DYNAMIC ENCRYPTION USING VIGENÈRE CIPHER AND IMAGE-DERIVED KEYS

A PROJECT REPORT

Submitted by

ABHEELASH MISHRA [RA2111003011143]
PRIYANSHI PUSAN [RA2111003011144]

Under the Guidance of

DR. MALARSELVI G

(Assistant Professor, Department of Computing Technologies)

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTING TECHNOLOGIES COLLEGE OF ENGINEERING AND TECHNOLOGY
SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR- 603 203

MAY 2025

Department of Computing Technologies
SRM Institute of Science & Technology
Own Work* Declaration Form

This sheet must be filled in (each box ticked to show that the condition has been met). It must be signed and dated along with your student registration number and included with all assignments you submit – work will not be marked unless this is done.

To be completed by the student for all assessments

Degree/ Course : B.Tech Computer Science and Engineering

Student Name : Abheelash Mishra, Priyanshi Pusan

Registration Number : RA2111003011143, RA2111003011144

Title of Work : Dynamic Encryption Using Vigenère Cipher And Image-Derived Keys

I / We hereby certify that this assessment compiles with the University's Rules and Regulations relating to Academic misconduct and plagiarism, as listed in the University Website, Regulations, and the Education Committee guidelines.

I / We confirm that all the work contained in this assessment is my / our own except where indicated, and that I / We have met the following conditions:

- Clearly referenced / listed all sources as appropriate
- Referenced and put in inverted commas all quoted text (from books, web, etc)
- Given the sources of all pictures, data etc. that are not my own
- Not made any use of the report(s) or essay(s) of any other student(s) either past or present
- Acknowledged in appropriate places any help that I have received from others (e.g. fellow students, technicians, statisticians, external sources)
- Compiled with any other plagiarism criteria specified in the Course handbook / University website

I understand that any false claim for this work will be penalized in accordance with the University policies and regulations.

DECLARATION:
I am aware of and understand the University's policy on Academic misconduct and plagiarism and I certify that this assessment is my / our own work, except where indicated by referring, and that I have followed the good academic practices noted above.
If you are working in a group, please write your registration numbers and sign with the date for every student in your group.

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR – 603 203

BONAFIDE CERTIFICATE

Certified that 18CSP111L - Major Project report titled "**DYNAMIC ENCRYPTION USING VIGENÈRE CIPHER AND IMAGE-DERIVED KEYS**" is the bonafide work of **ABHEELASH MISHRA [RA2111003011143], PRIYANSHI PUSAN [RA2111003011144]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

DR. MALARSELVI G
SUPERVISOR
Assistant Professor
Department of Computing Technologies

SIGNATURE

DR. NIRANJANA G
PROFESSOR & HOD
Department of Computing Technologies

SIGNATURE

DR. MALARSELVI G
PANEL HEAD
Associate Professor
Department of Computing Technologies

ACKNOWLEDGEMENTS

We express our humble gratitude to **Dr. C. Muthamizhchelvan**, Vice-Chancellor, SRM Institute of Science and Technology, for the facilities extended for the project work and his continued support.

We extend our sincere thanks to **Dr. T. V. Gopal ,** Dean - CET, SRM Institute of Science and Technology, for his invaluable support.

We wish to thank **Dr. Revathi Venkataraman**, Professor and Chairperson, School of Computing, SRM Institute of Science and Technology, for her support throughout the project work.

We encompass our sincere thanks to **Dr. M. Pushpalatha**, Professor and Associate Chairperson, School of Computing and **Dr. C. Lakshmi,** Professor and Associate Chairperson, School of Computing, SRM Institute of Science and Technology, for their invaluable support.

We are incredibly grateful to our Head of the Department, **Dr. G. Niranjana**, Professor, Department of Computing Technologies, SRM Institute of Science and Technology, for her suggestions and encouragement at all the stages of the project work.

We want to convey our thanks to our Project Coordinators **Dr. R. Vidhya**, **Dr. M. Arul Prakash** and **Dr. M. Revathi**, our Panel Head **Dr. G. Malarselvi**, and Panel Members **Mrs. Brindha R** and **Dr. S. Ramesh,** Department of Computing Technologies, SRM Institute of Science and Technology, for their inputs during the project reviews and support.

We register our immeasurable thanks to our Faculty Advisor **Dr. G. Malarselvi**, Department of Computing Technologies, SRM Institute of Science and Technology, for leading and helping us to complete our course.

Our inexpressible respect and thanks to our guide **Dr. G. Malarselvi**, Department of Computing Technologies, SRM Institute of Science and Technology, for providing us with an opportunity to pursue our project under her mentorship. She provided us with the freedom and support to explore the research topics of our interest. Her passion for solving problems and making a difference in the world has always been inspiring.

We sincerely thank all the staff and students of Computing Technologies, School of Computing, S.R.M Institute of Science and Technology, for their help with our project. Finally, we would like to thank our parents, family members, and friends for their unconditional love, constant support and encouragement.

Abheelash Mishra (RA2111003011143)
Priyanshi Pusan (RA2111003011144)

ABSTRACT

The Vigenère cipher, while of historical importance, is susceptible to frequency analysis and brute-force attacks because it is based on linguistic keys. This paper presents a new method that is more secure by producing random, high-entropy keys from digital images. Through the use of image information, the system avoids frequency-based attacks and eliminates key memorization. The framework consists of three key phases: image preprocessing, where inputs are standardized; localized patch hashing, ensuring tamper sensitivity; and iterative key derivation, producing cryptographically sound material. Additionally, the approach operates covertly, keeping encryption mechanisms hidden from adversaries. Keys can be stored inconspicuously within image collections, creating a significant challenge for attackers. Session-based encryption ensures that each encryption instance is unique, preventing replay attacks and key reuse. Experimental results demonstrate that image-derived keys significantly strengthen the Vigenère cipher. Even minor image modifications (e.g., cropping, watermarking, noise) lead to drastically different base keys, rendering decrypted text unintelligible. The method effectively disrupts traditional letter frequency biases, making cryptanalysis more difficult. This work contributes: (1) a deterministic algorithm for binding visual data to cryptographic keys, (2) empirical validation of image-based keys against cryptanalysis, and (3) session-based encryption to limit key exposure. The results confirm its robustness, with high cipher text entropy and sensitivity to image modifications, making decryption infeasible without the exact image. This work lays the foundation for future advancements in dynamic and secure encryption.

TABLE OF CONTENTS

CHAPTER	PAGE
ABSTRACT	**v**
TABLE OF CONTENTS	**vi**
LIST OF FIGURES	**viii**
LIST OF TABLES	**ix**
ABBREVIATIONS	**x**
1 INTRODUCTION	**1**
1.1 Evolution and Vulnerabilities of the Vigenère Cipher	1
1.2 Advantages of Visual Key Generation	2
1.3 Key Contributions	3
2 LITERATURE SURVEY	**5**
3 TECHNICAL SPECIFICATIONS	**7**
3.1 Programming Language and Execution Environment	7
3.1.1 Language Choice	7
3.1.2 Platform Compatibility	8
3.1.3 Development Environment	10
3.2 Libraries and Dependencies	11
3.2.1 Cryptographic and Hashing Libraries	11
3.2.2 Image Data Processing	13
3.2.3 Entropy Visualization and Analysis	14
3.3 Cryptographic Techniques Used	15
3.4 Security Features	16
4 ARCHITECTURE	**19**
4.1 Overview	19
4.2 Image Based Key Generation	21
4.2.1 Image Preprocessing	21
4.2.2 Key Extraction Process	22
4.2.3 Secret Word Extraction	23
4.3 Modified Vigenère Cipher	24
4.4 Session Management	25
5 MODULES	**28**
5.1 Image Preprocessing Module	28

5.2 Feature Extraction Module 29
5.3 Base Key Derivation Module 30
5.4 Dynamic Key Derivation Module 32
5.5 Secret Word Extraction Module 34
5.6 Vigenère Cipher Module 34
5.7 Security Validation and Statistical Analysis Module 37
6 PROJECT DEMONSTRATION AND RESULTS 40
5.1 Experimental Setup 40
5.2 Encryption Key Generation and Entropy Evaluation 42
5.3 Sensitivity to Image Alterations 45
5.4 Encryption Strength with Low-Detail Images 46
5.5 Execution Time Analysis 47
7 CONCLUSIONS 50
7 FUTURE ENHANCEMENT 51
REFERENCES 53
APPENDIX
A CODING 55
B CONFERENCE PUBLICATION 63
C PLAGIARISM REPORT 64

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
4.1	**Architecture diagram of the encryption system**	25
5.1	**Vigenère cipher representation**	26
5.2	**Encryption process**	27
6.1	**Image that is used for key derivation**	28
6.2	**Cipher text generated from the plain text input using the scenery image**	28
6.3	**Comparative frequency analysis between plain text and cipher text**	30
6.4	**Failure message displayed when using an invalid image**	32

LIST OF TABLES

TABLE NO.	TITLE	PAGE NO.
6.1	**Average execution times of the encryption and decryption system**	48

ABBREVIATIONS

AES	Advanced Encryption System
RSA	Rivest-Shamir-Adleman
PBKDF2	Password-Based Key Derivation Function 2
SHA	Secure Hash Algorithm
ASCII	American Standard Code for Information Interchange
ECC	Elliptic Curve Cryptography

CHAPTER 1

INTRODUCTION

The Vigenère cipher, created as a polyalphabetic substitution in the 16th century, transformed traditional cryptography by combining cyclical key shifts to hide plain text letter frequencies [1]. Although very common in earlier periods of time, short, repetitive, or linguistically patterned keys render the cipher susceptible to contemporary cryptanalysis methods like frequency analysis and brute-force attacks [2]. This weakness is present in both its original and some of its derivative modifications. Although contemporary encryption mechanisms such as AES and RSA made traditional ciphers impractical in real-world application [3], the Vigenère cipher is still worth studying as it provides an inspiration for creating innovative key generation protocols that combine ancient methods with the principles of advanced cryptography.

1.1. Evolution and Vulnerabilities of the Vigenère Cipher

Part of the traditional cryptography's central problem is relying on user-based keys, limited by human memories and linguistic leanings. Within classical systems such as the Vigenère cipher, which depend heavily on repeated-key polyalphabetic, security in the encrypting process itself is a matter of the fortitude and random nature of the key. However, user-created keys tend to be short, reused, and based on natural language, all of which reduce entropy and increase vulnerability to cryptanalysis techniques such as frequency analysis and known-plaintext attacks.

Recent advances in cryptographic research have sought to address these limitations by leveraging high-entropy sources for key generation. For instance, physical phenomena like van der Waals heterojunctions [4], which have distinctive quantum-level properties, have been put forward as hardware-based sources of entropy. In the same way, biometric systems based on features such as fingerprints or retinal patterns [5] provide a personalized yet fairly secure means of obtaining cryptographic material. Such approaches are an attempt to insert entropy into physical or biological processes in order to address the limitations of user-provided keys.

Even with all these developments, there is one promising direction left untapped: the application of digital images as cryptographic keys, specifically in the context of classical ciphers such as the Vigenère cipher. Natural digital images have some very appealing qualities. Their very visual complexity has a high degree of entropy, and their widespread presence in digital environments enables them to be used as stealthy cryptographic keys. Images can be simply embedded, concealed in plain sight, or covertly transmitted, hence minimizing the chances of detection by assailants.

This project aims to address the neglected potential of visual data in traditional cryptosystems. Through the incorporation of images as a source of material in the Vigenère cipher, this method is intended to give new life to the cipher's functionality without affecting computational efficiency, particularly in environments with limited resources like embedded systems, IoT devices, and mobile platforms.

1.2. Advantages of Visual Key Generation

One of the greatest advantages of this approach is that it is extremely easy and simple to implement. Since the system only needs an image for key generation, even those with no technical expertise can easily encrypt their information. With the assistance of generic digital images (e.g., self-portraits or digital images obtained from online image galleries), users can securely encrypt their information without needing to understand complex cryptographic principles or manage cryptographic keys. This renders the encryption process both easy to use and secure, opening up the capability to secure sensitive information with a high level of security to any user.

This research tackles such challenges using a new paradigm that strengthens the Vigenère cipher through image-key generation and session encryption. Rather than using traditional linguistic keys, our methodology uses digital images to produce strong, unbiased cryptographic keys, guaranteeing high entropy and immunity to widespread attacks. The encryption process involves three critical stages:

- **Preprocessing Images for Key Generation:** Images are resized to a standard size in order to have uniform key extraction while maintaining adequate entropy.

- **Feature-Based Key Extraction:** Multi-scale visual signatures are obtained using the hybrid approach of global hashing and local patch analysis, such that the scheme is sensitive to all forms of tampering (e.g., cropping, watermarking, or noise).
- **Iterative Key Derivation and Session-Based Encryption:** The derived key goes through several rounds of transformation in order to maximize entropy, and session-based encryption guarantees that every encryption instance is independent, and key reuse attacks are avoided.

1.3. Key Contributions

By linking keys to image material, the system guarantees that any small change to the original image will trigger devastating divergence in the decrypted message, making it incomprehensible. Also, the incorporation of session-based encryption boosts security through avoiding reuse of keys during successive encryptions, minimizing frequency-based and adaptive attack vulnerabilities even more. Lastly, the non-alphabetic character of image-generated keys sabotages letter frequency habits usually used to launch Vigenère cryptanalysis.

