Class Learning Outcomes

Students will learn about the Substitution Ciphers including:

- ✓ Hill Cipher
- ✓ PlayFair Cipher
- ✓ Homophonic Cipher
- ✓ ROT13
- ✓ Monoalphabetic Cipher
  - ➢ Additive Cipher (Shift Cipher)
  - ➢ Caeser Cipher
  - ➢ Multiplicative Cipher
  - ➢ Affine Cipher

Hill cipher

AIM:

To develop a program to encrypt and decrypt using the Hill cipher substitution technique

PRELAB DISCUSSION:

The Hill cipher is a substitution cipher invented by Lester S. Hill in 1929. Hill's major contribution was the use of mathematics to design and analyse cryptosystems. The Hill cipher is a polygraphic substitution cipher based on linear algebra. Hill ciphers are applications of linear algebra because a Hill cipher is simply a linear transformation represented by a matrix with respect to the standard basis. Groups of letters are represented by vectors. The domain of the linear transformation is all plaintext vectors, while the codomain is made up of all ciphertext vectors. Matrix multiplication is involved in the encoding and decoding process. And when trying to find the inverse key, we will use elementary row operations to row reduce the key matrix in order to find its inverse in the standard manner. Each letter is represented by a number modulo 26. To encrypt a message, each block of n letters is multiplied by an invertible n × n matrix, again modulus 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption. The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n × n matrices (modulo 26).The cipher can, be adapted to an alphabet with any number of letters. All arithmetic just needs to be done modulo the number of letters instead of modulo 26.

Encryption: Cipher text = (Plain text * Key) mod 26

Decryption: Plain text = (Cipher text * Key-1) mod 26.

ALGORITHM:

1. Get the n-by-n key matrix with vectors of length n.

2. Separate the plaintext from left to right into some number k of groups of n letters each. If you run out of letters when forming the final group, repeat the last plaintext letter or 'X' as many times as needed to fill out the final group of n letters.
3. Replace each letter by the corresponding number of its position (from 0 through m-1) in the alphabet to get k groups of n integers each.
4. Encryption:
    a. Reshape each of the k groups of integers into an n-row column vector called Plain text matrix
    b. Cipher Text Matrix = (Plain Text Matrix * Key Matrix) mod 26.
    c. Using step 4.b, perform matrix multiplication of Plain text matrix and Key matrix and modulus 26 to yield Cipher text matrix.
    d. Translate the Cipher text matrix to string representation by replacing each of the entries with the corresponding letter of the alphabet, after arranging all k of the resulting column vectors in order into a single vector of length k x n
5. Decryption:
    a. Find the inverse of the key matrix
        i. Find the determinant of the key matrix
        ii. Transpose the key matrix
        iii. Find minor matrix and then Cofactor of the key matrix iv. Key-1 = [[Det(key matrix)] -1 * Cofactor] mod 26 using modulus arithmetic
    b. Plain Text = (Cipher Text * Key-1) mod 26
    c. Using step 5.b, perform matrix multiplication of Cipher text matrix and Key inverse matrix and modulus 26 to yield Plain text matrix.
    d. Translate the Plain text matrix to string representation by replacing each of the entries with the corresponding letter of the alphabet, after arranging all k of the resulting column vectors and removing any letters padded in order into a single vector of length k x n
6. Stop

Tasks:

1. Implement a 2x2 Hill Cipher Encryption & Decryption

- Write a program that takes a **2x2 key matrix** and encrypts/decrypts a text.
- Ensure the key matrix is **invertible (mod 26)** for decryption.

2. Extend to a 3x3 Hill Cipher

- Modify your implementation to support a **3x3 key matrix** for stronger encryption.
- Implement modular matrix inversion for decryption.

PlayFair Cipher

ALGORITHM:

1. Generate the (5 x 5) key table or matrix using the key.
    a. Fill the spaces in the table with the letters of the key without any duplication of letters.
    b. Fill the remaining spaces with the rest of the letters of the alphabet in order by having 'I' & 'J' in the same space or omitting 'Q' to reduce the alphabet to fit.

