# PolyAlphabetic Cipher (Vigenère Cipher) Implementation

# **Overview**

This project provides a Python implementation of the **Vigenère Cipher**, a classic polyalphabetic substitution cipher used for encrypting and decrypting text. The implementation includes additional functionalities such as hashing, verification, and a brute-force attack mechanism to demonstrate the cipher's security aspects.

## **Features**

- **Encryption & Decryption**: Encrypt plaintext using a key and decrypt ciphertext back to plaintext.
- **Hash Function**: Generates a simple hash for plaintext to verify data integrity.
- **Verification**: Ensures the decrypted text matches the original plaintext using hashing.
- **Brute-Force Attack**: Attempts to discover the encryption key by exhaustively searching possible key combinations.
- **Random String Generation**: Generates random strings for testing encryption and decryption processes.

# Usage

1. **Set the Encryption Key**: Modify the KEY variable at the top of the script to set your desired encryption key.

```
KEY = "akpt"
```

2. Run the Script:

python code.py

The script will:

- Generate random strings.
- Append a hash to each string for verification.
- Encrypt each string using the Vigenère cipher.
- Decrypt the ciphertext to verify correctness.
- Attempt to brute-force the encryption key.

#### Classes and Functions

# PolyAlphabeticCipher Class

**Purpose**: Implements the Vigenère cipher with additional functionalities for hashing, verification, and brute-force attacks.

#### Methods:

• encrypt(plaintext: str, key: str) -> str

Encrypts the given plaintext using the provided key.

#### **Parameters:**

- plaintext: The text to encrypt.
- key: The encryption key.

### **Returns:**

- ciphertext: The resulting encrypted text.
- decrypt(ciphertext: str, key: str) -> str

Decrypts the given ciphertext using the provided key.

### **Parameters**:

- ciphertext: The text to decrypt.
- key: The decryption key.

#### **Returns:**

- plaintext: The resulting decrypted text.
- hash\_fn(text: str) -> str

Generates a simple 4-letter hash for the given text.

- 1. Sums the position values of each character in the alphabet (a=0, b=1, etc.)
- 2. Multiplies by the length of the text

#### **Parameters:**

text: The text to hash.

#### **Returns:**

- hash: The generated hash string.
- verify(plaintext: str) -> bool

Verifies if the plaintext has a valid hash appended.

### **Parameters:**

plaintext: The text with an appended hash.

#### **Returns**:

- True if verification succeeds, else False.
- brute\_force(ciphertext: str) -> str

Attempts to discover the encryption key by trying all possible 4-letter combinations.

### **Parameters:**

ciphertext: The text to decrypt.

### **Returns:**

The discovered key if successful, else "none".

```
generate_random_strings Function
```

**Purpose**: Generates a list of random lowercase strings.

### **Parameters:**

- num\_strings: Number of random strings to generate (default is 5).
- string length: Length of each random string (default is 10).

# **Returns**:

• random\_strings: A list containing the generated random strings.

# **Example**

Running the script will produce output similar to the following:

Random String 1: hournhddfyjtdn

Encrypted: hyjknrswfiymdx

Decrypted: hournhddfyjtdn

Attacking : akpt

• • •

# Each entry shows:

- The original random string with its hash.
- The encrypted ciphertext.
- The decrypted plaintext to verify correctness.
- The result of the brute-force attack attempting to discover the encryption key.