The primary contributions of this work are as follows:

- **First Image-Based Key Injection into the Vigenère Cipher:** To the best of our knowledge, this is the first formal framework for injecting image-based key generation into a traditional cipher. This work illustrates how legacy cryptosystems can utilize visual data to counter user-generated key vulnerabilities.
- **Self-Tampering Key Generation:** Keys are deterministically linked with image features so that a reproducible output is produced under normal circumstances while the key is invalidated whenever adversarial alterations are made. Such a property makes the integrity of the cipher more robust by removing the compromise between key stability and tamper resistance.
- **Session-Based Encryption for Stronger Security:** Traditional Vigenère cipher implementations are vulnerable to attacks when the same key is reused across multiple encryptions, allowing attackers to analyze repeated patterns in the ciphertext and apply frequency analysis to deduce the key. To counter this, our approach introduces session-based encryption, where each encryption session generates a unique variation of the key, even when

using the same image. This ensures that no two encrypted messages share the exact same key, preventing attackers from recognizing patterns or exploiting previously cracked keys. By dynamically altering the key for each session, our method strengthens resistance against brute-force attacks, frequency analysis, and replay attacks, significantly enhancing security while maintaining usability.

- **Stealth-Oriented Key Concealment:** Keys are camouflaged as normal pictures in a user's album, making it considerably harder for an adversary to discover them. Even if the user's storage falls into an attacker's hands, they need first to realize that a picture had been utilized as a cryptographic key and next which image from potentially thousands is the appropriate one (a needle-in-a-haystack situation).
- **Disrupting Frequency-Based Attacks:** Experimental results indicate that image-derived keys add entropy that disturbs the letter frequency patterns usually taken advantage of in standard Vigenère cryptanalysis, which compels attackers to drop traditional attack methods.

This research spans classical and contemporary cryptographic models, illustrating how older ciphers can be reinvigorated using new key generation methods. Although the Vigenère cipher's structural constraints mean it cannot meet contemporary security requirements, our results identify the greater potential for image-based cryptography in situations where computational resources or legacy system compatibility restrict the use of modern encryption protocols.

CHAPTER 2

LITERATURE SURVEY

The use of digital images as cryptographic primitives has gained significant attention in recent years. Zhao et al. [6] proposed one of the early frameworks for using image characteristics to generate encryption keys, though their approach focused primarily on digital watermarking rather than general-purpose encryption. Arambam Neelima and Manglem Singh [7] proposed using image hash functions for authentication purposes, which shares conceptual similarities with our localized patch hashing mechanism.

The PanAAVA algorithm proposed by Ihsan, A., & Doğan, N. [8] enhances encryption security by integrating multiple techniques. It is the first to use the Pan-Tompkins Algorithm (PTA) for key generation and employs Least Significant Bit (LSB) steganography for secure key embedding. It combines Affine (AA) and Vigenère (VA) encryption to strengthen security. Comparative analysis with standard encrypted images shows its effectiveness, with UACI of 33.4044\% and NPCR of 99.7442\%, indicating strong resistance to attacks. The algorithm passes NIST security tests and demonstrates adaptability to quantum communication with stable key rates and low QBER values. These improvements indicate the direction toward multi-layered encryption to overcome cryptographic weakness.

The paper by Dash et al. explores biometric-based cryptography, focusing on iris image data for key generation [9]. The authors introduced an ensemble L0 Gradient Minimization technique that had an edge-preserving filter to extract high-quality iris features. A statistical normalization method is used to ensure key stability, and by selecting hybrid features, it enhances security. The work uses an interval-based encoding scheme for efficient key conversion and extensively tests its efficiency on various iris datasets. It is applicable to data security, authentication, and blockchain systems and reinforces the role of biometric cryptography in modern security frameworks.

Li et al. introduced the SeSMR model, which enhances session-based recommendations and ensures data privacy in edge computing [10]. It incorporates BGV homomorphic encryption to protect user data, that mitigates privacy risks. To address over-smoothing in graph neural networks, a residual

attention mechanism is used to preserve feature independence. The model also uses a soft attention mechanism with location coding to improve recommendation accuracy. Experimental results show a 2-5\% improvement in Recall and MRR, demonstrating SeSMR's effectiveness in secure and accurate multimedia recommendations.

A recent study by Gupta et al. on image encryption for IoT devices introduces a lightweight symmetric encryption algorithm designed for secure and efficient image transmission [11]. It implements a session key mechanism, generating a unique key for each encryption to prevent key reuse attacks. The approach leverages genetic algorithms, using crossover and mutation operators to enhance key randomness and security. The algorithm supports both color and grayscale images, ensuring broad applicability. Performance evaluations in MATLAB 2017 demonstrate superior speed and security compared to existing methods, making it a strong candidate for real-world IoT applications.

CHAPTER 3

TECHNICAL SPECIFICATIONS

3.1. Programming Language and Execution Environment

3.1.1. Language Choice

Python 3 was chosen as the development language for this encryption system because of its combination of readability, speed of development, and a comprehensive set of built-in libraries specifically designed for data handling and security. This project involved extensive experimentation in various areas (image processing, cryptographic key generation, secure encoding, and file management) and Python's high-level abstractions facilitated easier integration of these elements together seamlessly.

Another one of Python's key strengths comes from its complete standard library that limits reliance on external packages. As an example, modules including hashlib, and collections were employed to perform cryptographic operations like hashing image data, encoding keys, and securely getting random values. Being able to implement these operations natively in Python's internal libraries helped in retaining control of the cryptographic process, which is extremely important when dealing with security-centric applications.

Additionally, Python's intuitive and clean syntax makes it possible to implement intricate logic quickly without sacrificing readability. During a project that involved repeated testing of derivation mechanisms and encryption schemes, writing and reading code was critical. It enabled the team to develop the system rapidly, detect bugs early, and maintain modular and clean logic. This is especially useful in cryptographic design, where knowing how the flow of data transformations evolves step by step is crucial for keeping the system in integrity.

Python's dynamic typing and interpreted nature allowed for rapid iteration loops as well. A great many features including patch-based image processing, dynamic key generation, and checking file validity were first created experimentally, Python's interactive nature during execution made it

immensely easier to verify design decisions at runtime.

Python also has support for various programming paradigms such as procedural, functional, and object-oriented programming. This was helpful in organizing the project as it became increasingly complex. Image processing and cryptographic logic, for instance, were maintained using a mix of utility functions and independent modules, whereas security assessments like entropy analysis and frequency analysis employed a more analytical procedural method.

Even though this project was constructed with a command-line interface in mind, Python gives the developer enough flexibility to enhance or port the same logic to interactive platforms such as Jupyter Notebooks, GUI applications, or even web frameworks. Such extensibility allows the system to be scaled for future educational, research, or enterprise deployment.

Lastly, Python has a huge, active community guaranteeing the sustainability of this endeavor in the long term. Ongoing improvements on its fundamental libraries, extensive documentation, and user support mean the system can scale over time with little technical debt.

Overall, Python's ease of use, readability, and flexibility made it the perfect language to create a security-oriented, image-based encryption system. It enabled the team to concentrate on cryptographic correctness, performance optimization, and modularity without being entangled in the intricacies of lower-level programming languages.

3.1.2. Platform Compatibility

The system was deliberately designed as cross-platform, with the aim of producing stable and consistent operation across principal operating systems like Linux and Windows. The design criterion was an underlying platform independence from the outset, given the variety of user environments in educational and private computing environments.

To provide complete cross-platform support, the implementation deliberately refrained from using operating-system-specific system calls, library functions, or environment-dependent setups. All

aspects of functionality ranging from file operations to image processing and cryptography handling were written using Python's standard and widely supported third-party libraries. These include such modules as os, hashlib, base64, and PIL (Python Imaging Library), all of which work satisfactorily on all operating systems without needing to be adapted.

File input and output operations, for instance, were handled using Python's native open() function and path-independent filename references, which hide the underlying file system differences between Linux (forward slashes) and Windows (backslashes). To make the system path-independent, hardcoded file paths were not used, and image files and text documents were accessed using relative paths, allowing for smooth transitions between development, deployment, and testing environments.

For image processing, the availability of the PIL (now Pillow) library provided for uniform image handling regardless of OS-specific differences in rendering. Images are resized and converted to grayscale in a deterministic, uniform way so that the same input image produces the same key derivation on both Linux and Windows.

Equivalently, cryptographical operations with modules like hashlib and cryptography.hazmat are designed to be platform-agnostic. These modules provide uniform implementations of hash algorithms (such as SHA-256 and SHA-512), key derivation functions (such as PBKDF2-HMAC), and encoding mechanisms (such as base64), which are essential for making the resultant cryptographic keys invariant to the execution environment.

While developing, the tool was comprehensively tested on both Linux and Windows platforms to ensure functional equivalence. This platform-independence architecture not only enhances the accessibility and ease of use of the tool but also renders it very deployable in a diverse set of fields. In academia, where the student population may use a combination of Linux and Windows machines, such uniformity is critical towards reproducibility as well as collaboration. Within the enterprise community, cross-platform capability ensures security utilities can be uniformly deployed over different infrastructures. For private use, it gives freedom to the end-user without placing limits on the operating system.

Ultimately, keeping the system platform-agnostic ultimately adds to both its usability and its maintainability, allowing the encryption system to act as a good utility in many different computing environments.

3.1.3. Development Environment

The entire development was performed within Visual Studio Code (VS Code), a lightweight but very extensible integrated development environment (IDE) that provides strong support for Python. Its modular nature and rich extension ecosystem made it ideally suited to the iterative development and testing of cryptographic algorithms.

A selection of important extensions and tools in VS Code played a major role in making the development process efficient and robust. Foremost among them was the use of Pylance to facilitate sophisticated type checking and smart code completion, thus making code correctness and readability better. This static analysis feature proved to be very useful to detect type inconsistencies and logical flaws in the crypto functions, which tend to involve complex data types and accurate mathematical computation.

For debugging purposes, the intrinsic support for the Python Debugger (pdb) enabled step-by-step running and runtime observation, making it easy to clearly understand control flow and variable values during encryption, decryption, and key scheduling operations. This detailed level of observation was crucial for checking the correctness of intermediate computations and edge-case tracing.

Also, VS Code's built-in Git capabilities ensured effortless version control. This allowed easy source code management with features like inline diff preview, navigation through commit history, and branch management within the same space. Integration of this type was critical for ensuring a tidy development history, particularly while repeatedly refining cryptographic algorithms.

The environment also allowed for real-time linting, syntax highlighting, and project navigation, which cumulatively maximized code legibility and minimized syntactic defects. These mechanisms helped to offer a smooth development experience by facilitating the project's ability to scale in

complexity while not compromising readability or organization.

Testing as well as visualizing output were done using native Python scripts that were run from VS Code's built-in terminal. This solution provided full control over input parameters, test vectors, and output formatting. Metrics like entropy scores, timing of execution logs, and key verification checks were calculated and presented in real time, allowing detailed analysis of both theory and practice of the cryptographic algorithms. Self-contained scripts avoided the need for external dependencies and provided results reproducibility across testing environments. In general, the development environment offered by VS Code allowed for a structured, efficient, and highly controllable workflow that was well-suited to the needs of designing and verifying cryptographic functionality in an academic research setting.

3.2. Libraries and Dependencies

3.2.1. Cryptographic and Hashing Libraries

The encryption process for this system is constructed on top of a base layer consisting of well-documented, peer-reviewed cryptographic implementations and algorithms, mostly facilitated by Python's hashlib and cryptography libraries. These libraries are sound and industry-standard mechanisms for safe and deterministic conversions and maintaining data confidentiality and integrity.

At the heart of the system is the Password-Based Key Derivation Function 2 (PBKDF2) algorithm, accessed through the PBKDF2 class from Python's cryptography.hazmat module. PBKDF2 is a highly respected algorithm defined in RFC 2898 and is renowned for its capability to derive cryptographic keys from low-entropy inputs (like passwords or in this instance, image-derived hashes). The algorithm includes a cryptographic salt and a lot of iterations, which are utilized to convert input into a key, essentially thwarting dictionary attacks and making it very expensive in terms of CPU time for brute-force attempts.

Usage of PBKDF2 in the system guarantees that even when the same image is recycled, the resulting key remains unpredictable unless the same salt is recycled, something that the system prevents by deriving the salt for binding to the hash of the image itself. This entropy-ful and dynamic derivation process increases the uniqueness of every encryption case even when near images or data are

employed.

Python's hashlib library is utilized heavily to calculate global and local hashes from image and text information. The SHA-512 algorithm is specifically utilized to calculate high-entropy feature hashes from flattened pixel values by combining the results of global image information and local image patches. This aids in capturing overall and granular features of the image for robust key derivation.

SHA-256 is also utilized to create secondary hashes, e.g., for salting and for generating the dynamic key component that is functionally connected to a session timestamp. These hashes provide the non-reversibility and collision resistance properties that are essential to the secure functioning of any cryptographic protocol. By employing SHA-256 in the formation of the secret key (i.e., base key with dynamic key formed from a timestamp concatenated), the system introduces time variability and session uniqueness, avoiding encryption key reuse and defense against replay attacks.

By combining secure design options and cryptographic building blocks, the system meets several goals of security:

- **Confidentiality:** Text text is concealed using a non-negligible cipher and random session-key based keying.
- **Integrity:** Any attempted compromise of encrypted metadata or data would render decryption unusable, through the consistent, deterministic operation of the hash-derived key.
- **Non-reversibility:** Hashing functions employed are one-way, such that it is computationally infeasible to recreate the original image or text from the key or hash.
- **Replay resistance:** Through associating key generation with an actual real-time timestamp and implementing time-bound validity, the system makes it impossible for previously generated ciphertexts to be reused in unauthorized situations.