2. Remove any punctuation or characters from the plain text that are not present in the key square.
3. Identify any double letters in the plaintext and insert ‘X’ between the two occurrences.
4. Split the plain text into digraphs (groups of 2 letters)
5. Encryption: Locate the digraph letters in the key table
    a. If the letters appear on the same row of the table, replace them with the letters to their immediate right respectively (wrapping around to the left side of the row if a letter in the original pair was on the right side of the row).
    b. If the letters appear on the same column of the table, replace them with the letters immediately below respectively (wrapping around to the top side of the column if a letter in the original pair was on the bottom side of the column).
    c. If the letters are in different rows and columns, replace the pair with the letters on the same row respectively but at the other pair of corners of the rectangle defined by the original pair. The order is important – the first encrypted letter of the pair is the one that lies on the same row as the first plaintext letter.
    d. Append the letters referred from the key table using the steps 5.a, 5.b and 5.c to generate Cipher text.
6. Decryption: Split the cipher text into digraphs and locate the digraph letters in the key table.
    a. If the letters appear on the same row of the table, replace them with the letters to their immediate left respectively (wrapping around to the right side of the row if a letter in the original pair was on the left side of the row).
    b. If the letters appear on the same column of the table, replace them with the letters immediately above respectively (wrapping around to the bottom side of the column if a letter in the original pair was on the top side of the column).
    c. If the letters are in different rows and columns, replace the pair with the letters on the same row respectively but at the other pair of corners of the rectangle defined by the original pair. The order is important – the first encrypted letter of the pair is the one that lies on the same row as the first plaintext letter.
    d. Append the letters referred from the key table using the steps 6.a, 6.b and 6.c to generate Plain text.
7. Stop

1. Implement a Standard Playfair Cipher (5x5 Grid)

- Encrypt and decrypt messages using a **given keyword**.
- Treat **"I" and "J" as the same letter**.

2. Automate Key Square Generation

- Allow users to enter any keyword.
- Generate the 5×5 key square dynamically.

Homophonic cipher

A homophonic cipher is a substitution cipher in which each plaintext letter is replaced with one of several possible ciphertext symbols. This makes frequency analysis more difficult compared to simple substitution ciphers because multiple symbols can represent the same letter, reducing letter frequency patterns.

How It Works

1. **Mapping Multiple Ciphertext Symbols to One Plaintext Letter**
    - Each letter in the plaintext has multiple possible ciphertext symbols.
    - For example:
      A → 12, 45, 78
      B → 34, 89
      C → 56, 90, 23

2. **Random Selection**
    - When encrypting, a symbol is randomly selected from the list assigned to the plaintext letter.
    - When decrypting, the reverse mapping is used.

Task: 1

Create a Homophonic Mapping Table

- Define a mapping where each letter (A-Z) is replaced with multiple possible numbers or symbols.

Implement Encryption

- Take a plaintext input and replace each letter with a **randomly selected** ciphertext value from the mapping table.

Implement Decryption

- Reverse the encryption process using the mapping table to recover the original text.

Optimize Random Selection

- Ensure randomness in choosing ciphertext symbols while encrypting.
- Use **random number generation** techniques effectively.

Task: 2

Dynamic Homophonic Cipher Generator

Instead of a fixed mapping, create a cipher that **dynamically** generates homophones based on the input text length and letter frequency.
The mapping should be different every time the program runs.

Example:

- If the input text is short, use 2-3 homophones per letter.
- If the text is **long**, increase the number of homophones dynamically.

ROT13

ROT13 is a simple **encryption method** and extension of the Ceaser Cipher. It shifts each character of the clear text string 13 positions forward in the alphabet. The algorithm can be explained in one sentence.

ROT13 = **Rot**ate the string to be encrypted by **13** positions (modulo 26) in the alphabet of 26 characters.

