Simple Encryption Tool

REPORT OF PROJECT SUBMITTED FOR PARTIAL FULFILLMENT OF THE

REQUIREMENT FOR THE DEGREE OF

BACHELOR OF COMPUTER APPLICATION

By

TUSHAR KANTI MAJUMDER

REGISTRATION NO – 221641010067

UNIVERSITY ROLL NO – 16401222020

UNDER THE SUPERVISION OF

Prof. Srikanta Sen

[Assistant Professor, George College of Management and Science]

AT

GEORGE COLLEGE OF MANAGEMENT AND SCIENCE

[Affiliated to **Maulana Abul Kalam Azad University of Technology**]

B.B.T. ROAD, KOLKATA – 141

August – 2024

GEORGE COLLEGE OF MANAGEMENT AND SCIENCE

KOLKATA – 7OO141, INDIA

CERTIFICATE

The report of the Project titled **Simple Encryption Tool** submitted by **TUSHAR KANTI MAJUMDER** (Roll No.: 16401222020 of BCA 5TH Semester Of 2024) has been prepared under our supervision for the partial fulfillment of the Requirements for BCA degree awarded by MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY.

The report is hereby forwarded.

_____

(Internal Supervisor)

ACKNOWLEDGEMENT

I express my sincere gratitude to respected **Prof. Srikanta Sen** of Department of BCA, GCMS and for extending their valuable times for me to take up this problem as a Project.

Last but not the least I would like to express my gratitude to my family and friends who helped me in their own way whenever needed.

Date: 15.11.2024

TUSHAR KANTI MAJUMDER

Reg. No.: 221641010067

Roll No.: 16401222020

BCA 5th Semester 2024,

George College of Management and Science

GEORGE COLLEGE OF MANAGEMENT AND SCIENCE

[Affiliated to Maulana Abul Kalam Azad University of Technology]

B.B.T. ROAD, KOLKATA – 141

CERTIFICATE of ACCEPTANCE

The report of the Project titled Cryptography and Hashing Techniques submitted by **TUSHAR KANTI MAJUMDER (Roll No.: 16401222020 of BCA 5th Semester of 2024)** is hereby recommended to be accepted for the partial fulfillment of the requirements for BCA degree in MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY.

	Name of the Examiner	Signature with Date
1.	……	……
2.	……	……
3.	……	……
4.	……	……

Introduction:

Project Proposal: SIMPLE ENCRYPTION TOOL

As a final year BCA student, I am eager to delve into the fascinating world of cryptography and hashing techniques. Cryptography plays a crucial role in securing data and communication in today's digital age. With the increasing reliance on digital platforms for sensitive transactions and communication, understanding cryptography and hashing techniques is paramount. This project aims to explore various cryptographic algorithms, their applications, and the significance of hashing techniques in data security.

Objectives:

- To understand the fundamentals of cryptography and hashing.
- To explore different cryptographic algorithms such as RSA, AES, and DES.
- To analyze the applications of cryptography in data encryption, digital signatures, and secure communication.
- To investigate the role of hashing techniques in data integrity and password storage.
- To implement selected cryptographic algorithms and hashing techniques in practical scenarios.
- To evaluate the performance and security aspects of implemented algorithms.

Methodology:

- **Literature Review: -** Conduct an extensive review of academic papers, textbooks, and online resources to understand the theoretical foundations of cryptography and hashing.
- **Algorithm Exploration: -** Study various cryptographic algorithms such as RSA, AES, DES, and hashing techniques like SHA-256 and MD5. Analyze their working principles, strengths, weaknesses, and real-world applications.
- **Implementation: -** Select a few cryptographic algorithms and hashing techniques for implementation using programming languages such as Python. Develop programs for encryption, decryption and hashing functions.
- **Practical Applications: -** Design and execute experiments to demonstrate the practical applications of implemented algorithms. This could include encrypting and decrypting messages, generating digital signatures, and verifying data integrity using hashing.
- **Security Analysis: -** Assess the security aspects of implemented algorithms by analyzing their resistance to common cryptographic attacks such as brute force, chosen plaintext, and chosen ciphertext attacks.

Deliverables:

- Research paper documenting the theoretical foundations, implementation details, and experimental results.
- Software prototypes demonstrating the practical applications of cryptographic algorithms and hashing techniques.
- Detailed report evaluating the performance and security aspects of implemented algorithms.

Timeline:

Week 1: Research and Planning

- Research ciphers (Caesar, ROT13, XOR, Multiplicative, Transposition).
- Plan project structure and objectives.

Week 2: Cipher Implementation (Part 1)

- Write and test Python code for **Caesar Cipher** and **ROT13**.
- Ensure encryption and decryption work correctly.

Week 3: Cipher Implementation (Part 2)

- Implement **XOR Cipher** and **Multiplicative Cipher** with coprime validation.
- Debug and integrate all ciphers into a menu-based program.

Week 4: Documentation

- Document algorithms, code explanations, advantages, and limitations.
- Add screenshots of online encryption tools.

Week 5: Final Touches

- Review and finalize the project.
- Organize sections, add bibliography, and prepare for submission.

Budget:

The project budget will primarily cover expenses related to software development tools, research materials, and any miscellaneous costs. As a student, I intend to utilize open-source software and freely available resources to minimize the budgetary requirements.

Project Supervisor:

Prof. Srikanta Sen

[Assistant Professor, George College of Management and Science]

We appreciate your consideration of this project proposal and look forward to the opportunity to enhance the cybersecurity skills of our students.

Sincerely,

TUSHAR KANTI MAJUMDER

[Student of BCA 5Th Sem]

System Used in this project

•OS Used: **Windows 11**

• System RAM: **8 GB Ram.**

• System Processor: **INTEL i3 12th generation processor.**

• System Disk space: **256 GB SSD**

Software Used in this project

Spyder: Spyder is a free and open-source scientific environment written in Python, for Python, and designed by and for scientists, engineers and data analysts. It features a unique combination of the advanced editing, analysis, debugging, and profiling functionality of a comprehensive development tool with the data exploration, interactive execution, deep inspection, and beautiful visualization capabilities of a scientific package.

Google Chrome: Google Chrome browser is a free web browser used for accessing the internet and running web-based applications. The Google Chrome browser is based on the open source Chromium web browser project. Google released Chrome in 2008 and issues several updates a year. Google Chrome is available for Microsoft Windows, Apple macOS and Linux desktop operating systems (OSes), as well as the Android and iOS mobile operating systems.

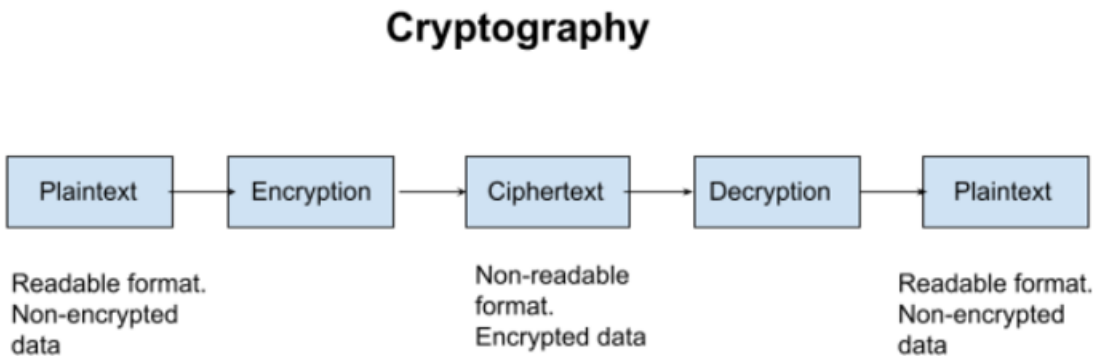
Index

"A Comprehensive Exploration of Cryptography: Techniques, Applications, and Security Implications"

1. What is Cryptography?

Cryptography is the science of securing information and communications through the use of codes so that only intended recipients can access and understand the information. This field involves transforming readable data into an encoded format, known as ciphertext, and then converting it back into readable form for authorized users. Derived from the Greek words "kryptos" (hidden) and "graphein" (writing), cryptography seeks to safeguard information by preventing unauthorized access.

The techniques used in cryptography are based on mathematical concepts and algorithmic rules, which help encode data securely. Cryptography is used extensively today in various applications, such as cryptographic key generation, digital signing, data verification, web browsing security, and safeguarding confidential transactions like online banking, credit, and debit card transactions.