Together, these methods form a secure, state-of-the-art method of lightweight cryptographic protection, illustrating how traditional algorithms (such as Vigenère) can be improved and integrated into a sound framework driven by modern cryptographic engines. The correctness and security of the system depend solely on thoroughly understood cryptographic foundations, and it is thus well-suited not just for experimental or educational use but also as a production prototype for encryption

workflows.

3.2.2. Image Data Processing

A central component of this system is generating cryptographic keys from images, and for this to be effective, image data must be transformed into a standardized numerical format that's well-suited to hashing and statistical analysis. To perform this process, the system makes use of a tightly coupled pipeline with two main libraries: Pillow (PIL) and NumPy.

The Pillow library is utilized for preliminary image preprocessing. It accommodates a broad variety of image formats such as PNG, JPEG, BMP, and TIFF, with support for flexible user input. When an image is loaded, it is initially resized, so dimension differences between various images do not bring about inconsistencies or unwanted entropy in key generation.

This representation lowers the data dimensionality without losing important structural information like texture, contrast, and spatial distribution. The resizing operation is important to normalize the input size, which means having the same vector lengths in different images. It makes downstream processing easier and ensures that the image can be mapped to a deterministic feature vector, which is essential to produce reproducible keys. Resizing also helps normalize the entropy space, aligning the key derivation process irrespective of original image resolution.

After preprocessing is finished, the image is converted to a matrix of pixel intensity values and passed over to NumPy for numerical computation. NumPy has a great handle on dealing with large data arrays and vectorized operations that considerably improve performance over regular Python loops. The pixel matrix is collapsed into a single one-dimensional vector, which now serves as the raw data input for additional cryptographic processes.

Besides flattening, NumPy is employed for patch-based operations, where smaller local pieces (e.g., 8×8 windows) are cropped from the image in order to create local entropy vectors. These are individually hashed and blended with global statistics to produce a stronger, noise-immune feature hash. This approach evades threats caused by image compression artifacts or localized distortions and enhances the reliability of key derivation even if the image is minimally edited.

The interdependence of Pillow and NumPy is key to the robustness and adaptability of the encryption system. Pillow maintains format consistency and aesthetically pleasing uniformity, while NumPy ensures computationally efficient and high-precision data handling. Together, they supply the preprocessing skeleton that directly influences the quality and entropy of the encryption key.

Effectively, the system's cryptographic security is limited by the image's structure, variability, and resolution, all of which are well-normalized and encoded during this preprocessing phase. By depending on this established image processing stack, the system achieves high reproducibility and cross-platform stability, rendering it a trustworthy instrument within environments needing both visual data management as well as cryptographic due diligence.

3.2.3. Entropy Visualization and Analysis

In order to analyze and confirm the randomness of the encrypted data, the system used matplotlib, which is a widely used Python package for data visualization. Graphical analysis was significant in determining the efficacy of the encryption process. In particular, the system created frequency histograms of ASCII character distributions prior to and after encryption. These histograms gave an instantaneous and intuitive visual sense of how the encryption mechanism impacted the underlying data structure. A spread-out and even distribution of character frequencies in the ciphertext, as opposed to the more organized distribution of the plaintext, was an early sign of successful obfuscation.

Aside from visual inspection, the project utilized a custom Shannon entropy computation on a byte basis. Shannon entropy, a common statistical figure of information unpredictability, was utilized to measure the level of randomness injected by the encryption. The special-purpose function examined the occurrence frequency for every byte value in the data and calculated the entropy based on it. Greater entropy in encrypted outputs, converging towards the theoretical maximum for uniformly random data, are universally accepted as a significant metric to assess the strength and resilience of cryptographic algorithms.

The integration of histogram display and entropy computation offered a two-pronged verification

method: graphical observations of distribution patterns, and quantitative assessment of unpredictability. In addition to reinforcing internal verification throughout development, these tools were also critical when testing or presenting the system's output in academia and the workplace. By providing both visual and statistical proof of encryption quality, the project was able to provide a thorough and stringent evaluation of cryptographic performance.

3.3. Cryptographic Techniques Used

This project blends multiple cryptographic building blocks to update and harden a historic cipher without sacrificing its instructional benefits and ease of use. The methods are implemented in layered stages, beginning with secure key derivation to ultimate ciphertext creation and metadata obscuration.

The basis of the encryption is that of deriving a high-entropy base key, which is derived from images provided by the user. The primary derivation procedure adheres to the PBKDF2 (Password-Based Key Derivation Function 2) protocol, which is commonly used in cryptographic applications to produce secure symmetric keys. The implementation, in particular, employs PBKDF2 along with HMAC-SHA256 as the incorporated pseudorandom function. For enhanced resistance against brute-force and rainbow table-based attacks, the algorithm applies a salt generated through a SHA-256 hash of the image. It also executes 100,000 hashes to elongate the entropy over time. The key that is produced is 256 bits (32 bytes) and is used as the root cryptographic material for all other operations.

To combat one of the fundamental weaknesses of original Vigenère ciphers, i.e. static keys, the system presents a dynamic session key. Every encryption session has a distinct timestamp (in UNIX format), which is appended to the derived base key and then fed into a SHA-256 hash function to generate the session-dependent dynamic key. This guarantees that even if the base key is the same (from the same image input), the encryption output will be different for every session, making ciphertext reuse detection and reverse engineering much harder. The last encryption key is a concatenation of the base key and the session-derived dynamic key, doubling the key material and adding session-level uniqueness.

The encryption scheme itself is a modernized form of the Vigenère cipher. Classic Vigenère ciphers are restricted to alphabet characters and are susceptible to frequency analysis. Conversely, this version increases the character set to cover all printable ASCII characters (range 32-126) so that arbitrary text content, including symbols, punctuation, and numeric data, can be encrypted. The extended Vigenère cipher computes ASCII shifts from the dynamic secret word created using the last key. Every plaintext character is converted using modulo arithmetic to keep it within the printable ASCII range, ensuring readability of ciphertext and significantly expanding cipher space.

In order to avoid metadata tampering or simple inspection of session validity and timestamps, a light-weight symmetric encryption process is applied via XOR-based obfuscation. While being simple in nature, XOR encryption provides adequate protection when applied with a long enough and non-repeating key, in this instance, the base key in base64. This encrypted metadata, added to the ciphertext file, prevents decryption routines from obtaining required session information without compromising its integrity or revealing session-specific data.

This stacked application of cryptographic methods with robust key derivation, session binding that is dynamic, large classical ciphering, and low-cost metadata encryption results in the system overall being robust to a range of attack vectors but simple and educational in nature.

3.4. Security Features

The security framework of the system is established upon both cryptographic strength and practical requirements designed to confine an attacker's opportunity. The structure incorporates a sensitive deliberation over key sensitivity, amplification of entropy, session expiration, and reduction of attack surfaces.

A key aspect of innovation within this system lies in the derivation of symmetric crypto keys from user-uploaded images. In contrast to conventional password schemes based on user-typed text, this approach uses pixel-level data and patch-based hashing to create a high-entropy base key. The image is initially normalized and resized to ensure consistency.

Global and local features are then extracted: the whole image is used to provide a SHA-512-based global hash, and 8×8 patches are hashed separately by SHA-256 and concatenated to achieve further entropy. The combined hash serves as the seed for the PBKDF2 process. Since this hash varies radically even with minor changes to the input image, the key space obtained is in effect non-deterministic unless the same exact image is used which is part of the deliberate design to prevent key reuse and impersonation threats.

Appreciating the fact that static encryption schemes are open to attack after a while, particularly in multi-user environments, this system includes a session validity period as a value in the encrypted file's metadata. The encryption process stores the epoch time at the moment of encryption and an arbitrary validity duration set by the user (in seconds).

On decryption, this time value is read from the decrypted metadata and compared with the current epoch time. If the validity period has expired, the system declines to decrypt the file. This method guarantees time-based access control, ensuring messages are read only within the defined time period. It successfully simulates forward secrecy principles, with each session being temporary and time-bound.

For the empirical evaluation of the strength of encryption, the system provides an in-built tool for carrying out frequency analysis and entropy calculation. Frequency analysis, normally a weakness of classical ciphers, is here overcome by using ASCII-range shifting and dynamically created secret words. The frequency distribution of the ciphertext is plotted as bar plots and compared to the plaintext.

The ciphertext should, in ideal circumstances, have a more even distribution, showing diffusion. The Shannon entropy is also computed for both plaintext and ciphertext so that the randomness of the output can be numerically estimated. A greater entropy value in the ciphertext, nearing the theoretical maximum for the ASCII space, indicates strong resistance against statistical attacks and verifies the success of the session-specific keying approach. Together, these attributes form a system that, though theoretically straightforward, is highly resilient in practice. It avoids dependence on external authentication mechanisms, resists prevalent attack vectors on traditional ciphers, and enhances its

integrity through real-time session constraints without sacrificing usability or educational simplicity.

CHAPTER 4

ARCHITECTURE

4.1. Overview

In this section, we introduce a new encryption system that builds on top of the Vigenère cipher and incorporates cryptographic keys derived from images. Conventional implementations use text-based keys created by the users, and these are compromised in terms of frequency analysis and brute-force decryption because they contain discernable patterns [2]. The scheme offered hereover overcomes such pitfalls by drawing high-entropy keys from computer images, based on their intricate feature distribution and randomness, to enhance security.

Passwords created by human users tend to lack entropy and linguistic unpredictability and, hence, are prone to cryptanalysis. The given technique overcomes such restrictions through the use of digital images as non-linguistic sources of keys with improved randomness and unpredictability. The model is based on various stages: image preprocessing, key extraction, encryption, decryption, and session handling. Image preprocessing normalizes sizes for compatibility among varied hardware and software platforms. Key extraction utilizes a localized hash function to compute cryptographic material that is deterministic and sensitive to changes in the image. Encryption provides a Vigenère transformation on a byte-by-byte basis that can handle all plain text characters, including symbols and digits. Through the use of images for key generation, this system brings an easy and user-friendly encryption technique, which enables secure communication even for those with little technical knowledge.

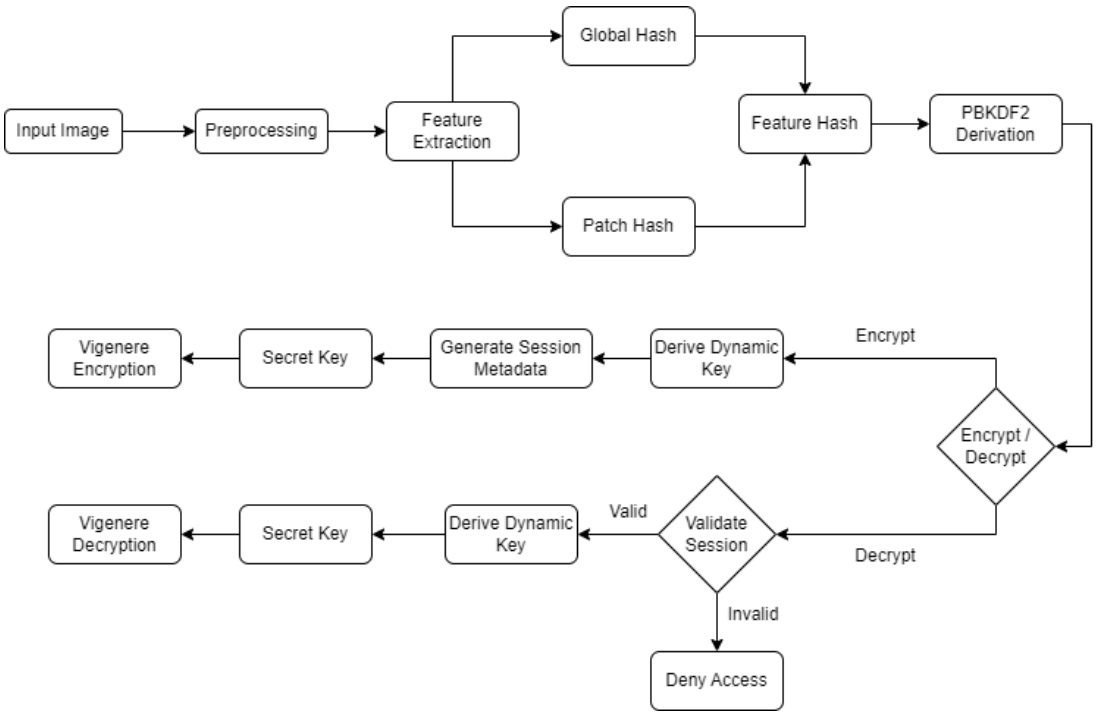

Fig 4.1: Architecture diagram of the encryption system

The above architecture diagram shows our safe, image-based encryption and authentication system. The operation starts with an input image, which is the major authentication factor. The image is preprocessed, a process that includes normalization, noise reduction, and resizing for standardization of input for uniform analysis. After preprocessing is done, the image is sent to the feature extraction module, where important visual features like edges, textures, or distinctive patterns are detected. These features are essential in creating a strong representation of the image that is utilized throughout the system.

After feature extraction, the system splits the process into two parallel streams: the creation of the "global hash" and the "patch hash". The global hash is an overall representation of the features of the image, capturing its essence, whereas the patch hash addresses localized areas of the image in order to yield higher granularity and increase uniqueness. A feature hash is also calculated, which summarizes important attributes of the image features. This feature hash is inputted into the PBKDF2 (Password-Based Key Derivation Function 2) module, which is a cryptographic system that fortifies the hash through the incorporation of computational work factors, making the brute-force mechanism harder to succeed. The end result is a derived key used for encryption tasks.

