CIPHERSHIELD: PIONEERING MULTI-KEY CRYPTOGRAPHY FRAMEWORK FOR FILE ENCRYPTION USING DYNAMIC KEYS

A Project report submitted in partial fulfilment of the requirements for the award of the Degree

of

Master of Technology

In

Cyber Forensics and Information Security (CFIS)

By

MOHAMMED ABDUL LATEEF

(22011DA802)

Under the guidance of

Dr. P. Swetha

Professor, Department of CSE

Deputy Director, Directorate of Affiliations and Academic Audit

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,

JNTUH UNIVERSITY COLLEGE OF ENGINEERING

SCIENCE AND TECHNOLOGY,

KUKATPALLY, HYDERABAD, TELANGANA – 500085.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,

JNTUH UNIVERSITY COLLEGE OF ENGINEERING

SCIENCE AND TECHNOLOGY,

KUKATPALLY, HYDERABAD, TELANGANA – 500 085.

DECLARATION BY THE CANDIDATE

I, **MOHAMMED ABDUL LATEEF,** bearing Roll Number: **22011DA802,** hereby declare that this project report entitled **CipherShield: Pioneering Multi-Key Cryptography Framework for File Encryption Using Dynamic Keys,** is carried out by me under the guidance of **Dr. P. Swetha, Professor, Department of CSE,** and is submitted in partial fulfilment of the requirements for the award of the degree of ***Master of Technology*** in ***Cyber Forensics and Information Security***. This is a record of bonafide work carried out by me and the results embodied in this project have not been reproduced/copied from any source.

Mohammed Abdul Lateef

Roll. No: 22011DA802

Date:

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,

JNTUH UNIVERSITY COLLEGE OF ENGINEERING

SCIENCE AND TECHNOLOGY,

KUKATPALLY, HYDERABAD, TELANGANA – 500 085.

CERTIFICATE BY THE SUPERVISOR

This is to certify that the project report entitled **CipherShield: Pioneering Multi-Key Cryptography Framework for File Encryption Using Dynamic Keys,** being submitted by **Mohammed Abdul Lateef (22011DA802)** in partial fulfilment of the requirements for the award of the degree of ***Master of Technology*** in ***Cyber Forensics and Information Security***, is a record of bonafide work carried out by him. The results are verified and found satisfactory.

Dr. P. Swetha

Professor

Department of CSE

JNTUHUCESTH

Date:

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,

JNTUH UNIVERSITY COLLEGE OF ENGINEERING

SCIENCE AND TECHNOLOGY,

KUKATPALLY, HYDERABAD, TELANGANA – 500 085.

CERTIFICATE BY THE HEAD OF DEPARTMENT

This is to certify that the project report entitled **CipherShield: Pioneering Multi-Key Cryptography Framework for File Encryption Using Dynamic Keys,** being submitted by **Mohammed Abdul Lateef (22011DA802)** in partial fulfilment of the requirements for the award of the degree of ***Master of Technology*** in ***Cyber Forensics and Information Security***, is a record of bonafide work carried out by him. The results are verified and found satisfactory.

Dr. K. P. Supreethi
Professor and Head of the Department
Department of CSE
JNTUHUCESTH

Date:

ACKNOWLEDGEMENT

I would like to express sincere thanks to my Supervisor**, Dr. P. Swetha, Professor, Department of Computer Science and Engineering** for her admirable guidance and inspiration both theoretically and practically and most importantly for the drive to complete project successfully. Working under such an eminent guidewas my privilege.

I owe a debt of gratitude to **Dr. K. P. Supreethi, Professor & Head of the Dept. of Computer Science and Engineering** for this kind consideration and encouragement in carrying out this project successfully.

I am grateful to the project review committee members **Dr. B. Padmaja Rani, Professor, Department of Computer Science and Engineering** and **Dr. J. Ujwala Rekha, Professor, Department of Computer Science and Engineering**,who have helped in successfully completing this project by giving their valuable suggestions andsupport.

I express thanks to my parents for their love, care and moral support, without which I would have not been able to complete this project. Finally, I would like to thanks my friends for their co-operation to complete this project.

Mohammed Abdul Lateef

Roll. No: 22011DA802

Date:

ABSTRACT

The proliferation of digital content consumption, particularly in the form video files, underscores the paramount importance of securing these assets against unauthorized access and copyright infringement.

In response to this imperative, this work presents a comprehensive approach to video file encryption, tailoring the cryptographic strategy based on the nature of the content. Video based data is emphasized to utilize a blend between RSA and Elliptic Curve Cryptography (ECC) equation-based approach, adapting to the dynamic nature of video data through multiple key generation. The implemented system is realized through dynamic software featuring dedicated module video-based encryption and decryption along with video chunking and chunk concatenation aspects.

In this innovative approach, real-time key generation, based on unique identifiers, dynamically encrypts and decodes file chunks, adapting to evolving data. To assess the effectiveness of the system, rigorous evaluations of its performance were conducted. The empirical results underscore the proposed approach's superiority, showcasing advancements in both performance and security metrics with the current one[1]. The work stands at the forefront of addressing contemporary challenges in the realm of file security, offering a resilient and efficient solution to combat copyright infringement and piracy risks through cryptographic implementation.

Key Words: Multi-Key Cryptography, Hybrid Cryptography, Elliptic Curve Equation, GUID, Dynamic keys generation, Dynamic Encryption, Performance Metrics

Table of Contents

List of Figures

CHAPTER - 01
INTRODUCTION

1.1 Project Description

In today’s interconnected world, where data is the new currency, ensuring its security is paramount. This is where cryptography comes into play. Derived from the Greek words "kryptos" (hidden) and "graphein" (writing), cryptography is the art and science of converting information into a form incomprehensible to unauthorized individuals. At its core, cryptography provides three fundamental security services: confidentiality, integrity, and authenticity. Integrity safeguards data from unauthorized modification, ensuring it remains unaltered during transmission or storage. Authenticity verifies the identity of the sender and the origin of the data.

From Caesar's simple substitution cipher to the complex algorithms used today, it has evolved to meet the increasing sophistication of threats. Modern cryptography relies on mathematical principles and computational complexity to create robust encryption schemes.

One of the most critical applications of cryptography is in securing digital communications. When you send an email, make an online purchase, or access your bank account, cryptography protects your sensitive information from being intercepted and misused. It is the backbone of secure websites (HTTPS), virtual private networks (VPNs), and digital signatures.

Beyond securing communications, cryptography also plays a vital role in protecting data at rest. Encryption safeguards sensitive data stored on hard drives, servers, and cloud storage from unauthorized access. This is particularly crucial for industries such as healthcare, finance, and government, where data breaches can have severe consequences.

Video-based cryptography is a specialized field that focuses on securing video content. With the proliferation of video streaming services and video surveillance systems, protecting the integrity and confidentiality of video data has become increasingly important. Video-based cryptography involves applying cryptographic techniques to encrypt, decrypt, and authenticate video content.

This technology has numerous applications, including secure video conferencing, digital rights management (DRM) for content protection, and secure video surveillance. By encrypting video data, unauthorized parties cannot access or modify the content, preventing illegal distribution and ensuring the privacy of individuals captured on video.

It protects our sensitive information, enables secure communication, and safeguards our privacy. As technology continues to advance and cyber threats become more sophisticated, cryptography will remain at the forefront of safeguarding our digital world.

1.1.1 Elliptic Curve Cryptography (ECC) Equation

Elliptic Curve Cryptography (ECC) is a public-key cryptographic system based on the algebraic properties of elliptic curves over finite fields. Unlike traditional public-key cryptosystems like RSA, which rely on the difficulty of factoring large numbers, ECC's security is derived from the complexity of solving the discrete logarithm problem on an elliptic curve.

The Basics of Elliptic Curves

The fundamental equation used in elliptic curve cryptography is as below,

$$y^2 = x^3 + ax + b \ (mod\ p)$$

where,

- x and y are curve points
- a and b are constants of the curve
- p is a prime number defining the finite field

To visualize an elliptic curve, imagine plotting the points that satisfy this equation on a graph. However, in ECC, we're working with points over a finite field, not a continuous space.

Advantages of ECC

- **Smaller Key Sizes**: ECC offers equivalent security to other cryptosystems using much smaller key sizes.
- **Faster Computations**: ECC operations are generally more efficient than those in other cryptosystems.
- **Energy Efficiency**: Smaller key sizes and faster computations make ECC suitable for resource-constrained devices.

Security of ECC

While there's no mathematical proof that elliptic curve discrete logarithm problem(ECDLP) is as hard as factoring large numbers, there's no known efficient algorithm to solve it, making ECC a secure choice for cryptographic applications due to difficulty in finding the scalar k.

By leveraging the rich mathematical structure of elliptic curves, ECC provides efficient and secure cryptographic solutions for a wide range of applications, from secure communication to digital signatures.

1.1.2 AES: The Advanced Encryption Standard (Symmetric)

AES, or Advanced Encryption Standard, is a symmetric block cipher chosen by the U.S. government as a standard for secure data encryption. AES (Advanced Encryption Standard) is a widely adopted symmetric encryption algorithm known for its security and efficiency[2]. In our application, AES is used to encrypt each video chunk, ensuring that the video content remains secure during storage and transmission. The choice of AES is driven by its ability to provide strong encryption with relatively fast performance, making it suitable for handling large video files.

How AES Works

AES operates on a fixed-size data block of 128 bits. The key length can vary, offering three levels of security: 128-bit, 192-bit, and 256-bit.

The core of AES is a substitution-permutation network (SPN). This involves multiple rounds of transformations applied to the data block. Each round consists of several steps:

1. **Byte Substitution**: Each byte in the data block is replaced with another byte according to a substitution table.

2. **Shift Rows**: The rows of the data block are shifted cyclically.

3. **Mix Columns**: Each column of the data block is transformed using a matrix multiplication.

4. **Add Round Key**: The data block is combined with a round key derived from the main key.

These steps are repeated multiple times, with the number of rounds depending on the key length. The final round is slightly different, omitting the Mix Columns step.

Key Features of AES in Our Application:

1. **Block Cipher**: AES operates as a block cipher, meaning it encrypts data in fixed-size blocks. In our application, the video chunks are split into blocks of 128 bits (16 bytes) before encryption. This ensures that the encryption process aligns with the AES algorithm's requirements.

2. **Key Length**: We utilize AES with a 256-bit key length, offering a high level of security. This key length is considered secure against most forms of cryptographic attacks, providing confidence in the protection of our video data.

3. **Mode of Operation**: In our implementation, we use the CBC (Cipher Block Chaining) mode of operation for AES encryption. CBC mode ensures that each block of plaintext is XORed with the previous ciphertext block before being encrypted, providing enhanced security by making patterns less discernible in the ciphertext.

Security of AES

AES is considered highly secure. Its strength comes from:

- **Key Length**: Longer keys offer greater protection against brute-force attacks.
- **Number of Rounds**: The multiple rounds of transformations significantly increase the complexity of breaking the encryption.
- **Mathematical Basis**: The underlying mathematical operations are carefully designed to resist various types of attacks.

While there have been theoretical attacks on reduced-round versions of AES, no practical attacks have been found against the full algorithm.

Applications of AES

AES is widely used in various applications due to its speed, efficiency, and security:

- Secure Communication: Protecting data transmitted over networks, such as HTTPS.
- **File Encryption**: Securing sensitive data stored on hard drives or in cloud storage.
- **Disk Encryption**: Protecting data on entire storage devices.
- **Wireless Security**: Securing Wi-Fi networks.

AES has become a cornerstone of modern cryptography, providing robust protection for sensitive information in the digital age.

1.1.3 RSA: Rivest Shamir Aldleman Algorithm (Asymmetric)

Its Named after its creators Ron Rivest, Adi Shamir and Leonard Adleman, RSA is one of the first practical public-key cryptosystems. While the symmetric encryption algorithms like AES uses a single secret key for both encryption and decryption, RSA used public-key cryptography to enable different keys form encrypting and decrypting. A public key is used to encrypt data and a private key is required to read (Decrypt) this Encrypted data [3].