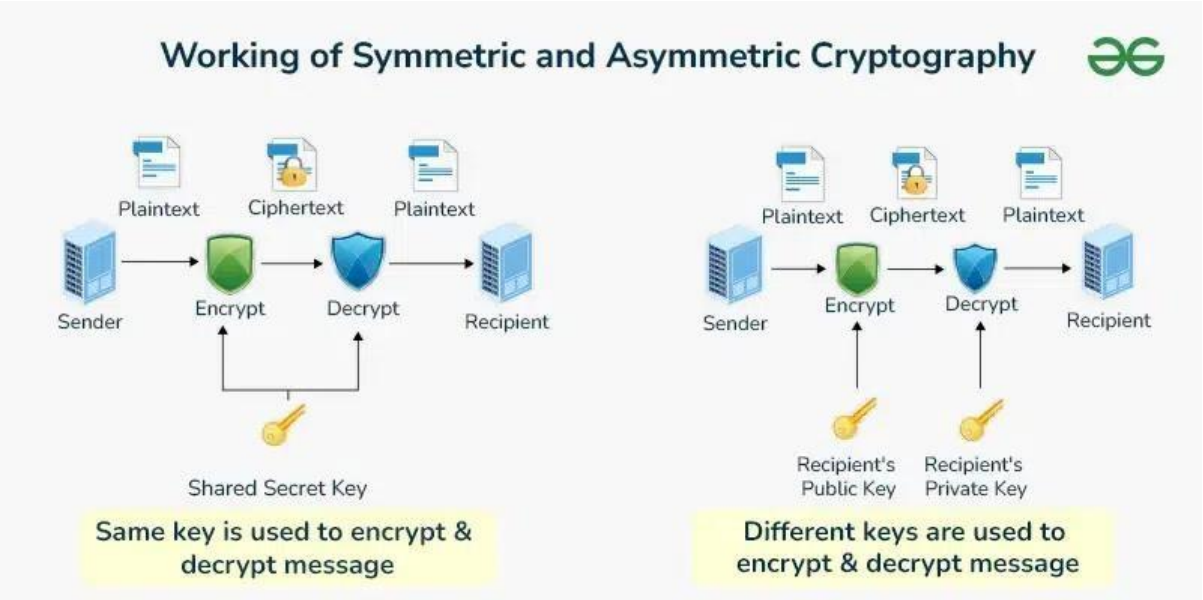


---

2. How Does Cryptography Work?

Cryptography relies on mathematical algorithms, known as ciphers, to encrypt text. These algorithms combine the data (plaintext) with keys—values like phrases, numbers, or words—to transform readable text into unreadable ciphertext. The strength of the cryptographic algorithm and the secrecy of the key are critical for security. The process is generally divided into two types:

- **Symmetric-Key Cryptography**: The same key is used for both encryption and decryption, as seen in algorithms like AES and DES.
- **Asymmetric-Key Cryptography**: Different keys are used for encryption and decryption, commonly utilized in systems like RSA.



In modern cryptographic systems, keys are central to security, as the effectiveness of encryption depends on the algorithm's robustness and the key's confidentiality.

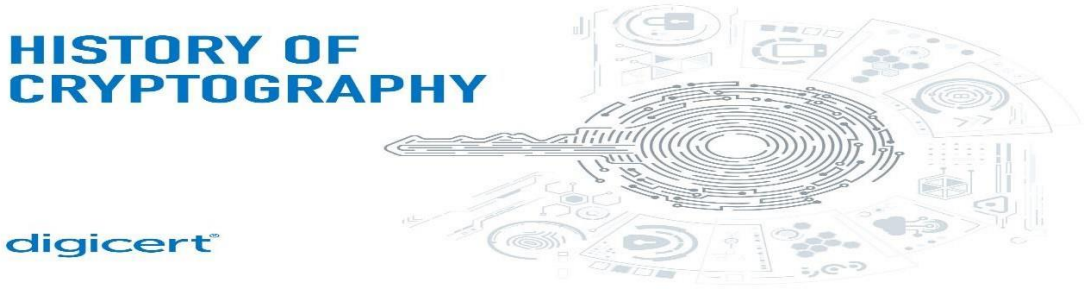
---

3. Examples of Cryptography in Real Life

- **End-to-End Encryption in Messaging Apps**: WhatsApp's use of end-to-end encryption is a prime example, leveraging asymmetric cryptography (public-key encryption). Only the intended recipient can decrypt the message, ensuring that even the platform's servers cannot read the messages exchanged.
- **Digital Signatures**: Digital signatures use cryptographic algorithms to validate the authenticity of electronic documents, contracts, and emails. They are crucial in business transactions to verify identity and build trust among remote parties who may not have physically met.

4. History of Cryptography

The art of cryptography is as old as human civilization itself, evolving as societies developed complex structures, hierarchies, and communication needs. As communities grew, so did the necessity to safeguard information.



A. Ancient Origins of Cryptography

The origins of cryptography trace back to early civilizations like Egypt and Mesopotamia, where people used symbols and hieroglyphs to encode secret messages. Around 1900 BC, non-standard hieroglyphs in ancient Egyptian tombs served as one of the earliest examples of encrypted text. Such codes were typically only known to scribes and officials, allowing rulers to communicate securely.

Hieroglyph - The First Known Cryptographic Technique
Hieroglyphs were special pictographic symbols used by Egyptian scribes to convey confidential messages. One of the oldest cryptographic methods, this form of writing was used to convey messages among elites and religious authorities.

As time progressed, scholars in ancient Greece and Rome introduced simple substitution ciphers, where each letter of the alphabet was shifted by a predetermined number. This early technique laid the foundation for more sophisticated encryption methods.

B. Classical Era of Cryptography

- **The Caesar Cipher**: Julius Caesar used a substitution cipher to communicate securely with his generals. Each letter in his message would be shifted by a set number of positions in the alphabet, rendering the message unreadable without knowing the shift value (the key).
- **Polybius Square**: Around 180 BC, Greek philosopher Polybius invented a technique now known as the Polybius Square. It involved arranging letters in a grid and then encoding each letter with two coordinates,

making it a simple yet effective method for secure communication. Polybius' ideas on coded messaging influenced government and military strategies for centuries.

c. Modern Cryptography

The 20th century saw rapid advances in cryptography, especially during times of war. Mechanical and electromechanical cipher machines like the **Enigma** became crucial for encoding military communications.

- **The Enigma Machine**: Developed by Arthur Scherbius and later modified by German forces during WWII, the Enigma machine was an advanced electromechanical device used for encoding strategic messages. It relied on rotating rotors to scramble letters in an exceedingly complex manner, making it highly secure until the Allies managed to break the code with the help of mathematicians and early computers.
- **Rotor Machines**: Around the same time, American Edward H. Hebern developed a machine using electrical rotors to create complex encryption. His invention and other similar devices marked a leap in cryptographic technology, as they allowed for polyalphabetic substitutions of arbitrary complexity, setting the stage for modern cryptography.

The efforts of these inventors laid the groundwork for the computerized cryptographic systems we rely on today. After WWII, the increasing demand for secure telecommunications led to new encryption protocols that eventually formed the basis of modern internet security.

5. Modern Cryptographic Techniques and Uses

Today's cryptography is essential to secure communications in the digital world. Algorithms like AES, RSA, and ECC are used across various industries for data protection.

- **Advanced Encryption Standard (AES)**: A symmetric encryption standard adopted by the U.S. government, known for its speed and security.
- **RSA (Rivest-Shamir-Adleman)**: An asymmetric encryption algorithm widely used for secure data transmission and digital signatures.
- **Elliptic Curve Cryptography (ECC)**: A compact and secure encryption technique ideal for mobile devices due to its low computational power requirements.

Cryptographic methods today protect our emails, web browsing, and financial transactions. They are embedded in digital certificates, secure messaging, and online payment systems, ensuring data integrity and confidentiality in the digital age.

---

6. The Future of Cryptography

As technology advances, so does the need for more robust cryptographic methods. Emerging areas like **Quantum Cryptography** and **Post-Quantum Cryptography** are being explored to counter the potential of quantum computing, which could break many current cryptographic algorithms. Quantum key distribution and new, quantum-resistant algorithms may eventually transform cryptography, making it even more secure against future threats.

Cryptography: An Overview

1. Terminology Related to Cryptography

A. Encryption

Encryption is the technique of converting readable data (plaintext) into a scrambled, unreadable format (ciphertext) to ensure only authorized parties can access and understand the information. This transformation uses a cryptographic key—a mathematical value agreed upon by both sender and recipient. In simple terms, encryption secures data by making it appear random, and it requires a cipher (or encryption algorithm) to encode and decode messages.

B. Key

In cryptography, a key is a string of characters used within an encryption algorithm to alter data, making it appear random. Like a physical key, it locks (encrypts) data so that only someone with the correct key can unlock (decrypt) it. The length of the key affects the difficulty of decrypting a message and, therefore, its security level.

C. Decryption

Decryption is the process of transforming encrypted (ciphertext) information back into its original readable format (plaintext). This is the reverse of encryption, allowing intended users to understand the content. Decryption is a core aspect of cybersecurity, ensuring that scrambled data sent securely across networks can be accessed only by authorized parties.

D. Cipher

A cipher is an algorithm used for encryption and decryption. It transforms plaintext into ciphertext, making it appear as random data. Ciphers are categorized as:

- **Stream Ciphers**: Encrypt data bit-by-bit in a stream.
- **Block Ciphers**: Break data into fixed-sized blocks and encrypt each separately.

To ensure adequate security, modern ciphers typically use a key length of at least 128 bits.

