Python Report - Python for security

Hugo François - INGE 2C

April 5, 2025

Contents

1 Introduction

This document presents the fundamental elements of the Python programming language in the context of cybersecurity. Python is a high-level, interpreted, and cross-platform language, making it particularly convenient for scripting and rapid development in security-related tasks.

2 Basic Syntax and Data Types

Python uses a clean and readable syntax. It is an object-oriented language but also supports functional programming.

2.1 Printing and Types

```
print("Hello")
type(1)
```

2.2 Numeric Types

Python supports three numeric types:

- `int` (integer)
- `float` (floating point)
- `complex` (complex numbers)

2.3 Arithmetic Operations

Common arithmetic operations include:

```
11 * 3     # Multiplication
1 + 2      # Addition
6 - 2      # Subtraction
10 % 3     # Modulo
_ / 2      # Division using last result (_ is a special variable)
```

3 Variables and Booleans

Variables are assigned using `=`, and Python supports multiple assignments:

```
x = 1
x, y = 1, 2
```

Boolean values are either `True` or `False`. Empty structures or zero values evaluate to `False`.

4 Control Structures

4.1 Conditions

```
x = int(input("Write Integer A: "))
y = int(input("Write Integer B: "))
if x == y:
    print("A and B are equal")
elif x < y:
    print("A is lesser than B")
else:
    print("A is greater than B")
```

4.2 Loops

For loop:

```
for i in range(5):
    print('*' * i)
```

While loop:

```
A = 15
while A > 1:
    A /= 2
    print(A, end=" ")
```

5 Functions and Modules

Functions are defined using `def`:

```
def Abs(n):
    if n < 0:
```

```
        return -n
    return n
```

Modules allow code reuse. A function in a file (e.g., `Module.py`) can be imported in another:

```
from Module import Abs
```

6 Strings

Python offers a rich set of string manipulation features including slicing, repetition, and formatting:

```
msg = "Hello"
msg2 = msg + " World"
msg3 = msg * 3
len(msg)
msg[::-1]  # Reversed string
```

7 Lists and Comprehensions

Lists are mutable and can contain any type:

```
my_list = [1, "a", 3.14]
my_list.append("new")
my_list[0] = 42
```

Comprehension example:

```
squares = [x**2 for x in range(10)]
```

8 Tuples, Sets and Dictionaries

Tuples are immutable:

```
tuple_example = (1, 2, 3)
```

Sets are unordered and store unique items:

```
unique_items = {1, 2, 3}
```

Dictionaries store key-value pairs:

```
info = {'name': 'Alice', 'age': 25}
```

9 File Handling

Python can read and write files:

```
with open("file.txt", 'r', encoding='utf8') as f:
    content = f.read()
```

10 Exceptions

Errors can be managed using `try/except`:

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Cannot divide by zero")
```

11 Classes and Inheritance

Python supports OOP with classes:

```
class Car:
    def __init__(self, brand):
        self.brand = brand
    def show(self):
        print("Brand:", self.brand)
```

Inheritance is defined by passing the parent class:

```
class SmartCar(Car):
    def __init__(self, brand, model):
        super().__init__(brand)
        self.model = model
```

12 Classical Cryptography: Caesar Cipher

The Caesar cipher is one of the simplest and most well-known encryption techniques. It is a type of substitution cipher where each letter in the plaintext is shifted by a fixed number of positions down the alphabet.

12.1 Script Overview

The provided script allows to encrypt, decrypt, and break Caesar ciphers. It makes use of basic string operations and file handling. A dictionary file is used to identify valid English words when attempting decryption.

12.2 Files Used

- `Hello.txt`: contains the original plaintext.
- `output.txt`: contains the encrypted text.
- `decrypted.txt`: contains the decrypted text and the key used.
- `english_dico.json`: dictionary of English words.

12.3 Key Features

- The cipher shifts each alphabetical character while preserving the case.
- Non-alphabetical characters (spaces, punctuation) are preserved.
- A brute-force method is used to find the correct key by testing all possible shifts and evaluating the result using the dictionary.

12.4 Code

```
Caesar Cipher Script

import argparse
import json
import matplotlib.pyplot as plt

def caesar_cipher(text, step, encrypt=True):
    result = []
    step = step if encrypt else -step
    for char in text:
        if char.isalpha():
```

```
            base = ord('A') if char.isupper() else ord('a')
            result.append(chr((ord(char) - base + step) % 26 + base))
        else:
            result.append(char)
    return ''.join(result)


def brute_force_with_dictionary(text, dictionary_file):
    with open(dictionary_file, 'r', encoding='utf-8') as file:
        english_words = set(json.load(file))

    match_percentages = []
    for step in range(26):
        decrypted_text = caesar_cipher(text, step, encrypt=False)
        words = decrypted_text.split()
        matches = sum(1 for word in words if word.lower() in english_words)
        match_percentage = (matches / len(words)) * 100 if words else 0
        match_percentages.append((step, match_percentage))
        print(f"Step {step}: {decrypted_text} (Match:
        ↪  {match_percentage:.2f}%) ---------------")


    keys = [step for step, _ in match_percentages]
    percentages = [percent for _, percent in match_percentages]
    plt.plot(keys, percentages, marker='o')
    plt.title("Caesar Cipher Brute Force Match Percentage")
    plt.xlabel("Key")
    plt.ylabel("Match Percentage")
    plt.xticks(range(26))
    plt.grid(True)
    plt.show()

    # Return the best match
    best_key, best_match = max(match_percentages, key=lambda x: x[1]) # Find
    ↪  the key with the highest match percentage
    print(f"\nBest match found with key {best_key} ({best_match:.2f}%
    ↪  match).")
    return best_key

def main():
    parser = argparse.ArgumentParser(description="Caesar Cipher Tool with
    ↪  Brute Force and Dictionary Matching")
    parser.add_argument('-i', '--input', required=True, help="Input file")
    parser.add_argument('-o', '--output', help="Output file (not required
    ↪  for cracking)")
```

```
    parser.add_argument('-K', '--key', type=int, help="Shift step (required
    ↪  for encryption/decryption)")
    parser.add_argument('-e', '--encrypt', action='store_true',
    ↪  help="Encrypt the file")
    parser.add_argument('-d', '--decrypt', action='store_true',
    ↪  help="Decrypt the file")
    parser.add_argument('-c', '--crack', action='store_true', help="Crack
    ↪  the cipher using brute force")
    parser.add_argument('-w', '--words', help="Path to English dictionary
    ↪  JSON file (required for cracking)")
    args = parser.parse_args()

    if args.crack:
        if not args.words:
            raise ValueError("You must specify a dictionary file (-w) for
            ↪  cracking.")
        with open(args.input, 'r', encoding='utf-8') as infile:
            text = infile.read()
        best_key = brute_force_with_dictionary(text, args.words)
        if args.output:
            with open(args.output, 'w', encoding='utf-8') as outfile:
                outfile.write(f"Best key: {best_key}\n")
                outfile.write(caesar_cipher(text, best_key, encrypt=False))
    else:
        if args.encrypt == args.decrypt:
            raise ValueError("You must specify either -e (encrypt) or -d
            ↪  (decrypt), not both.")
        if args.key is None:
            raise ValueError("You must specify a key (-K) for encryption or
            ↪  decryption.")
        if args.output is None:
            raise ValueError("You must specify an output file (-o) for
            ↪  encryption or decryption.")

        with open(args.input, 'r', encoding='utf-8') as infile:
            text = infile.read()

        processed_text = caesar_cipher(text, args.key, encrypt=args.encrypt)

        with open(args.output, 'w', encoding='utf-8') as outfile:
            outfile.write(processed_text)

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

12.5 Output Results

Plaintext (`Hello.txt`):

> That all men are by nature equally free and independent and have certain inherent rights, of which, when they enter into a state of society, they cannot, by any compact, deprive or divest their posterity; namely, the enjoyment of life and liberty, with the means of acquiring and possessing property, and pursuing and obtaining happiness and safety.