Key Generation

The foundation of RSA lies in number theory. Here's a simplified overview of key generation:

1. **Choose two large prime numbers:** p and q.
2. **Calculate the modulus:** n = p * q.
3. **Calculate the totient:** φ(n) = (p-1)*(q-1).
4. **Choose a public exponent (e):** e should be relatively prime to φ(n) and typically has a small value like 3 or 65537.
5. **Calculate the private exponent (d):** d is the modular multiplicative inverse of e modulo φ(n), meaning (d * e) mod φ(n) = 1.

   The public key consists of the pair (e, n), while the private key is (d, n).

Encryption and Decryption

- **Encryption:** To encrypt a message M, the following formula is used:
  - Ciphertext C = M^e mod n
- **Decryption:** To decrypt the ciphertext C, the following formula is used:
  - Plaintext M = C^d mod n

Security of RSA

While there have been advancements in factorization algorithms, the key size used in RSA (typically 2048 bits or more) makes it computationally infeasible for current computers to break. Also, large factoring of numbers make this algorithm more secure.

Applications of RSA

RSA is widely used in various applications:

- **Secure Communication:** Protecting data transmitted over networks, especially for key exchange in hybrid cryptosystems.
- **Secure Email:** Providing confidentiality and integrity for email communication.
- **Secure Remote Login:** Protecting user credentials.

  While RSA is a powerful tool, it's important to note that its computational overhead is higher compared to symmetric encryption algorithms like AES. Therefore, it's often used in conjunction with symmetric encryption for practical implementations.

1.2 Problem Statement

Protecting copyright and piracy has become a key concern in real-time video streaming systems as it has become a daily routine and source of income to many individuals today. This study introduces a novel multi-key and hybrid cryptography technique for security purposes. In this work, the software implementation of video encryption and decryption is achieved for discrete systems with pseudorandom binary sequence as an approach to generate secure keys. This technique generates multiple keys to encrypt and decrypt the piece of small video chunks which is having a dynamic nature based on the video content. The proposed method was applied for Android, and we developed the sender and receiver application of casting on this platform. We implemented the proposed system to real devices and streaming video, tested for not only security but also performance. The outcomes demonstrate superiority in terms of performance and security but with high computational time and less security due to the use of MAC address which can be addressed using GUID data.

1.3 Project Summary

The project is a comprehensive web application developed and designed for secure video uploading, storage, and streaming. The application is built with a focus on user authentication, encryption, and decryption of video files, and features a user-friendly frontend with modern design principles. Below is a detailed summary of the key aspects of the project:

1. Encryption and Decryption Mechanisms:

- Encryption: Videos are encrypted upon upload using a hybrid cryptographic technique that combines RSA and AES algorithms. The RSA algorithm is used for encrypting the AES keys, while AES is used for chunk-wise encryption of the video files. This ensures robust security, making it difficult for unauthorized users to access the video content.

- Decryption: When a user attempts to watch a video, the application decrypts the video on-the-fly. The decryption process is triggered only if the user is authorized, ensuring that only the intended recipient can view the video. The decrypted video is stored temporarily and streamed to the user.

2. Video Streaming and Mini-Player Implementation:

- Streaming Function: The application streams video files using a Python generator function, which reads the video file in chunks and streams it to the client in real-time. This method is efficient and suitable for handling large video files without overloading the server[4].

- Mini-Player Design: The application includes a mini-player format, allowing users to watch videos in a smaller, fixed window on the screen. The mini-player is designed to be non-intrusive, providing a seamless viewing experience while maintaining the video quality.

3. Frontend UI/UX Design:

- Responsive Design: The application's frontend is designed with a focus on responsiveness and modern aesthetics. The design includes a transparent screen overlay on a background image, similar to the login page, ensuring visual consistency across the application.

- Navbar and Navigation: The application features a top navigation bar with links to essential pages, such as Home, Metrics, Upload, and Logout. The navbar is fixed at the top and styled to blend with the dark theme of the application.

- Scrollable Video List: The home screen displays a scrollable list of uploaded videos, with each video numbered and linked for easy access. This section is designed to be responsive, adjusting to the screen size and number of videos available.

4. Metrics and Analytics:

- Metrics Page: The application includes a metrics page that displays various statistics related to video encryption and decryption processes. These metrics are plotted using Matplotlib and are saved in user-specific folders for easy access[5].

- Dynamic Key Management: The application supports dynamic key management, where the AES keys for each video chunk are dynamically generated and optionally saved, depending on a configuration flag (`SAVE_DYNAMIC_KEYS`).

5. Security Considerations:

- Hardware UUID Integration: The application stores and uses the hardware UUID as part of the encryption process, adding an additional layer of security. This ensures that the video decryption is tied to a specific hardware setup, preventing unauthorized access from other machines.

- Session and Flash Messages: Security is further enhanced with Flask sessions and flash messages, providing users with timely feedback on their actions and ensuring secure session management.

6. Modularity and Code Separation:

- Separation of Concerns: The application follows a clean and modular structure, with a clear separation between frontend and backend code. CSS and JavaScript files are kept separate, ensuring that the HTML remains uncluttered and easy to maintain.

- Reusability: Functions and templates are designed to be reusable, allowing for easy scaling and modification of the application. This modularity ensures that new features can be added without disrupting the existing functionality.

7. User Experience and Accessibility:

- Hover Effects and Custom Icons: The frontend design includes modern UI elements, such as hover effects, rounded input fields, and custom icons from external libraries. These elements enhance the user experience by providing a visually appealing interface.

- Fullscreen Toggle: Users can toggle full screen mode when watching videos, allowing for a more immersive viewing experience. The full screen feature is implemented using JavaScript, ensuring compatibility across different browsers.

Overall, this project represents a robust and secure video management system that combines advanced cryptographic techniques with a user-friendly interface, making it suitable for real-world applications where video content needs to be securely managed and streamed to authorized users. The focus on modularity, security, and a modern UI/UX design ensures that the application is not only functional but also visually appealing and easy to use.

CHAPTER 2: LITERATURE SURVEY

2.1 Existing System

The proliferation of online video streaming has transformed entertainment consumption, but it has also intensified the challenge of copyright protection. Traditional security measures have proven inadequate against the sophisticated tactics of pirates. This necessitates innovative approaches to safeguard valuable content.

2.1.1 Existing Challenges in Video Streaming Security

Current video streaming platforms primarily rely on digital rights management (DRM) systems and encryption techniques. However, these methods have shown vulnerabilities.

- **Single-key encryption:** Many systems employ a single key to encrypt the entire video, making it a prime target for attackers. If the key is compromised, the entire video is vulnerable[6].
- **Static encryption:** Encryption keys are often static, making them susceptible to brute-force attacks and reverse engineering[7].
- **Real-time challenges:** Real-time streaming introduces additional complexities, as encryption and decryption must occur efficiently without impacting video quality or user experience[8].

2.1.2 The Need for a Robust Solution

To address these shortcomings, a more robust and dynamic approach is required. A multi-key, hybrid cryptography system offers a promising solution. By employing elliptic curve cryptography (ECC) to generate pseudorandom encryption keys, this method introduces a higher level of complexity and security.

2.2 Key advantages of this approach

- **Dynamic key generation:** Continuously changing encryption keys based on video content make it significantly harder for attackers to compromise the entire stream.
- **Enhanced security:** The combination of multiple keys and ECC provides a robust defense against various cryptographic attacks.
- **Improved performance:** By encrypting small video chunks, the system can optimize processing efficiency and reduce latency[13].

2.3 Disadvantages

While the proposed multi-key hybrid cryptography system offers significant advantages, it's essential to consider potential drawbacks:

- **Key generation:** The continuous generation of new encryption keys can be computationally intensive, especially for resource-constrained devices[10].
- **System implementation:** The system involves multiple components (key generation, encryption, decryption, synchronization) which can increase development and maintenance complexity[11].
- **Key management:** Securely managing and distributing a large number of keys is challenging and requires robust key management infrastructure[11].
- **Backward compatibility:** Older devices or platforms might not support the required cryptographic algorithms or processing capabilities[12].

CHAPTER – 03

PROPOSED SYSTEM

3.1 Data Encryption

In this section, we will delve into the detailed process of data encryption within the Flask based video encryption and decryption application. This documentation will cover the video file segmentation, key generation, and the initial key encryption process, all integral to ensuring secure video communication.

3.1.1 Video File Segmentation

Process of Splitting Video Files into Smaller Chunks:

The first step in the encryption process is the segmentation of video files into smaller chunks. This step is crucial because encrypting a large video file in its entirety would be computationally expensive and could lead to inefficiencies in both storage and transmission. The process is as follows:

1. **Input Video Analysis**: The system first analyses the video file to determine its size and format. This analysis helps in defining how the video will be segmented.
2. **Chunking Mechanism**: The video file is then split into smaller chunks. Each chunk is treated as a separate unit for the encryption process. The size of each chunk is predetermined based on the application's requirements or can be dynamically set depending on the file size and network constraints.
3. **Storing Chunks**: Once the video is segmented, each chunk is temporarily stored in a secure location for subsequent encryption. These chunks are named systematically (e.g., video_chunk_01, video_chunk_02) to maintain their order during reassembly.

3.1.2 Rationale Behind Chunk Size Selection

The choice of chunk size is a critical factor in the video encryption process. The rationale for selecting a specific chunk size includes:

1. **Efficiency**: Smaller chunks allow for faster encryption and decryption processes, as the computational load is distributed across smaller data units. This improves the overall speed and efficiency of the encryption process.

2. **Security**: Encrypting video in chunks enhances security by ensuring that even if one chunk is compromised, the entire video is not exposed. Each chunk can be encrypted with a unique dynamic key, making it harder for unauthorized entities to decrypt the entire video.

3. **Transmission and Storage**: Segmented video files are easier to transmit over networks and can be more efficiently stored. Chunking also allows for parallel processing during encryption and decryption, which can be advantageous in high-performance systems.

4. **Error Handling**: In case of transmission errors, only the affected chunks need to be retransmitted, not the entire video file. This reduces the potential for data loss and increases the reliability of the system.

3.2 The Use of ECC Equation

ECC Equation and Its Role:

The fundamental equation of Elliptic Curve Cryptography (ECC) is employed within the key management process. Although the full ECC algorithm is not implemented, a specific equation from ECC is used to generate keys or parameters within the encryption scheme. ECC offers high security with smaller key sizes, making it efficient for cryptographic operations on devices with limited processing power.

1. **Curve Selection**: A specific elliptic curve is chosen, denoted as $y^2 = x^3 + ax + b \ (mod\ p)$, where (a) and (b) are parameters defining the curve. The selection of the curve is based on the security requirements of the application.

2. **Key Derivation**: Using a point on the selected curve, a key is derived that can be used in the encryption process. The key derived from the ECC equation is utilized within the dynamic key generation process, ensuring that each chunk has a unique and secure key[14].

3. **Security Assurance**: ECC's robustness against attacks such as brute force and its efficiency in generating secure keys make it a valuable addition to the overall encryption framework.

3.3 Key Generation

3.3.1 RSA Key Pair Generation

1. **RSA Key Pair**: The RSA (Rivest-Shamir-Adleman) algorithm is used to generate a public-private key pair. The public key is used for encrypting the AES keys (discussed below), while the private key is retained securely by the server for decrypting these keys when needed.

2. **Key Siz**e: Typically, a key size of 2048 bits or higher is selected to ensure strong encryption. This choice balances security and performance.

3. **Key Storage**: The RSA keys are securely stored, with the private key being kept confidential. The public key is made available for use in the initial key encryption process.

3.3.2 AES Symmetric Key Generation for Chunk Encryption

1. **AES Key Generation**: The Advanced Encryption Standard (AES) is used to generate symmetric keys that will encrypt each video chunk. AES is chosen for its efficiency and security, with a key size of 256 bits to ensure robust encryption.