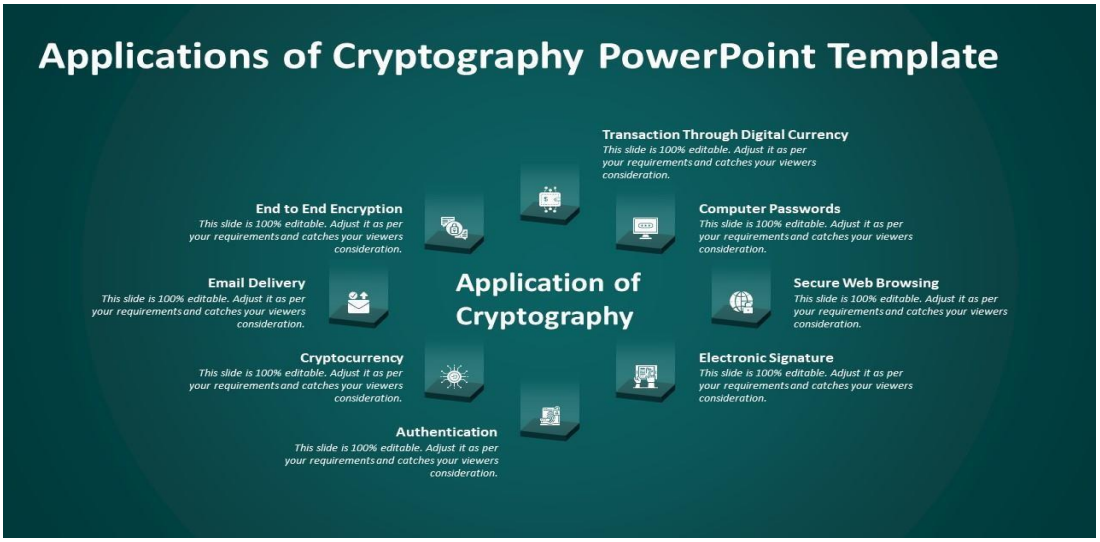
Applications of Cryptography

Cryptography is used in various applications to secure data and ensure confidentiality and authenticity. Common applications include:

- **Computer Passwords**: Cryptography secures passwords by hashing them. When a user logs in, their password is hashed and compared to the stored hash. This technique keeps passwords unreadable even if accessed by unauthorized users.
- **Digital Currencies**: Cryptography secures digital currencies like Bitcoin by using complex algorithms and cryptographic keys to prevent tampering or fraudulent transactions.
- **Secure Web Browsing**: Cryptography ensures safe browsing through SSL/TLS protocols, which use public-key encryption to secure data transferred between users and web servers, protecting against eavesdropping.
- **Electronic Signatures**: Digital signatures, secured by cryptography, serve as the digital equivalent of a handwritten signature. These signatures authenticate document ownership and integrity.
- **Authentication**: Cryptographic techniques verify user identities in various scenarios, such as banking, computer login, and network access. Authentication protocols confirm user credentials, ensuring secure access to resources.
- **Cryptocurrencies**: Cryptocurrencies, such as Bitcoin and Ethereum, use cryptography to secure transactions, prevent fraud, and maintain network integrity.
- **End-to-End Encryption**: Used in communication platforms like WhatsApp and Signal, end-to-end encryption ensures only intended recipients can read messages.
- **Online Banking**: Cryptography secures sensitive transactions and protects customers' financial information.
- **Email**: Many email providers, such as Gmail, use encryption protocols like TLS to protect messages during transmission.
- **Mobile Devices**: Smartphones and tablets use encryption to secure stored data. For example, Apple's iOS uses hardware encryption to

protect user data on iPhones and iPads.

- **Cloud Storage**: Cloud providers like Amazon Web Services use encryption algorithms, such as AES, to secure data stored in services like S3 and EBS.

- 

2. Key Features of Cryptography

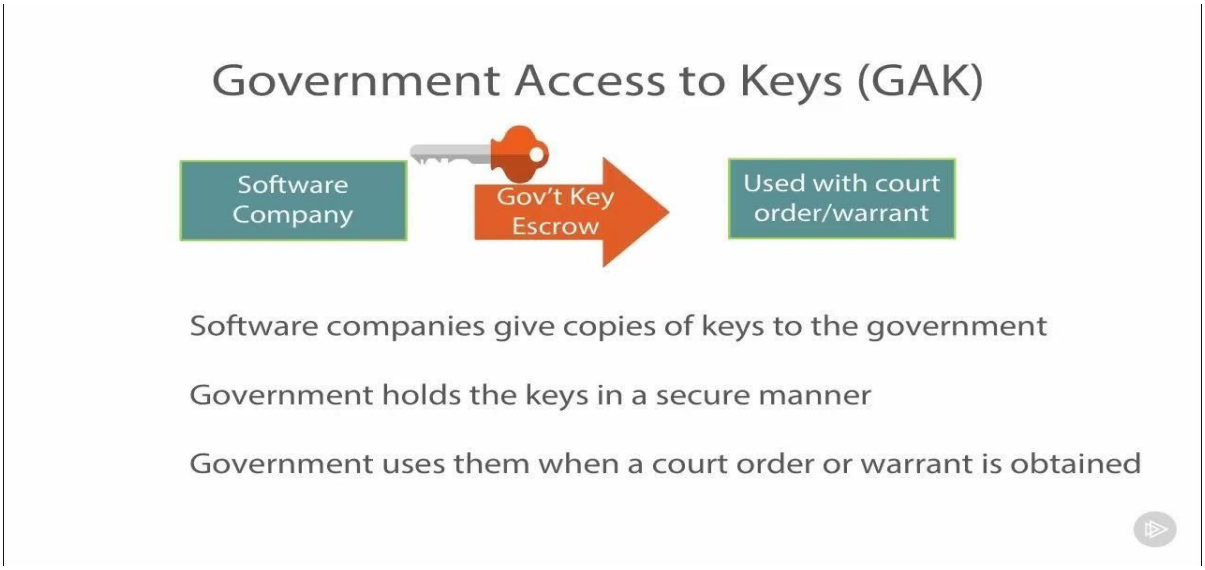
Key features that make cryptography essential in data security:

- **Confidentiality**: Ensures that data is accessible only to those authorized to access it.
- **Integrity**: Maintains data accuracy, ensuring that information remains unchanged during transmission.
- **Authentication**: Verifies the identity of the sender and receiver, establishing trust in communication.
- **Non-Repudiation**: Prevents the sender from denying they sent a message, which is essential in secure communication and transactions.
- **Availability**: Ensures authorized users have access to necessary information when needed.
- **Key Management**: Involves the secure generation, distribution, storage, and replacement of cryptographic keys.
- **Algorithm**: Mathematical formulas that govern encryption and decryption processes.
- **Encryption/Decryption**: The processes of transforming readable data to ciphertext and back to plaintext.
- **Digital Signatures**: Authenticates the sender's identity and ensures message integrity.

3. Government Access to Keys (GAK)

Government Access Keys (GAK) is a protocol that allows government authorities access to cryptographic keys in certain circumstances to enforce the law. In key escrow systems, encryption keys are stored with a trusted third party, enabling government access if required by law, such as during criminal investigations.

For example, if a company encrypts data and later goes out of business, the government may need access to the encryption keys to retrieve that information legally. This system ensures that essential data remains accessible for lawful purposes, while also balancing individual privacy rights with the need for security and accountability in national and international matters.



ADVANTAGES AND DISADVANTAGES

- **Advantages of Cryptography:**
- **Confidentiality:** Ensures that only authorized parties can understand and access the information being transmitted or stored, protecting it from unauthorized viewing.
- **Integrity:** Guarantees that data remains unchanged or unaltered during transmission or storage, preserving its accuracy and trustworthiness.
- **Authentication:** Confirms the sender's identity, ensuring messages come from verified and trusted sources, reducing the risk of tampering.
- **Data Integrity Verification:** Uses digital signatures to certify a document's authenticity and assure it remains unmodified by unauthorized users.
- **Access Control:** Enhances data confidentiality by encrypting it and restricting decryption only to privileged users.
- **Disadvantages of Cryptography:**
- **Complexity:** Cryptographic systems require significant technical knowledge and expertise to manage, which can complicate implementation and usage.
- **Key Management:** Proper handling and distribution of cryptographic keys is crucial, especially for large-scale applications, making it a challenging task to maintain securely.
- **Performance Overhead:** Encryption and decryption processes can reduce system performance, particularly in environments with limited resources.
- **Vulnerabilities:** Cryptographic algorithms may have known weaknesses or flaws that hackers could exploit, potentially compromising the entire security infrastructure.
- **Misuse:** Cryptographic techniques can be misapplied for unlawful purposes, such as encrypting malware or creating encrypted channels for criminal activities, hindering law enforcement efforts to monitor and intercept malicious actions.

Understanding Ciphers in Cryptography

What is a Cipher?

A cipher is a fundamental algorithm in cryptology, the study of cryptographic methods for securing information. It is a systematic method used for encrypting and decrypting data, converting readable text (plaintext) into an unreadable format (ciphertext) to protect the information from unauthorized access.

How Does a Cipher Work?

- **Transformation Process:** Ciphers use encryption methods to change plaintext, which is readable communication, into ciphertext, which appears to be a random string of letters. This transformation is essential for protecting sensitive information.
- **Stream and Block Ciphers:** Ciphers can operate as stream ciphers, which encrypt or decrypt data bit by bit, or as block ciphers, which process data in fixed-size blocks or units.
- **Encryption Method and Secret Key:** Modern cipher implementations alter data as it is encrypted using an encryption method and a secret key. The key length significantly impacts the cipher's resistance to brute-force attacks; longer keys generally offer stronger security.
- **Key Secrecy:** In real-world applications, the key used for encryption is kept secret. Even if someone knows the encryption method, a robust cipher requires the correct key to decrypt the ciphertext effectively.
- **Sender and Recipient Keys:** For a cipher to work, both the sender and recipient must possess the necessary key or keys. This key management is crucial for maintaining the confidentiality and integrity of the encrypted information.

