Final project: MTH 349

Giovanny Rodriguez

09/12

0.1 Math Functions

This file has 4 basic math functions used in cryptography:

- `mod_exp(a,k,n)`: Computes $a^k n$ using fast squaring.
- `gcd(a,b)`: Finds greatest common divisor using Euclid.
- `mod_inv(a,n)`: Finds x where $ax \equiv 1 \pmod{n}$.
- `is_prime(n)`: Tests if n is prime using Miller-Rabin.

Listing 1: crypto_math.py

```
import math, random

def mod_exp(a, k, n):
    return pow(a, k, n)

def gcd(a, b):
    return math.gcd(a, b)

def mod_inv(a, n):
    return pow(a, -1, n)

def is_prime(n, k=10):
    if n < 2: return False
    if n == 2 or n == 3: return True
    if n % 2 == 0: return False
    r, d = 0, n - 1
    while d % 2 == 0: r += 1; d //= 2
    for _ in range(k):
        a = random.randrange(2, n - 1)
        x = pow(a, d, n)
        if x == 1 or x == n - 1: continue
        for _ in range(r - 1):
            x = pow(x, 2, n)
            if x == n - 1: break
        else: return False
```

```
    return True

def gen_prime(bits=512):
    while True:
        p = random.getrandbits(bits) | (1 << bits - 1) | 1
        if is_prime(p): return p
```

0.2 Classic Ciphers

This file has 7 classic encryption methods. Each has encrypt and decrypt:

- **Caesar**: Shifts each letter by a fixed number.
- **Monoalphabetic**: Replaces each letter with another letter.
- **Vigenère**: Uses a keyword to shift letters (polyalphabetic).
- **Transposition**: Rearranges letters using a grid.
- **Playfair**: Uses a 5x5 matrix to encrypt letter pairs.
- **ADFGVX**: Combines substitution with transposition.
- **DIANA**: Reciprocal cipher (same operation encrypts and decrypts).

Listing 2: classic_ciphers.py - Caesar and Vigenere

```
import numpy as np, string

def clean(txt):
    return ''.join(c for c in txt.upper() if c in string.
        ascii_uppercase)

# CAESAR: shift each letter by s positions
def caesar_enc(txt, s):
    return ''.join(chr((ord(c)-65+s)%26+65) if c.isalpha()
        else c for c in txt.upper())

def caesar_dec(txt, s):
    return caesar_enc(txt, -s)

# VIGENERE: use keyword to shift letters
def vig_enc(txt, key):
    txt, key = clean(txt), clean(key)
    return ''.join(chr((ord(txt[i])-65+ord(key[i%len(key)])
        -65)%26+65) for i in range(len(txt)))

def vig_dec(txt, key):
    txt, key = clean(txt), clean(key)
    return ''.join(chr((ord(txt[i])-65-ord(key[i%len(key)])
        +65)%26+65) for i in range(len(txt)))
```

Listing 3: classic_ciphers.py - Transposition

```
# TRANSPOSITION: rearrange letters in a grid
def trans_enc(txt, key):
    txt, key = clean(txt), clean(key)
    pad = (len(key) - len(txt) % len(key)) % len(key)
    txt += 'X' * pad
    grid = np.array(list(txt)).reshape(-1, len(key))
    return ''.join(grid[:, np.argsort(list(key))].flatten('F
        '))

def trans_dec(txt, key):
    txt, key = clean(txt), clean(key)
    rows = len(txt) // len(key)
    grid = np.array(list(txt)).reshape(len(key), rows).T
    return ''.join(grid[:, np.argsort(np.argsort(list(key)))
        ].flatten()).rstrip('X')
```

Listing 4: classic_ciphers.py - Playfair

```
# PLAYFAIR: use 5x5 matrix for letter pairs
def play_matrix(key):
    key = clean(key).replace('J','I')
    seen, chars = set(), []
    for c in key:
        if c not in seen: seen.add(c); chars.append(c)
    for c in string.ascii_uppercase:
        if c != 'J' and c not in seen: chars.append(c)
    return np.array(chars).reshape(5,5)

def play_enc(txt, key):
    txt, m = clean(txt).replace('J','I'), play_matrix(key)
    # Split into pairs, apply Playfair rules
    # Same row: move right, Same col: move down, Rectangle:
        swap corners
    ...
```

Listing 5: classic_ciphers.py - DIANA

```
# DIANA: reciprocal cipher (encrypt = decrypt)
def diana_enc(txt, key):
    txt, key = clean(txt), clean(key)
    return ''.join(chr((26-ord(txt[i])+65-ord(key[i%len(key)
        ])+65)%26+65) for i in range(len(txt)))

def diana_dec(txt, key):
    return diana_enc(txt, key)  # Same operation!
```

0.3 RSA

RSA is a public-key cryptosystem. It uses two keys:

- **Public key** (e, n): Anyone can use this to encrypt.
- **Private key** (d, n): Only owner can decrypt.