Encrypted Text (`output.txt`):

> Uibu bmm nfo bsf cz obuvsf frvbmmz gsff boe joefqfoefou boe ibwf dfsubjo joifsfou sjhiut, pg xijdi, xifo uifz foufs joup b tubuf pg tpdjfuz, uifz dboopu, cz boz dpnqbdu, efqsjwf ps ejwftu uifjs qptufsjuz; obnfmz, uif fokpznfou pg mjgf boe mjcfsuz, xjui uif nfbot pg bdrvjsjoh boe qpttfttjoh qspqfsuz, boe qvstvjoh boe pcubjojoh ibqqjoftt boe tbgfuz.

Decryption Result (`decrypted.txt`):

> That all men are by nature equally free and independent and have certain inherent rights, of which, when they enter into a state of society, they cannot, by any compact, deprive or divest their posterity; namely, the enjoyment of life and liberty, with the means of acquiring and possessing property, and pursuing and obtaining happiness and safety.

13 Classical Cryptography: Monoalphabetic Cipher

The monoalphabetic substitution cipher replaces each letter in the plaintext with another letter using a fixed permutation of the alphabet. Unlike the Caesar cipher which shifts characters by a constant value, this cipher allows a fully custom mapping. Although it increases the complexity compared to Caesar, it remains vulnerable to frequency analysis due to the preserved structure of the language.

13.1 Script Overview

The script analyzes the frequency of each letter in the encrypted text and attempts to match it to the frequency of letters in a reference English text. It then uses the resulting guessed key to decrypt the message.

13.2 Files Used

- `mylongtext.txt`: a long English reference text used for statistical frequency comparison.
- `encrypted.txt`: the encrypted message.
- `cracked.txt`: the final decrypted output.

13.3 Code

```
import argparse
import string
import sys
import matplotlib.pyplot as plt
from collections import Counter

def validate_key(key):
    if len(key) != 26 or not key.isalpha() or len(set(key.lower())) != 26:
        raise ValueError("The key must be a 26-character string containing
        ↪  all letters of the alphabet exactly once.")

def create_cipher_maps(key, decrypt=False):
    alphabet = string.ascii_lowercase
    if decrypt:
        return dict(zip(key.lower(), alphabet)), dict(zip(key.upper(),
        ↪  alphabet.upper()))
    else:
        return dict(zip(alphabet, key.lower())), dict(zip(alphabet.upper(),
        ↪  key.upper()))

def process_file(input_file, output_file, cipher_map_lower,
↪  cipher_map_upper):
    try:
        with open(input_file, 'r') as infile, open(output_file, 'w') as
        ↪  outfile:
            for line in infile:
                processed_line = ''.join(
                    cipher_map_lower.get(char, cipher_map_upper.get(char,
                    ↪  char)) for char in line
                )
                outfile.write(processed_line)
    except FileNotFoundError:
        print(f"Error: The file '{input_file}' was not found.")
        sys.exit(1)

def count_letters(text):
    text = text.lower()
    return Counter(char for char in text if char.isalpha())

def plot_letter_frequency(letter_counts):
    letters, counts = zip(*sorted(letter_counts.items()))
    plt.bar(letters, counts)
```

```
    plt.xlabel("Letters")
    plt.ylabel("Frequencies")
    plt.title("Letter Frequency in Text")
    plt.show()

def crack_cipher(input_file, output_file):
    try:
        with open(input_file, 'r') as infile:
            encrypted_text = infile.read()
        letter_counts = count_letters(encrypted_text)
        cipher_freq_order = sorted(letter_counts, key=letter_counts.get,
        ↪  reverse=True)
        english_freq_order = "etaoinshrdlucmfwypvbgkjqxz"

        mapping = {}
        for i, letter in enumerate(cipher_freq_order):
            if i < len(english_freq_order):
                mapping[letter] = english_freq_order[i]

        cipher_map_lower = mapping
        cipher_map_upper = {k.upper(): v.upper() for k, v in
        ↪  mapping.items()}

        with open(output_file, 'w') as outfile:
            for char in encrypted_text:
                if char.islower():
                    outfile.write(cipher_map_lower.get(char, char))
                elif char.isupper():
                    outfile.write(cipher_map_upper.get(char, char))
                else:
                    outfile.write(char)

        print("Cracked text written to output file:", output_file)
        print("Mapping used:")
        for cipher_letter in sorted(mapping):
            print(f"  {cipher_letter} -> {mapping[cipher_letter]}")
    except FileNotFoundError:
        print(f"Error: The file '{input_file}' was not found.")
        sys.exit(1)

def main():
    parser = argparse.ArgumentParser(description="Monoalphabetic Cipher
    ↪  Tool")
    parser.add_argument("-i", required=True, help="Input file")
    parser.add_argument("-o", required=True, help="Output file")
```