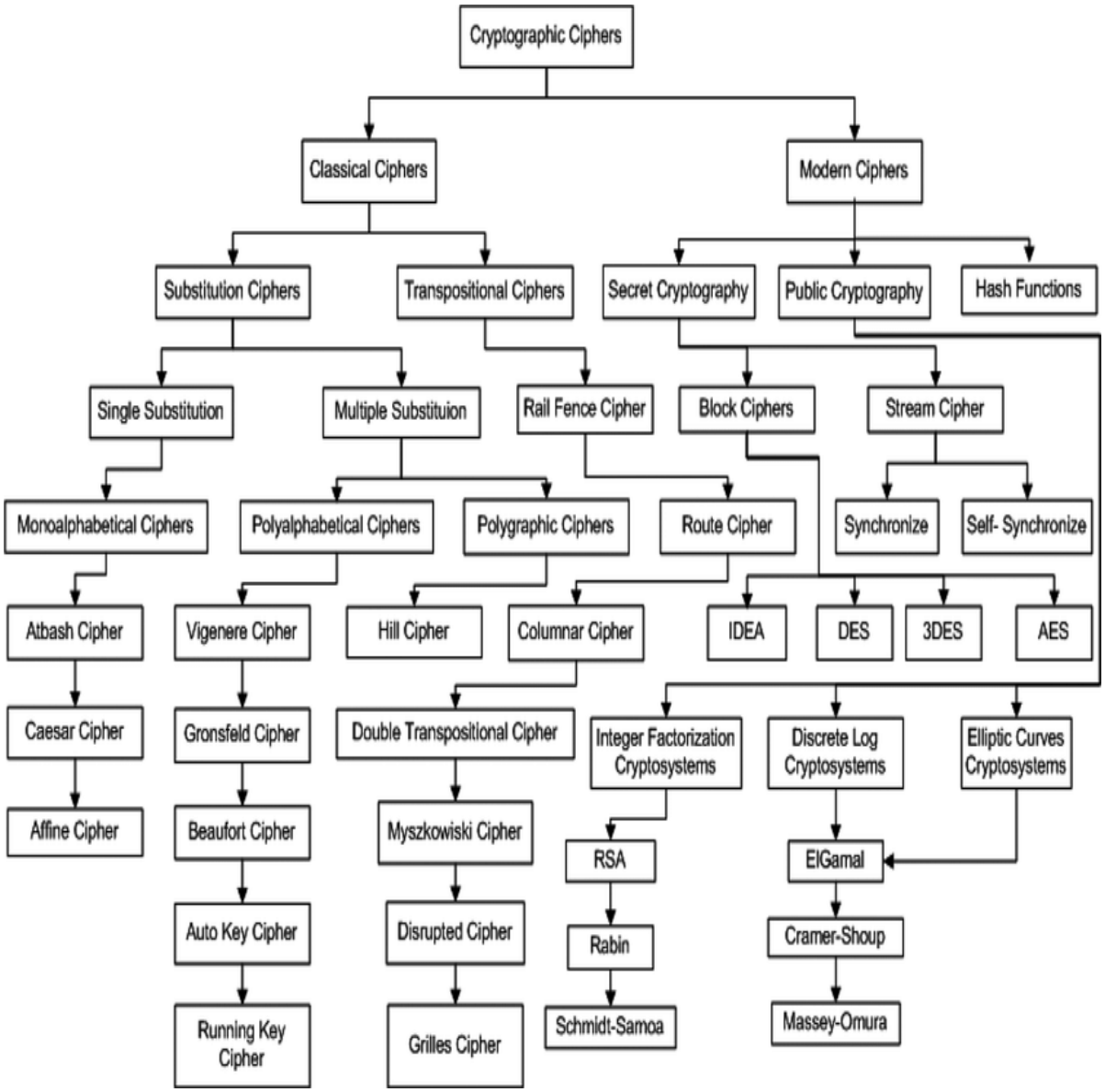
Types of Ciphers

In the context of your project, you have implemented four different types of ciphers:

1. **Caesar Cipher:** A simple substitution cipher where each letter in the plaintext is shifted by a fixed number of positions down the alphabet. For example, with a shift of 4, 'A' becomes 'E', 'B' becomes 'F', and so on.
2. **ROT13 Algorithm:** A specific case of the Caesar cipher that shifts letters by 13 positions. This method is commonly used in online forums to obscure text, allowing readers to easily decode it by applying the same

transformation.

3. **XOR Cipher:** A more complex encryption technique that combines the plaintext with a key using the XOR logical operation. This process ensures that the same key is used to both encrypt and decrypt the data, provided the key length matches the message length.
4. **Multiplicative Cipher:** A method that encrypts by multiplying each character's numeric value by a fixed key and then taking the remainder with respect to the alphabet length. This technique offers a unique approach to encryption by altering the numeric values associated with each character.
5. **Vigenère Cipher:** A method that uses a keyword to create a repeating series of Caesar ciphers based on the letters of the keyword. Each letter of the plaintext is shifted according to the corresponding letter in the keyword. This technique enhances security by combining multiple Caesar ciphers.
6. **Playfair Cipher:** An encryption technique that encrypts pairs of letters (bigrams) instead of single letters. A 5x5 grid of letters is created using a keyword, and letters are replaced according to their positions in the grid. If a pair consists of the same letter, a filler letter is added to separate them.
7. **Affine Cipher:** A substitution cipher that uses a mathematical function to encrypt characters. Each letter is first assigned a numerical value, and then a linear function is applied to produce the encrypted value, which is then converted back to a letter.
8. **RSA (Rivest-Shamir-Adleman):** A widely used asymmetric cryptographic algorithm that relies on the mathematical difficulty of factoring large prime numbers. RSA is used for secure data transmission, particularly in digital signatures and secure web browsing (HTTPS).
9. **AES (Advanced Encryption Standard):** A symmetric encryption algorithm that encrypts data in fixed-size blocks (128 bits) using keys of various lengths (128, 192, or 256 bits).
10. **DES (Data Encryption Standard):** An older symmetric key algorithm that encrypts data in 64-bit blocks using a 56-bit key. Although once widely used, it has largely been replaced by more secure algorithms like AES due to its vulnerability to brute-force attacks.


Cryptographic Ciphers
Classical Ciphers
Modern Ciphers
Substitution Ciphers
Transpositional Ciphers
Secret Cryptography
Public Cryptography
Hash Functions
Single Substitution
Multiple Substituion
Rail Fence Cipher
Block Ciphers
Stream Cipher
Monoalphabetical Ciphers
Polyalphabetical Ciphers
Polygraphic Ciphers
Route Cipher
Synchronize
Self- Synchronize
Atbash Cipher
Vigenere Cipher
Hill Cipher
Columnar Cipher
IDEA
DES
3DES
AES
Caesar Cipher
Gronsfeld Cipher
Double Transpositional Cipher
Integer Factorization Cryptosystems
Discrete Log Cryptosystems
Elliptic Curves Cryptosystems
Affine Cipher
Beaufort Cipher
Myszkowiski Cipher
RSA
ElGamal
Auto Key Cipher
Disrupted Cipher
Rabin
Cramer-Shoup
Running Key Cipher
Grilles Cipher
Schmidt-Samoa
Massey-Omura

Understanding Classical Ciphers

Classical ciphers, also known as Hand Ciphers or Pen & Paper Ciphers, refer to early encryption methods that were widely used historically but are largely obsolete today. They were foundational in cryptography, using manual or simple mechanical means for encryption and decryption.

Types of Classical Ciphers

There are two primary types of classical ciphers:

A. Simple Substitution Cipher

In a substitution cipher, each character in the plaintext is replaced by another character based on a fixed system or "key." This process shifts or substitutes the characters according to a predetermined rule. For example, using a shift of 1, the letter A would become B, B would become C, and so forth.

Algorithm for Substitution Cipher

Input:

- A string of plaintext containing both uppercase and lowercase letters.
- An integer value representing the key or shift.

Procedure:

1. **Initialize Characters**:
    - Create a list of all alphabet characters (A-Z for uppercase, a-z for lowercase).
2. **Generate Cipher Key**:
    - Create a shuffled copy of the character list to serve as the key sequence.
    - The key will define how each letter in the plaintext will be substituted in the ciphertext.
3. **Encrypt Each Character**:
    - For each character in the plaintext:
        - **Identify Position**: Locate the position of the character in the original alphabet list.
        - **Substitute Using Key**: Replace it with the character at the same position in the shuffled key list.
        - **Maintain Case**: If the character is uppercase in the plaintext, ensure the substituted character remains uppercase in the ciphertext.
4. **Return Ciphertext**:
    - After substituting all characters, combine them to form the final encrypted message.

Output:

- The encrypted string (ciphertext) created by substituting each plaintext character based on the key.

