



문제의 설명은 'Do you know how to Vigenere?' 이다. 비제네르 암호란 빈도분석법에 취약했던 과거의 암호, 예를 들어 카이사르 암호 등의 단점을 개선한 암호로, 다중 치환 암호이다. 비제네르 암호는 문자의 위치를 키 값을 이용해 이동시키는 방식으로 동작한다.

최근 암호학에 대해서 공부하고 있기에 이 문제를 선택하였다.

✓ 오늘

|  |                    |              |     |
|--|--------------------|--------------|-----|
|  output | 2024-06-01 오후 9:47 | 텍스트 문서       | 1KB |
|  prob   | 2024-06-01 오후 9:47 | Python 원본 파일 | 1KB |

일단 파일에는 prob라는 파이썬 파일과 output이라는 텍스트 문서가 있다.

```
import random
from string import ascii_lowercase, ascii_uppercase, digits

words = ascii_uppercase + ascii_lowercase + digits
class Vigenere:
    def __init__(self, key):
        self._key = key

    def shift(self, a, d):
        if a not in words:
            return a
        index = words.index(a)
        return words[(index + d) % len(words)]

    def encrypt(self, pt):
        ct = ""
        for i in range(len(pt)):
            ct += self.shift(pt[i], self._key[i % len(self._key)])
        return ct

    def decrypt(self, ct):
        pt = ""
        for i in range(len(ct)):
            pt += self.shift(ct[i], -self._key[i % len(self._key)])
        return pt

def main():
    key = [random.randint(0, len(words)) for _ in range(16)]
    with open("secret", "r") as f:
        secret = f.read()

    assert "Vigenere", "cipher" in secret
    cipher = Vigenere(key)
    secret_enc = cipher.encrypt(secret)
    print(f"my encrypted sentence > {secret_enc}")

if __name__ == '__main__':
    main()
```

Prob.py 파일을 열어보았다. 코드를 전체적으로 볼 때 prob 파일은 비제네르 암호를 구현하고 어떤 문자열(혹은 문장)을 암호화하여 출력하는 역할을 하며, output은 그 제목과 형식으로 볼 때 prob 파일이 암호화하여 출력해낸 값을 저장 혹은 출력하는 역할을 하는 것으로 보인다.

일단 prob 파일의 코드를 분석해보자.

```
class Vigenere:
    def __init__(self, key):
        self._key = key
```

Vigenere 클래스는 초기화될 때 값 'key'를 받는다. 비제네르 암호의 작동 원리를 생각해 볼 때, 'key'는 비제네르 암호의 키값에 해당할 것이다. 즉 'key'에 포함된 숫자에 따라 암호의 문자가 얼마나 이동되는지 결정된다.

```
def shift(self, a, d):
    if a not in words:
        return a
    index = words.index(a)
    return words[(index + d) % len(words)]
```

이 부분은 문자를 이동시키는 역할을 한다. 'shift' 멤버 함수는 문자 a를 d만큼 이동시킨다. 먼저, a가 문자열 'words'에 포함되어 있는지 확인한다. 포함되어 있지 않다면 a를 그대로 반환한다. 포함되어 있다면 a의 현재 위치를 찾고, 그 위치에 d를 더해 새로운 위치를 계산한다. 새로운 위치가 words의 길이를 초과하지 않도록 len(words)로 나눈 나머지를 사용한다.

```
def encrypt(self, pt):
    ct = ""
    for i in range(len(pt)):
        ct += self.shift(pt[i], self._key[i % len(self._key)])
    return ct
```

이 부분은 암호화를 담당한다. 'encrypt' 멤버 함수는 주어진 평문 'pt'를 암호화한다. 평문의 각 문자에 대해 shift 멤버 함수를 사용해 문자를 이동시키는 일을 반복해 나간다. 이 과정에서 키 '\_key'의 현재 위치 값을 사용한다. 'i % len(self.\_key)'는 키값이 평문보다 부족한 상황에 대처하기 위해 키값을 계속해서 반복한다. 이 과정을 통해 이동된 문자를 문자열 'ct'에 추가하고, 모든 문자를 이동 처리했다면 'ct'를 반환한다.

```
def decrypt(self, ct):
    pt = ""
    for i in range(len(ct)):
        pt += self.shift(ct[i], -self._key[i % len(self._key)])
    return pt
```