2. **Key Per Chunk**: A separate AES key is generated for each video chunk. This ensures that even if one key is compromised, only a single chunk is affected, not the entire video.

3.3.3 Dynamic Key Generation and Management

1. **Dynamic Keys**: For added security, dynamic keys are generated for each video chunk. These keys are derived using a combination of the AES key and additional parameters, such as timestamps or unique identifiers.

2. **Key Management**: The dynamically generated keys are either stored securely (if `SAVE_DYNAMIC_KEYS` is enabled) or discarded after use. The management of these keys is critical to ensure that they are available for decryption when needed.

3.4 Initial Key Encryption

3.4.1 Deriving VID

The VID (Video Identifier) is a unique identifier derived from specific video parameters or metadata. It is used to label the video chunks and associated keys, ensuring that the encryption and decryption processes can correctly match chunks to their respective keys.

Process of Encrypting AES Keys with RSA:

1. **Key Encryption**: Each AES key generated for the video chunks is encrypted using the RSA public key. This process ensures that the symmetric keys are securely stored and transmitted.

2. **Security Layer**: By encrypting the AES keys with RSA, the system adds a layer of security. Even if the AES keys are intercepted, they cannot be used without the RSA private key to decrypt them.

3. **Key Distribution**: The encrypted AES keys are stored alongside the video chunks or transmitted to the recipient securely. During decryption, the RSA private key is used to decrypt the AES keys, allowing the chunks to be decrypted and reassembled into the original video.

3.4.1 Video Chunk Encryption

In our Flask-based video encryption and decryption application, video chunk encryption is a critical process that ensures the confidentiality of the video content. This process leverages AES (Advanced Encryption Standard) encryption, utilizing dynamically generated keys for each video chunk. This section will explore the AES encryption process and how each chunk is encrypted using dynamically generated keys, tailored specifically to the implementation in our application.

1. Preparation of Video Chunks:

- The video file is first segmented into smaller chunks. Each chunk is then divided into 128-bit blocks, ready for encryption.

- Padding is applied if the final block of a chunk is not 128 bits long. The padding ensures that the last block conforms to the block size required by AES.

2. Key Association:

- Each chunk is associated with a unique, dynamically generated AES key. This key is crucial for the encryption process and is derived using the dynamic key generation methodology outlined earlier.

3. Encryption Execution:

- The AES encryption algorithm takes the 128-bit block and the associated dynamic key as inputs. In CBC mode, the first block is XORed with an initialization vector (IV) before encryption and next blocks with previous blocks.

- The AES algorithm processes each block, producing a corresponding block of ciphertext. This process continues until all blocks within a chunk are encrypted.

- The final output is the fully encrypted chunk, which consists of ciphertext blocks linked by the chaining mechanism inherent to CBC mode.

Security Considerations:

- **Confidentiality**: AES encryption ensures that the content of each video chunk remains confidential. The use of unique keys for each chunk further enhances security by preventing a single key compromise from exposing the entire video.

- **Integrity**: The use of CBC mode in AES encryption helps protect against certain types of cryptographic attacks, such as replay attacks, by ensuring that each block of ciphertext is dependent on all preceding plaintext blocks.

3.4.2 Encryption of Each Chunk Using Dynamically Generated Keys

Dynamic Key Generation:

In our application, dynamic key generation plays a pivotal role in securing video chunks. Each chunk is encrypted with a unique AES key, dynamically generated to provide an additional layer of security. This approach ensures that even if an attacker gains access to one key, only a single chunk is at risk, not the entire video.

Process of Dynamic Key Generation:

1. **Key Derivation**: Dynamic keys are derived using a combination of factors, including the chunk's index, a base key, and potentially other unique parameters such as timestamps or random numbers. This combination ensures that each key is unique to its corresponding chunk.
2. **Temporal and Contextual Factors**: Dynamic keys are influenced by temporal factors (e.g., the exact time the chunk is processed) and contextual factors (e.g., the specific video or user). This ensures that even repeated encryptions of the same video result in different keys and, consequently, different ciphertexts.
3. **Key Management**: Once generated, these keys are either stored securely if `SAVE_DYNAMIC_KEYS` is enabled or used solely for the encryption process before being discarded. The management of these keys is integral to ensuring that they can be securely retrieved for decryption if needed.

Encryption Using Dynamically Generated Keys

1. **Key Assignment:** Each dynamically generated key is assigned to its corresponding chunk. This key will be used exclusively for encrypting that specific chunk.
2. **Encryption Process:** The AES encryption algorithm is initialized with the dynamic key and the chunk data. In our application, each chunk undergoes the AES encryption process described earlier, with the dynamic key providing the cryptographic strength.

- The encryption is performed in CBC mode, ensuring that each block of the chunk is securely encrypted, and the output is a series of ciphertext blocks that represent the encrypted chunk.

3. **Secure Storage**: The resulting encrypted chunks are then securely stored in the application's designated storage system. If dynamic keys are saved, they are securely stored in a corresponding file. If not, they are discarded, making the ciphertext only decryptable by regenerating the exact key using the same dynamic process.

Advantages of Dynamic Key Use:

- **Enhanced Security**: The use of dynamic keys ensures that even if one chunk's encryption is compromised, it does not affect the security of other chunks.

- **Unpredictability**: The dynamic nature of key generation makes it extremely difficult for attackers to predict or replicate the encryption keys, thus securing the video content from unauthorized access.

- **Efficiency**: Despite the enhanced security, dynamic key generation and chunk encryption are designed to be efficient, minimizing the performance impact on the system.

3.5 Data Decryption

Data decryption is the reverse process of encryption, where the original video is reconstructed from its encrypted chunks. This section will explore the key retrieval and decryption process, as well as the decryption of each video chunk using AES and the retrieved keys, in the context of our Flask-based application.

3.5.1 Key Retrieval and Decryption

Retrieving the Encrypted AES Keys in Our App:

1. **Encrypted Key Storage**: During the encryption process, the AES keys used for each chunk are encrypted using the RSA public key and stored securely. The location and management of these encrypted keys depend on the application's configuration, specifically whether `SAVE_DYNAMIC_KEYS` is enabled.

2. **Key Retrieval Process**:

- Upon initiating the decryption process, the application retrieves the encrypted AES keys corresponding to the video chunks. The retrieval process ensures that the correct keys are matched to their respective chunks.

- If the keys were saved during encryption, they are retrieved from their secure storage location (e.g., a separate file in the user's directory). If not, the system regenerates the keys using the same dynamic process employed during encryption.

3.5.1 RSA Decryption of AES Keys

1. **RSA Decryption Proces**s:

- The encrypted AES keys are decrypted using the RSA private key. The RSA decryption process is computationally intensive but secure, ensuring that only authorized users with access to the private key can decrypt the AES keys.

- The application initializes the RSA decryption module with the private key and the encrypted AES key as inputs. The decryption algorithm processes the encrypted data, producing the original AES key.

2. **Reconstructing the Original AES Keys**:

- The decrypted AES keys are then reconstructed and prepared for use in the subsequent video chunk decryption process. Each key is linked to its corresponding video chunk, ensuring that the decryption process aligns with the original encryption structure.

- The keys are temporarily stored in memory during the decryption process to maintain security and prevent unauthorized access.

3.5.1 Video Chunk Decryption

Decrypting Video Chunks Using AES and the Retrieved Keys

1. **Initialization of AES Decryption**:

- With the AES keys retrieved and decrypted, the application initializes the AES decryption module. The same key length (256 bits) and mode of operation (CBC) used during encryption are employed here to ensure the integrity of the decryption process.

2. **Decryption Process**:

- The encrypted video chunks are processed in sequence, with each chunk being decrypted using its corresponding AES key. The AES algorithm decrypts each block of the chunk, reversing the encryption process and producing the original plaintext data.

- In CBC mode, the decryption of each block depends on the previous ciphertext block, ensuring that the chaining mechanism is correctly reversed. The initialization vector (IV) used during encryption is also utilized during decryption to properly decrypt the first block.

3.6 Reassembly of Video Chunks

Once all blocks within a chunk are decrypted, the plaintext data is reassembled into the original chunk. This chunk is then temporarily stored in its decrypted form, ready to be combined with other chunks to reconstruct the entire video.

3.6.1 Handling the Sequence of Chunk Decryption

1. **Sequential Decryption**:

- The video chunks are decrypted in the same order in which they were encrypted. The sequence is crucial to ensure that the video can be correctly reassembled without any data corruption or loss.

- The application manages the decryption sequence by referencing the chunk identifiers (e.g., video_chunk_01, video_chunk_02) that were assigned during encryption.

2. **Error Handling**:

- The decryption process includes error-handling mechanisms to manage any discrepancies in chunk sequence or key retrieval. For example, if a key cannot be retrieved or a chunk is missing, the application flags the issue and either attempts a recovery or notifies the user of the decryption failure.

3. **Final Reassembly**

- After all chunks are successfully decrypted, the application reassembles them into the final video file. This reassembly involves concatenating the chunks in the correct order and ensuring that any padding applied during encryption is removed.

- The result is a fully decrypted video file that matches the original input video, ready for playback or further processing by the user.