The ROT13 cipher is not very secure, as it is just a special case of the Caesar cipher. The Caesar cipher can be broken by either frequency analysis or by just trying out all 25 keys, whereas the ROT13 cipher can be broken by just shifting the letters to 13 places. Therefore, it has no practical use.

ROT 13 Encryption Table

A	B	C	D	E	F	G	H	I	J	K	L	M
N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Cleartext: FINXTER

Encrypted: ???????

Algorithm

a. **Input**: A string of text.
b. **Initialize**: An empty output string.
c. **Iterate through each character** in the input string:
    - If the character is an **uppercase letter** (A-Z):
        - Replace it with the letter 13 places ahead in the alphabet.
        - If shifting goes beyond 'Z', wrap around to the start of the alphabet.
    - If the character is a **lowercase letter** (a-z):
        - Replace it with the letter 13 places ahead in the alphabet.
        - If shifting goes beyond 'z', wrap around to the start of the alphabet.
    - If the character is **not a letter**, leave it unchanged.
d. **Return the transformed string** as output.

How is ROT13 Related to the Caesar Cipher?

The Caesar cipher is the generalization of the ROT13 algorithm. ROT13 does nothing but fix the "number of positions down the alphabet" to +13. Why do we shift the original text, called 'cleartext' or 'plaintext', by 13 positions and not another number? It ensures that applying the encryption twice returns the original cleartext. Hence, you do not have to define two separate methods for encryption and decryption – one method to rule them all!

This is not the case if you use any other number. If you shift the cleartext by 5 positions, ROT5, and apply it twice, you get ROT10 encryption (5+5=10).

Is There a Library for ROT13 in Python?

Yes! It's a built-in library called **codecs**. Using the ROT13 algorithm with the library is simple. Just import the library and call the encode function.

Here is an example:

```
import codecs
phrase = "The Russians are coming!"
# Apply twice to get back original string
print(codecs.encode(codecs.encode(phrase, 'rot_13'), 'rot_13'))
# The Russians are coming!
print(codecs.encode('hello', 'rot_13'))
# uryyb
```

The encode function from the codecs library takes up to three parameters.

- The first parameter is the string object to encode.
- The second parameter is the encoding scheme (default: 'utf-8').
- The third parameter allows you to customize error handling.

In most cases, you can skip the last parameter and use the default error handling.

What Are the Applications of the ROT13 Algorithm?

The ROT13 algorithm is easy to decrypt. An attacker can easily crack your code by running a probabilistic analysis on the distribution of the letters in your encrypted text. You should never rely on this algorithm to encrypt your messages!

So, you may ask, ***what are the applications of the ROT13 algorithm?***

Here are some:

- Obscure potentially offensive jokes in online forums.
- Obscure the result of puzzles in online forums.
- Obscure possible spoilers for movies or books.
- Obscure email addresses on websites against not very sophisticated email bots (the 99%).
- Use it as a game to find phrases that make sense in both forms, encrypted or decrypted. Examples: (png, cat), (be, or).
- ROT13 is a special case of the popular **Caesar cipher**. ROT13 serves as an educational tool to explain it.

In summary, ROT13 is more a fun encryption method that has been a popular running gag in internet culture.

Tasks:

Task 1: File Encryption & Decryption using ROT13

- **Objective**: Implement a program that applies ROT13 to encrypt and decrypt a file.
- **Requirements**:
    1. Read text from an input file (input.txt).
    2. Apply ROT13 transformation.
    3. Save the encrypted text to encrypted.txt.
    4. Implement decryption by applying ROT13 again.

Bonus Challenge: Allow users to specify the filename through command-line arguments.

Task 2: ROT13 for Character Streams

- **Objective**: Modify individual characters using ROT13 while reading from standard input.
- **Requirements**:
    1. Continuously read user input character by character.
    2. Convert only alphabetic characters using ROT13.
    3. Display the transformed characters immediately.