The derived key is implemented in two paths: encryption and decryption. For the encryption process, the derived key is merged with a dynamic key, which is obtained after generating the session metadata, to generate the final secret key. This secret key is session-specific and prevents reuse of encryption keys within different sessions, thus improving security. The secret key is further utilized in the encryption process for encrypting sensitive information or communication.

On the decryption branch, the reverse of the process is followed. The encrypted data is decrypted by obtaining the secret key in the same way, by decrypting the session metadata and generating the dynamic key component. The session needs to be validated before decryption can continue. This security check proves critical to verify that only valid sessions are allowed to access encrypted data. In case the session is considered valid, decryption is facilitated and the data made available. Otherwise, the system denies access completely, preventing unauthorized users from accessing or manipulating sensitive information.

Overall, this design uses image-based authentication in combination with cryptographic hashing, key derivation, and traditional encryption techniques to develop a secure, session-aware data protection system. The layered architecture improves security as well as resilience, and thus it is applicable to applications where strict access control and data confidentiality are needed.

4.2. Image-Based Key Generation

4.2.1. Image Preprocessing

To ensure consistency in key derivation, all images undergo a preprocessing phase where they are resized to a fixed dimension (1024 x 1024) before key extraction. This standardization is important for maintaining deterministic key generation, as variations in image size or format could lead to inconsistent cryptographic outputs.

The system has minimal constraints on image formats and sizes. Standard image formats such as JPEG, and PNG are supported, and the resizing process tries to maintain an optimal balance between computational efficiency and randomness preservation. Although reducing image resolution may lead

to minor information loss, the hashing mechanism ensures that the essential entropy required for key derivation is retained.

4.2.2. Key Extraction Process

After preprocessing, the system generates the cryptographic key by using a multi-stage hashing and key expansion algorithm on the pixel information of the image. This process ensures that the resulting key has high entropy, non-repeatability, and resistance against frequency-based cryptanalysis. The key derivation process involves three main steps: image segmentation, patch hashing, and key expansion, each of which adds to the strength and unpredictability of the resulting cryptographic material.

The system is initiated with image segmentation that divides the resized image into several non-overlapping patches of a specified size (8 × 8 pixels). Instead of applying a global hash to the entire image, in this scheme the key extraction process is localized such that the cryptographic key depends on variation across small regions of pixels as opposed to constant image-wide attributes.

The application of visual entropy is a central element in the production of high-entropy keys. Natural images possess inherently varied pixel intensities and structural complexity, thus making them perfect sources for cryptographic key derivation. In contrast to conventional user-created passwords that display linguistic predictability, image-based keys introduce statistical randomness, making it very difficult for an adversary to recreate the key through brute-force or frequency analysis. This segmentation technique using localized patches also greatly enhances sensitivity to small image changes, in that even minor changes like cropping, small color adjustments, or watermarking result in a completely different key.

The patch hashing stage then applies the SHA-256 cryptographic hash function to every single patch. By processing each patch as an independent source of entropy, the system ensures that the resulting key is based on a rich and highly random mix of local pixel arrangements. Each hashed patch adds to a collection of entropy-dense hash segments, which are concatenated to produce an intermediate hash. A global SHA-512 hash of the whole image is also calculated to add strength to the randomness of the derived key, so that both local and global image characteristics add to the cryptographic

material. In order to further enhance security, the system computes a salt from the SHA-256 hash of the global image hash and adds it to the combined patch hashes prior to final processing. The salt makes precomputed attacks impossible even if an attacker has access to the original image since the derived key will be independently affected by the salt.

The system then uses the PBKDF2 algorithm with SHA-256 and performs 100,000 iterations on the obtained hash to produce a cryptographically secure, fixed-size key [12]. The iterative operation performs several functions: it makes computational resistance against brute-force attacks harder, prevents repetitiveness, and eliminates any correlations between similar images. Also to avoid key reuse for multiple encryption sessions, the system includes a dynamic key derivation mechanism on a session basis which will be explained in the upcoming sections.

4.2.3. Secret Word Extraction

To improve usability while maintaining security, the system integrates a dynamic key derivation and secret word extraction mechanism. The dynamic key is generated by using a UNIX timestamp into the base cryptographic key, ensuring that each encryption session has a unique, time-based variation. This dynamic key adds an additional layer of complexity and security, preventing replay attacks and ensuring that the same input does not result in the same encryption output.

The dynamic key is generated by combining the base key with the timestamp at the time of encryption. The timestamp is first converted into a byte string, and then the SHA-256 cryptographic hash function is applied to the concatenated base key and timestamp. This process generates a unique, session-specific key.

Once the dynamic key is generated, the system concatenates it with the base key and derives a human-readable secret word from this key. The secret word is extracted by mapping each byte of the final key to a corresponding uppercase English letter using modular arithmetic. The transformation ensures that each byte from the final key is converted into a letter, forming a readable string.

This transformation is performed using modular arithmetic:

$$Si = (Ki \, mod \, 26) + 65 \quad (1)$$

where *Ki* is a byte from the final key, and 65 is the ASCII value of 'A'. This ensures that each byte is mapped to a letter from 'A' to 'Z', generating a readable sequence of characters. By selecting the first N bytes of the key (where N is the predefined secret word length, 32 in this case), the function produces a consistent yet unpredictable alphanumeric sequence.

The secret key developed is then used as the basis for encrypting files with the Vigenère cipher. With the key originating from the picture and supplemented by a timestamp, every encryption session is unique as well as secure. The recovered secret word supplies a human-friendly representation of the key, being more usable for the cipher but not less secure. This hidden key is then used inside the Vigenère cipher to encrypt and decrypt files in order to provide confidentiality and integrity to the data.

4.3. Modified Vigenère Cipher

The proposed encryption system extends the classical Vigenère cipher by replacing user-defined textual keys with cryptographically strong image-derived keys. In its traditional form, the Vigenère cipher operates as a polyalphabetic substitution, using a cyclically repeating key to encrypt plain text characters.

However, this approach is susceptible to frequency analysis attacks when the key exhibits predictable linguistic patterns. By incorporating high-entropy keys extracted from images, the modified version significantly enhances security by eliminating these vulnerabilities.

Unlike the standard Vigenère cipher, which only operates on alphabetic characters, the modified implementation extends encryption to all possible byte values (0-255) to support arbitrary data encryption, including non-textual content. Instead of mapping plain text characters to a fixed alphabet, each byte of the plain text is encrypted using modular addition with the corresponding byte

of the image-derived key:

$$C_i = (P_i + K_i) \, mod \, 256 \quad (2)$$

where C_i is the encrypted byte, P_i is the plain text byte, and K_i is the corresponding byte of the derived key. The key is expanded as necessary using the iterative derivation process, ensuring that no predictable repetition occurs. This modification disrupts the letter frequency patterns typically leveraged in cryptanalysis, as the statistical distribution of cipher text characters is directly influenced by the entropy of the image-derived key rather than linguistic biases.

Moreover, due to the localized hashing mechanism in key generation, even minor alterations in the input image produce dramatically different encryption keys. This ensures that any brute-force attack attempting to reconstruct the key from known images becomes infeasible, as an attacker would need not only the correct image but also an exact replica at the pixel level.

4.4. Session Management

The encryption system implements a session-based approach to ensure both security and controlled access to encrypted files. This is achieved through a combination of dynamic key derivation, metadata encryption, and time-based access control mechanisms.

At the encryption stage, a UNIX timestamp is generated that serves as a unique session identifier. This timestamp is then incorporated into the key derivation process to ensure that each encryption session produces a unique cryptographic key. The base key generated from image-derived features is combined with the timestamp to create a dynamic key component using the SHA-256 hashing algorithm:

$$k_{dynamic} = SHA - 256(k_{base} \,||\, T) \quad (3)$$

where k_{base} is the primary key extracted from the image, T is the session timestamp, $k_{dynamic}$ is the

dynamic key derived from the encryption timestamp, and || represents concatenation. This approach ensures that even if an identical image is used in multiple encryption attempts, different keys are produced, preventing key reuse attacks.

The final encryption key is then obtained by concatenating the base key with the dynamic key:

$$k_{final} = k_{base} \,||\, k_{dynamic} \tag{4}$$

From this combined key, a readable secret word is extracted, which serves as the session-specific encryption key for the Vigenère cipher. Since the dynamic key varies based on the timestamp, each encryption session generates a unique secret word while maintaining deterministic key reconstruction for authorized decryption.

Once the plain text is encrypted using the secret word, the system embeds metadata containing the session timestamp and a validity period (e.g., 3600 seconds) to enforce access restrictions. The metadata is structured as follows:

$$metadata = T \,||\, V \tag{5}$$

where T is the encryption unix timestamp, and V is the validity duration.

This metadata is encrypted using an XOR-based transformation with the base key, ensuring that it remains protected from unauthorized modifications. The transformation is defined as follows:

$$E(metadata) = metadata \oplus K \tag{6}$$

where $E(metadata)$ is the encrypted metadata, and K is the base64-encoded base key. Using the XOR operation, the metadata is encrypted securely so that only a person with the appropriate base

key can decrypt it. The metadata is then written to the file after encryption, and then the encrypted content is written. The file now has both the cipher text and metadata securely bound together so that tampering and unauthorized access are not possible. The file is now prepared for storage or transmission.

In the decryption process, the system initially extracts and decrypts the metadata. The encrypted file is loaded and the metadata is extracted and decrypted by applying the same XOR operation with the base key, which is extracted from the image that is used to encrypt the file. From the decrypted metadata, timestamp and validity period are extracted. These are utilized to verify the file's validity.

If the present time is greater than the validity period from the timestamp, the file is deemed expired, and access is denied. If the file is valid, the system re-generates the dynamic key by applying the timestamp to the base key so that the same encryption key is used for decryption. The final decryption key is reconstructed, and the secret word is obtained. With this secret word, the system then decrypts the cipher text with the Vigenère cipher, reversing the encryption and returning the original plain text. The last step is to write the decrypted plain text to a file, finishing the decryption session.

This technique ensures that the file is secured and only accessible within the time specified, which provides an added layer of security against unauthorized access. Therefore, this approach creates time-sensitive encryption and follows strong cryptographic practices to provide confidentiality and integrity of the file throughout its life cycle.

CHAPTER 5

MODULES

5.1. Image Preprocessing Module

The image preprocessing module has a central part to play in the consistency and robustness of the encryption process. It takes on the duty of converting indiscriminate images to a structured, standardized format in which they may be reliably applied to cryptographic use. The function "preprocess_image" receives a file reference to the target image and performs well-crafted transformations to assure that the resulting image is to be a sound foundation for future calculations.

The initial transformation that is carried out is resizing. Images are all resized into a uniform size of 1024×1024 pixels, irrespective of their original size or aspect ratio. This process is important because cryptographic systems are very sensitive to input variation. A slight discrepancy in image size, such as a one-pixel difference, would yield a totally different derived key, and hence decryption would fail. Resizing guarantees consistency between various users and systems, rendering the encryption and decryption operations predictable, repeatable, and deterministic.

After resizing, the image is loaded into memory and translated into a NumPy array. This process essentially converts the image into a structured numerical matrix that stores pixel intensity values. NumPy is used for this reason because it offers very fast, memory-efficient operations on large data structures such as images, which contain millions of pixels. The fact that the image can be treated as a direct numerical object allows subsequent transformations like flattening, normalization, hashing, and feature extraction to be very efficient.

While the initial design did plan to incorporate grayscale conversion, the process still maintains visual standardization through resizing, which greatly diminishes input variability. If grayscale had been incorporated, it would have further minimized the three-channel (RGB) information into a single luminance channel, reducing the noise introduced by colors and prioritizing structure over color variation.

Notably, the module does not include operating system-specific dependencies. Utilizing the Pillow (PIL) and NumPy libraries, both of which are stable, popular, and cross-platform, the image preprocessing phase acts the same regardless of whether the system is Linux, Windows, macOS. This choice helps make the encryption and decryption phases platform-independent and available for mass distribution without adjustment. Yet another significant benefit is resistance to errors. By preprocessing all images in a uniform manner prior to any hashing or cryptographic function, the module significantly eliminates the possibility of sender-receiver key mismatches. This ensures that if two users are using the same image file, their resultant cryptographic keys will also match, provided they both go through this very same preprocessing pipeline.

This module is the foundation of the input normalization task of the project. It allows for cryptographic safety, cross-platform usability, speed performance, and user dependability, while yet having the openness to process input images of immense diversity in various formats, dimensions, and colour depths. The thoughtful, modest architecture of this module facilitates its broader purpose in securing the entirety of the encryption-decryption system as consistent, reproducible, and trustworthy.

5.2. Feature Extraction Module

It is the task of this module to create a highly resilient, tamper-evident cryptographic base key from a given input image. The module takes the preprocessed image array as input and extracts from it a feature hash that captures both the global shape and the fine-grained local textures of the image. The construction of this feature means that the extracted cryptographic key is immune to any image alterations, cropping, or compression artifacts.

Normalization of pixel values comes first. By dividing the pixel values by 255.0, the function normalizes all values into the [0, 1] range. This provides scale invariance and numeric stability between various image formats, color depths, and software environments. Using normalized floating-point values minimizes the chance for cryptographic divergence due to small encoding variations.

A global hash is then calculated. The whole flattened pixel array is serialized into bytes and fed through the SHA-512 hashing algorithm, generating a secure, 512-bit digest. This global hash encodes the overall visual organization of the image. Any drastic change to the entire image, e.g., large-scale distortion, color change, or extensive cropping would yield an entirely different global hash because of the avalanche effect of cryptographic hash functions. This imposes tamper detection at the macro level.