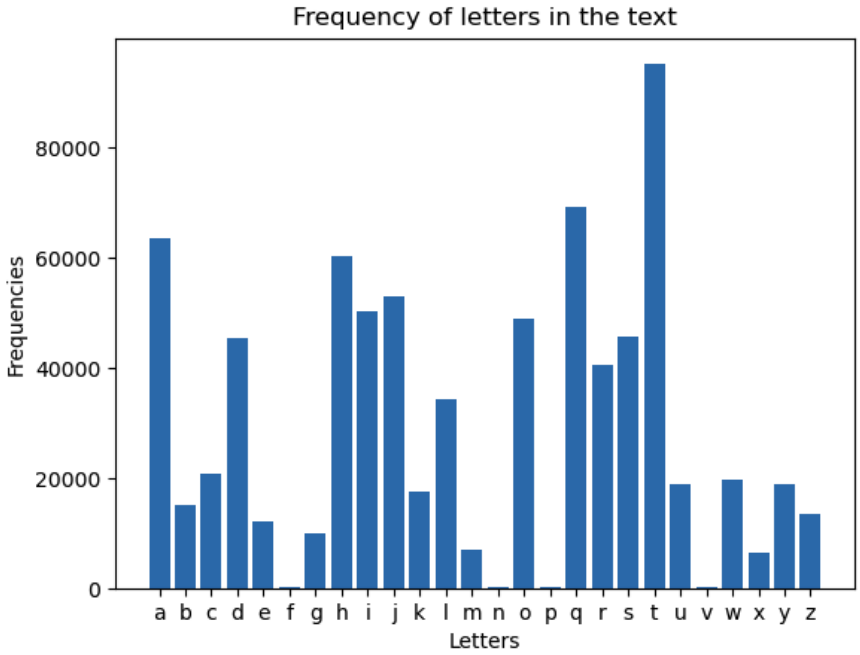
```
    parser.add_argument("-K", help="Key (26 letters, required for
    ↪  encryption/decryption)")
    parser.add_argument("-e", action="store_true", help="Encryption mode")
    parser.add_argument("-d", action="store_true", help="Decryption mode")
    parser.add_argument("-c", action="store_true", help="Crack mode
    ↪  (frequency analysis)")

    args = parser.parse_args()

    if sum([args.e, args.d, args.c]) != 1:
        print("Error: Specify exactly one mode: -e (encrypt), -d (decrypt),
        ↪  or -c (crack).")
        sys.exit(1)

    if args.c:
        crack_cipher(args.i, args.o)
        sys.exit(0)

    if (args.e or args.d) and args.K is None:
        print("Error: Key (-K) is required for encryption or decryption.")
        sys.exit(1)

    try:
        if args.K:
            validate_key(args.K)
    except ValueError as e:
        print(f"Error: {e}")
        sys.exit(1)

    if args.K:
        cipher_map_lower, cipher_map_upper = create_cipher_maps(args.K,
        ↪  decrypt=args.d)
        process_file(args.i, args.o, cipher_map_lower, cipher_map_upper)

    # Analyze processed text and display letter frequency
    with open(args.o, 'r') as outfile:
        processed_text = outfile.read()
        letter_counts = count_letters(processed_text)
        plot_letter_frequency(letter_counts)
        print("Frequency analysis of the processed text:")
        for letter, count in sorted(letter_counts.items()):
            print(f"  {letter}: {count}")

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

13.4 Output Results

Letter mapping printed by the script:

```
a -> a
b -> y
c -> u
d -> r
e -> v
f -> j
g -> b
h -> o
i -> n
j -> i
k -> w
l -> l
m -> g
n -> z
o -> q
p -> s
q -> d
r -> x
s -> h
t -> e
u -> f
v -> x
w -> c
x -> k
y -> m
z -> p
```

Letter Frequency Plot:



Decrypted Text (`cracked.txt`):

```
Q R R Tolgsei Tne Lord Om Tne Rsifh
 _Tne Lord om tne Rsifh_ nah peei read py waiy beoble hsive st msially
  ↪ abbeared si brsit; aid S hnocld lsge to hay howetnsif nere ustn
  ↪ remereive to tne waiy obsisoih or fcehheh tnat S nake revesked or
  ↪ nake read voiverisif tne wotskeh aid weaisif om tne tale. Tne brswe
  ↪ wotske uah tne dehsre om a tale-teller to try nsh naid at a really
  ↪ loif htory tnat uocld nold tne atteitsoi om readerh, awche tnew,
  ↪ delsfnt tnew, aid at tsweh waype exvste tnew or deebly woke tnew.
  ↪ Ah a fcsde S nad oily wy oui meelsifh mor unat sh abbealsif or
  ↪ woksif, aid mor waiy tne fcsde uah siekstaply omtei at maclt. Howe
  ↪ uno nake read tne poog, or at aiy rate nake rekseued st, nake mocid
  ↪ st porsif, aphcrd, or voitewbtsple; aid S nake io vache to
  ↪ vowblasi, hsive S nake hswslar obsisoih om tnesr uorgh, or om tne
  ↪ gsidh om urstsif tnat tney eksdeitly bremer. Pct ekei mrow tne
  ↪ bosith om kseu om waiy uno nake eiqoyed wy htory tnere sh wcvn tnat
  ↪ maslh to bleahe. St sh bernabh iot bohhsple si a loif tale to
  ↪ bleahe ekerypody at all bosith, ior to dshbleahe ekerypody at tne
  ↪ hawe bosith; mor S msid mrow tne letterh tnat S nake revesked tnat
  ↪ tne bahhafeh or vnabterh tnat are to howe a plewshn are all py
  ↪ otnerh hbevsally abbroked. Tne woht vrstsval reader om all, wyhelm,
  ↪ iou msidh waiy demevth, wsior aid waqor, pct pesif mortciately
  ↪ cider io oplsfatsoi estner to rekseu tne poog or to urste st afasi,
  ↪ ne usll bahh oker tnehe si hsleive, exvebt oie tnat nah peei ioted
  ↪ py otnerh: tne poog sh too hnort.
```

```
_Tne Lord om tne Rsifh_ sh iou shhced si a ieu edstsoi, aid tne
↪ obbortcisty nah peei tagei om rekshsif st. A icwper om errorh aid
↪ sivoihshteivseh tnat htsll rewasied si tne text nake peei
↪ vorrevted, aid ai attewbt nah peei wade to broksde simorwatsoi oi a
↪ meu bosith unsvn atteitske readerh nake rashed. S nake voihsdered
↪ all tnesr vowweith aid eijcsrseh, aid sm howe heew to nake peei
↪ bahhed oker tnat way pe pevache S nake masled to geeb wy ioteh si
↪ order; pct waiy eijcsrseh vocld oily pe aihuered py addstsoial
↪ abbeidsveh, or sideed py tne brodcvtsoi om ai avvehhory kolcwe
↪ voitasisif wcvn om tne watersal tnat S dsd iot sivlcde si tne
↪ orsfsial edstsoi, si bartsvclar wore detasled lsifcshtsv
↪ simorwatsoi. Si tne weaitswe tnsh edstsoi ommerh tnsh Moreuord, ai
↪ addstsoi to tne Brolofce, howe ioteh, aid ai sidex om tne iaweh om
↪ berhoih aid blaveh. Tnsh sidex sh si siteitsoi vowblete si stewh
↪ pct iot si remereiveh, hsive mor tne breheit bcrbohe st nah peei
↪ ievehhary to redcve sth pclg. A vowblete sidex, wagsif mcll che om
↪ tne watersal brebared mor we py Wrh. I. Hwstn, peloifh ratner to
↪ tne avvehhory kolcwe.
```

Explication of the result :