3.7 Process flow for Encryption operation

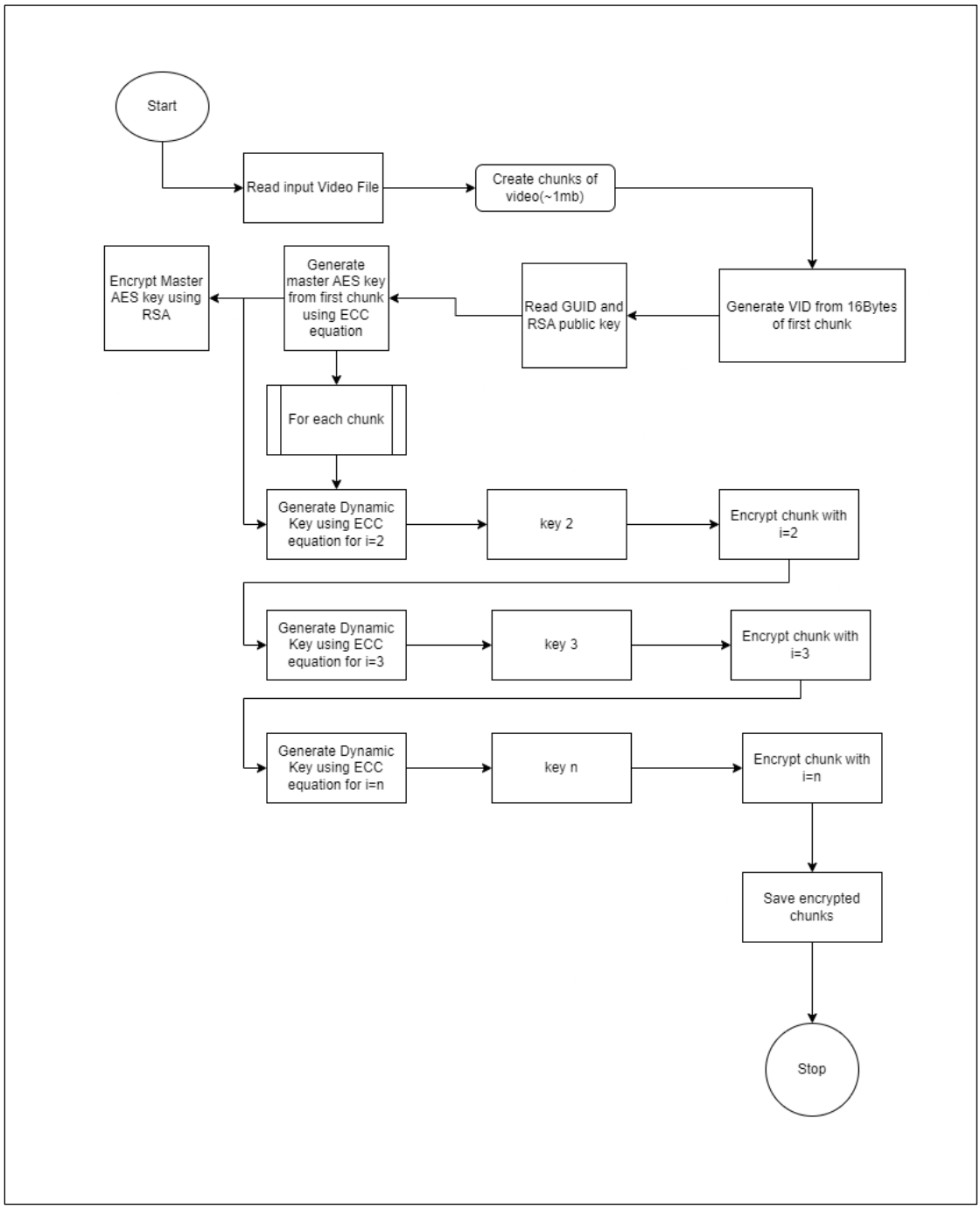

Figure 1: Illustration of the key process involved during encryption operation

3.8 Process flow for Decryption operation

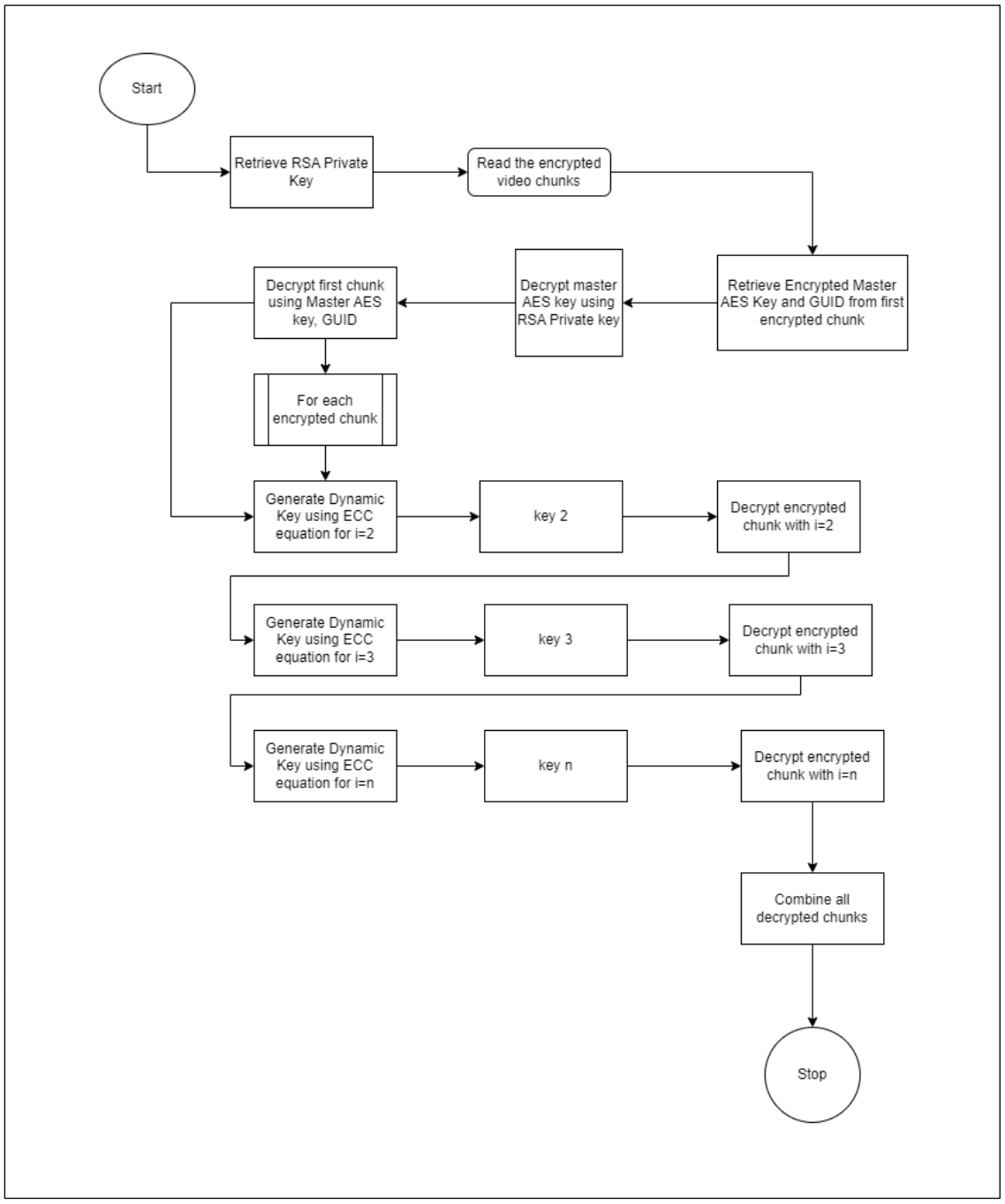

Figure 2: Illustration of the key process involved during decryption operation

CHAPTER – 04

IMPLEMENTATION

4.1 Complete List of Libraries

Below is the list of libraries used in our Flask application, along with details about their purpose and how they contribute to the functionality of our application.

1. base64

- Contribution to Our Application:

In our Flask application, `base64` is likely used to encode binary data such as encryption keys or video chunks into a format suitable for storage or transmission. For example, encoding an AES key before storing it in a database ensures that the data remains in a consistent and transportable format.

2. logging

- Contribution to Our Application:

The `logging` module is used in our Flask application to record various events, such as errors, warnings, or informational messages. This helps in tracking the execution flow and identifying issues during encryption, decryption, or user interactions within the application.

3. math

- Contribution to Our Application:

The `math` library might be used in our application for calculations related to video chunking, encryption processes, or key generation algorithms. For instance, determining the chunk size or calculating cryptographic parameters may rely on mathematical operations provided by this library.

4. os

- Contribution to Our Application:

In our application, `os` is crucial for file handling operations such as reading video files, saving encrypted chunks, or managing directories where encrypted keys and videos are stored. It also helps in defining file paths dynamically based on the environment.

5. time

- Contribution to Our Application:

The `time` library may be used to timestamp events, such as when a video was uploaded or encrypted. Additionally, it can be used to measure the time taken for encryption or decryption processes, which is vital for performance analysis and optimization.

6. hashlib

- Contribution to Our Application:

`hashlib` could be used in our application to generate checksums or digital signatures for video files or chunks. This ensures data integrity by verifying that files have not been altered during storage or transmission.

7. collections (Counter)

- Contribution to Our Application:

`Counter` might be utilized in our application to track occurrences of certain events or data patterns, such as counting user uploads or monitoring the frequency of specific actions. This can be useful for generating metrics or performing statistical analysis.

8. bcrypt

- Contribution to Our Application:

In our application, `bcrypt` is likely used to hash and verify user passwords securely. This ensures that even if the password database is compromised, the hashed passwords are not easily decipherable, thereby protecting user credentials.

9. datetime

- Contribution to Our Application:

`datetime` is used in our application for timestamping activities, such as recording the date and time of video uploads, encryptions, or user logins. It ensures that time-related data is managed consistently throughout the application.

10. cryptography

- Contribution to Our Application:

In our Flask application, the `cryptography` library is central to the encryption and decryption processes. It enables RSA and AES encryption, key generation, and secure data handling.

11. flask (Flask, request, redirect, url_for, session, flash, Response, render_template)

- Contribution to Our Application:

Flask forms the core of our application, managing the web server, handling HTTP requests, rendering templates, and maintaining user sessions. Each import from Flask (e.g., `request`, `redirect`, `url_for`) supports different aspects of web development, such as processing user input, navigating between pages, and rendering HTML pages dynamically.

12. flask_mysqldb (MySQL)

- Contribution to Our Application:

In our application, `flask_mysqldb` is used to connect to the MySQL database, where user information, video metadata, and possibly encrypted keys are stored. This extension allows seamless interaction between our Flask application and the underlying MySQL database, ensuring data is stored and retrieved efficiently.

13. matplotlib (pyplot)

- Contribution to Our Application:

`matplotlib.pyplot` is used in our application to generate visualizations, such as plotting metrics related to video encryption and decryption. These visualizations are crucial for analyzing performance and presenting data in a comprehensible manner to users or for work documentation.

14. random

- Contribution to Our Application:

In our Flask application, `random` may be used for generating random initialization vectors (IVs) for encryption, selecting random dynamic keys, or other cryptographic operations that require randomness to enhance security.

15. sympy (isprime)

- Contribution to Our Application:

The `isprime` function in `sympy` might be used in our application to verify the primality of numbers during the RSA key generation process. Since RSA relies on prime numbers for generating secure keys, ensuring the primality of these numbers is critical for maintaining encryption strength.

This comprehensive list details the purpose and application of each library used in our Flask-based video encryption and decryption system. Each library plays a specific role, contributing to the overall functionality and security of the application.

Comprehensive List of Hardware and Software Requirements for the Flask-Based Video Encryption and Decryption Application.

4.2 Hardware Requirements

1. Processor:

- Minimum: Dual-core processor (Intel Core i3 or equivalent)

- Recommended: Quad-core processor (Intel Core i5 or equivalent)

- Rationale: A multi-core processor is required to handle multiple tasks concurrently, such as video processing, encryption/decryption, and web server management.

2. Memory (RAM):

- Minimum: 4 GB

- Recommended: 8 GB or more

- Rationale: Sufficient memory is needed to efficiently process video files, manage user sessions, and perform cryptographic operations without performance bottlenecks.

3. Storage:

- Minimum: 20 GB of available disk space

- Recommended: SSD with at least 50 GB of free space

- Rationale: Storage is needed to store video files, encrypted chunks, logs, and database files. SSDs offer faster read/write speeds, improving the overall performance.

4. Graphics:

- Minimum: Integrated graphics card

- Recommended: Dedicated graphics card (NVIDIA GTX series or equivalent)

- Rationale: If video processing involves intensive tasks such as transcoding, a dedicated graphics card can offload these tasks from the CPU, improving efficiency.

4.3 Software Requirements

1. Operating System:

- Minimum:

  - Windows 10 (64-bit)

- Recommended:

  - Windows 11 (64-bit)

- Rationale: The application is cross-platform, running on both Windows and Linux. A 64-bit OS is necessary to handle large file processing and modern software dependencies.

2. Python:

- Version: Python 3.8 or later

- Rationale: The application is built on Python 3, and it's crucial to use a version that supports the latest features and security patches. Python 3.8 or later is recommended to ensure compatibility with all libraries.

3. Web Server:

- Flask:

  - Version: Flask 2.0 or later

  - Rationale: Flask serves as the web framework for the application, handling routing, session management, and rendering templates. Flask 2.0 provides the necessary features and security improvements.

- Gunicorn (for production deployment):

- Version: Gunicorn 20.0 or later

- Rationale: Gunicorn is a Python WSGI HTTP Server for UNIX, used for serving the Flask application in production. It is lightweight and scalable, making it ideal for deploying Flask applications.

4. Database:

- MySQL:

- Version: MySQL 5.7 or later

- Rationale: MySQL is used to store user information, video metadata, and encryption keys. MySQL 5.7 or later ensures compatibility with `flask_mysqldb` and provides the necessary performance and security features.

5. Development Tools:

- Integrated Development Environment (IDE):

- Recommended: PyCharm Professional Edition, Visual Studio Code with Python and Flask extensions

- Rationale: A powerful IDE with support for Python, Flask, and database integration will enhance development efficiency and debugging capabilities.

6. Version Control:

- Git: - Version: Git 2.20 or later

- Rationale: Git is essential for version control, allowing developers to manage code changes, collaborate with others, and deploy updates efficiently.

This comprehensive list of hardware and software requirements provides a detailed overview of the necessary components to successfully run and develop your Flask-based video encryption and decryption application. Ensuring that these requirements are met will help in achieving optimal performance, security, and reliability of your application.

