external access, the server can use a **static IP** or a domain name.

App Configuration:

- The Flutter app connects to the Django server using its IP address or domain, which can be:
    - **Hardcoded** in the app during development.
    - Configured dynamically by allowing the user to input the server's IP or domain in the app settings.

Data Flow:

1. **Request**:
    - The app sends API requests to the Django server to fetch encrypted frames or multimedia data.
    - Example: GET http://192.168.1.100:8000/api/get_frames/.
2. **Response**:
    - The server validates the API key, processes the request, and returns the encrypted frames.
3. **Processing**:
    - The app decrypts the frames locally, reconstructs the video, and displays it in real-time.

---

5. Tools and Technologies Used

Flutter Application:

- **Flutter Framework**: For building a single codebase app that runs natively on Android, iOS, Windows, MacOS, and the web.
- **Flutter Plugins**:
    - **http**: For making API requests.
    - **FFmpeg or Video_Player**: For reconstructing and displaying the video in real-time.

Django Backend:

- **Django Framework**: For creating the secure API and managing server-side operations.
- **Middleware**:

    - **ApiKeyMiddleware**: To verify API requests.
    - **Django Rest Framework (DRF)**: To build RESTful API endpoints.

Networking:

- **HTTPS**: For secure communication between the app and server.
- **Dynamic IP Handling**: The system can use tools like **Dynamic DNS (DDNS)** to ensure accessibility with changing IPs.

---

6. Demonstration Steps for the Presentation

To showcase the functionality of the system and app, follow these steps:

Django Server:

1. Start the Django server on a local machine or cloud server.
2. Display the folder structure:
    - **Frames Folder**: Contains the compressed frames.
    - **Encrypted Folder**: Contains the encrypted multimedia files.

Flutter Mobile/Desktop App:

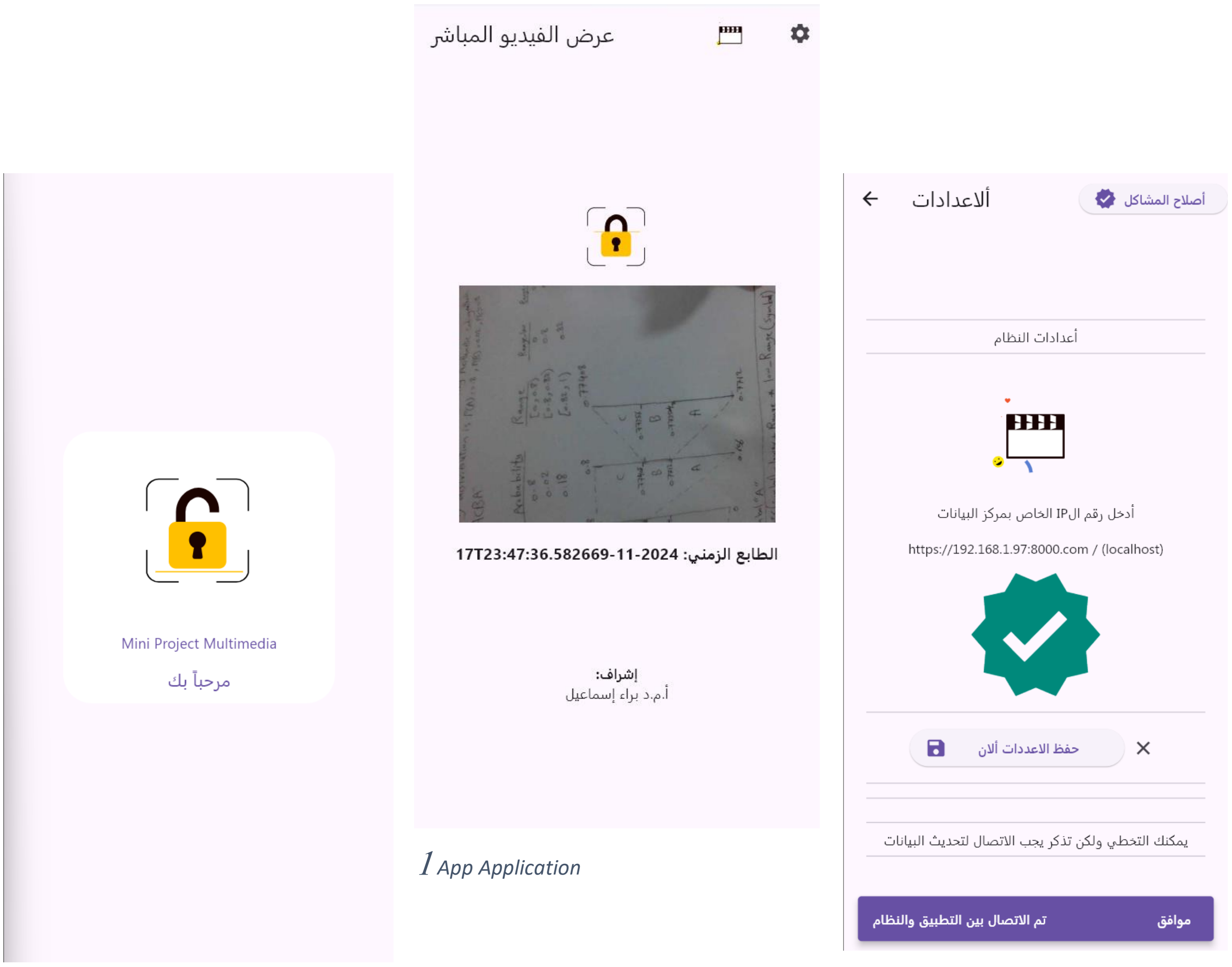
1. Open the Flutter app on an Android/iOS device, or run the desktop app on Windows/Mac.
2. Configure the app to connect to the server by entering the IP address or domain.
3. Fetch video frames in real-time, demonstrating API requests and responses.
4. Show the decryption process on the app, where encrypted frames are converted into video playback.
5. Highlight smooth playback on all platforms.