```
Since the input text was relatively short, the frequency analysis used
↪ for cracking the substitution cipher was distorted. Frequency-based
↪ decryption techniques require longer texts to identify consistent
↪ statistical patterns, and the insufficient data led to an incomplete
↪ decryption
```

14 Classical Cryptography: Polyalphabetic Cipher

Polyalphabetic Cipher The polyalphabetic cipher is a method of encrypting alphabetic text using a sequence of substitution alphabets derived from a secret key. Unlike monoalphabetic ciphers, it applies different substitution rules across the message, significantly increasing resistance to frequency analysis. The key is repeated cyclically to match the length of the plaintext, and each character in the key determines how the corresponding character in the message is substituted.

14.1 Script Overview

The script implements a polyalphabetic cipher by using a key-driven multi-substitution system. The key characteristics include:

- A user-defined key used cyclically across the message.
- An encryption function that applies a unique alphabet substitution per character based on the key.
- A decryption function that reverses each substitution accurately.
- Case sensitivity is preserved, and non-alphabetic characters remain unchanged.

The script supports command-line arguments allowing the user to choose between encryption and decryption modes, specify the key, and provide the input text

14.2 Files used

- `pac.py` : Contains the script
- `text.txt` : Contains the original message to encrypt.
- `encrypted.txt` : Contains the encrypted message.
- `cracked.txt` : Contains the decoded text.

14.3 Code

```
#!/usr/bin/env python3
import argparse
import os
import random

LETTERS = "abcdefghijklmnopqrstuvwxyz"

def encrypt(text: str, key_list: list) -> str:
```

```
    """
    Encrypts the provided text using a polyalphabetic substitution cipher.
    Each letter is replaced using a different substitution alphabet, cycling
    ↪ through
    the provided key list.

    Args:
        text (str): The input text to be encrypted.
        key_list (list): A list of substitution alphabets.

    Returns:
        str: The encrypted text.

    Examples:
        >>> encrypt("hello", ["defghijklmnopqrstuvwxyzabc",
        ↪ "bcdefghijklmnopqrstuvwxyza"])
        'kfomr'
        >>> encrypt("hello", ["ndjemywsxchlqizauvbkptrogf",
        ↪ "bcdefghijklmnopqrstuvwxyza", "cdefghijklmnopqrstuvwxyzab"])
        'sfnlp'
    """
    result = ""
    total_keys = len(key_list)
    for pos, char in enumerate(text.lower()):
        if char in LETTERS:
            key_index = pos % total_keys
            result += key_list[key_index][LETTERS.index(char)]
        else:
            result += char
    return result

def decrypt(text: str, key_list: list) -> str:
    """
    Decrypts text that was encrypted with the polyalphabetic substitution
    ↪ cipher.
    It restores each letter by finding its position in the corresponding key
    ↪ alphabet.

    Args:
        text (str): The encrypted text.
        key_list (list): A list of substitution alphabets.

    Returns:
        str: The decrypted text.
```

```
    Examples:
        >>> decrypt("kfomr", ["defghijklmnopqrstuvwxyzabc",
        ↪  "bcdefghijklmnopqrstuvwxyza"])
        'hello'
        >>> decrypt("dbvmbw", ["ndjemywsxchlqizauvbkptrogf",
        ↪  "bcdefghijklmnopqrstuvwxyza", "cdefghijklmnopqrstuvwxyzab"])
        'bateau'
    """
    result = ""
    total_keys = len(key_list)
    for pos, char in enumerate(text.lower()):
        if char in LETTERS:
            key_index = pos % total_keys
            original_index = key_list[key_index].index(char)
            result += LETTERS[original_index]
        else:
            result += char
    return result

def create_random_key() -> str:
    """
    Generates a random substitution alphabet by shuffling the letters.

    Returns:
        str: A randomized alphabet string containing 26 characters.
    """
    letters_list = list(LETTERS)
    random.shuffle(letters_list)
    return "".join(letters_list)

def main_cli():
    """
    Command-line interface for encrypting or decrypting text using a
    ↪  polyalphabetic cipher.
    """
    parser = argparse.ArgumentParser(description="Polyalphabetic Cipher
    ↪  Utility")
    parser.add_argument("-i", type=str, help="Input text or path to a text
    ↪  file")
    parser.add_argument("-K", nargs="+", type=str, help="List of
    ↪  substitution key alphabets")
    parser.add_argument("-o", type=str, help="Name of the output file")
    parser.add_argument("-e", action="store_true", help="Encrypt the input")
    parser.add_argument("-d", action="store_true", help="Decrypt the input")
```

```
parser.add_argument("--create-key", action="store_true", help="Generate
↪  a random key alphabet and exit")
parser.add_argument("--print-only", action="store_true", help="Print the
↪  result instead of saving to a file")

args = parser.parse_args()

# If the user just wants to generate a key and exit.
if args.create_key:
    print(create_random_key())
    exit(0)

# Validate the input text.
if not args.i:
    print("Error: You must specify an input text or file using the -i
    ↪  option.")
    exit(1)

# Process key alphabets: if none provided, generate them.
if not args.K:
    print("Warning: No key alphabets provided; generating random
    ↪  keys...", end=" ")
    count = input("How many keys would you like to generate? (type exit
    ↪  to quit): ")
    if count.lower() == "exit":
        exit(0)
    generated_keys = [create_random_key() for _ in range(int(count))]
    args.K = generated_keys
    print(f"Generated keys: {generated_keys}")
else:
    for key in args.K:
        if len(key) != 26:
            print("Error: Each key alphabet must contain exactly 26
            ↪  characters.")
            exit(1)
    print(f"Using provided keys: {args.K}")

# Ensure that exactly one operation (encrypt or decrypt) is specified.
if (not args.e and not args.d) or (args.e and args.d):
    print("Error: Please choose a single operation: use -e for
    ↪  encryption or -d for decryption.")
    exit(1)

# Validate output option.
if not args.o and not args.print_only:
```

```
        print("Error: You must specify an output file name with -o or use
        ↪  --print-only to display the result.")
        exit(1)

    # Determine if input is a text file or a direct text string.
    if args.i.endswith(".txt"):
        if not os.path.exists(args.i):
            print("Error: The specified file does not exist.")
            exit(1)
        with open(args.i, "r", encoding="utf-8") as file:
            content = file.read()
    else:
        content = args.i

    # Execute the appropriate operation.
    final_output = encrypt(content, args.K) if args.e else decrypt(content,
    ↪  args.K)

    # Output the result: either print or write to file.
    if args.print_only:
        print("\n" + final_output)
    else:
        with open(args.o, "w", encoding="utf-8") as file:
            file.write(final_output)

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

14.4 Results

- `text.txt` Message :

```
abcdefghijklmnopqrstuvwxyz
aaaa
This is a test text for the poly_alphabetic_cipher
```

- `encrypted.txt` Encrypted message :

```
urxyqvwynezshbqavpnsuvtyjh
uchu
egob ob c lqbl etxe fqm egq alsj_hbwfurtlox_unaytm
```

- `cracked.txt` Decoded text :

```
abcdefghijklmnopqrstuvwxyz
aaaa
This is a test text for the poly_alphabetic_cipher
```

15 Classical Cryptography: One-Time Pad (OTP)

The One-Time Pad (OTP) is a theoretically unbreakable cipher when implemented correctly. It operates by XORing the plaintext with a key that is as long as the message, completely random, and used only once. Because of its perfect secrecy, OTP is often studied as a cryptographic ideal.

15.1 Script Overview

The provided Python script implements the OTP encryption and decryption process. It converts plaintext and key characters to their binary representations, performs bitwise XOR, and converts the result back to text. This simple implementation assumes the key and plaintext are both available and of the same length.