A salt is then obtained from this global hash using SHA-256. A salt is a randomizing factor employed within cryptographic systems to avoid precomputed attacks such as rainbow table attacks. Here, having the first 16 bytes of the SHA-256 digest guarantees a fixed length, high-quality entropy source that shall be utilized later during key derivation procedures. Besides the global representation, local features are also extracted to preserve fine-grained structure and texture. The image is partitioned into non-overlapping 8×8 pixel patches. Each patch is hashed separately with SHA-256, and a collection of resulting patch hashes is gathered. By extracting features at a localized, smaller scale, the system achieves sensitivity to small image differences, such as noise textures, subtle patterns, or edges. This renders the global feature descriptor more detailed and able to differentiate images that are globally similar but locally dissimilar.

Each of the individual patch hashes is appended into a large single byte stream. This aggregated stream, in combination with the global hash and salt, becomes the input to a second round of SHA-512 hashing. This second-level hashing process combines global features, local features, and randomness into one strong feature hash. This final digest is used as the cryptographic anchor for all subsequent calculations such as key generation and encryption. This module's multi-layered structure, with global view, localized texture, and salted randomness all mixed together, makes the system very robust to adversarial attack. It makes it such that only the right, non-altered image can replicate the original cryptographic key, imposing image-bound identity upon the encrypted information. Through the effective application of NumPy for pixel management and hashlib for safe hashing, this module ensures optimal performance, robust entropy, and superb cross-platform reproducibility that are all critical for the assurance of the overall system's reliability and security.

5.3. Base Key Derivation Module

This module constitutes the cryptographic kernel of the system by safely transforming the image feature extracted into a high-entropy encryption key. This process is essential since it guarantees that even if the two images look visually identical, the resultant keys will be vastly different unless the pixel-level structures are identical.

This module uses the PBKDF2 (Password-Based Key Derivation Function 2) algorithm via Python's cryptographic library interface. PBKDF2 is a tried-and-tested, standardized key derivation method, recommended by many security standards, including NIST and PKCS#5. It makes the relationship between the input and the output base key more secure by performing multiple iterations of hashing, thereby offering computational resistance to brute-force attacks.

The module is well parameterized:

- **Algorithm:** SHA-256 is selected as the underlying pseudorandom function used in PBKDF2. SHA-256 is commonly thought of as being secure and computationally efficient as a hash function with excellent collision resistance. With SHA-256, the resulting keys retain an extremely high level of cryptographic unpredictability.
- **Length:** The key derived is fixed at 32 bytes (256 bits), which is the standard key size for robust symmetric encryption systems such as AES-256. A 256-bit key size guarantees that the system remains future-proof against even quantum-level computational attacks.
- **Salt:** A 16-byte salt generated during feature extraction is used here. The salt ensures that identical feature hashes across different sessions or users will yield completely different derived keys, thus neutralizing any threat from precomputed dictionary or rainbow table attacks.
- **Iterations:** The algorithm is set with 100,000 iterations, a well-selected number weighing security versus performance. Large iteration numbers make brute-force attempts slow down immensely because each guess takes thousands of hash computations. Meanwhile, performance is still reasonable for legitimate users during encryption and decryption.

Through the application of PBKDF2-HMAC with SHA-256 and 100,000 iterations, the module implements a computational hardness that serves as an effective defense against password-guessing attacks and key-extraction attempts. The implementation by the module depends on the use of the package's PBKDF2-HMAC class with C-optimized routines and secure default backend to ensure consistent cross-platform behavior along with optimized performance. Ultimately, this module guarantees that the system meets:

1. High unpredictability of keys
2. Immunity to precomputed attacks
3. Consistency across platforms and runs
4. High conformance to contemporary cryptographic best practices

In the broader system architecture, this module acts as the bridge between the image-driven feature extraction layer and the encryption-decryption core. It is crucial because it transforms a possibly high-dimensional, variable-length input (the feature hash) into a fixed-length, standardized, high-entropy symmetric key that drives all further cryptographic operations.

5.4. Dynamic Key Derivation Module

The dynamic key derivation module is responsible with inserting a time-varying component into the encryption key so that even when using the same image and the same base key, the resulting encryption key varies between sessions. This dynamic action strengthens the system's immunity to replay attacks and session hijacking and renders it extremely difficult for attackers to reuse existing encryption material.

This module operates by merging the static base key with a timestamp to produce a session-specific dynamic key. The following are the steps carried out:

- The timestamp (being the encryption time) is translated to byte representation. This way, even slight variations in the encryption time result in varying key material.
- The base key (obtained securely from image features) is appended with the timestamp bytes.

- The combined data is then run through the SHA-256 hashing function, resulting in a 32-byte digest that is used as the dynamic key.

The choice of SHA-256 as a design decision here is important. Hashing rather than direct concatenation or simple mathematical combination is employed to:

- Uniformly distribute the entropy throughout the keyspace and eliminate any predictable pattern that might result from the structure of timestamps.
- Make it non-reversible, i.e., an attacker observing the end key cannot simply go back to obtaining the initial base key or timestamp.
- Generate an output of fixed length, which is crucial for subsequent cryptographic operations that require keys with precise lengths (such as Vigenère cipher computation or XOR encryptions).

The advantages this module contributes to the system as a whole are extensive:

- **Time-sensitivity:** Each encryption action generates a key specific to that time, greatly restricting the potential for key reuse or ciphertext correlation.
- **Stateless dynamic behavior:** Dynamic keys do not have to be stored independently, limiting storage threats. Because the key can always be recalculated as long as there are the proper inputs, the system is not only lightweight but also secure.
- **Cryptographic soundness:** Utilizing SHA-256 guarantees the dynamic key full diffusion and randomness, which are crucial for good encryption.

In the overall architecture, this module serves as a session-based hardening layer on top of the secure base key. It ensures that encryption keys are dynamic and unpredictable between sessions, making the system more temporally attack-resilient and significantly improving the overall cryptographic hygiene of the platform. Therefore, this module is a minimal but important improvement which changes the system from static key encryption to a more secure, dynamic encryption paradigm appropriate for contemporary threat models.

5.5. Secret Word Extraction Module

The module serves as the pivotal part of the encryption process to produce a viable key for use with the Vigenère cipher. It transforms the binary key derivation function's high-entropy output to a structured string consisting only of uppercase alphabetic characters (A-Z). The derived binary key's each byte is reduced mod 26 and encoded into the ASCII range for uppercase letters so that the secret word is reproducible, deterministic, and fits within the cipher's operating constraints.

This conversion is important since it provides for smooth integration between the secure key derivation pipeline (PBKDF2 and SHA-256-based) and the traditional character-based Vigenère cipher without compromising cryptographic quality. By converting the randomness of the binary key to a printable form, the module maintains the required entropy to protect against pattern recognition and brute-force attacks in the printable ASCII range. In addition, this method makes the practical handling of keys in the system easier. Because the secret word is deterministic and based on image-based features and temporal data (timestamp), it enables encryption and decryption to be done independently and uniformly across systems without having to store or pass keys manually. This enhances security by reducing sensitive material exposure and facilitates a clean, user-friendly workflow.

In short, the module plays the essential role of a link between contemporary key generation in cryptography and traditional encryption schemes. It makes the system secure while keeping it easy to operate, allowing for powerful encryption while it remains compatible with the adapted Vigenère cipher employed in the project.

5.6. Vigenère Cipher Module

The Vigenère cipher is an old polyalphabetic substitution cipher that was created in the sixteenth century and was still one of the most used encryption methods for hundreds of years. Unlike basic monoalphabetic ciphers like the Caesar cipher, which shift each plaintext letter by a fixed value, the Vigenère cipher applies a key word to decide the shift value dynamically per character. This adaptive movement greatly reinforces security, leaving frequency analysis attacks much harder to accomplish relative to more straightforward ciphers.

By its very nature, the Vigenère cipher works by overwriting a recurring key word upon the plaintext message. Every plaintext letter is coded by advancing the letter in the alphabet by as many positions as the value of the key's letter at an equivalent position. Particularly, if the key character is 'B', it is a shift of 1 position, whereas a key character of 'D' would be a shift of 3 positions, and so forth. The alphabet is handled cyclically, so that after 'Z', it loops around to 'A'.

Plain Text	**P**	**A**	**S**	**S**	**W**	**O**	**R**	**D**
Key	**K**	**E**	**Y**	**K**	**E**	**Y**	**K**	**E**
Cipher Text	**Z**	**E**	**Q**	**C**	**A**	**M**	**B**	**H**

Fig 5.1: Vigenère cipher representation

Since the key repeats itself if it is shorter than the message, the encryption pattern depends on both the key length and also the particular letters in the key. A longer and more random key gives higher security since it reduces repetition that would otherwise be used for cryptanalysis.

In spite of being resistant to frequency analysis for a while, the Vigenère cipher is not computationally unbreakable. With the introduction of contemporary cryptographic methods and statistical tests, e.g., the Kasiski test and Friedman test, it is now possible to estimate the key length and breach the cipher provided sufficient ciphertext. However, within the domain of controlled settings, lightweight encryption, and teaching tools, the Vigenère cipher continues to be an open study and practice of polyalphabetic substitution.

In the context of this project, the Vigenère cipher is adapted. Rather than confining operations to alphabetic characters (A–Z), the cipher is expanded to cover all printable ASCII characters. This enables the encryption of not only letters but also numbers, punctuation, and special characters, hence expanding the cipher's usefulness for arbitrary text data. The key for this general Vigenère cipher is extracted from image characteristics and temporal metadata so that the system is not only usable but also cryptographically secure. The simplicity, readability, and flexibility of the Vigenère

cipher make it well adapted to inclusion in hybrid systems such as that created in this project. Its character-based nature is easily compatible with secret keys created as readable words, facilitating secure and beautiful text file encryption. The two main functions, “vigenere_encrypt” and “vigenere_decrypt”, each deal with the tasks of encryption and decryption by taking advantage of the secret word.

During encryption, for every character of the plaintext, the character from the secret word corresponding to it based on position is used with the help of modular arithmetic. The ASCII value of the plaintext character is normalized by first subtracting 32, reducing it to a zero-based index in the printable ASCII set. A shift is then computed as the ASCII value of the secret word character modulo 95, which keeps the shift within the 95 printable characters between ASCII 32 (space) and ASCII 126 (tilde). Once the shift is applied, the output is brought back into the printable ASCII range by adding 32. This ensures that every encrypted character is a valid printable symbol, so encrypted files can be safely stored, sent, and displayed without corruption or misinterpretation. Fig 5.2. showcases how the text transforms.

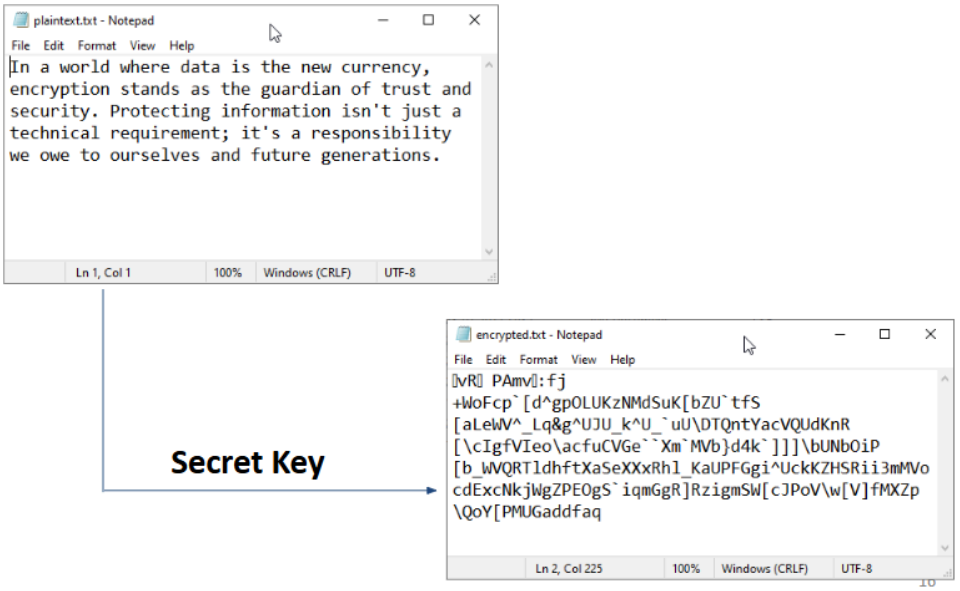

Fig 5.2: Encryption process

The decryption function is an inverse of the encryption logic and uses a negative shift to invert the transformation and return the plaintext to its original form. The cyclic behavior of ASCII printable

characters is handled cautiously by wrapping the range if necessary. Encryption and decryption are performed with strict one-to-one correspondence between input and output characters for lossless and deterministic operation, which is crucial for data integrity in a security system.

Applying the Vigenère cipher in this modified manner is highly advantageous for the project. First, it allows for secure encryption of random text files that can consist of a combination of letters, numbers, punctuation, and symbols, as opposed to alphabetic-only strings only. Second, since the secret word itself is dynamically generated from distinctive image characteristics and timestamps, each encryption session employs a functionally unique key, avoiding key repetition exploitation like that of the classical Vigenère weaknesses. This design minimizes the system's vulnerability to pattern-based cryptanalysis attacks that broke earlier simpler Vigenère implementations.

Additionally, simplicity of this module and the fact that it only uses simple arithmetic and intrinsic Python functions have made the system portable and easy to maintain. There are no external cryptographic library dependencies for this component, which is in line with the objective of the project to reduce third-party tools where robust native options are available. This also benefits knowledge comprehension and auditing of the encryption process for study, research, or instructional purposes. By extending the traditional Vigenère cipher to a complete ASCII range and combining it with dynamic, image-based keying, this module updates an ancient encryption scheme to the status of a contemporary, adaptable part of a greater hybrid security system.