Implementing in SPYDER software:

```python
import random
import string

def generate_substitution_key():
    # Generate a shuffled alphabet to act as the substitution key
    alphabet = list(string.ascii_uppercase)
    shuffled_alphabet = alphabet.copy()
    random.shuffle(shuffled_alphabet)
    return dict(zip(alphabet, shuffled_alphabet))

def encrypt_substitution_cipher(plaintext, key):
    ciphertext = []
    for char in plaintext:
        if char.isalpha():
            # Preserve case by converting to uppercase and back if needed
            is_upper = char.isupper()
            char = char.upper()
            encrypted_char = key.get(char, char)  # Substitute the character using the key
            ciphertext.append(encrypted_char if is_upper else encrypted_char.lower())
        else:
            ciphertext.append(char)  # Keep non-alphabet characters as they are
    return ''.join(ciphertext)

# Generate a random substitution key
key = generate_substitution_key()
print("Substitution Key:", key)

# Example usage
plaintext = "Hello, World!"
ciphertext = encrypt_substitution_cipher(plaintext, key)
print("Plaintext:", plaintext)
print("Ciphertext:", ciphertext)
```

OUTPUT:

```
IPython 8.27.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: %runfile 'D:/New folder spyder/1.py' --wdir
Substitution Key: {'A': 'J', 'B': 'A', 'C': 'H', 'D': 'M', 'E': 'U', 'F': 'P', 'G': 'T', 'H': 'I',
'I': 'N', 'J': 'S', 'K': 'V', 'L': 'W', 'M': 'K', 'N': 'Z', 'O': 'F', 'P': 'E', 'Q': 'Q', 'R': 'O',
'S': 'D', 'T': 'G', 'U': 'Y', 'V': 'R', 'W': 'C', 'X': 'L', 'Y': 'X', 'Z': 'B'}
Plaintext: Hello, World!
Ciphertext: Iuwwf, Cfowm!

In [2]:
```

B. **Transposition Cipher**

Definition: The Transposition Cipher is a method of encryption where the positions of the characters in the plaintext are shifted according to a certain system, without altering the actual characters. This rearrangement of the plaintext characters results in the ciphertext. Transposition ciphers can create complex encodings by scrambling the characters in a structured manner, which only those who know the system can decrypt.

How It Works: In a transposition cipher, characters are shuffled according to a predefined system or pattern. The order of characters is altered, making it hard to decipher without the correct key. This technique doesn't change the actual characters in the message but reorders them according to the encryption rule.

Algorithm:

1. Choose the text you wish to encrypt (the plaintext).
2. Decide on a key that specifies the transposition rule (e.g., arranging characters in a matrix and reading them in a different order).
3. Write the plaintext in rows based on the chosen key.
4. Read characters column by column to form the encrypted text.
5. For decryption, arrange the characters back into the matrix using the same key and read row by row.

Example Code for Transposition Cipher:

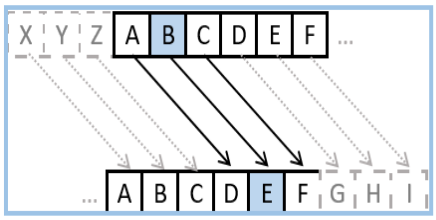
```
def encrypt_transposition(plain_text, key):
    # Create a grid with columns equal to the key length
    cipher_text = [''] * key

    for column in range(key):
        cursor = column

        # Add every key-th character to the current column in the gri
        while cursor < len(plain_text):
            cipher_text[column] += plain_text[cursor]
            cursor += key

    # Join all columns to get the final cipher text
    return ''.join(cipher_text)

def decrypt_transposition(cipher_text, key):
    # Calculate number of rows and unused boxes
    num_rows = len(cipher_text) // key
    num_extra_cells = len(cipher_text) % key

    # Fill the grid with placeholders for easier handling
    grid = [''] * key
    column_length = num_rows

    # Fill each row based on how many characters it should have
    cursor = 0
    for i in range(key):
        if i < num_extra_cells:
            column_length = num_rows + 1
        else:
            column_length = num_rows

        grid[i] = cipher_text[cursor:cursor + column_length]
        cursor += column_length

    # Read off characters in a row-by-row manner to decrypt
    decrypted_text = ''
    for i in range(num_rows + 1):
        for column in grid:
            if i < len(column):
                decrypted_text += column[i]

    return decrypted_text

# Example usage
plain_text = "HELLOTRANSPOSITION"
key = 5

cipher_text = encrypt_transposition(plain_text, key)
print("Encrypted Text:", cipher_text)

decrypted_text = decrypt_transposition(cipher_text, key)
print("Decrypted Text:", decrypted_text)
```

OUTPUT:

```
In [2]: %runfile 'D:/New folder spyder/1.py' --wdir
Encrypted Text: HTPIEROOLASNLNIOST
Decrypted Text: HELLOTRANSPOSITION
```

Caesar Cipher

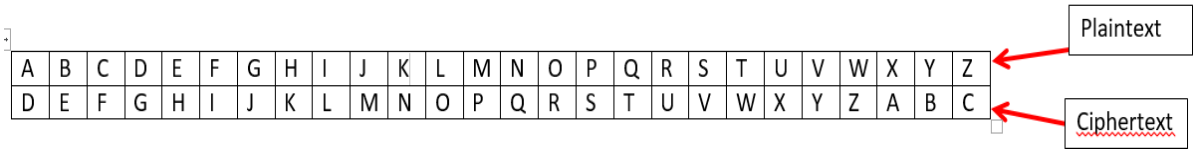
Introduction: The Caesar Cipher is one of the oldest and simplest forms of encryption. It's a type of substitution cipher where each letter in the plaintext is "shifted" a certain number of places down the alphabet. Named after Julius Caesar, who reportedly used it to communicate with his generals, this cipher uses a fixed shift key to scramble text, making it readable only to those who know the shift.



SHIFT +3

This Caesar cipher has a shift of 3 characters.

The letter 'A' becomes a 'D'. The letter 'B' becomes 'E'.



How It Works:

- The cipher shifts each letter in the plaintext by a fixed number of positions.
- For example, if the shift is `3`, then `A` becomes `D`, `B` becomes `E`, and so on.
- The recipient reverses the shift to decrypt the message.

Applications of Caesar Cipher:

- Today, it's mainly used as a teaching tool for understanding encryption, though variations of substitution ciphers are still applied in more advanced cryptographic algorithms.

Algorithm for Caesar Cipher

Input: A string of uppercase or lowercase letters, called plaintext, and an integer called the key (shift amount).

Procedure:

1. For each character in the plaintext:
    - Check if the character is an uppercase or lowercase letter.
    - Shift the letter forward by the key amount within the alphabet.
    - If the shift goes past `Z` (for uppercase) or `z` (for lowercase), wrap around to the beginning of the alphabet.
2. The resulting shifted letters form the ciphertext.

Output: The ciphertext obtained after shifting each letter by the key value.

Encryption Code for Caesar Cipher

Here is the code to encrypt plaintext using a Caesar Cipher. This code handles both uppercase and lowercase letters.

```python
def caesar_encrypt(plaintext, key):
    ciphertext = ""
    for char in plaintext:
        # Check if character is an uppercase letter
        if char.isupper():
            ciphertext += chr((ord(char) + key - 65) % 26 + 65)
        # Check if character is a lowercase letter
        elif char.islower():
            ciphertext += chr((ord(char) + key - 97) % 26 + 97)
        else:
            ciphertext += char  # Non-alphabet characters are added unchanged
    return ciphertext

# Example usage
plaintext = "HelloWorld"
key = 3
ciphertext = caesar_encrypt(plaintext, key)
print("Ciphertext:", ciphertext)
```

OUTPUT:

```
IPython 8.27.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: %runfile 'D:/New folder spyder/1.py' --wdir
Ciphertext: KhoorZruog
```

Explanation of the Code:

- The function `caesar_encrypt` takes a `plaintext` string and a `key` integer.
- It checks each character to see if it's uppercase, lowercase, or non-alphabetic.
- For uppercase letters, it adjusts the ASCII code so that `A` is 0, then applies the shift within the 26 letters.
- For lowercase letters, it does the same but adjusts `a` to 0.
- Non-alphabet characters remain unchanged.

Example Output: If `plaintext` is `"HelloWorld"` and `key` is `3`, the output will be:

Ciphertext: KhoorZruog

Decryption Code for Caesar Cipher

To decrypt, we simply reverse the encryption by shifting each character back by the key amount.

```python
def caesar_decrypt(ciphertext, key):
    plaintext = ""
    for char in ciphertext:
        # Check if character is an uppercase letter
        if char.isupper():
            plaintext += chr((ord(char) - key - 65) % 26 + 65)
        # Check if character is a lowercase letter
        elif char.islower():
            plaintext += chr((ord(char) - key - 97) % 26 + 97)
        else:
            plaintext += char  # Non-alphabet characters are added unchanged
    return plaintext

# Example usage
ciphertext = "KhoorZruog"
key = 3
decrypted_text = caesar_decrypt(ciphertext, key)
print("Decrypted Text:", decrypted_text)
```

OUTPUT:

```
In [4]: %runfile 'D:/New folder spyder/1.py' --wdir
Decrypted Text: HelloWorld
```

Explanation of the Decryption Code:

- The function caesar_decrypt takes ciphertext and key.
- It shifts each character back by the key using a similar approach as in the encryption function.
- Non-alphabet characters are added directly to the output without modification.

Example Output: If ciphertext is "KhoorZruog" and key is 3, the decrypted output will be:

Decrypted Text: HelloWorld

Advantages of Caesar Cipher

1. **Simplicity**: The Caesar Cipher is easy to understand and implement, making it an excellent tool for teaching the basics of encryption.
2. **Speed**: Due to its straightforward nature, Caesar Cipher encryption and decryption are fast and require minimal computational resources, ideal for environments where efficiency is needed.
3. **Low Resource Requirement**: Caesar Cipher only requires basic alphabetic shifting, meaning it doesn't need complex algorithms or large amounts of storage.
4. **Historical Importance**: The Caesar Cipher played a foundational role in the history of cryptography, helping to develop more advanced encryption methods.