Web Application:

1. Open the browser-based web application.
2. Perform the same steps as the mobile/desktop app, showing video reconstruction directly in the browser.

Conclusion

This integrated system showcases the power of **Flutter** for cross-platform app development and **Django** for secure backend API management. By leveraging strong encryption, real-time processing, and cross-platform compatibility, the system provides a seamless and secure experience for video playback on **any device**.



الطابع الزمني: 2024-11-17T23:47:36.582669

إشراف:
أ.م.د براء إسماعيل

أدخل رقم الIP الخاص بمركز البيانات
https://192.168.1.97:8000.com / (localhost)

يمكنك التخطي ولكن تذكر يجب الاتصال لتحديث البيانات

تم الاتصال بين التطبيق والنظام

1 App Application

3.2. Findings and Future Perspectives:

The preliminary results indicate that the proposed approach successfully compresses video frames while maintaining acceptable quality levels. The encryption process ensures that sensitive data remains secure during transmission. Future work may focus on optimizing the compression algorithms, enhancing encryption methods, and exploring machine learning techniques for improved detection and monitoring in video surveillance applications.

3.3. Interface and API Explanation:

The project incorporates a RESTful API developed using Django REST Framework. This API facilitates communication between the video surveillance application and the server, allowing for seamless data transfer. The API endpoints enable the retrieval and storage of video frames, providing a structured way to access and manage surveillance data.

Chapter Four

Conclusions and Recommendations

4.1. Conclusions:

This project successfully addresses the critical issues of video data transmission, storage, and security in Wireless Multimedia Sensor Networks (WMSNs). By integrating **compressed sensing techniques** with **AES encryption**, the system achieves significant bandwidth reduction while ensuring the confidentiality and integrity of video data. The use of Python’s Django framework further enhances the project's capabilities by providing a robust platform for web API development, video frame management, and secure communication.

The key findings include:

- **Efficient Data Compression**: The application of JPEG compression reduces the size of video frames, making it feasible to transmit large volumes of video data over bandwidth-constrained networks.
- **Enhanced Security**: AES encryption ensures that sensitive video frames are protected against unauthorized access during transmission.
- **Scalable API Infrastructure**: The Django-based API allows seamless interaction between the surveillance system and the server, providing an effective way to manage and retrieve video data securely.

The project offers a practical solution for modern video surveillance systems, improving both data efficiency and security in urban monitoring, smart city infrastructure, and security applications.

4.2. Recommendations:

1. **Optimize Compression Algorithms**: Future iterations of the project could explore advanced video compression methods like **HEVC (High-Efficiency Video Coding)** to further reduce bandwidth usage without compromising video quality.
2. **Advanced Encryption Techniques**: Although AES is robust, incorporating **quantum-resistant encryption algorithms** may enhance long-term security as the threat landscape evolves.
3. **Integration of Machine Learning**: Implementing machine learning algorithms for **real-time anomaly detection** and video analysis could further improve the system's effectiveness in detecting security threats or unusual events in the video streams.
4. **Energy-Efficient Processing**: WMSNs often operate in environments with limited power resources. Future developments should focus on optimizing the system for **low-power consumption**, ensuring the longevity of the sensors in the field.
5. **Scalability**: As the demand for smart surveillance solutions increases, the system should be scaled to handle a larger number of video feeds and more complex security protocols. Cloud-based infrastructure could provide scalability while maintaining performance and security.