5.7. Security Validation and Statistical Analysis Module

This module is tasked with assessing the strength, randomness, and validity of the encryption process using both qualitative and quantitative measures. This module integrates classical cryptographic analysis methods like frequency analysis, Shannon entropy calculation, and character-wise similarity scoring between plaintext and encrypted results.

The frequency analysis function takes a provided text string and extracts all characters in the printable ASCII range and counts their frequency using Python's counter module. Frequency analysis is an ancient method in cryptography, whereby attackers analyze the letter distribution in the ciphertext to deduce the underlying plaintext, especially effective against simple substitution ciphers.

By giving a frequency breakdown of character use, this function allows the project to compare the extent to which the encryption has randomized and flattened the frequency distribution, which is one measure of encryption strength.

The “plot_frequency_analysis” function plots these results. It reads plaintext and ciphertext, tallies their character frequencies over the entire printable ASCII range, and normalizes the tallies to get probability distributions. The normalized frequencies are thereafter displayed side-by-side in bar charts. In the plot, if the frequency distribution of the ciphertext is fairly level compared to the plaintext's peaky distribution (with some characters such as vowels occurring heavily), it ensures that the encryption has successfully hidden underlying patterns. The ability to use Matplotlib to output the frequency plot as a high-quality JPEG file not only makes results reportable but also enables archiving of security validation evidence for auditing or additional academic scrutiny.

The other crucial element is the Shannon entropy function that returns the entropy of the input text string. Shannon entropy quantifies the uncertainty or randomness in the text. Large entropy values suggest that the text is more random and therefore more attack-resistant like frequency analysis. By comparing initial and final entropy values, the system ensures that encryption indeed enhanced randomness, further consolidating the argument for safety.

Shannon entropy, developed by Claude Shannon in 1948, is a mathematical model of uncertainty or randomness of a system. When used with text, Shannon entropy measures how random or diverse the characters of the text are. It is an important concept in information theory and cryptography since it can be used to estimate how much information is transmitted by a message and how immune it is to predictability-based attacks. In the case of text data, entropy is determined according to the probability distribution of the occurrence of each character in the text. If a text employs a very small number of characters in repetition (e.g., “aaaaabbbbbccccc”), the entropy is low since the next character is predictable. Conversely, if all characters occur with almost equal probability, the entropy is greater, reflecting a more randomized and less predictable organization.

So in encryption systems, the greater value of entropy post-encryption demonstrates that the ciphertext is more random and less subject to frequency attacks or pattern-attack attacks. For the sake

of this project, calculation of Shannon entropy prior to and after encryption serves the purpose of ensuring that the resulting ciphertext contains drastically higher uncertainty relative to the initial plaintext, substantiating the point that the encryption has successfully disguised the original message structure.

The module finishes with a similarity score calculation between the decrypted text and the original plaintext. This process is to ensure that encryption and decryption are functionally lossless. The similarity score is determined by comparing each corresponding character between the decrypted text and the original plaintext. A high percentage of similarity (nearly 100%) assures that the system maintains data integrity perfectly, i.e., no characters are lost, changed, or misaligned during encryption or decryption.

Together, this module is crucial in confirming the strength of the encryption system. It offers qualitative (visual) and quantitative (entropy, frequency distribution, and similarity score) measures to illustrate the effectiveness of the system. In addition, by incorporating these analyses as standard components of the encryption pipeline, the project encourages transparency, reproducibility, and scientific integrity in its security assertions — critical qualities for academic projects, research publications, or practical cryptographic implementations.

CHAPTER 6

PROJECT DEMONSTRATION AND RESULTS

6.1. Experimental Setup

To assess the effectiveness and security of the proposed encryption scheme, a controlled experiment is set up. The experiment involved selecting a representative plain text sample, and a reference image for key derivation to evaluate performance and security robustness.

For this test, a structured English paragraph was chosen as the plain text input. The selected text reads:

“In today's digital age, where data has become one of the most valuable commodities, encryption plays a critical role as the guardian of trust, privacy, and security. As information flows rapidly across various networks, it is no longer just a technical necessity to protect it, but a fundamental responsibility we hold, not only to ourselves but also to future generations. Every byte of data we share, every transaction we make, and every communication we send is susceptible to threats. Without proper encryption, this data is left vulnerable to interception and misuse. Therefore, encryption is not just about shielding information from unauthorized access; it is about preserving the integrity of personal privacy and upholding the values of trust and security in the digital world. In this context, securing data is a collective responsibility that will ensure the safety and stability of future digital ecosystems.”

This passage was deliberately chosen for its diverse linguistic structure, moderate entropy, and real-world relevance to cybersecurity. It contains a mix of common words, technical terms, and punctuation, making it an ideal candidate for evaluating encryption performance. The text consists of 920 characters and features a balanced mix of high-frequency words alongside less predictable terms (e.g., "responsibility," "generations"). The inclusion of punctuation and contractions increases

character variety, making frequency analysis slightly more challenging than a simple, repetitive phrase.

For our key, we chose a high-resolution (4032 × 3024) JPEG picture from a personal gallery. Fig 6.1. is a photograph of the idyllic beauty of a hill station with a sweeping view of a sprawling area of green trees under an open sky. From a high position, the image offers a dynamic combination of texture, color, and natural features, featuring fine details in the foliage and changing light effects on the terrain.

Fig 6.1: Image that is used for key derivation

The selection of this image plays a functional and a visually appealing role in the process of encryption. The enormous amount of distinct pixel information contained within a high-resolution nature photo adds to high entropy, which guarantees that the resulting cryptographic key is non-repetitive and attack-resistant. Contrary to artificially produced patterns or low-detail images, a nature photo inherently is not predictable, and hence it is very effective to generate secure keys.

Moreover, the inherent randomness of the scene with trees of different sizes, shapes, and shades introduces an additional layer of randomness to the process of feature extraction. Even small variations in lighting or position of the camera would produce a very different set of features extracted, making the tamper sensitivity of the encryption scheme even stronger. This implies that even if an attacker was in possession of a comparable picture, they could not recreate the very same cryptographic key without the original image.

Aside from its implications for security, this picture also represents visual evidence of real-world entropy and the way nature's randomness can be used to develop cryptographic applications. The way that light and darkness interact, the changing density of the foliage, and the smooth color shifts all add up to the unpredictability of the encryption key, which makes this method both technically feasible and visually attractive.

6.2. Encryption Key Generation and Entropy Evaluation

Using the selected plain text and high-resolution nature photograph, we evaluated the effectiveness of our encryption scheme. The key derived from the image served as the foundation for both encryption and decryption, ensuring that no external key storage was required. In this case, the secret key is:

AHODCXPHWYGXFFBEZIKOQUAGBELZTWMC

The cryptographic key extracted from the image was analyzed using Shannon entropy calculations. The entropy value of the generated key was 4.3125 bits per character, indicating a strong degree of randomness while still allowing reproducibility from the image. Although this entropy is lower than the theoretical maximum of approximately 6.57 bits per character for printable ASCII [13], it is sufficiently high to resist brute-force attacks. The presence of image-dependent features introduces variability that enhances security beyond typical user-generated passwords.

The encryption process transforms the plain text into an obfuscated cipher text that lacks any identifiable linguistic patterns. The entropy of the plain text was measured at 4.2575 bits per character, while the entropy of the cipher text increased to 5.6958 bits per character, demonstrating a

significant reduction in predictability. This increase suggests that the encryption process effectively disperses character frequency distributions, making it resistant to frequency analysis attacks. Fig 6.2. shows the cipher text generated after the encryption process is performed on the plain text.

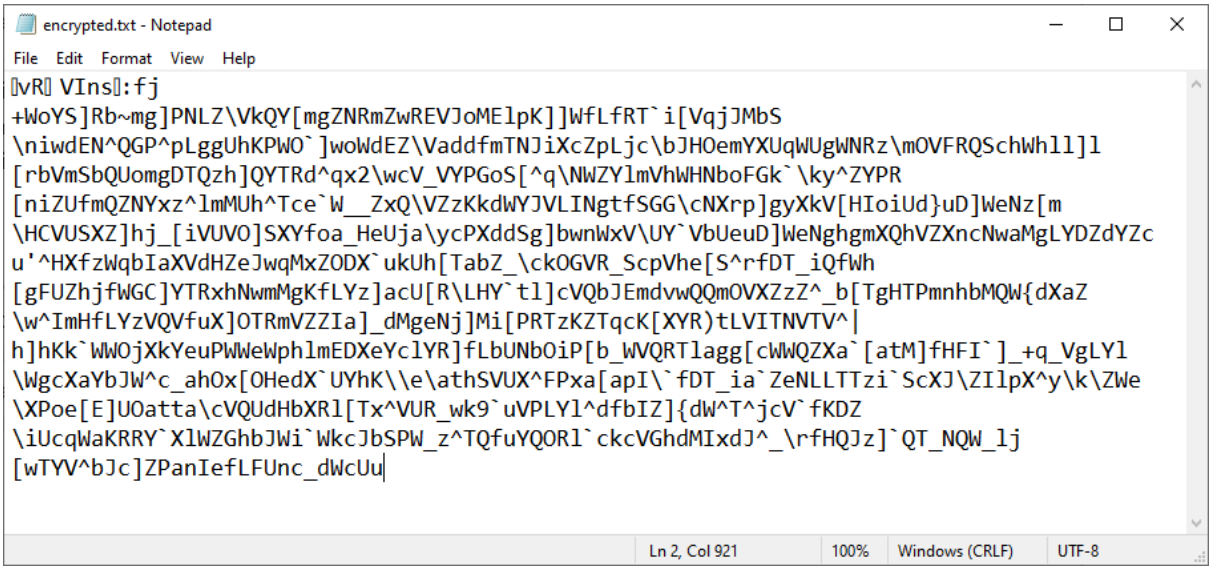

Fig 6.2: Cipher text that is generated from the plain text input using the scenery image

Unlike conventional text-based keys that may retain some structural predictability, the dynamically generated key from the image introduced non-linguistic randomness, further enhancing security. The resulting cipher text appeared as an evenly distributed set of characters, ensuring that even repeated segments in the plain text did not produce discernible patterns in the encrypted output.

Fig 6.3. presents a comparative frequency analysis of characters in plain text and cipher text when encrypted using the Vigenère cipher over the full range of ASCII printable characters (32 - 126). The horizontal axis represents ASCII characters, while the vertical axis denotes their normalized frequency of occurrence.

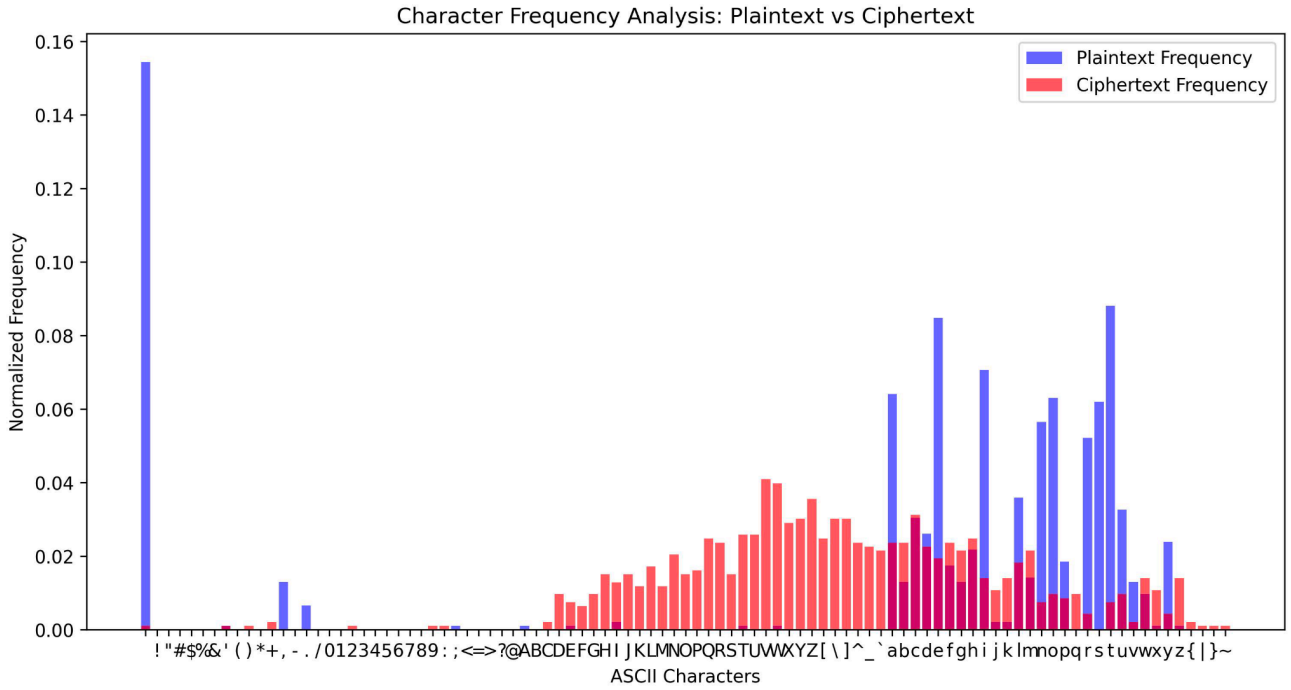

Fig 6.3: Comparative frequency analysis between plain text and cipher text

The distribution of the plain text exhibits characteristic non-uniformity, with specific characters such as spaces and commonly used letters appearing at higher frequencies. This reflects natural linguistic patterns present in the source text.