15.2 Files Used

- `plaintext.txt`: contains the original message to encrypt.
- `key.txt`: contains the secret key (must match length of plaintext).
- `ciphertext.txt`: stores the encrypted binary message.
- `plaintext_verif.txt`: stores the decrypted message after OTP decryption.

15.3 Code

```
def otp(plaintext, key):
    with open('plaintext.txt', 'r') as f:
        plaintext = f.read().strip()
    with open('key.txt', 'r') as f:
        key = f.read().strip()
    if len(key) < len(plaintext):
        raise ValueError("len(key) needs to be equal to len(plaintext)..")

    plaintext_bin = ''.join(format(ord(p), '08b') for p in plaintext)
    key_bin = ''.join(format(ord(k), '08b') for k in key)
```

```
    ciphertext_bin = ''.join(str(int(p) ^ int(k)) for p, k in
    ↪  zip(plaintext_bin, key_bin))
    with open('ciphertext.txt', 'w') as f:
        f.write(ciphertext_bin)
    return ciphertext_bin

def otp_decrypt(ciphertext, key):
    with open('ciphertext.txt', 'r') as f:
        ciphertext_bin = f.read().strip()
    with open('key.txt', 'r') as f:
        key = f.read().strip()
    key_bin = ''.join(format(ord(k), '08b') for k in key)
    plaintext_bin = ''.join(str(int(c) ^ int(k)) for c, k in
    ↪  zip(ciphertext_bin, key_bin))
    plaintext = ''.join(chr(int(plaintext_bin[i:i+8], 2)) for i in range(0,
    ↪  len(plaintext_bin), 8))
    with open('plaintext_verif.txt', 'w') as f:
        f.write(plaintext)
    return plaintext


with open('plaintext.txt', 'r') as f:
    plaintext = f.read().strip()
with open('key.txt', 'r') as f:
    key = f.read().strip()

plaintext_bin = ''.join(format(ord(p), '08b') for p in plaintext)
key_bin = ''.join(format(ord(k), '08b') for k in key)
ciphertext = otp('plaintext.txt', 'key.txt')
plaintext_verif = otp_decrypt('ciphertext.txt', 'key.txt')

print('\n')
print(plaintext_bin, '<-- Plaintext (in binary)')
print(key_bin, '<-- Key (in binary)')
print(ciphertext, '<-- Ciphertext (in binary)')
print(plaintext_verif, '<-- Plaintext verification')


def verif():
    with open('plaintext.txt', 'r') as f:
        plaintext = f.read().strip()
    with open('plaintext_verif.txt', 'r') as f:
        plaintext_verif = f.read().strip()
    if plaintext == plaintext_verif:
        print('Plaintext is verified ')
```

```
        print('\n')
    else:
        print('Plaintext is not verified ')
        print('\n')

verif()
```

15.4 Results

- `plaintext.txt`: "secret text"
- `plaintext.txt in binary`:

```
01110011011001010110001101110010011001010111010000100000011101000110010101110000111010
↪ 010111100001110100
```

- `key.txt`: "mysecretkey"
- `key.txt in binary`:

```
01101101011110010111001101100101011000110111001001100101011101000110100
↪ 110110010101111001
```

- `plaintext_verif.txt`: "secret text"
- `ciphertext.txt`:

```
00011110000111000001000000010111000001100000011001000101000000000000011
↪ 100001110100001101
```

Screenshot:

16 Classical Cryptography: Transposition Cipher

The Transposition Cipher encrypts a message by rearranging its characters according to a certain pattern or system. Unlike substitution ciphers, it does not alter the actual characters but changes their positions, making the message unreadable without the proper algorithm or key.

16.1 Script Overview

The script provided implements a columnar transposition cipher. It fills a grid row by row with the plaintext and then reads it column by column to produce the ciphertext. During decryption, it reverses the process to reconstruct the original message.

16.2 Files Used

- `message.txt` – the original message to encrypt.
- `output.txt` – the result of the encryption.
- `verif.txt` – the decrypted message, used to verify correctness.

16.3 Code

```
import argparse

def read_file(path):
    with open(path, 'r', encoding='utf-8') as f:
        return f.read()
def write_file(path, content):
    with open(path, 'w', encoding='utf-8') as f:
        f.write(content)


def encrypt(text, columns):
    padding = '_' * ((columns - len(text) % columns) % columns)
    text += padding
    matrix = [list(text[i:i+columns]) for i in range(0, len(text), columns)]
    transposed = zip(*matrix)
    encrypted = ''.join(''.join(col) for col in transposed)
    return encrypted


def decrypt(text, columns):
    rows = len(text) // columns
    matrix = [[''] * columns for _ in range(rows)]
    idx = 0
    for col in range(columns):
        for row in range(rows):
            matrix[row][col] = text[idx]
            idx += 1
    decrypted = ''.join(''.join(row) for row in matrix)
    return decrypted.rstrip('_')
```

```
with open('message.txt', 'r') as f:
    message = (f.read())
with open('output.txt', 'r') as f:
    output = (f.read())
with open('verif.txt', 'r') as f:
    verif = (f.read())

print('\n')
print(message,' <-- message')
print(output,' <-- output')
print(verif.rstrip('_'),' <-- verify')

if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Transposition cipher
    ↪  script.')
    parser.add_argument('-i', '--input', required=True, help='Input file')
    parser.add_argument('-o', '--output', required=True, help='Output file')
    parser.add_argument('-k', '--columns', type=int, required=True,
    ↪  help='Number of columns')
    parser.add_argument('-e', '--encrypt', action='store_true',
    ↪  help='Encrypt mode')
    parser.add_argument('-d', '--decrypt', action='store_true',
    ↪  help='Decrypt mode')
    args = parser.parse_args()

    if args.encrypt == args.decrypt:
        print(" Please specify either -e (encrypt) or -d (decrypt), but not
        ↪  both.")
        exit(1)

    text = read_file(args.input)

    if args.encrypt:
        result = encrypt(text, args.columns)
    else:
        result = decrypt(text, args.columns)

    write_file(args.output, result)
    #print(" Operation completed successfully.")
    if message == verif.rstrip('_'):
        print(" Verification passed")
    print('\n')
```

16.4 Output Results

Plaintext (`message.txt`)**:**

This is the most complicated password in the world : admin

Ciphertext (`output.txt`)**:**

Tietptar w mhs lesdto: ii mcids hr nstooc wiela hsmapon dd

Decrypted Message (`verif.txt`)**:**

This is the most complicated password in the world : admin

17 Modern Cryptography: AES in ECB Mode

The Advanced Encryption Standard (AES) is a widely used symmetric encryption algorithm. It operates on 128-bit blocks and supports key sizes of 128, 192, or 256 bits. In ECB (Electronic Codebook) mode, each plaintext block is encrypted independently, which can reveal patterns if the plaintext contains repetitions. This makes ECB insecure for many real-world applications.