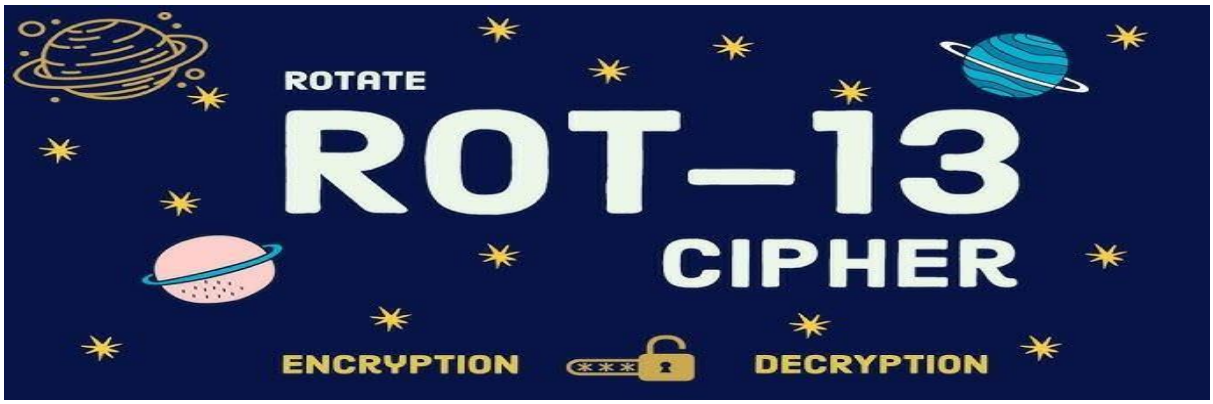
Disadvantages of Caesar Cipher

1. **Weak Security**: The Caesar Cipher is vulnerable to brute-force attacks, as there are only 25 possible shifts. This makes it extremely easy to break, even for beginners.
2. **Limited Key Space**: With a key space of only 26 (for the English alphabet), Caesar Cipher is not suitable for secure communications because all possible keys can be easily tried.
3. **Predictable Pattern**: Because the cipher only shifts letters in a predictable way, frequency analysis (analyzing the occurrence of letters) can quickly reveal the key, especially in languages like English where certain letters are more common.
4. **Not Suitable for Digital Data**: The Caesar Cipher only works on alphabetical characters, making it ineffective for encrypting digital data (such as binary files, numbers, or symbols) without extensive modifications.

ROT13 Algorithm

Introduction: The ROT13 (rotate by 13 places) algorithm is a special case of the Caesar Cipher, using a fixed shift of 13. This cipher is useful because it encrypts and decrypts text using the same process, making it a "self-inverse" cipher. It's often used for simple obfuscation rather than true encryption.

s



How It Works:

- Each letter in the plaintext is replaced by the letter 13 places after it in the alphabet.
- For example, A becomes N, B becomes O, and so on. After reaching Z, it wraps back to A.
- Since the alphabet has 26 letters, applying ROT13 twice returns the original text.

Applications of ROT13:

- Primarily used for obscuring text, like hiding spoilers in forums or simple coding exercises.
- Not suitable for secure encryption due to its simplicity.

Algorithm for ROT13

Input: A string (plaintext).

Procedure:

1. For each character in the plaintext:
    - Check if it's an uppercase or lowercase letter.
    - Shift the letter by 13 positions in the alphabet.
    - Wrap around to the start of the alphabet if the shift goes beyond Z or z.
2. Output the modified string as the ciphertext.

Output: The encoded/decoded text.

Code for ROT13 Algorithm

ROT13 applies the same function for both encryption and decryption.

```python
def rot13(text):
    result = ""
    for char in text:
        if char.isupper():
            result += chr((ord(char) - 65 + 13) % 26 + 65)
        elif char.islower():
            result += chr((ord(char) - 97 + 13) % 26 + 97)
        else:
            result += char
    return result

# Example usage
plaintext = "good day"
ciphertext = rot13(plaintext)
print("Ciphertext:", ciphertext)
print("Decrypted Text:", rot13(ciphertext))  # Applying ROT13 again to decrypt

```

```
In [6]: %runfile 'D:/New folder spyder/1.py' --wdir
Ciphertext: tbbq qnl
Decrypted Text: good day
```

Explanation of Code:

- This function takes a text string and processes each character.
- It shifts uppercase and lowercase letters by 13 positions.
- Non-alphabet characters are appended to the result unchanged.
- Reapplying ROT13 on the output restores the original text.

Example Output: If plaintext is "HelloWorld", the output will be:

Ciphertext: tbbq qnl

Decrypted Text: good day

Advantages of ROT13

1. **Simple to Use**: ROT13 is easy to understand and implement, requiring only a 13-character shift.
2. **Reversible Encryption**: Applying ROT13 twice returns the original text, making it convenient for basic obfuscation.
3. **Useful for Informal Obfuscation**: Often used in forums to hide spoilers or answers.
4. **No Key Management**: Does not require a key, simplifying its use in non-sensitive scenarios.

Disadvantages of ROT13

1. **Weak Security**: ROT13 is easy to break and provides minimal security.
2. **Susceptible to Frequency Analysis**: Letter frequencies remain intact, making it easy to analyze and decipher.
3. **Limited to Alphabetic Data**: Only works on alphabetic characters without further adaptation.
4. **Fixed Key**: Uses a fixed 13-character shift, offering no flexibility in key choice.

XOR Cipher

Introduction: The XOR Cipher is a symmetric key cipher that uses the XOR (exclusive or) logical operation for encryption and decryption. This cipher is extremely simple and works well if the key length matches the message length.

How It Works:

- Each bit of the plaintext is XORed with the corresponding bit of the key.
- Since XORing twice with the same key reverts to the original bit, it makes decryption as simple as encryption.

Applications of XOR Cipher:

- Used in low-level encryption for simple data.
- Often employed as a building block in more complex encryption systems.

Algorithm for XOR Cipher

Input: A binary string (plaintext) and a binary string (key).

Procedure:

1. Convert each character to binary.
2. XOR each bit of the plaintext with the corresponding bit of the key.
3. Convert the result back to a string.

Output: The ciphertext after XORing.

Code for XOR Cipher

This example assumes the key length matches the plaintext length.

```
def xor_encrypt_decrypt(text, key):
    result = ""
    for i in range(len(text)):
        result += chr(ord(text[i]) ^ ord(key[i % len(key)]))
    return result

# Example usage
plaintext = "HelloWorld"
key = "mysecretkey"
ciphertext = xor_encrypt_decrypt(plaintext, key)
print("Ciphertext:", ciphertext)
print("Decrypted Text:", xor_encrypt_decrypt(ciphertext, key))
```

OUTPUT:

```
Ciphertext: %└▼ %
↑0
Decrypted Text: HelloWorld
```

Explanation of Code:

- Each character in the text is XORed with a character from the key.
- The key [i % len(key)] ensures the key wraps around if it's shorter than the plaintext.

Example Output: If plaintext is "HelloWorld" and key is "mysecretkey", the output will vary due to the binary XOR operation.

Advantages of XOR Cipher

1. **Simplicity**: XOR is straightforward to implement and understand, ideal for beginner-level encryption concepts.
2. **Efficient Processing**: XOR operations are fast, allowing quick encryption and decryption even for large data sets.
3. **Perfect Secrecy with One-Time Pad**: When used with a random key equal in length to the plaintext (one-time pad), XOR provides theoretically unbreakable encryption.
4. **Symmetric Key**: The same XOR operation is used for both encryption and decryption, simplifying the process.

Disadvantages of XOR Cipher

1. **Weak Security with Repeated Keys**: Reusing a key for multiple messages makes it vulnerable to attacks, as patterns emerge.
2. **No Built-in Key Management**: XOR requires secure key storage and transmission, which can be challenging in practice.
3. **Limited to Binary Operations**: XOR operates at the binary level, making it less adaptable for more complex encryption requirements.
4. **Easily Broken without One-Time Pad**: When not used with a one-time pad, XOR encryption can be easily cracked by simple analysis techniques.

Multiplicative Cipher

Introduction

The **Multiplicative Cipher** is a type of substitution cipher, where each letter in the plaintext is encrypted by multiplying it by a fixed integer, called the key, modulo the size of the alphabet. This cipher belongs to classical cryptographic techniques, and it's also known as the **Multiplicative Caesar Cipher**.

In English, where the alphabet has 26 letters, the mathematical operation is done modulo 26. Importantly, the key used for encryption must be a number that is coprime with 26 (i.e., it shares no common factors with 26 other than 1) to ensure that each character has a unique encryption.

Encryption Process

The encryption formula for the Multiplicative Cipher is:

Ciphertext=(Plaintext×Key)mod 26\text{Ciphertext} = (\text{Plaintext} \times \text{Key}) \mod 26Ciphertext=(Plaintext×Key)mod26

1. Choose a key that is coprime with 26 (e.g., 3, 5, 7, etc.).
2. Convert each character in the plaintext to its numeric position (A = 0, B = 1, … Z = 25).
3. Multiply each numeric value by the key and take modulo 26.
4. Convert the resulting values back to letters to form the ciphertext.

Decryption Process

To decrypt the ciphertext, you need the **multiplicative inverse** of the encryption key modulo 26. The multiplicative inverse k−1k^{-1}k−1 of a key kkk is a number such that:

(k×k−1)mod 26=1(k \times k^{-1}) \mod 26 = 1(k×k−1)mod26=1

The decryption formula is:

Plaintext=(Ciphertext×k−1)mod 26\text{Plaintext} = (\text{Ciphertext} \times k^{-1}) \mod 26Plaintext=(Ciphertext×k−1)mod26

1. Find the multiplicative inverse of the encryption key.
2. Convert each letter in the ciphertext to its numeric equivalent.
3. Multiply each number by the multiplicative inverse, then take modulo 26.
4. Convert the result back to letters to get the plaintext.