The complete implementation of this system is uploaded in the github repo as in https://github.com/lateefcode2101/CipherShield along with required results.

CHAPTER – 05

RESULTS AND DISCUSSIONS

5.1 Implementation

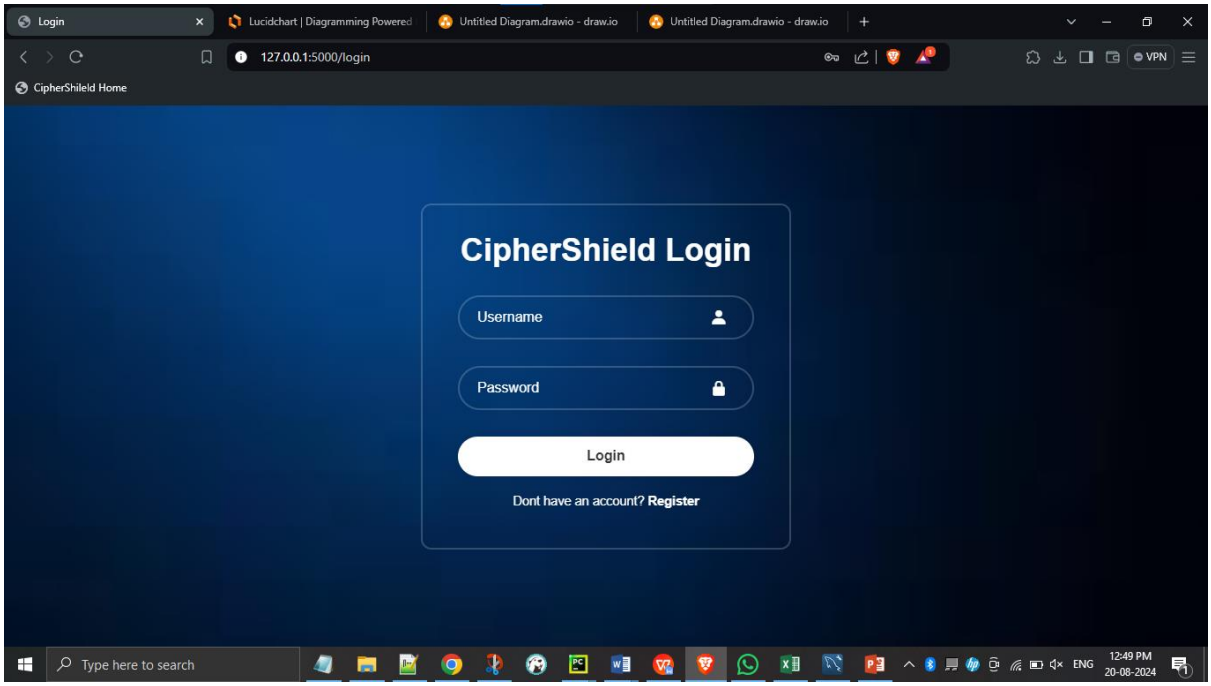

Figure 3: Login page for CipherShield

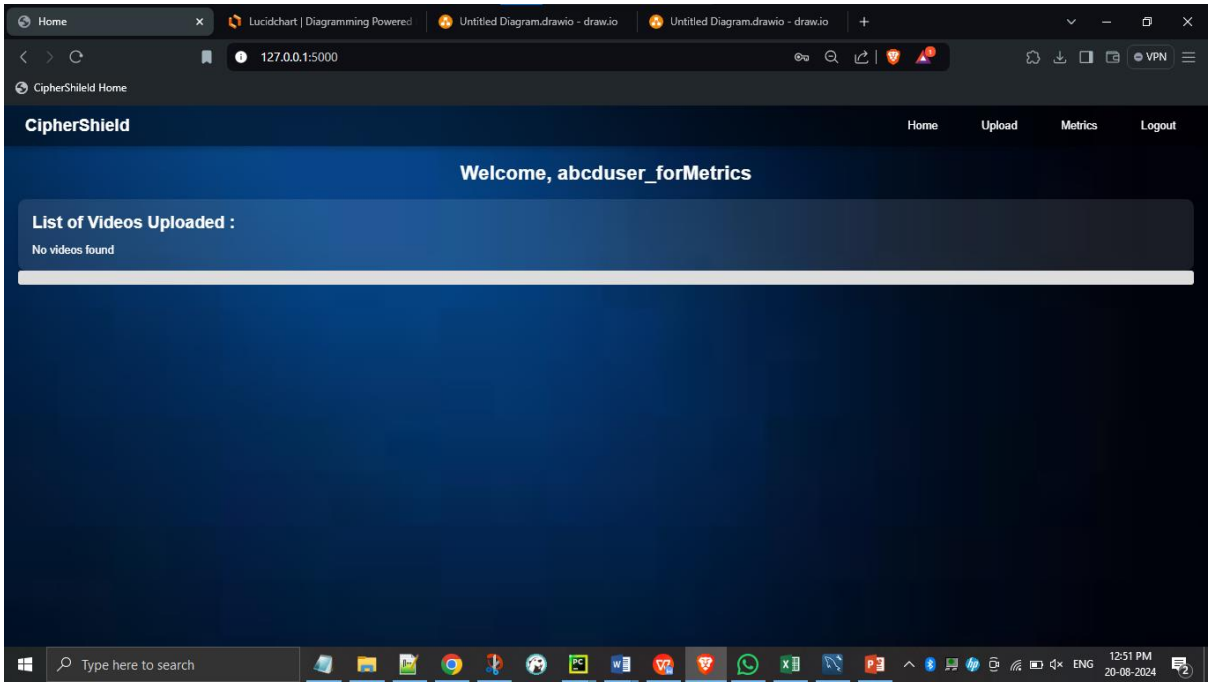

Figure 4: Index page after the user logs into the system

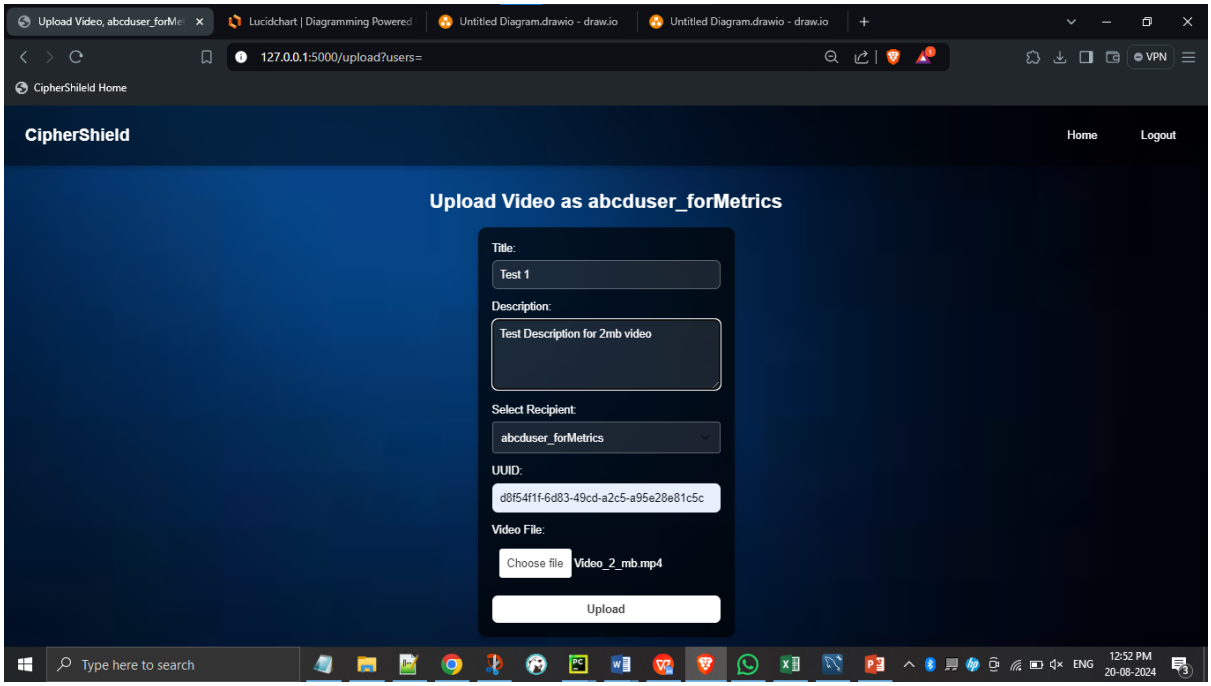

Figure 5: Upload page for test upload 1 with 2 mb video file