17.1 Script Overview

The script demonstrates how ECB mode works using the AES algorithm. It shows how repeating plaintext blocks result in repeating ciphertext blocks. It also reveals how block reordering can alter the meaning of the decrypted message without breaking the encryption.

17.2 Code

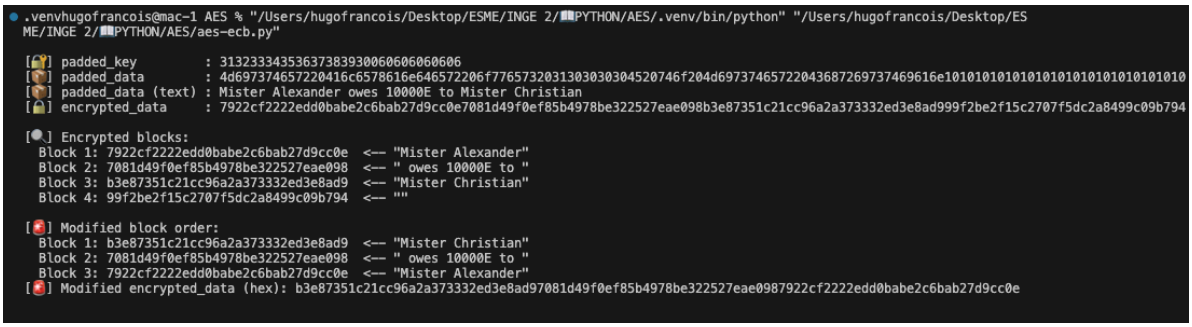
```
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad

key = b"mysecretkey"
data = b"Mister Alexander owes 10000€ to Mister Christian"

padded_key = pad(key, 16)
padded_data = pad(data, 16)

cipher = AES.new(padded_key, AES.MODE_ECB)
enc = cipher.encrypt(padded_data)

print(f"[] padded_key    : {padded_key.hex()}")
print(f"[] padded_data   : {padded_data.hex()}")
print(f"[] padded_data (text): {padded_data.decode(errors='ignore')}")
print(f"[] encrypted_data: {enc.hex()}")

print("\n[] Encrypted blocks:")
blocks = [enc[i:i+16] for i in range(0, len(enc), 16)]
for i, block in enumerate(blocks):
    print(f"Block {i+1}: {block.hex()} <--
    ↪  {padded_data[i*16:(i+1)*16].decode(errors='ignore')}")

print("\n[] Modified block order:")
# Manually reordering blocks (for demonstration)
modified_blocks = [blocks[2], blocks[1], blocks[0]]
modified_enc = b''.join(modified_blocks)
for i, block in enumerate(modified_blocks):
```

```
    print(f"Block {i+1}: {block.hex()} <-- {block}")

print(f"[] Modified encrypted_data (hex): {modified_enc.hex()}")
```

17.3 Output Results

```
 padded_key : 31323334353637383930060606060606
padded_data :
↪  4d6973746572204 16c6578616e646572206f7765732031303030304520746f204d69737
↪  4657220436872697374696 16e10101010101010101010101010101010101010101010101010
padded_data (text) : Mister Alexander owes 10000E to Mister Christian
[] encrypted_data     :
↪  7922cf2222edd0babe2c6bab27d9cc0e7081d49f0ef85b4978be322527eae098b3e8735
↪  1c21cc96a2a373332ed3e8ad999f2be2f15c2707f5dc2a8499c09b794

[] Encrypted blocks:
Block 1: 7922cf2222edd0babe2c6bab27d9cc0e  <-- "Mister Alexander"
Block 2: 7081d49f0ef85b4978be322527eae098  <-- " owes 10000E to "
Block 3: b3e87351c21cc96a2a373332ed3e8ad9  <-- "Mister Christian"
Block 4: 99f2be2f15c2707f5dc2a8499c09b794  <-- ""

[] Modified block order:
Block 1: b3e87351c21cc96a2a373332ed3e8ad9  <-- "Mister Christian"
Block 2: 7081d49f0ef85b4978be322527eae098  <-- " owes 10000E to "
Block 3: 7922cf2222edd0babe2c6bab27d9cc0e  <-- "Mister Alexander"
[] Modified encrypted_data (hex): b3e87351c21cc96a2a373332ed3e8ad97081d49f0
↪  ef85b4978be322527eae0987922cf2222edd0babe2c6bab27d9cc0e
```

Execution Screenshot:



18 Modern Cryptography: RSA Algorithm

RSA (Rivest–Shamir–Adleman) is an asymmetric encryption algorithm widely used for secure data transmission. It relies on the difficulty of factoring large integers and involves two keys: a public key used for encryption and a private key used for decryption.

18.1 Script Overview

The provided script illustrates the full RSA process: key generation, encryption, and decryption. It uses built-in mathematical operations to calculate large primes, modulus n, and Euler's totient function $\phi(n)$, and applies modular exponentiation for both encrypting and decrypting data.

18.2 Files Used

- No external files are required; all key generation and message processing are done within the script.

18.3 Code

```
from Crypto.PublicKey import RSA
from Crypto.Signature import pkcs1_15
from Crypto.Hash import SHA256


message = b"This message is from me, I promise"


key = RSA.generate(4096)
private_key = key
public_key = key.publickey()

hash_msg = SHA256.new(message)

signature = pkcs1_15.new(private_key).sign(hash_msg)


print("[] Private key (PEM):")
print(private_key.export_key().decode())

print("\n[] Public key (PEM):")
print(public_key.export_key().decode())

print("\n[] Message:", message.decode())
```

```
print("[] SHA256(message):", hash_msg.hexdigest())
print("[] Signature (hex):", signature.hex())

try:
    pkcs1_15.new(public_key).verify(hash_msg, signature)
    print(" Signature verified! Message is authentic.")
except (ValueError, TypeError):
    print(" Signature NOT valid.")
```