Algorithm for Multiplicative Cipher

1. **Input**:
    - Plaintext (string of uppercase letters).
    - Key (an integer coprime with 26).
2. **Encryption Steps**:
    - For each character in the plaintext:
        - Convert to its numeric position.
        - Apply the encryption formula.
        - Convert back to a letter.
3. **Decryption Steps**:
    - Calculate the multiplicative inverse of the key.
    - For each character in the ciphertext:
        - Convert to its numeric position.
        - Apply the decryption formula.
        - Convert back to a letter.
4. **Output**:
    - The encrypted ciphertext or decrypted plaintext.

Python Code for Multiplicative Cipher

Here is the Python code for the Multiplicative Cipher, with functions for encryption and decryption.

```python
# Function to compute the Greatest Common Divisor (GCD)
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

# Function to find the Multiplicative Inverse of a key under modulo 26
def multiplicative_inverse(key, mod):
    for i in range(1, mod):
        if (key * i) % mod == 1:
            return i
    return None

# Encryption Function for Multiplicative Cipher
def encrypt_multiplicative(plaintext, key):
    if gcd(key, 26) != 1:
        return "Invalid key! Key must be coprime with 26."

    ciphertext = ""
    for char in plaintext:
        if char.isalpha():  # Only process alphabetic characters
            char_num = ord(char.upper()) - ord('A')
            encrypted_num = (char_num * key) % 26
            ciphertext += chr(encrypted_num + ord('A'))
        else:
            ciphertext += char  # Non-alphabet characters remain unchanged
    return ciphertext

# Decryption Function for Multiplicative Cipher
def decrypt_multiplicative(ciphertext, key):
    key_inverse = multiplicative_inverse(key, 26)
    if key_inverse is None:
        return "Invalid key! No multiplicative inverse exists for this key."

    plaintext = ""
    for char in ciphertext:
        if char.isalpha():  # Only process alphabetic characters
            char_num = ord(char.upper()) - ord('A')
            decrypted_num = (char_num * key_inverse) % 26
            plaintext += chr(decrypted_num + ord('A'))
        else:
            plaintext += char  # Non-alphabet characters remain unchanged
    return plaintext

# Example usage
plaintext = "HELLO"
key = 5

encrypted_text = encrypt_multiplicative(plaintext, key)
print("Encrypted Text:", encrypted_text)

decrypted_text = decrypt_multiplicative(encrypted_text, key)
print("Decrypted Text:", decrypted_text)
```

Example Walkthrough

Plaintext: HELLO
Key: 5

1. Convert each letter of "HELLO" to its numeric position: H = 7, E = 4, L = 11, L = 11, O = 14.
2. Encrypt each number using the key:
    - H (7) -> (7 × 5) % 26 = 9 → J
    - E (4) -> (4 × 5) % 26 = 20 → U
    - L (11) -> (11 × 5) % 26 = 5 → F
    - L (11) -> (11 × 5) % 26 = 5 → F
    - O (14) -> (14 × 5) % 26 = 18 → S
3. **Ciphertext**: JUFFS

Advantages:

1. Simple to Implement: Straightforward calculations make it easy to understand and code.
2. Moderate Security for Short Texts: Provides basic encryption for short messages, suitable for educational or low-stakes scenarios.
3. Efficient Processing: Mathematical operations are fast, making it efficient for small data.
4. Useful for Cryptography Education: Helps learners understand concepts like modular arithmetic and multiplicative inverses.

Disadvantages:

1. Limited Key Options: Only numbers coprime with 26 are valid, reducing flexibility.
2. Vulnerable to Frequency Analysis: Character frequencies are unchanged, making it easy to analyze patterns.
3. Not Secure for Long Texts: With larger messages, patterns are more obvious, reducing security.
4. Difficulty with Key Management: Finding a valid key and its inverse can be complex and is impractical for modern security needs.

Combined Cipher Implementation

Objective:

This program demonstrates the implementation of multiple encryption and decryption techniques: **Caesar Cipher**, **ROT13 Cipher**, **XOR Cipher**, and **Multiplicative Cipher**. It allows the user to interactively choose between different cipher methods, making it versatile and user-friendly.

Purpose:

The purpose of combining these ciphers in a single program is to showcase the differences in their mechanisms and the flexibility of using multiple algorithms in real-world applications. This interactive tool helps understand:

- How each cipher encrypts and decrypts data.
- The importance of cryptographic keys, including the concept of **coprime numbers** in the Multiplicative Cipher.
- The relevance of encryption in ensuring data security.

Features of the Program:

1. **User-Friendly Interface**:
    - The program allows the user to select their desired cipher method by pressing a number corresponding to their choice.
    - Options for encryption and decryption are clearly outlined for each cipher.

2. **Coprime Validation**:
    - The Multiplicative Cipher requires the key to be coprime with 262626. The program automatically checks this condition to avoid errors.

3. **Multi-Algorithm Support**:
    - **Caesar Cipher**: A basic substitution cipher with a user-defined shift key.
    - **ROT13 Cipher**: A variant of Caesar Cipher with a fixed shift of 13.
    - **XOR Cipher**: Uses a repeating key for bitwise encryption and decryption.
    - **Multiplicative Cipher**: Encrypts data by multiplying character values with a key that is coprime to 262626, with support for modular inverse decryption.
4. **Robust Input Validation**:
    - The program provides feedback for invalid input, ensuring a smooth user experience.

5. **Educational Value**:
    - This program serves as a practical demonstration of classical and basic cryptographic techniques, aiding understanding of fundamental cryptographic principles.

Code Implementation

Below is the Python code for the program that implements all the above features. Users can interact with the program to see how different ciphers work for both encryption and decryption.

```python
# Import necessary library
import sys

# Function to check if two numbers are coprime
def gcd(a, b):
    while b:
        a, b = b, a % b
    return a

def is_coprime(a, b):
    return gcd(a, b) == 1

# Caesar Cipher
def caesar_cipher_encrypt(text, key):
    result = ""
    for char in text:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char) - base + key) % 26 + base)
        else:
            result += char
    return result

def caesar_cipher_decrypt(text, key):
    return caesar_cipher_encrypt(text, -key)

# ROT13 Cipher
def rot13_cipher(text):
    return caesar_cipher_encrypt(text, 13)

# XOR Cipher
def xor_cipher(text, key):
    result = ''.join(chr(ord(char) ^ ord(key[i % len(key)])) for i, char in enumerate(text))
    return result

# Multiplicative Cipher
def multiplicative_cipher_encrypt(text, key):
    encrypted_text = ""
```

```
    for char in text:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            encrypted_text += chr(((ord(char) - base) * key) % 26 + base)
        else:
            encrypted_text += char
    return encrypted_text

def multiplicative_cipher_decrypt(text, key):
    # Find modular inverse of the key
    for i in range(1, 26):
        if (key * i) % 26 == 1:
            mod_inverse = i
            break
    else:
        return "Decryption not possible (no modular inverse found for the key)."

    decrypted_text = ""
    for char in text:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            decrypted_text += chr(((ord(char) - base) * mod_inverse) % 26 + base)
        else:
            decrypted_text += char
    return decrypted_text

# Main menu
while True:
    print("\nSelect an option:")
    print("1. Caesar Cipher")
    print("2. ROT13 Cipher")
    print("3. XOR Cipher")
    print("4. Multiplicative Cipher")
    print("5. Exit")

    choice = input("Enter your choice (1-5): ")
```

```
    if choice == '1':
        text = input("Enter the text: ")
        key = int(input("Enter the key (shift): "))
        mode = input("Choose mode (encrypt/decrypt): ").strip().lower()
        if mode == 'encrypt':
            print("Encrypted Text:", caesar_cipher_encrypt(text, key))
        elif mode == 'decrypt':
            print("Decrypted Text:", caesar_cipher_decrypt(text, key))
        else:
            print("Invalid mode.")
    elif choice == '2':
        text = input("Enter the text: ")
        print("Encrypted/Decrypted Text:", rot13_cipher(text))
    elif choice == '3':
        text = input("Enter the text: ")
        key = input("Enter the key (string): ")
        print("Encrypted/Decrypted Text:", xor_cipher(text, key))
    elif choice == '4':
        text = input("Enter the text: ")
        key = int(input("Enter the key (must be coprime with 26): "))
        if not is_coprime(key, 26):
            print(f"The key {key} is not coprime with 26. Please choose another key.")
            continue
        mode = input("Choose mode (encrypt/decrypt): ").strip().lower()
        if mode == 'encrypt':
            print("Encrypted Text:", multiplicative_cipher_encrypt(text, key))
        elif mode == 'decrypt':
            print("Decrypted Text:", multiplicative_cipher_decrypt(text, key))
        else:
            print("Invalid mode.")
    elif choice == '5':
        print("Exiting the program.")
        sys.exit()
    else:
        print("Invalid choice. Please select again.")
```

OUTPUT:

```
In [1]: %runfile 'D:/New folder spyder/1.py' --wdir

Select an option:
1. Caesar Cipher
2. ROT13 Cipher
3. XOR Cipher
4. Multiplicative Cipher
5. Exit
Enter your choice (1-5): 1
Enter the text: hello world
Enter the key (shift): 3
Choose mode (encrypt/decrypt): encrypt
Encrypted Text: khoor zruog
```

```
Select an option:
1. Caesar Cipher
2. ROT13 Cipher
3. XOR Cipher
4. Multiplicative Cipher
5. Exit
Enter your choice (1-5): 1
Enter the text: khoor zruog
Enter the key (shift): 3
Choose mode (encrypt/decrypt): decrypt
Decrypted Text: hello world
```

Introduction to Online Cryptographic Tools

In the digital age, online tools have become an essential way to understand, experiment with, and implement cryptographic concepts. These tools provide a convenient and interactive platform for performing encryption and decryption without the need for complex setups or programming expertise.