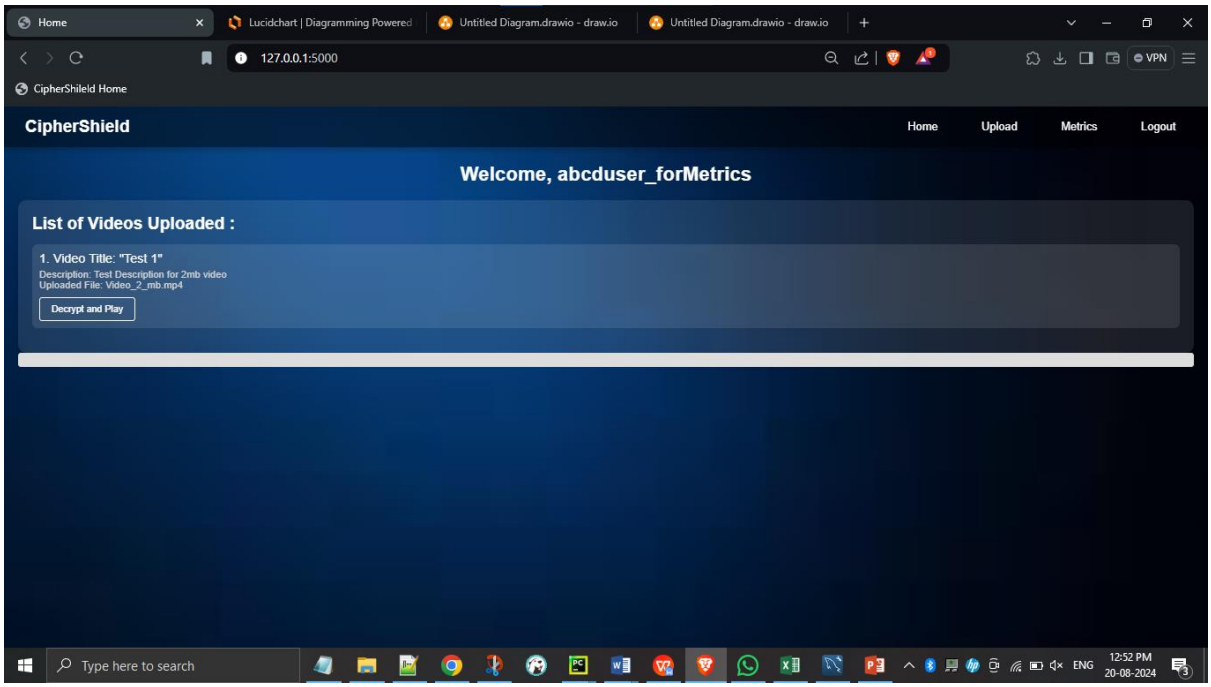

Figure 6: Index page showing the test upload 1- video is uploaded successfully

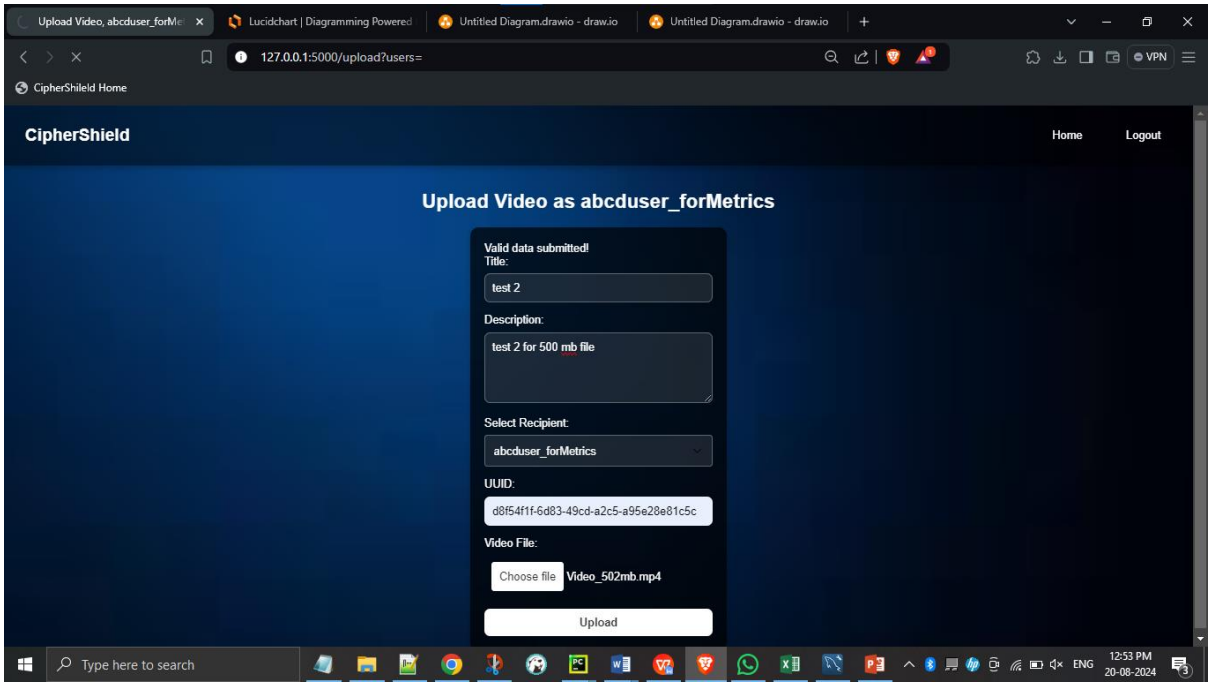

Figure 7: Upload page for test upload - 2 with 500 mb video file

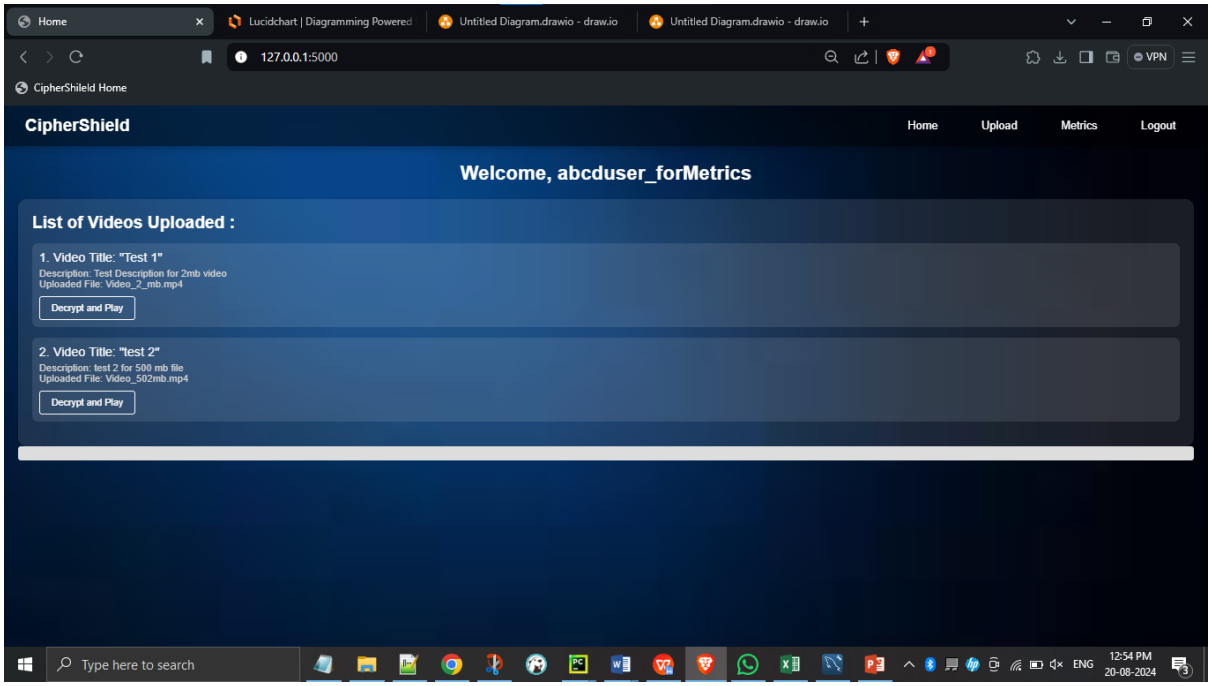

Figure 8: Index page showing the test upload - 2 video is uploaded successfully

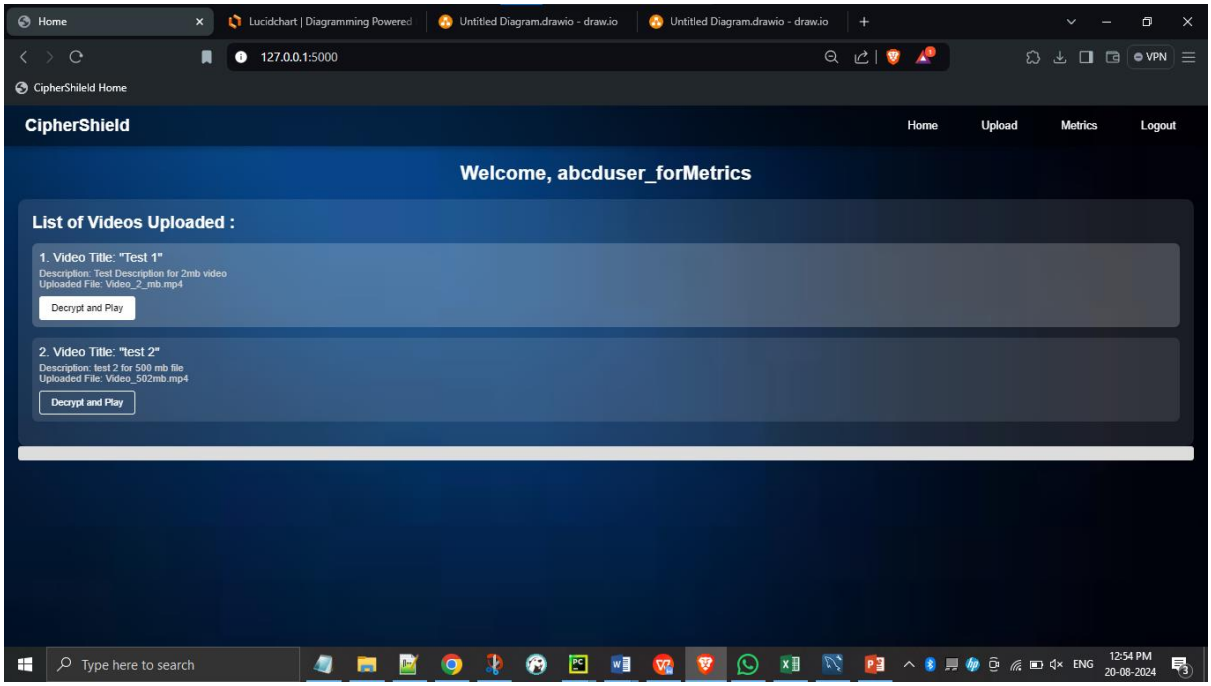

Figure 9: Figure showcasing Decryption of Test upload -1

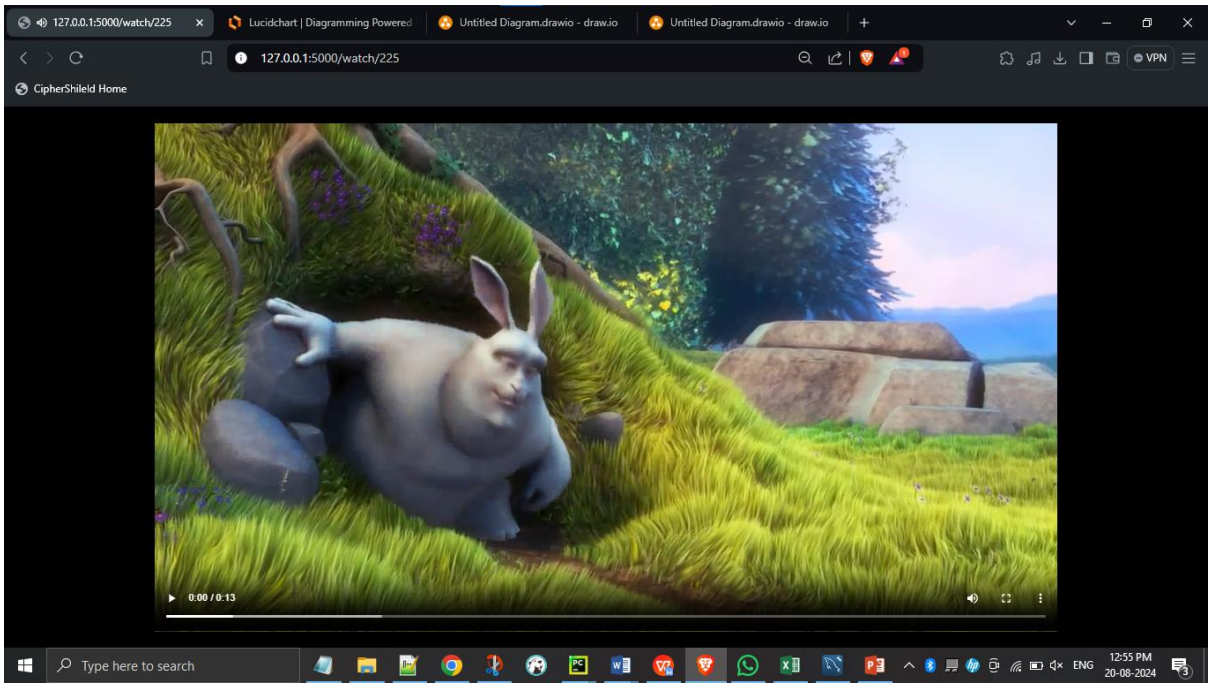

Figure 10: Figure showcasing decrypted Test upload1