In contrast, the cipher text distribution demonstrates a significant deviation from the plain text distribution, appearing more uniform. This is a consequence of the Vigenère cipher's polyalphabetic nature, which redistributes character occurrences based on the varying shifts determined by the secret key. The reduced distinguishability between characters in the cipher text suggests an increase in entropy, thereby enhancing resistance to frequency-based cryptanalysis.

This visualization underscores the cipher's effectiveness in obfuscating direct character correlations, making statistical attacks more challenging compared to classical monoalphabetic substitution techniques. However, further statistical tests, such as entropy measurement and correlation analysis, are necessary to comprehensively assess the cipher's strength against more advanced cryptanalytic methods.

6.3. Sensitivity to Image Alterations

Furthermore, the encryption framework demonstrated high sensitivity to modifications in the source image. Even minor adjustments such as cropping, brightness changes, or compression artifacts result in entirely different key derivations, ensuring that an attacker cannot reconstruct the key using an altered version of the original image.

To evaluate the robustness of the encryption scheme, we conducted an experiment where the original 4032 × 3024 image was slightly cropped to 4025 × 3024, reducing its width by just seven pixels. Despite this minor modification which is imperceptible to the human eye, the base key derived from the cropped image failed to match the original base key, causing decryption to fail as it cannot decrypt the metadata attached to the cipher text, which is necessary to be able to generate the dynamic component of the encryption key.

This failure occurs because the key generation process relies on image feature extraction via patch hashing, which is highly sensitive to pixel-level alterations. Even a small change in the spatial composition of the image modifies the extracted feature set and ultimately leads to a different cryptographic key. Since the decryption process depends entirely on key reproducibility, any deviation in the image results in an entirely different key, making decryption infeasible. Fig 6.4. shows the output received when attempting to decrypt with this modified image.

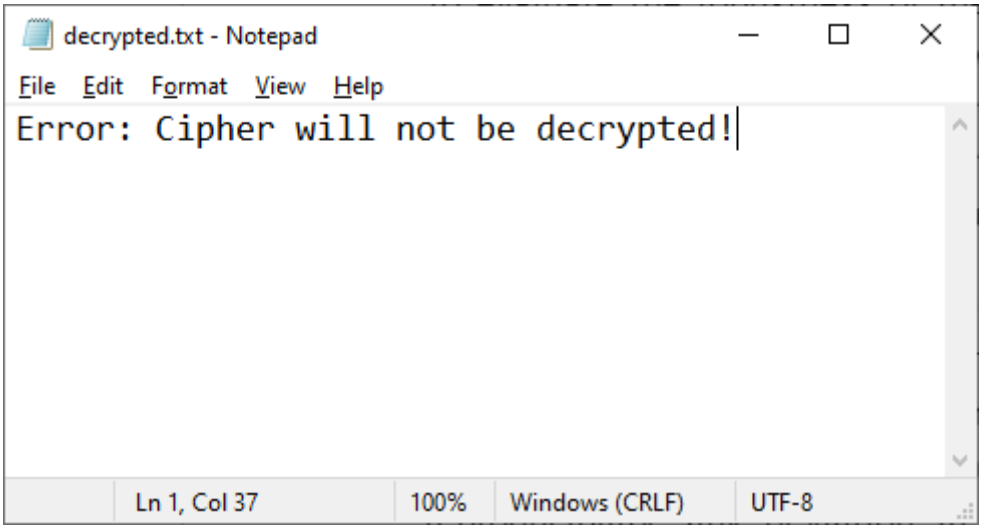

Fig 6.4: Failure message displayed when using an invalid image as the key for decryption

This test reinforces the tamper resistance of the system, ensuring that an adversary cannot reconstruct the correct decryption key using a modified version of the original image. Any manipulations such as cropping, resizing, compression, or color adjustments disrupt the feature extraction pipeline, making unauthorized decryption virtually impossible. This property significantly enhances security, as an attacker would need access to the exact unaltered image to successfully decrypt the data.

6.4. Encryption Strength with Low-Detail Images

A potential concern is whether low-complexity images affect the security of the system. To evaluate this, we conducted an experiment using an image completely devoid of complex details: a pure black (RGB: 0,0,0) image of size 1920 × 1080. Upon generating the secret key, we get:

HFEWKJTQDFKBFQOHJEWYZSPXTJCWLGPG

Using the same plain text as earlier, the Shannon entropy of the cipher text comes out to 5.6265 bits per character. This outcome shows that even when an image with minimum entropy and no features present is used, the encryption process still generates a high-entropy cipher text, which guarantees good security. The secret key extracted is still adequately randomized since the feature extraction process does not just depend on the complexity of the image alone but also on the exclusive transformation and hashing methods used to the image data. The resilience of the encryption stems from two key factors:

- **Dynamic Key Augmentation:** The feature extraction process still generates a hash from the input image using our patches technique. Since the base key is also dependent on the dimensions of our image via patch hashing, the feature hash generated remains random enough. Additionally, the system further strengthens security by incorporating time-dependent dynamic key derivation, preventing reuse of the same key across different sessions.

- **Vigenère Cipher with Strong Key Material:** The actual encryption process is independent of image complexity, as it relies on the extracted feature hash, which is further processed to generate the secret encryption key. Even if an image lacks detail, the secret key derived from it is sufficiently randomized and maintains a high resistance to frequency analysis attacks.

Also, the entropy of 5.6265 bits per character shows a high level of randomness in the cipher text, which resists frequency analysis attacks. This is similar to the entropy of cipher texts from very detailed images, which supports that encryption strength isn't significantly weakened even using low-complexity input images. Therefore, this experiment validates that the system is not based on complex image details to ensure security.

6.5. Execution Time Analysis

To evaluate the performance of our encryption system, we conducted a series of timing experiments using different plain text lengths and images. The encryption and decryption times were measured using Python's time module to ensure accurate benchmarking. Each test was repeated five times, and the average execution time was recorded. We selected three plain texts of varying lengths:

- **Short plain text:** 275 characters
- **Medium plain text:** 920 characters
- **Long plain text:** 4189 characters

For the image-based key generation, we used the following three images:

- **High-Detail Image (4032 × 3024):** A nature photograph with rich textures and complex details.
- **Low-Detail Image (1920 × 1080):** A completely uniform image to test how low entropy affects key derivation.
- **Blurred Image (3060 × 4080):** A blurry image of a nature photograph to test how lower detailed images affect encryption.

The execution times for encryption and decryption across different plain text and image combinations are summarized in Table 6.1.

Image Type	Text Length	Avg Encryption Time (s)	Avg Decryption Time (s)
High-Detail (4032x3024)	Short	0.3962	0.3852
	Medium	0.4009	0.3912
	Long	0.3918	0.3811
Pure Black (1920x1080)	Short	0.2775	0.2626
	Medium	0.2715	0.2662
	Long	0.2714	0.2618
Blurred Image (3060 × 4080)	Short	0.3773	0.3608
	Medium	0.3744	0.3598
	Long	0.3652	0.3581

Table 6.1: Average execution times of the encryption and decryption system against different plain texts and images

The execution times demonstrate a clear relationship between image complexity and encryption/decryption performance. High-detail images exhibit slightly higher processing times due to the increased entropy in their feature-derived keys. In contrast, the pure black image results in significantly faster execution, as the derived key is computationally simpler to generate and apply. The blurred image maintains a performance profile similar to high-detail images.

Overall, while image complexity has a measurable impact on execution time, the differences remain within an acceptable range. This confirms that the method can operate efficiently across diverse image types without compromising security. The encryption scheme also exhibits a low

computational footprint, making it practical for real-time applications while maintaining strong security guarantees.

CHAPTER 7

CONCLUSIONS

This paper introduces an image-derived key generation system for cryptography, which uses the distinct structural properties of an input image to generate a cryptographic key. By combining this with the Vigenère Cipher, we introduce an encryption scheme that is secure and extremely sensitive to changes in an image. The system makes sure that any slight changes to the input image yield completely different keys, discouraging any form of unauthorized decryption.

Experimentation analysis proves our method robust and measurements of the entropies of cipher texts show that the encryption adds adequate randomness even with the use of low-detail images like a pure black image. Also, our benchmarks illustrate that the process of encryption and decryption remains computationally efficient. The high image modification sensitivity of the system was also verified through cropping the original image, upon which decryption processes failed. This attests to the robustness of the approach proposed in preserving data confidentiality through the requirement for the precise original image in case any unauthorized action is taken towards reconstructing the key.

In conclusion, the proposed encryption system effectively leverages image-derived entropy for secure key generation, ensuring strong encryption and attack resilience. The results confirm its robustness, with high cipher text entropy and sensitivity to image modifications, making decryption infeasible without the exact image. This work lays the foundation for future advancements in dynamic and secure encryption.

CHAPTER 8

FUTURE ENHANCEMENT

While the proposed encryption system demonstrates strong security properties and resilience to attacks, several opportunities exist for further refinement and expansion. One area of interest is improving key expansion mechanisms. Currently, the system derives keys from image features, but future implementations could integrate adaptive key expansion techniques, such as deep learning-based feature extraction or hash chaining, to further enhance the unpredictability and strength of the generated keys.

Another avenue for advancement involves integrating modern cryptographic standards. While the Vigenère cipher proves effective within this framework, hybrid approaches incorporating stronger encryption schemes such as AES, RSA, or ECC could enhance both security and efficiency. This would make the system more viable for applications requiring robust data protection against cryptographic advancements.

Optimizing performance for real-time applications is another crucial aspect. Although the algorithm operates efficiently in controlled experiments, handling large-scale data encryption or real-time streaming requires additional optimizations. Implementing parallel processing or GPU-accelerated computations could significantly reduce encryption and decryption times, making the system more practical for high-throughput environments.

Additionally, the flexibility of the key generation process can be expanded by allowing multiple images as input. Instead of relying on a single image, incorporating multiple images for key derivation could add another layer of security. This would require an attacker to possess all the images used in key generation, further complicating any unauthorized access attempts. This approach can also be improved by making it so that files can only be decrypted when the images are entered in

a specific order, making unauthorized access much more difficult.

Finally, future research can explore the application of this encryption method in distributed systems, such as secure cloud storage or blockchain-based data protection. With increasing concerns over data security in decentralized environments, integrating this approach into modern security architectures could provide enhanced confidentiality while maintaining efficiency.

REFERENCES

[1] Rubinstein-Salzedo, S. (2018). The Vigenère Cipher. In: Cryptography. Springer Undergraduate Mathematics Series. Springer, Cham. https://doi.org/10.1007/978-3-319-94818-8_5

[2] Bale, A. S., Ghorpade, N., K, B., Hashim, M. F., H., C., & R, H. (2023). *Modification of Vigenère Cipher to Overcome Kasiski and Friedman Attacks*. 1–5. https://doi.org/10.1109/ccpis59145.2023.10291990

[3] Katema, E. C., & Chatola, F. (2024). Leveraging the Techniques of Vigenère Cipher and Modern Cryptographic Algorithms. *International Journal of Advanced Research in Science, Communication and Technology*, 692–699. https://doi.org/10.48175/ijarsct-22294

[4] Abraham, N. S., Watanabe, K., Taniguchi, T., & Majumdar, K. (2022). A High-Quality Entropy Source Using van der Waals Heterojunction for True Random Number Generation. *ACS Nano*, *16*(4), 5898–5908. https://doi.org/10.1021/acsnano.1c11084

[5] Ahuja, M.S., Chabbra, S. (2011). Biometric Encryption: Combining Fingerprints and Cryptography. In: Mantri, A., Nandi, S., Kumar, G., Kumar, S. (eds) High Performance Architecture and Grid Computing. HPAGC 2011. Communications in Computer and Information Science, vol 169. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-22577-2_69

[6] Yang Zhao, P. Campisi and D. Kundur, "Dual domain watermarking for authentication and compression of cultural heritage images," in IEEE Transactions on Image Processing, vol. 13, no. 3, pp. 430-448, March 2004, doi: 10.1109/TIP.2003.821552.