- **Example Interaction:**

```
Enter character: a
Transformed: n
Enter character: Z
Transformed: M
```

Monoalphabetic Cipher

A **monoalphabetic substitution cipher** is a type of cipher where each letter in the plaintext is replaced with another fixed letter to form ciphertext. It's a part of the substitution technique in which a single cipher alphabet is used per message (mapping is done from plain alphabet to cipher alphabet). The letter mapping remains **constant throughout the encryption process.** Monoalphabetic cipher converts plain text into cipher text and re-convert a cipher text to plain text. Monoalphabetic Cipher eliminates the brute-force techniques for cryptanalysis. Moreover, the cipher line can be a permutation of the 26 alphabetic characters.

Key Features:

- **Each plaintext letter maps to a unique ciphertext letter.**
- **Fixed substitution** makes it easy to break with frequency analysis.
- Unlike **polyalphabetic ciphers**, which use multiple alphabets, **monoalphabetic ciphers use only one**.

Monoalphabetic Substitution with Random Mapping

Unlike shift-based ciphers (**Caesar, Multiplicative, Affine**), a **Random Mapping Cipher** randomly maps each letter in the alphabet to another unique letter. This makes it significantly harder to break using frequency analysis.

How It Works

1. **Encryption**
    - Create a random mapping of letters (A → Q, B → M, C → Z, etc.).
    - Replace each letter in the plain text with its mapped letter.
2. **Decryption**
    - Reverse the mapping to reconstruct the original text.

Key Features

- Randomized encryption for each execution
- Hard to break using simple frequency analysis
- Maps letters uniquely
- Decryption uses reverse mapping

Types of Monoalphabetic Substitution Ciphers

There are several types, each with different encryption methods:

A. Additive Cipher (Shift Cipher)

- **Also known as:** Shift Cipher

Encryption Formula:

$$C = (P + K) \mod 26$$

Decryption Formula:

$$P = (C - K) \mod 26$$

- **Example:**
  - Input: P = GEEKS, Key = 4
  - Output: C = KIIOW

Variation: The **Caesar Cipher** is a special case where K = 3.

B. Caesar Cipher

- **A specific type of Additive Cipher with key = 3.**

Encryption Formula:

$$C = (P + 3) \mod 26$$

Decryption Formula:

$$P = (C - 3) \mod 26$$

- **Example:**
  - Input: P = GEEKS
  - Output: C = JHHNV

Security Issue: Since there are only 25 possible shifts, **it can be broken easily by brute force**.

C. Multiplicative Cipher

- **Each letter is mapped using multiplication instead of addition.**

Encryption Formula:

$$C = (P \times K) \mod 26$$

Decryption Formula:

$$P = (C \times K^{-1}) \mod 26$$

- **Example:**
  - Input: P = VMH, Key = 3
  - Output: C = HEL

Issue: The key **must be coprime with 26**, meaning gcd(K, 26) = 1 for unique decryption.

D. Affine Cipher

- **A combination of Additive and Multiplicative Ciphers.**

Encryption Formula:

$$C = (P \times K_1 + K_2) \mod 26$$

Decryption Formula:

$$P = ((C - K_2) \times K_1^{-1}) \mod 26$$

- **Example:**
  - Input: P = ARM, Key1 = 3, Key2 = 5
  - Output: C = HEL

Security Strength: More secure than Additive and Multiplicative alone.

Advantages of Monoalphabetic Ciphers

- Better security than Caesar Cipher (since mapping is not uniform).
- Maintains letter frequency in ciphertext (useful for statistical decryption).
- Supports encryption and decryption with simple substitution

Disadvantages of Monoalphabetic Ciphers

- Easily breakable with frequency analysis (e.g., 'E' is the most common letter in English).
- Prone to guessing attacks using letter frequency data.
- Not as secure as Polyalphabetic Ciphers (e.g., Vigenère Cipher).
- Fixed letter substitution makes it vulnerable to cryptanalysis.

Tasks:

Task 1: Implement the discussed ciphers under monoalphabetic using the cpp code.