Listing 6: rsa_tool.py

```
from crypto_math import mod_exp, mod_inv, gen_prime, gcd

def gen_keys(bits=512):
    p, q = gen_prime(bits), gen_prime(bits)  # Two big
        primes
    n, phi = p * q, (p-1) * (q-1)
    e = 65537  # Standard public exponent
    d = mod_inv(e, phi)  # Private exponent
    return (e, n), (d, n)  # (public, private)

def enc(msg, pub):
    e, n = pub
    return [mod_exp(ord(c), e, n) for c in msg]  # c = m^e
        mod n

def dec(cipher, priv):
    d, n = priv
    return ''.join(chr(mod_exp(c, d, n)) for c in cipher)  #
         m = c^d mod n
```

0.4 Frequency Analysis

These tools help break ciphers by looking at letter patterns:

- `freq(txt)`: Counts how often each letter appears.
- `adj(txt)`: Shows which letters come before/after each letter.
- `entropy(txt)`: Measures randomness (higher = more random).

Listing 7: analysis_tool.py

```
import pandas as pd
from scipy.stats import entropy as sp_entropy
from collections import Counter

def freq(txt):
    txt = clean(txt)
    s = pd.Series(list(txt))
    df = s.value_counts().reset_index()
    df.columns = ['Letter', 'Count']
    df['%'] = (df['Count'] / len(txt) * 100).round(2)
    return df
```

```
def entropy(txt):
    txt = clean(txt)
    probs = pd.Series(list(txt)).value_counts(normalize=True
        )
    return sp_entropy(probs, base=2)  # Shannon entropy in
        bits
```

0.5 Main Program

This is the main menu. User can choose what to do:

- Option 1: Test all ciphers at once.
- Option 2: Vigenère encrypt or decrypt only.
- Option 3: Vigenère frequency tables (for breaking the cipher).
- Option 4: RSA only.
- Option 5: Frequency analysis of text.

Listing 8: main.py

```
from crypto_math import mod_exp, gcd, mod_inv, is_prime
from classic_ciphers import *
from rsa_tool import gen_keys, enc, dec
from analysis_tool import freq, adj, entropy, vig_freq

def main():
    print("\n=== CRYPTO PORTFOLIO ===")
    print("1. All ciphers demo")
    print("2. Vigenere only")
    print("3. Vigenere freq tables")
    print("4. RSA only")
    print("5. Frequency analysis")

    opt = input("Option: ") or "1"

    if opt == "1":
        msg = input("Text: ") or "HELLOWORLD"
        key = input("Key: ") or "SECRET"
        print(f"\nInput: {msg} | Key: {key}")
        for name, e, d in [
            ("Caesar", caesar_enc(msg,3), caesar_dec(
                caesar_enc(msg,3),3)),
            ("Vigenere", vig_enc(msg,key), vig_dec(vig_enc(
                msg,key),key)),
            ("Playfair", play_enc(msg,key), play_dec(
                play_enc(msg,key),key)),
            ("Diana", diana_enc(msg,key), diana_dec(
                diana_enc(msg,key),key)),
```

```
        ]:
            print(f"  {name}: {e} -> {d}")

    elif opt == "2":
        msg = input("Text: ") or "ATTACK"
        key = input("Key: ") or "LEMON"
        mode = input("(e)ncrypt or (d)ecrypt? ") or "e"
        if mode == "e":
            print(f"Ciphertext: {vig_enc(msg, key)}")
        else:
            print(f"Plaintext: {vig_dec(msg, key)}")

    elif opt == "3":
        cipher = input("Ciphertext: ") or "LXFOPVEFRNHR"
        k = int(input("Key length: ") or "3")
        tables = vig_freq(cipher, k)
        for t in tables:
            print(f"Pos {t['pos']}: IC={t['IC']:.4f}")

    elif opt == "4":
        msg = input("Text: ") or "HELLO"
        pub, priv = gen_keys(64)
        c = enc(msg, pub)
        print(f"Decrypted: {dec(c, priv)}")

    elif opt == "5":
        txt = input("Text: ") or "THE QUICK BROWN FOX"
        print(f"Entropy: {entropy(txt):.4f} bits")

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

0.6 Example Output

```
Input: ATTACK | Key: LEMON

Caesar:   DWWDFN -> ATTACK
Vigenere: LXFOPV -> ATTACK
Playfair: BSSBIC -> ATTACK
DIANA:    QDJQUS -> ATTACK

RSA: enc=[123...], dec=ATTACK

Entropy: 1.92 bits
Top letters: [A, T, C]
```