Online cryptographic tools are particularly useful for:

1. **Learning and Demonstration**: They help users understand how encryption algorithms work by providing immediate feedback.
2. **Quick Implementation**: For scenarios where simple encryption is needed, such tools save time by offering ready-to-use functionality.
3. **Comparison and Testing**: Users can compare outputs from various ciphers and validate their own implementations.

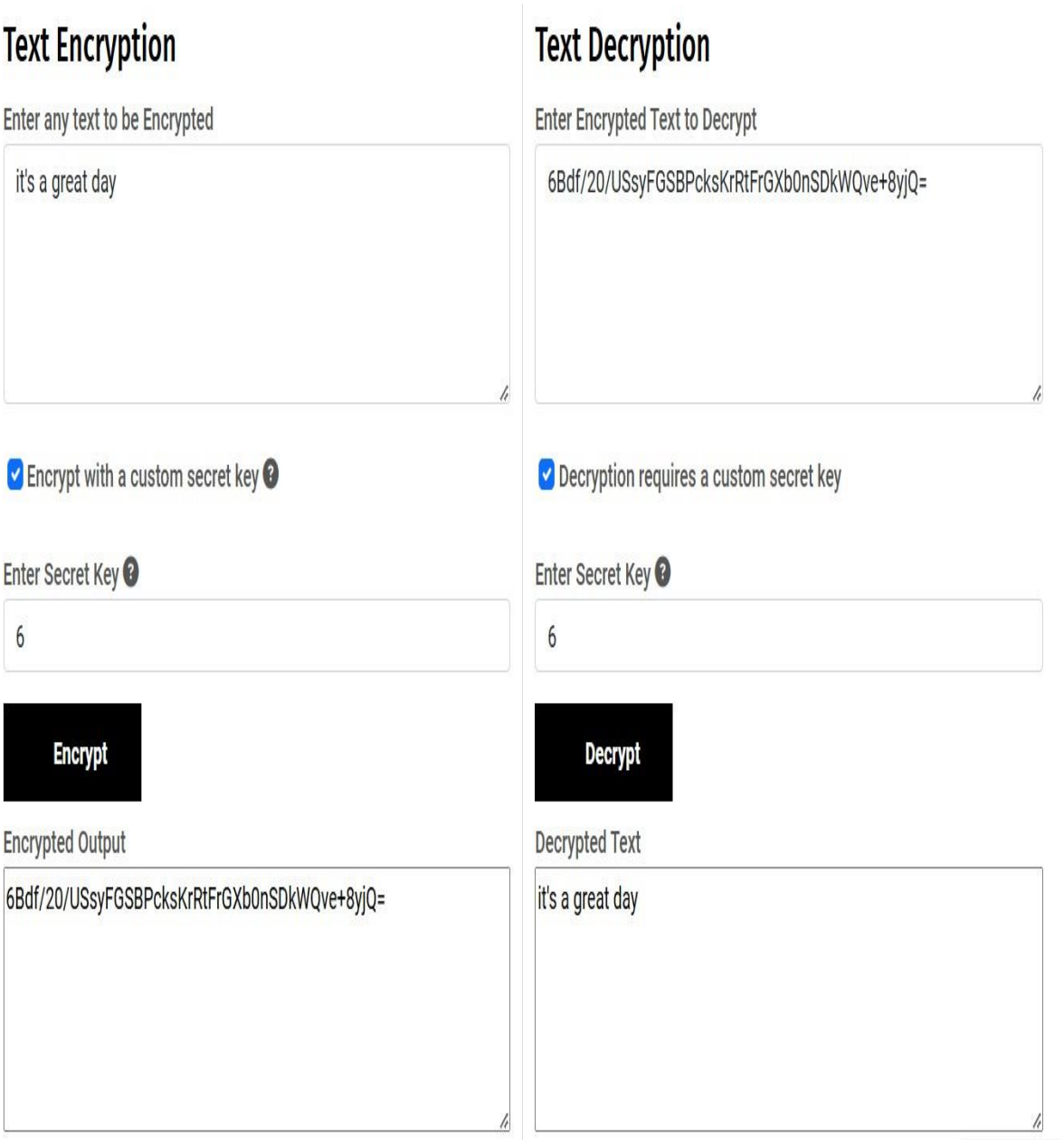
In this project, we explore the theoretical and practical aspects of cryptography by combining:

- **Custom Implementations**: Using Python to build encryption and decryption algorithms for popular ciphers.
- **Online Tools**: Demonstrating their use to understand and validate the working of different ciphers.

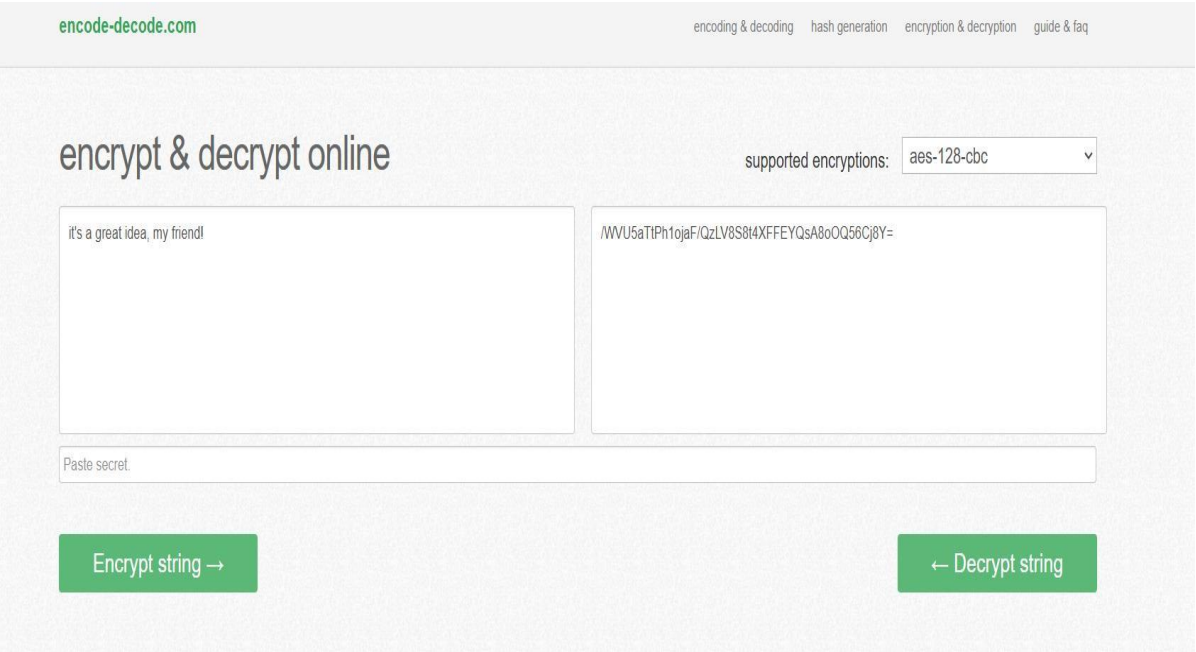
Some of the **online encryption and decryption tools** included in this project are:

These tools serve as a complement to the custom implementations, providing insights into the broader cryptographic landscape.

- **https://www.devglan.com/online-tools/text-encryption-decryption**



- **https://encode-decode.com/encryption-functions/**



Limitations of the Project

This project highlights classical encryption techniques, but it has certain limitations that restrict its scope and real-world applicability:

1. **Focus on Classical Ciphers:**

   The project includes classical ciphers like Caesar, ROT13, XOR, Multiplicative, and Transposition Cipher. While educational, these methods are outdated and not suitable for modern secure communications.

2. **Lack of Modern Standards:**

   The project does not implement modern cryptographic algorithms like AES or RSA, which are essential for current secure systems.

3. **Key Management Challenges:**

   Key handling is manual and lacks secure key exchange protocols like those used in real-world systems (e.g., Diffie-Hellman).

4. **Vulnerability to Attacks:**
   - Caesar Cipher and ROT13 are easily broken using brute force or frequency analysis.
   - XOR Cipher is secure only with a perfect random key.
   - Multiplicative Cipher is mathematically weak and limited by the need for keys coprime with 26.
   - Transposition Cipher can be decrypted using pattern recognition.

5. **No Authentication or Integrity Checks:**

   This project focuses on encryption and decryption but does not ensure data integrity or authenticate the sender, essential aspects of modern cryptography.

6. **Limited Usability:**

   The system is designed for small text inputs and relies on user inputs, which may introduce errors. It is not scalable for large datasets or files.

Bibliography

Web Resources:

- **TutorialsPoint. "Cryptography Overview." https://www.tutorialspoint.com**

Software and Tools:

- **Python Programming Language: https://www.python.org**
- **Spyder software : https://docs.spyder-ide.org/current**

Online Articles and Tutorials:

- **Cloudflare. "The History and Evolution of Cryptography." https://www.cloudflare.com**