[7] Arambam Neelima, Kh Manglem Singh, Perceptual Hash Function based on Scale-Invariant Feature Transform and Singular Value Decomposition, *The Computer Journal*, Volume 59, Issue 9, September 2016, Pages 1275–1281, https://doi.org/10.1093/comjnl/bxv079

[8] Ihsan, A., & Doğan, N. (2024). An innovative image encryption algorithm enhanced with the Pan-Tompkins Algorithm for optimal security. *Multimedia Tools and Applications*. https://doi.org/10.1007/s11042-024-18722-x

[9] Dash, P., Pandey, F., Sarma, M., & Samanta, D. (2023). Efficient private key generation from iris data for privacy and security applications. *Journal of Information Security and Applications*, *75*, 103506. https://doi.org/10.1016/j.jisa.2023.103506

[10] Li, F., Liu, H., Li, G., Wang, Y., Zhou, H., Cao, S., & Li, T. (2024). SeSMR: Secure and Efficient Session-based Multimedia Recommendation in Edge Computing. *ACM Transactions on Multimedia Computing, Communications, and Applications*. https://doi.org/10.1145/3687473

[11] Gupta, M., Gupta, K., & Shukla, P. (2021). Session key based fast, secure and lightweight image encryption algorithm. *Multimedia Tools and Applications*, *80*(7), 10391–10416. https://doi.org/10.1007/S11042-020-10116-Z

[12] Luorio, A.F., Visconti, A. (2019). Understanding Optimizations and Measuring Performances of PBKDF2. In: Woungang, I., Dhurandher, S. (eds) 2nd International Conference on Wireless Intelligent and Distributed Environment for Communication. WIDECOM 2018. Lecture Notes on Data Engineering and Communications Technologies, vol 27. Springer, Cham. https://doi.org/10.1007/978-3-030-11437-4_8

[13] Rodríguez, L., Madarro-Capó, E. J., Legón-Pérez, C. M., Rojas, O., & Sosa-Gómez, G. (2021). Selecting an Effective Entropy Estimator for Short Sequences of Bits and Bytes with Maximum Entropy. *Entropy*, *23*(5), 561. https://doi.org/10.3390/E23050561

APPENDIX A

CODING

```
import time
import math

from PIL import Image
import numpy as np
import hashlib
import base64
from cryptography.hazmat.primitives.kdf.pbkdf2 import PBKDF2HMAC
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.backends import default_backend
import matplotlib.pyplot as plt
from collections import Counter
import string

# LOGIC
# User A and B derive the base key from the image
# User A generates the dynamic key using the timestamp and the base key, and appends it to the base key to form the final key
# User B does the same, but gets the timestamp from the encrypted file's metadata, and decrypts the file

# --- Lightweight XOR Encryption for Metadata ---
def xor_encrypt_decrypt(text, key):
    """
    XOR encrypt the text with a given key.
    """
    return ''.join(chr(ord(c) ^ ord(key[i % len(key)])) for i, c in enumerate(text))


# --- Image Processing ---
def preprocess_image(image_path, resize_dim=(1024, 1024)):
    """
    Preprocess the image: convert to grayscale and resize.
    """
    with Image.open(image_path) as img:
        img = img.resize(resize_dim)  # Resize for uniformity

    # Display the processed image using Matplotlib
    plt.imshow(img)
    plt.title("Preprocessed Image")
    plt.axis("off")
    plt.show()
```

```
    pixel_array = np.array(img)
    return pixel_array


# --- Feature Extraction ---
def extract_features(image_array, patch_size=(8, 8)):
    """
    Extract robust features from the image array by combining global and local hashes.
    """
    normalized_pixels = image_array / 255.0
    flat_pixels = normalized_pixels.flatten()
    global_hash = hashlib.sha512(flat_pixels.tobytes()).digest()
    salt = hashlib.sha256(global_hash).digest()[:16]

    # Patch-based hashing
    h, w, _ = image_array.shape
    patch_hashes = []
    for i in range(0, h, patch_size[0]):
        for j in range(0, w, patch_size[1]):
            patch = normalized_pixels[i:i + patch_size[0], j:j + patch_size[1], :]
            patch_flat = patch.flatten()
            patch_hash = hashlib.sha256(patch_flat.tobytes()).digest()
            patch_hashes.append(patch_hash)

    combined_patches = b"".join(patch_hashes)
    combined_data = global_hash + combined_patches + salt
    feature_hash = hashlib.sha512(combined_data).digest()

    return feature_hash, salt


# --- Key Derivation ---
def derive_key(feature_hash, salt, iterations=100000, key_length=32):
    """
    Derive a secure key using PBKDF2.
    """
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=key_length,
        salt=salt,
        iterations=iterations,
        backend=default_backend()
    )
    return kdf.derive(feature_hash)


def derive_dynamic_key(base_key, timestamp):
    """
```

```
    Derives the dynamic key using the timestamp.
    """
    timestamp_bytes = str(timestamp).encode()
    dynamic_key = hashlib.sha256(base_key + timestamp_bytes).digest()
    return dynamic_key


# --- Secret Word Extraction ---
def extract_secret_word(key, length=32):
    """
    Generate a readable secret word from the binary key.
    """
    return ''.join(chr((byte % 26) + 65) for byte in key[:length])


# --- Vigenère Cipher ---
def vigenere_encrypt(text, secret_word):
    """
    Encrypt the text using the Vigenère cipher for all characters.
    Works within ASCII printable range (32–126).
    """
    encrypted_text = []
    word_len = len(secret_word)

    for i, char in enumerate(text):
        ascii_val = ord(char)
        shift = ord(secret_word[i % word_len]) % 95  # Shift within printable ASCII range
        new_ascii = ((ascii_val - 32 + shift) % 95) + 32  # Keep within printable range (32–126)
        encrypted_text.append(chr(new_ascii))

    return ''.join(encrypted_text)


def vigenere_decrypt(encrypted_text, secret_word):
    """
    Decrypt the text using the Vigenère cipher for all characters.
    Works within ASCII printable range (32–126).
    """
    decrypted_text = []
    word_len = len(secret_word)

    for i, char in enumerate(encrypted_text):
        ascii_val = ord(char)
        shift = ord(secret_word[i % word_len]) % 95  # Shift within printable ASCII range
        new_ascii = ((ascii_val - 32 - shift) % 95) + 32  # Keep within printable range (32–126)
        decrypted_text.append(chr(new_ascii))

    return ''.join(decrypted_text)
```

```
def read_file(filename):
    """
    Read the content of a file and return it.
    """
    with open(filename, 'r') as file:
        return file.read()


def frequency_analysis(text):
    """
    Perform frequency analysis of letters in a given text.
    """
    # Filter only alphabetic characters and convert to lowercase
    letters = [char.lower() for char in text if char in string.ascii_letters]
    return Counter(letters)


def plot_frequency_analysis(plaintext, ciphertext):
    # Count character frequencies
    plain_freq = Counter(plaintext)
    cipher_freq = Counter(ciphertext)

    # Get all possible ASCII printable characters
    ascii_chars = [chr(i) for i in range(32, 127)]

    # Extract frequencies (default to 0 if character is absent)
    plain_counts = [plain_freq[char] for char in ascii_chars]
    cipher_counts = [cipher_freq[char] for char in ascii_chars]

    # Normalize frequencies
    total_plain = sum(plain_counts)
    total_cipher = sum(cipher_counts)
    plain_norm = [count / total_plain for count in plain_counts]
    cipher_norm = [count / total_cipher for count in cipher_counts]

    # Plot bar charts
    plt.figure(figsize=(12, 6))

    plt.bar(ascii_chars, plain_norm, alpha=0.6, label="Plaintext Frequency", color='blue')
    plt.bar(ascii_chars, cipher_norm, alpha=0.6, label="Ciphertext Frequency", color='red')

    plt.xlabel("ASCII Characters")
    plt.ylabel("Normalized Frequency")
    plt.title("Character Frequency Analysis: Plaintext vs Ciphertext")
    plt.legend()
```

```
    plt.savefig("freq_analysis.jpg", dpi=300, format="jpg", bbox_inches="tight")
    plt.show()


def shannon_entropy(text):
    """Calculate Shannon entropy of a given text string."""
    if not text:
        return 0

    freq = Counter(text)
    text_length = len(text)

    entropy = -sum((count / text_length) * math.log2(count / text_length) for count in freq.values())

    return entropy


if __name__ == "__main__":
    # File paths
    image_path_for_encryption = "image1.jpg"
    image_path_for_decryption = "image1.jpg"
    input_file = "plaintext2.txt"

    encrypted_file = "encrypted.txt"
    decrypted_file = "decrypted.txt"

    total_encryption_time = 0
    total_decryption_time = 0

    for i in range(1):
        encryption_start_time = time.time()

        # --- Key Generation ---
        image_array = preprocess_image(image_path_for_encryption)
        feature_hash, salt = extract_features(image_array)
        base_key = derive_key(feature_hash, salt)

        # --- Encryption ---
        timestamp = int(time.time())
        dynamic_key = derive_dynamic_key(base_key, timestamp)
        final_key = base_key + dynamic_key
        secret_word = extract_secret_word(final_key)
        print(secret_word)

        with open(input_file, 'r') as f:
            plaintext = f.read()
```

```
encrypted_text = vigenere_encrypt(plaintext, secret_word)
validity = 10  # File validity in seconds

metadata = f"{timestamp}\n{validity}"
base_key_str = base64.b64encode(base_key).decode()
encrypted_metadata = xor_encrypt_decrypt(metadata, base_key_str)

with open(encrypted_file, 'w') as f:
    f.write(encrypted_metadata + "\n" + encrypted_text)

encryption_end_time = time.time()
print(f"Encrypted text written to: {encrypted_file}")
print(f"Encryption time: {encryption_end_time - encryption_start_time:.4f} seconds")
total_encryption_time += (encryption_end_time - encryption_start_time)


decryption_start_time = time.time()

# --- Decryption ---
image_array = preprocess_image(image_path_for_decryption)
feature_hash, salt = extract_features(image_array)
base_key = derive_key(feature_hash, salt)

with open(encrypted_file, 'r') as f:
    encrypted_content = f.read()

# Extract metadata and decrypt it
encrypted_metadata, encrypted_text = encrypted_content.split("\n", 1)
base_key_str = base64.b64encode(base_key).decode()
metadata = xor_encrypt_decrypt(encrypted_metadata, base_key_str)

try:
    timestamp_str, validity_str = metadata.split("\n")
    timestamp, validity = int(timestamp_str), int(validity_str)
except ValueError:
    with open(decrypted_file, 'w') as f:
        print("Exiting!")
        f.write("Error: Cipher will not be decrypted!")
    exit()


# Uncomment to test file validity expiration
# time.sleep(validity + 1)

# Check if the file is still valid
current_time = int(time.time())
if current_time - timestamp > validity:
    print("Error: Encrypted file has expired!")
```

```
    exit()

# Reconstruct the dynamic key using the retrieved timestamp
dynamic_key = derive_dynamic_key(base_key, timestamp)
final_key = base_key + dynamic_key
secret_word = extract_secret_word(final_key)

# Decrypt the text
decrypted_text = vigenere_decrypt(encrypted_text, secret_word)

with open(decrypted_file, 'w') as f:
    f.write(decrypted_text)

decryption_end_time = time.time()
print(f"Decrypted text written to: {decrypted_file}")
print(f"Decryption time: {decryption_end_time - decryption_start_time:.4f} seconds")
total_decryption_time += (decryption_end_time - decryption_start_time)


# Perform frequency analysis
file1 = 'encrypted.txt'
file2 = 'plaintext2.txt'

text1 = read_file(file1)
text2 = read_file(file2)

freq1 = frequency_analysis(text1)
freq2 = frequency_analysis(text2)

plot_frequency_analysis(text2, text1)

entropy_plaintext = shannon_entropy(text2)
entropy_ciphertext = shannon_entropy(text1)

print(entropy_plaintext)
print(entropy_ciphertext)

file1 = 'decrypted.txt'
file2 = 'plaintext2.txt'

text1 = read_file(file1)
text2 = read_file(file2)

length = min(len(text1), len(text2))
text1, text2 = text1[:length], text2[:length]

# Count matching characters
matches = sum(1 for c1, c2 in zip(text1, text2) if c1 == c2)
```

```
    # Calculate percentage
    score = (matches / length) * 100
    print(score)

print(f"Average encryption time: {total_encryption_time / 5:.4f} seconds")
print(f"Average decryption time: {total_decryption_time / 5:.4f} seconds")
```

APPENDIX B

CONFERENCE PUBLICATION

A manuscript of the project has been submitted to the International Conference on Innovations in Intelligent Systems 2025 and is currently under review by the organizers.

2025 International Conference on Innovations in Intelligent Systems: Advancements in Computing, Communication, and Cybersecurity : Submission (684) has been created.

1 message

Microsoft CMT <email@msr-cmt.org> Fri, Apr 4, 2025 at 11:15 PM
To: ap2907@srmist.edu.in

Hello,

The following submission has been created.

Track Name: ISAC32025

Paper ID: 684

Paper Title: Image-Derived Cryptographic Keys: A Novel Mechanism for Enhancing Classical Cipher Security

Abstract:
The Vigenère cipher, while of historical importance, is susceptible to frequency analysis and brute-force attacks because it is based on linguistic keys. This paper presents a new method that is more secure by producing random, high-entropy keys from digital images. Using image information, the system avoids frequency-based attacks and eliminates key memorization. The framework consists of three key phases: image preprocessing, where inputs are standardized; localized patch hashing, ensuring tamper sensitivity; and iterative key derivation, producing cryptographically sound material. Additionally, the approach operates covertly, keeping encryption mechanisms hidden from adversaries. Keys can be stored inconspicuously within image collections, creating a significant challenge for attackers. Session-based encryption ensures that each encryption instance is unique, preventing replay attacks and key reuse. Experimental results demonstrate that image-derived keys significantly strengthen the Vigenère cipher. Even minor image modifications (like cropping, watermarking, noise) lead to drastically different base keys, rendering decrypted text unintelligible. The method effectively disrupts traditional letter frequency biases, making cryptanalysis more difficult. This work contributes: (1) a deterministic algorithm for binding visual data to cryptographic keys, (2) empirical validation of image-based keys against cryptanalysis, and (3) session-based encryption to limit key exposure. The results confirm its robustness, with high cipher text entropy and sensitivity to image modifications, making decryption infeasible without the exact image. This work lays the foundation for future advancements in dynamic and secure encryption.

Created on: Fri, 04 Apr 2025 17:45:50 GMT

Last Modified: Fri, 04 Apr 2025 17:45:50 GMT

Authors:
- ap2907@srmist.edu.in (Primary)
- ps6668@srmist.edu.in
- malarseg@srmist.edu.in

Primary Subject Area: Track 6: Advances in Cyber Security and Blockchains

Secondary Subject Areas: Not Entered

Submission Files:
Image-Derived Cryptographic Keys.pdf (442 Kb, Fri, 04 Apr 2025 17:45:35 GMT)

Submission Questions Response: Not Entered

Thanks,
CMT team.

APPENDIX C

PLAGIARISM REPORT