이 부분은 복호화를 담당한다. 'decrypt' 멤버 함수는 'encrypt' 멤버 함수에서 넘겨받은 암호문 문자열 'ct'를 복호화한다. 주목할 점은 이 멤버 함수의 형식이 'encrypt' 멤버 함수와 거의 동일하다는 것이다. 복호화된 문자열을 문자열 'pt'에 저장한다는 점과, 문자를 이동시킬 때 키값이 음수로 사용된다는 점만이 다르다. 이렇게 하면 문자의 이동 방향이 반대가 되어 복호화를 실행할 수 있으며, 그 결과 원래의 평문이 문자열 'pt'에 저장되어 반환된다.

이제 메인 함수를 살펴보자.

```
def main():
    key = [random.randint(0, len(words)) for _ in range(16)]
```

'key'는 16개의 무작위 숫자로 이루어져 있다. 각 숫자는 0부터 'words' 사이의 값이다. 이를 통해 키값 'key'가 정수의 리스트임을 알 수 있다.

```
with open("secret", "r") as f:
    secret = f.read()
```

'secret' 이라는 별도의 파일에서 메시지를 읽어온다.

```
assert "Vigenere", "cipher" in secret
```

'Vigenere'와 'cipher'라는 단어가 'secret' 파일의 메시지에 포함되어 있는지 확인하는 문장이다. 이를 통해 'secret' 파일의 메시지에 포함되어 있는 단어를 알 수 있다.

```
cipher = Vigenere(key)
secret_enc = cipher.encrypt(secret)
print(f"my encrypted sentence > {secret_enc}")
```

생성된 키값으로 'Vigenere' 객체를 초기화한다. 이후 'secret' 파일의 메시지를 암호화하고, 해당 암호화된 메시지를 출력한다.

요약하면, 위 코드는 비제네르 암호를 사용하여 메시지를 암호화하고 복호화하는 코드이다. 암호화 과정은 키 생성, 파일에서 메시지 읽기, 암호화, 출력의 과정을 거쳐 시행된다. 비제네르 암호는 특정 키값을 알고 있지 않다면 풀 수 없지만, 위 코드에서 'secret' 파일의 메시지 내에 있는 단어 몇 개를 알아낼 수 있기에 이를 이용하면 복호화를 할 수 있다고 추정이 가능하다.

최대한 이용할 수 있는 정보가 주어진 단어밖에 없기 때문에 이를 이용하여 복호화를 시행하려면 암호화된 메시지의 특정 위치가 알아낸 단어일 것이라는 예측을 반복하는 방법을 선택할 수밖에 없다.

따라서 이 문제를 해결할 코드의 구조는 기존의 문자 집합 정의와 Vigenere 클래스, 암호화와 복호화를 위한 기존의 멤버 함수를 그대로 가져온 후 키값을 예측하고 복호화하는 함수를 추가하면 된다고 볼 수 있겠다.

```
def solve():
    # 초기 키는 모두 0으로 설정
    key = [0] * 16
    print("초기 키:", " ".join(map(str, key)))

    # 암호문을 파일에서 읽어오기
    with open("output.txt", "r") as f:
        ciphertext = f.read().split(' > ')[1]

    # 16자리씩 암호문 출력
    for i in range(0, len(ciphertext), 16):
        if i + 16 < len(ciphertext):
            print(ciphertext[i:i+16], end=' || ')
        else:
            print(ciphertext[i:])
```

기존 prob 파일의 문자 집합 정의와 Vigenere 클래스, 암호화와 복호화를 위한 멤버 함수를 제외하고 새롭게 작성한 코드는 다음과 같다. 먼저 초기 키를 0으로 설정하고, 암호문을 주어진 'output' 파일에서 읽어온다. 이후 키값 'key'는 16개의 무작위 숫자로 이루어져 있으므로 암호문을 16자리씩 출력해낸다.

이후 사용자에게 복호화 전후의 텍스트를 입력받아 키값을 추정하고 알아내어 암호문을 복호화하는 코드가 필요한데, 이는 본인의 파이썬 실력 부족으로 구현해낼 수 없었다.