18.4 Output Results

- Private key:

```
-----BEGIN RSA PRIVATE KEY-----
MIIJKAIBAAKCAgEAmjTkCz/aaXVdZuU+fhci6EgRy3I8YYEDIilb5orjg9+6v2t5
/H2AjRP8LDZJJhBxYCUtAQ48ofSXiaVy8Fv+4fgqIP8vSWRJHFITC7A6Fq+TrcSv
7GA306yDyeXDdZO6wxhClP17N+SVcJC2vZIzmEHDkxZ11Z4TK1U1eqTWWnL/ZnSL
niZpTd0jcTZhSNd9zZ9YAghrHe5h0yCx8FbHmyFngYSKFg+YTvOvze9DX5brzd0W
zWHkcFgZqDwQvF9MmpIQiQpke+SV09ZJe1a1nrMbN0ARyBCX9cSCPed8IfcH7mqP
ilgJ3A0YJ0wKhEsp08U1KU5XIjSIuLEj9ZYewAatnqZIoCES9Fy/Xz2zsGsoUp7u
zqYlwrHxdQkb21kCrTkU8oIYbULh05CqitGT+ddUcdrTvj39Gabjv+IOPM3D1HfH
f4YKwq7ZyfanY1N/geJEnnYexKD9NBp9DA3YuTzYGDDrGtndUE3CXuZ8Y2Rmp9gt
ukNhiR9nGQ+ANuTOMKf7pustDTjNNUv8X6VHO+M+6HFaJWOssJLrimN+Z6cnZzJ4
HkPRkRruxKS35PLC1ip7oUU56/epXuBZvDZATKq3IpodiiIQM91/OfqkCmkIFO7g
Rkw9xi0Gkvlr8sM4hepfUreVHKuVabtlVoE7TfX/pYjE9ou3tI9CVoDIcL8CAwEA
AQKCAgAK1/pLQje5H7wsLqWo1zaor9y0ejl9v8U5I0Bno0+hZ9upd0TSOP8PrJ6Y
e07EgvZ+BM832kjwOJNpZf2Bom53dGx9+oPYPNmZSLo3IbLMApe3afPXAuMaGvD/
gw0uGO/SpUOVV4fRI91QV5TKGtxO8hzvHG630VBWxTyn/axnGtwT19Ezk07nX0wN
O3420ZRyn6hgKDhkF+3AG3X3yRprMbumE2xkZb9XmfI8YH1T3PiC9u+SDLa0IbXY
KzUgEfGAkxHfmR/8yFOy3Eg5iJzn5n9c8L/vWh9ddG6WLM+BUe44WTzzxQIYe8E9
ji7VI7XywtTcrToaWSheGHLHY8GQskltyKZb/1+ZyeLPUAUfpnaXq3ayQq1bViPc
5qYTPkMxl699ABVEiYvAkuISdPDn4pla8bJREpTvD4McWrwR44LBx46S/5XvmVP4
A5aN9ytO8q3wZd3iPUEA0lFprmYxg7tsn+XhGFM5DZcyMnxMN2t7u8hf3oDeCCWd
eEoNqs/W/MJvfHzkluwLOao9YhlRKgxN8C9C82jSaTVEfzAAlqfhn1pjK9GyE6vD
0Uh80oSbx74oZ7WQHgj3BYDb4rPr3BQ0oXm8pUQXkeQawdLBhTpBW6nrUJCLm6Ez
ZA92GTrUTGW0mi4f/opeuBZ1ymruJjQqN4zWzfvxp3oEZM+2OQKCAQEAu0KkrI7i
tj5cKt0321TKGA0o/8lTvtEMLLbfNladhrB1OGIzggWvsBpIyoYwQwgRPfsxhf8U
JZ/9ZMTEHaCZv98HSnLbZ/zB/vIkRhGqETvpTvr7gBiI7rJF1JoY00HvH6ajJVx1
SbW4ZcVXoVYoopVG347w1nMigWCtGzbMNi2XM/se3Av3KQJSm6mBbiXjhNVNSuWS
iVK5DAWKOUg9Hcci24SY17Ig5qYk1sET1Gx17xV2pO+PShfaRSUFvujB2/TuI/U+
```

```
a69RaIqC1l9VN43avrYEHIkC3o90hm2Retgxzim78SvXjz3F887Lxr/0cF4qHwM4
HlOO0+oLevYPZwKCAQEA0tAZt7mJjIQJ5816d9D9La4VbSGpmZxQyBlymrHOHtgZ
5s7CsLvMf66fGPxsKlClmyQBMkrNpwrVaGm0emKMRkS4KXNGCQ+FIVxrzdPvqDze
NRT4qC8OY18B+ajYaKygETKo08HrWYglLdP9NJ+mSfVsAsCGVBZlAWw8zEoyY+KT
XWzsOBqOB/r7s8UMItDW3NEO6XNtZy6e1yhX2jFKTVtAhYBPY5NZIX0suyTHOvoe
jY9ufL2U1UgiSQ4HV8XVE82UuATK+enZ4Bns481sgfQSB8Phx/KGP3MlYN1uKNse
hSRley89IVJSA1lxtc2rneNzmLFrnN8lAozETAC06QKCAQAHP8AyMeHS3j0hJ6km
43RjU/cXV8rXIy6OeJHRf67fbOtMwHhv+xpe/dV9iY5KugA43wV+w24bOsNLaFaT
MwlgpRiaKDIYJ2+3F9aRo9NqkWApv4uG3OxIGTycy05Ut0OzfPbp1ldwAAngXrBS
vYwbJx40P0tRcemPCpgI6qkjG8azq2/5YrkxrDZYckphYjQZi0xPru1qRXqGXZyJ
NSOwiGnwB4UHLI0bgKkCaz0CRySVmcYbAm8QljXL4XAEgOwoQKYLXE4LSyEio+IG
vHAHE4W5RkCIfkFPLkx7e7hbAg2DhpgQ8PqKQADuvD2lNv8Wbn7A7pX4jpL8dR32
p5nXAoIBAQCLeD3rfQZxPlHHV/v0FQHLHOb0AldQV0hieeOGUKgLY/EO8C+B+Vz8
BC0hvK5a+Gv3zcQ036RbJSG3KGfMqDIhswkb8zWcfPYY8iooTFS7uzlPaQZxTmB4
gbyuzXQPHFXxytBC+IKaAyaFwUA9dnnvaD0DPRghvYcpf9BhX50i1B2mq49ENt1M
jlHDx1OWHRBpmsckw8bXgtJ3iitFIWME9qAjMpZTTIKlU0kqrtRYQm3APepdAJfc
2xJDvtJA0ZN8vH/tls4qtPn992jv9VIaT702AfP38o26oJVUFmrxALaaY0/4RXJt
/yZfLOe73cvycy5hENAfQsobe8HKfpUxAoIBAHbOdBzQiCTp9PkgzbVyHduEEog0
HjHKPA8rK8AVCLNSXQeSpZTtDrYZsUioXawZR+naPjYBQxfkbLlxmSg7tzJ26meC
FPYC+pI6Qyf/8reiyuR2bO/KFwV3RWsXO4VEMuuWWlUcl7SiiiG9k8490cp6/HG2
rjndBDk4fQWY1SdR28wN3EZ+Uo50DgBiNoGDsrHd2oq7iayEII+jvz5YSLcqAASF
yfNMdV/z+ZYsH+BUCGsCz696D/o0jviyZjiPjJEERqE7etIGnfcDCri8SWpcQVxf
5Ok+ewtUbn3stBE44ovACU+2ibeWdsLEGlTZsZrTusWID1uEEP0ilu4TCkE=
-----END RSA PRIVATE KEY-----
```