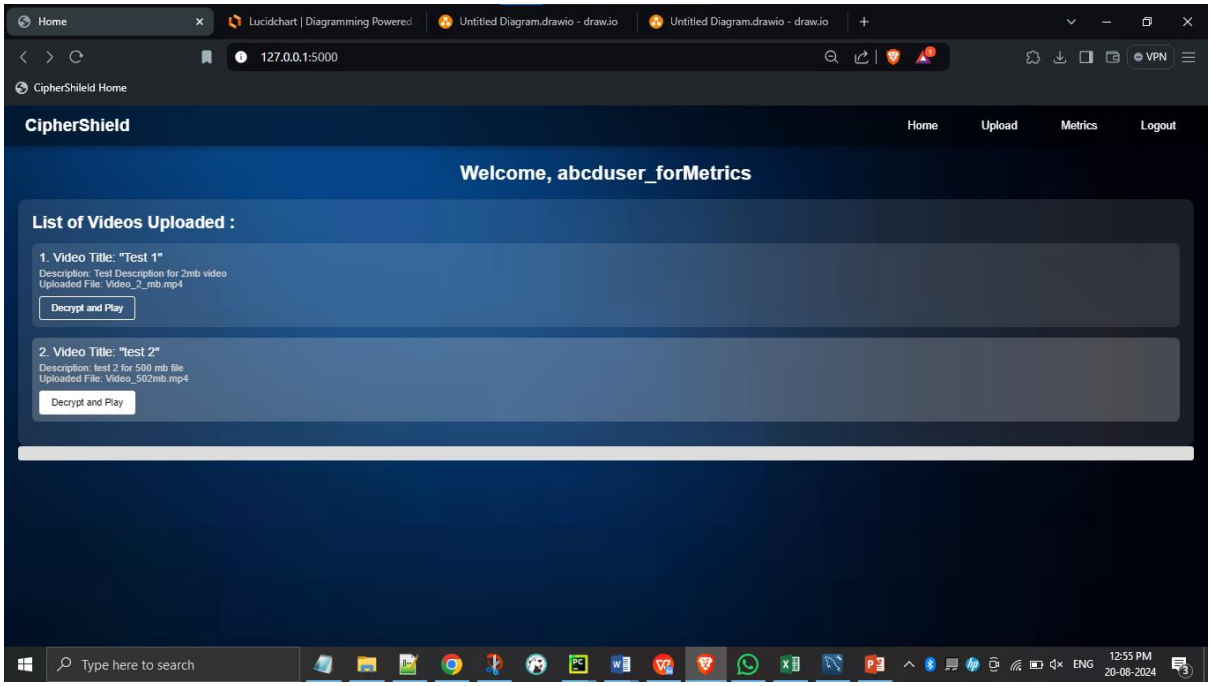

Figure 11: Figure showcasing Decryption of Test upload 2

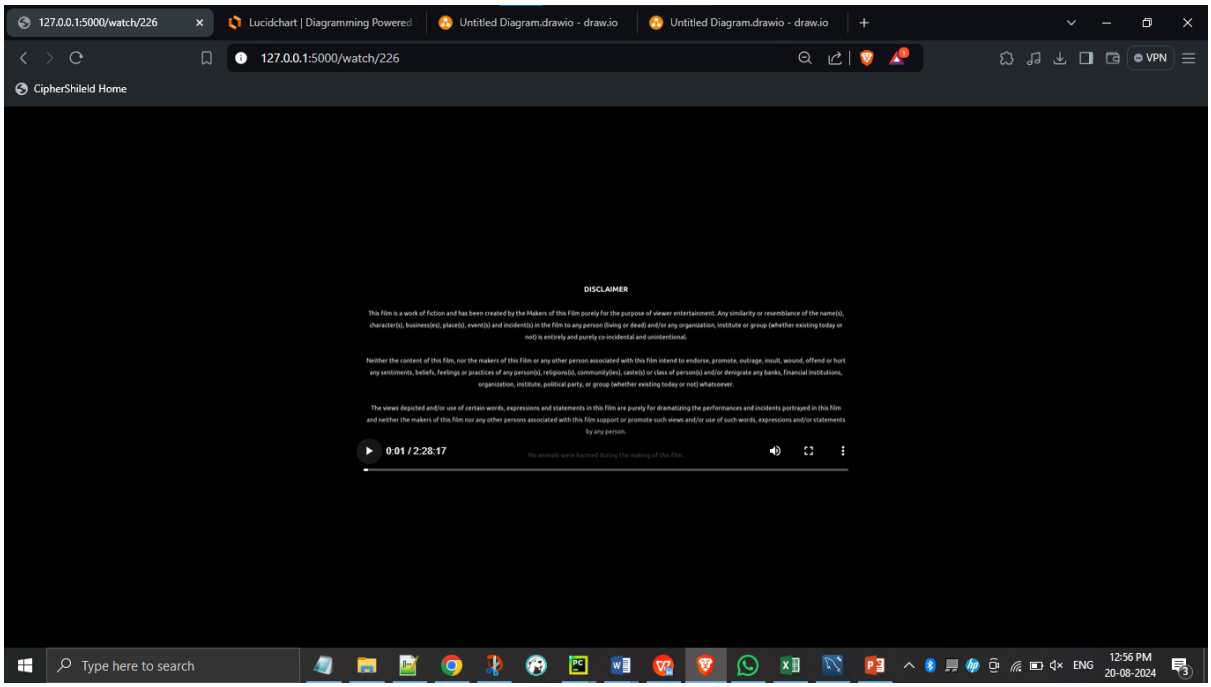

Figure 12: Figure showcasing decrypted Test upload 2

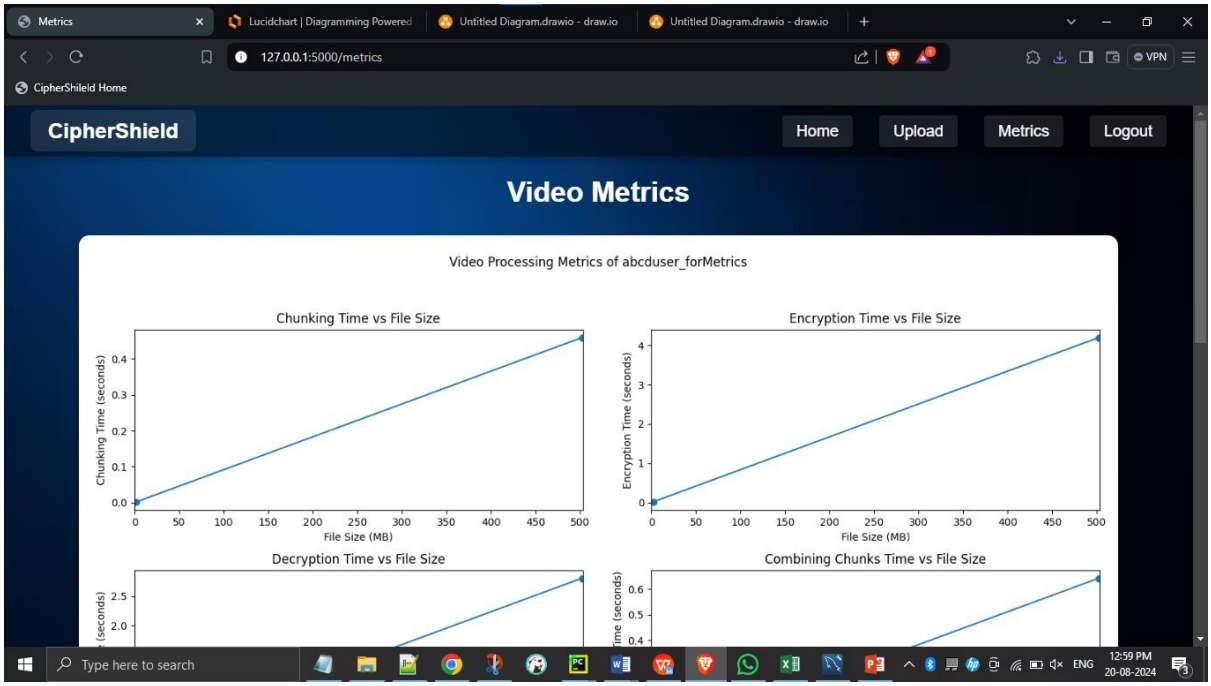

Figure 13: Metrics page for user [chunking time, encryption time]

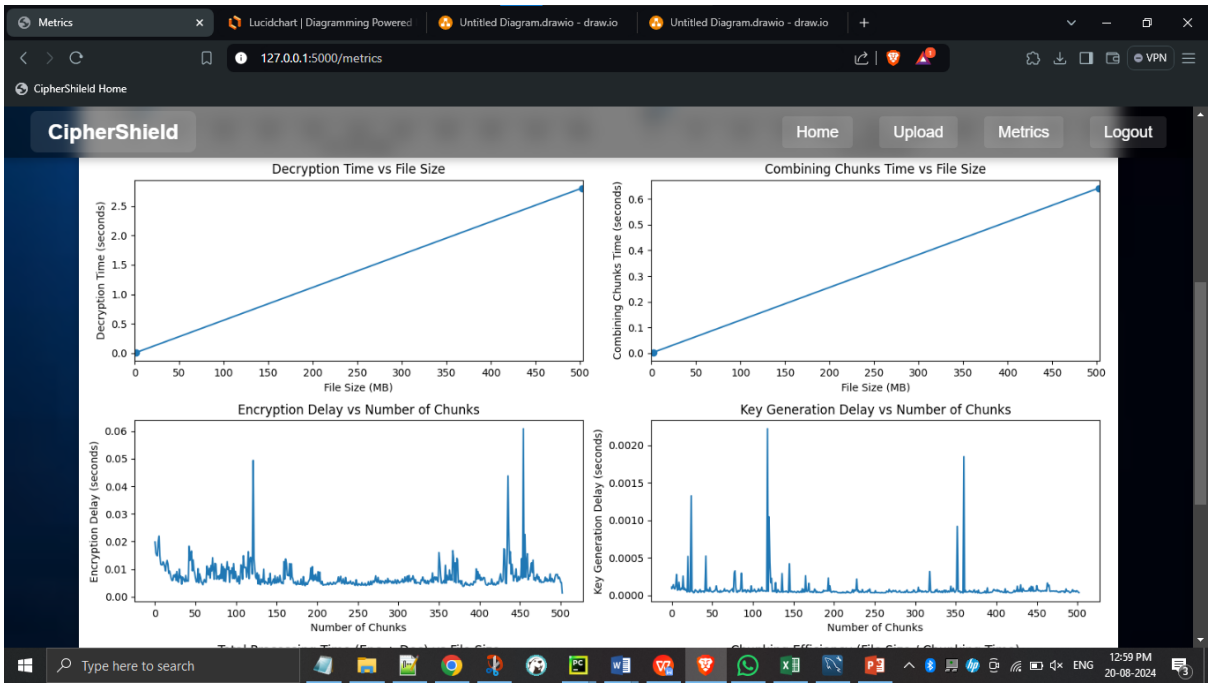

Figure 14: Metrics page for user [decryption time, combining time, encryption delay and key generation delay– only for 500mb video file]