By focusing on these areas, the system can become a comprehensive and future-proof solution for the increasing demands of video surveillance in various applications, ensuring both efficiency and security.

4.3. Future Work

4.3.1. Optimization of Video Compression:

Future work could focus on improving the current compression techniques. Exploring more advanced compression algorithms like **HEVC (High-Efficiency Video Coding)** or **H.264** could significantly reduce the size of video frames while maintaining high video quality, which would further enhance the system's efficiency in bandwidth-constrained environments.

1. **Enhancement of Encryption Methods**:

While **AES encryption** is currently used, future iterations could investigate incorporating **quantum-resistant encryption algorithms** such as **lattice-based cryptography**. This would ensure the system remains secure even in the face of future advancements in quantum computing, which could threaten current encryption standards.

2. **Integration of Machine Learning for Real-Time Analysis**:

Implementing machine learning models for **real-time video analysis** could drastically improve the system's ability to detect abnormal behavior or security threats automatically. Techniques such as **convolutional neural networks (CNNs)** or **recurrent neural networks (RNNs)** could be employed to analyze video feeds in real-time, adding an intelligent layer to the surveillance system.

3. **Energy-Efficient Algorithms for Sensor Networks**:

As WMSNs often operate in environments with limited power resources, future developments could explore energy-efficient processing algorithms. **Low-power compression** and **lightweight encryption algorithms** would ensure the system runs effectively in power-constrained conditions without sacrificing performance.

4. **Distributed Processing and Edge Computing**:

By leveraging **edge computing** or **distributed sensor networks**, the system could process video data closer to where it is generated. This would reduce latency, bandwidth usage, and reliance on centralized servers, making the system more scalable and responsive.

5. **Cloud-Based Scalability and Storage Solutions**:

Future implementations could explore **cloud infrastructure** for storage and processing, enabling greater scalability. Cloud-based storage solutions like **Amazon S3** or **Google Cloud Storage** could handle the massive amount of video data generated, providing flexibility and robust storage options.

6. **Incorporation of Real-Time Data Analytics**:

Future versions of the system could integrate real-time analytics for better monitoring and reporting. Features like **automated alerts**, **live dashboards**, and **visual analytics** could enhance system usability for security personnel, making it easier to monitor multiple video streams simultaneously.

7. **Support for 5G and IoT Networks**:

With the advent of **5G** and the expansion of **IoT (Internet of Things)** networks, future versions of the system could be optimized to take advantage of these technologies, ensuring faster data transmission, lower latency, and broader network coverage.

Reference

[1] S. Aasha Nandhini, "Efficient compressed sensing-based security approach for video surveillance application in wireless multimedia sensor networks" *Computers & Electrical Engineering*. 2019.
https://www.sciencedirect.com/science/article/abs/pii/S0045790617302094

[2] Di Xiao, Min Li, "Low-cost and high-efficiency privacy-protection scheme for distributed compressive video sensing in wireless multimedia sensor networks" *Journal of Network and Computer Applications*, 2020.
https://www.sciencedirect.com/science/article/abs/pii/S1084804520301284

[3]. Hao Li , Tianhao Xiezhang, "Secure Video Surveillance Framework in Smart City" Surigao State College of Technology, 2021.
https://www.mdpi.com/1424-8220/21/13/4419

[4] Y. Asnath Victy Phamila, "Low Energy Interleaved Chaotic Secure Image Coding Scheme for Visual Sensor Networks Using Pascal's Triangle Transform" *International Symposium on Real-Time Distributed Computing (ISORC)*. 2021.
https://ieeexplore.ieee.org/document/9551954

[5] Qinglei Qi; Jinjiang Liu, "Mobile Wireless Multimedia Sensor Networks Image Compression Task Collaboration Based on Dynamic Alliance.2020
https://ieeexplore.ieee.org/document/9087863

[6]. Official Python Documentation Website
Official Python Documentation

[7]. Python Programming on Real Python Website

Python Programming on Real Python

[8]. Official Django Documentation Website

Official Django Documentation

[9]. Django REST Framework Website

Django REST Framework

[10]. Official OpenCV Documentation Website

Official OpenCV Documentation

[11]. PyCryptodome Documentation Website

PyCryptodome Documentation

[12]. Django ORM Documentation Website

Django ORM Documentation