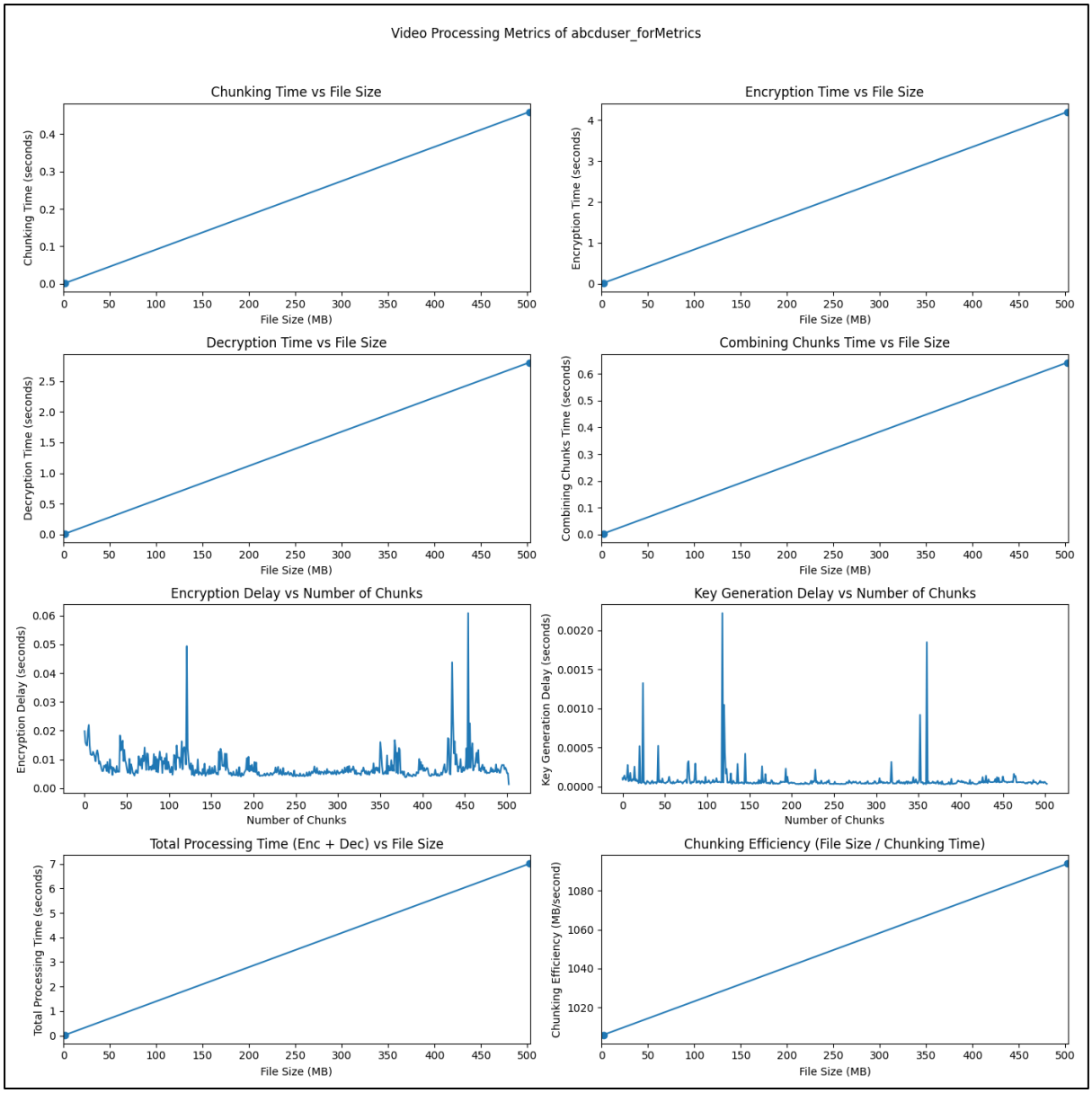

Figure 15: Complete plot made by CipherShield system for user

5.2 Comparison Results

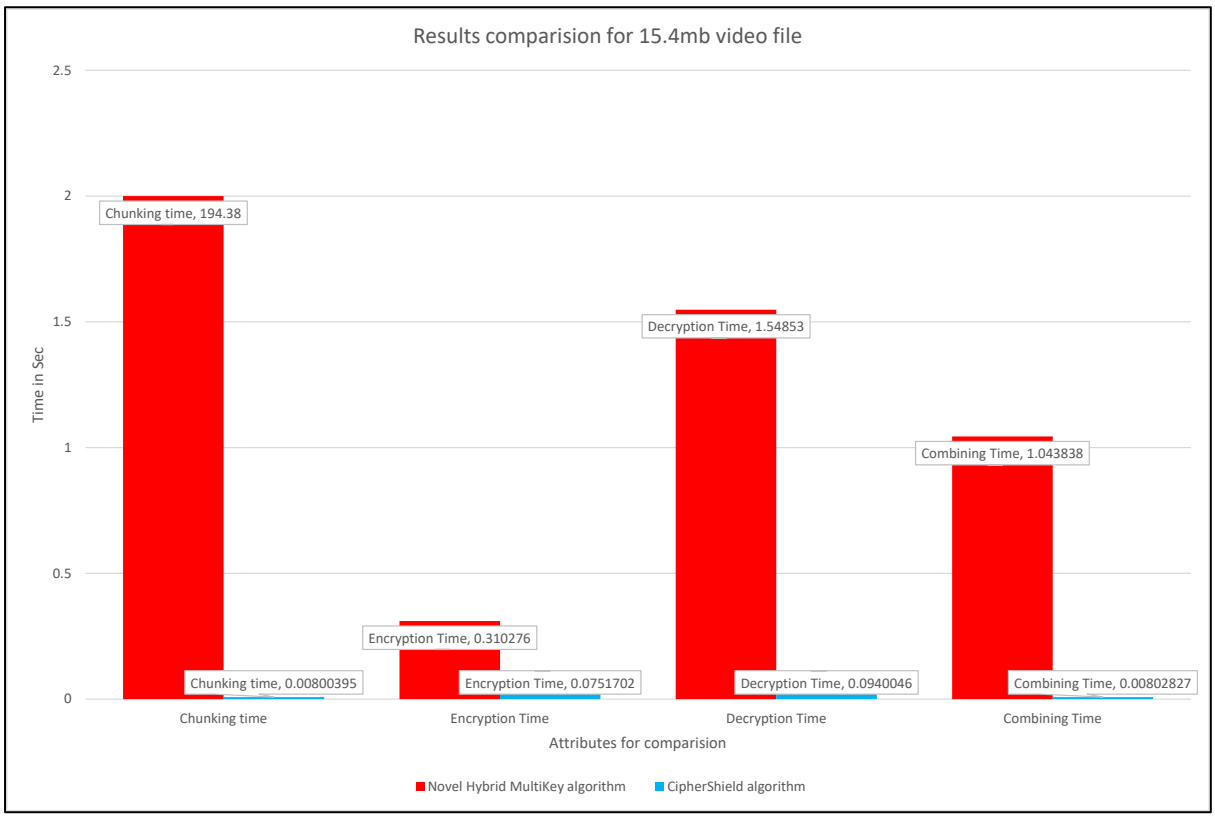

Figure 16: Figure comparing Novel Hybrid Multi Key Algorithm [1] vs CipherShield Algorithm

CHAPTER – 06
CONCLUSION AND FUTURE ENHANCEMENTS

6.1 CONCLUSION

This work presents a sophisticated and highly secure method for video encryption and decryption, designed to protect sensitive content while maintaining efficiency. The developed system leverages advanced cryptographic techniques, including dynamic key generation and chunk-wise encryption, to ensure that video data remains secure throughout the transmission and storage processes. By employing these innovative methods, the proposed system not only enhances the security of video files but also optimizes processing times, making it suitable for real-time applications.

The results of this work demonstrate significant improvements in both security and performance compared to existing solutions. The dynamic key generation mechanism, coupled with the chunk-based encryption approach, ensures that even if one segment of the video is compromised, the integrity of the remaining content is preserved. The results in figure 14 depict the huge increase in efficiency with lesser computation time for chunking, encryption, decryption and decrypted chunk concatenation/combination. Furthermore, the system's ability to handle large video files efficiently without sacrificing security makes it a promising solution for a wide range of applications, from secure video conferencing to protected video-on-demand services.

In conclusion, this work contributes to the field of video encryption by offering a robust and efficient method that addresses the limitations of traditional encryption techniques. The system's ability to adapt to various scenarios, combined with its strong security guarantees, positions it as a valuable tool for protecting video content in an increasingly digital world. Future work could explore the integration of this system with other security protocols, further enhancing its applicability in different environments.

6.2 FUTURE ENHANCEMENT

Implement a distributed decryption mechanism using Secure Multi-Party Computation (SMPC) to decentralize the decryption process. Instead of relying on a single server or device for decryption, this enhancement involves splitting the decryption key into multiple parts and distributing them across different trusted nodes. Each node would perform a part of the decryption process without revealing its key share. The final video is decrypted only when all nodes collaborate.

This enhancement involves integrating SMPC protocols, managing the distribution and synchronization of decryption tasks, and ensuring that the system remains efficient despite the added complexity. The design must also ensure the integrity and confidentiality of the decryption process, even if some nodes are compromised.

SMPC enhances the security and resilience of the decryption process by removing single points of failure and reducing the risk of key exposure. It also opens up possibilities for collaborative decryption in distributed environments, making the application suitable for highly secure and sensitive scenarios, such as government or military communications.

REFERENCES

[1] Y. Fouzar, A. Lakhssassi and M. Ramakrishna, "A Novel Hybrid Multikey Cryptography Technique for Video Communication," in IEEE Access, vol. 11, pp. 15693-15700, 2023, doi: 10.1109/ACCESS.2023.3242616.

[2] M. Iavich, S. Gnatyuk, E. Jintcharadze, Y. Polishchuk and R. Odarchenko, "Hybrid Encryption Model of AES and ElGamal Cryptosystems for Flight Control Systems," 2018 IEEE 5th International Conference on Methods and Systems of Navigation and Motion Control (MSNMC), Kiev, Ukraine, 2018, pp. 229-233, doi: 10.1109/MSNMC.2018.8576289.

[3] Y. Hu, L. Gong, J. Zhang and X. Luo, "An Efficient Hybrid Encryption Scheme for Encrypting Smart Grid Business Data," 2023 IEEE 11th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, China, 2023, pp. 1490-1494, doi: 10.1109/ITAIC58329.2023.10408778.

[4] C. Prashanth, M. Mohamed, K. Latha, S. Hemavathi and D. Venkatesh, "Enhanced Hybrid Encryption Through Slicing and Merging of Data with Randomization of Algorithms," 2021 4th International Conference on Computing and Communications Technologies (ICCCT), Chennai, India, 2021, pp. 376-380, doi: 10.1109/ICCCT53315.2021.9711883.

[5] T. Yue, C. Wang and Z. -x. Zhu, "Hybrid Encryption Algorithm Based on Wireless Sensor Networks," 2019 IEEE International Conference on Mechatronics and Automation (ICMA), Tianjin, China, 2019, pp. 690-694, doi: 10.1109/ICMA.2019.8816451.

[6] Huahong Ma, Bowen Ji, Honghai Wu, Ling Xing, "Video data offloading techniques in Mobile Edge Computing: A survey", Physical Communication, Volume 62, 2024, 102261, ISSN 1874-4907, doi: 10.1016/j.phycom.2023.102261

[7] M. A. El-Mowafy, S. M. Gharghory, M. A. Abo-Elsoud, M. Obayya and M. I. Fath Allah, "Chaos Based Encryption Technique for Compressed H264/AVC Videos," in IEEE Access, vol. 10, pp. 124002-124016, 2022, doi: 10.1109/ACCESS.2022.3223355

[8] Q. Zhang, "An Overview and Analysis of Hybrid Encryption: The Combination of Symmetric Encryption and Asymmetric Encryption," 2021 2nd International Conference on Computing and Data Science (CDS), Stanford, CA, USA, 2021, pp. 616-622, doi: 10.1109/CDS52072.2021.00111.

[9] X. Zhou, H. Wang, K. Li, L. Tang, N. Mo and Y. Jin, "A Video Streaming Encryption Method and Experimental System Based on Reconfigurable Quaternary Logic Operators," in IEEE Access, vol. 12, pp. 25034-25051, 2024, doi: 10.1109/ACCESS.2024.3365523

[10] T. Shanableh, "HEVC Video Encryption With High Capacity Message Embedding by Altering Picture Reference Indices and Motion Vectors," in IEEE Access, vol. 10, pp. 22320-22329, 2022, doi: 10.1109/ACCESS.2022.3152548.

[11] B. Jiang, Q. He, P. Liu, S. Maharjan and Y. Zhang, "Blockchain Empowered Secure Video Sharing With Access Control for Vehicular Edge Computing," in IEEE Transactions on Intelligent Transportation Systems, vol. 24, no. 9, pp. 9041-9054, Sept. 2023, doi: 10.1109/TITS.2023.3269058.

[12] J. Arif et al., "A Novel Chaotic Permutation-Substitution Image Encryption Scheme Based on Logistic Map and Random Substitution," in IEEE Access, vol. 10, pp. 12966-12982, 2022, doi: 10.1109/ACCESS.2022.3146792.

[13] M. Yu, H. Yao, C. Qin and X. Zhang, "A Comprehensive Analysis Method for Reversible Data Hiding in Stream-Cipher-Encrypted Images," in IEEE Transactions on Circuits and Systems for Video Technology, vol. 32, no. 10, pp. 7241-7254, Oct. 2022, doi: 10.1109/TCSVT.2022.3172226.

[14] A. Al-Hyari, C. Obimbo, M. M. Abu-Faraj and I. Al-Taharwa, "Generating Powerful Encryption Keys for Image Cryptography With Chaotic Maps by Incorporating Collatz Conjecture," in IEEE Access, vol. 12, pp. 4825-4844, 2024, doi: 10.1109/ACCESS.2024.3349470