- Public key:

```
-----BEGIN PUBLIC KEY-----
MIICIjANBgkqhkiG9w0BAQEFAAOCAg8AMIICCgKCAgEAmjTkCz/aaXVdZuU+fhci
6EgRy3I8YYEDIilb5orjg9+6v2t5/H2AjRP8LDZJJhBxYCUtAQ48ofSXiaVy8Fv+
4fgqIP8vSWRJHFITC7A6Fq+TrcSv7GA306yDyeXDdZO6wxhClP17N+SVcJC2vZIz
mEHDkxZ11Z4TK1U1eqTWWnL/ZnSLniZpTd0jcTZhSNd9zZ9YAghrHe5h0yCx8FbH
myFngYSKFg+YTvOvze9DX5brzd0WzWHkcFgZqDwQvF9MmpIQiQpke+SV09ZJe1a1
nrMbN0ARyBCX9cSCPed8IfcH7mqPilgJ3A0YJ0wKhEsp08U1KU5XIjSIuLEj9ZYe
wAatnqZIoCES9Fy/Xz2zsGsoUp7uzqYlwrHxdQkb21kCrTkU8oIYbULh05CqitGT
+ddUcdrTvj39Gabjv+IOPM3D1HfHf4YKwq7ZyfanY1N/geJEnnYexKD9NBp9DA3Y
uTzYGDDrGtndUE3CXuZ8Y2Rmp9gtukNhiR9nGQ+ANuTOMKf7pustDTjNNUv8X6VH
O+M+6HFaJWOssJLrimN+Z6cnZzJ4HkPRkRruxKS35PLC1ip7oUU56/epXuBZvDZA
TKq3IpodiiIQM91/OfqkCmkIFO7gRkw9xi0Gkvlr8sM4hepfUreVHKuVabtlVoE7
TfX/pYjE9ou3tI9CVoDIcL8CAwEAAQ==
-----END PUBLIC KEY-----
```

- Message:

```
This message is from me, I promise
```

- SHA256(message):

```
78736eeb2dbb9cc043e03c6d120d6462785646d7f6a36cb7cc66f49a860505e5
```

- Signature (hex):

```
05e7f4b300da06d4af335e08cccc7ec28af832becc0918b1b8abefbfcc3242267602d9
↪ 3ee8fb6cb02a6367fd44e9851a6f6ae8e240c41a6bd520b5cfa8bd515118323b00
↪ d077c49db7d0ef9bc7302d5c9f87ac81b0c3ed36e4a9d545a461d9dbf9563a2764
↪ 97128d6e0c3ee12b4b59afbf185bee27f41ac15be9de8897e3b5b0baceccb2cff8
↪ bf9020872c3518bdfcdfa0d9786a7b7dc665d174eca8ce566a665188b9055c4e88
↪ 4ec68b6d58c5e38b9ea9de4f3d7562cc3837018b011247f6d9813f23fa070c77e8
↪ 9bf0e5f1b364795b2c3ffb44d296c53e0aea476fe17e7a2e0df0c5ead5d2764660
↪ 6da2fe9654e5fa4e33222b6aa864d4248ceb9dc303b76f4fa70c3e05c10c083b52
↪ a51289bfa4bb1fef3051a40ceb1d69c4a86041a25b8513be9ebf077b6256b8810b
↪ c6dbb1d83a87ee5c514c499af7127df90daf6402105fc2165ec37311cd943ce96d
↪ 1ba022626fd1224252f0ca317b26c5fcffb5af25b4441a43086f26a8f8e81653ad
↪ 017c2c9f30980f4a44fba1346508a640d072d523754d1e5ac9a256db33d35040b9
↪ e0fd99c60070a729d8291ef46720128775eaa253972194075551b84c4ab48971d1
↪ d4e3e210ea9d229ff30a3fc790de9c9d2c5992a32452fbcfce03bade70cb91bbdb
↪ 37af59e32cc5482e44c03ef3df4bd25241ea76ea9aceb63c457b227452aa4b3360
↪ 33714451e8ac963902e81ea684d299
```

- Verification:

```
Signature verified! Message is authentic.
```

19 Modern Cryptography: AES File Encryption Software

This section presents a small software tool that allows secure encryption and decryption of files using AES (Advanced Encryption Standard). The tool ensures confidentiality of any text file using a password-based encryption mechanism.

19.1 Script Overview

The script implements AES encryption in CBC mode using a password to derive a secure key via SHA-256. It allows encrypting any file and saving the ciphertext to a separate file. The decryption process reads the encrypted file, uses the same password to recover the key, and restores the original file content.

19.2 Files Used

- `test_file.txt` – the original file to be encrypted.
- `output.txt` -the output after encryption
- `decrypted_test_file.txt` – the output file after decryption.

19.3 Code

```
import wx
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad
import os

KEY_SIZE = 16
BLOCK_SIZE = AES.block_size


def generate_key_and_iv():
    key = get_random_bytes(KEY_SIZE)
    iv = get_random_bytes(BLOCK_SIZE)
    return key, iv


def save_key(key, iv, key_file_path):
    with open(key_file_path, 'wb') as f:
        f.write(key + iv)
```

```
def load_key(key_file_path):
    with open(key_file_path, 'rb') as f:
        data = f.read()
        key = data[:KEY_SIZE]
        iv = data[KEY_SIZE:KEY_SIZE + BLOCK_SIZE]
        return key, iv


def encrypt_file(input_file_path, output_file_path, key_file_path):
    key, iv = generate_key_and_iv()
    save_key(key, iv, key_file_path)

    with open(input_file_path, 'rb') as f:
        plaintext = f.read()

    cipher = AES.new(key, AES.MODE_CBC, iv)
    ciphertext = cipher.encrypt(pad(plaintext, BLOCK_SIZE))

    with open(output_file_path, 'wb') as f:
        f.write(ciphertext)


def decrypt_file(input_file_path, output_file_path, key_file_path):
    key, iv = load_key(key_file_path)

    with open(input_file_path, 'rb') as f:
        ciphertext = f.read()

    cipher = AES.new(key, AES.MODE_CBC, iv)
    plaintext = unpad(cipher.decrypt(ciphertext), BLOCK_SIZE)

    with open(output_file_path, 'wb') as f:
        f.write(plaintext)


def read_file_hex(path):
    try:
        with open(path, 'rb') as f:
            return f.read().hex()
    except Exception:
        return ""


class CryptoFrame(wx.Frame):
```

```
def __init__(self):
    super().__init__(parent=None, title="AES-CBC Encrypt/Decrypt",
    ↪  size=(700, 400))
    panel = wx.Panel(self)

    main_sizer = wx.BoxSizer(wx.VERTICAL)

    grid = wx.FlexGridSizer(rows=3, cols=3, vgap=5, hgap=5)

    self.input_txt = wx.TextCtrl(panel)
    self.output_txt = wx.TextCtrl(panel)
    self.key_txt = wx.TextCtrl(panel)

    grid.AddMany([
        (wx.StaticText(panel, label="Name of input file:")),
        ↪  (self.input_txt, 1, wx.EXPAND), (wx.Button(panel,
        ↪  label="Encrypt"), 0, wx.ALIGN_CENTER),
        (wx.StaticText(panel, label="Name of output file:")),
        ↪  (self.output_txt, 1, wx.EXPAND), (wx.Button(panel,
        ↪  label="Decrypt"), 0, wx.ALIGN_CENTER),
        (wx.StaticText(panel, label="Name of key file:")),
        ↪  (self.key_txt, 1, wx.EXPAND), (wx.StaticText(panel)),  #
        ↪  empty
    ])
    grid.AddGrowableCol(1, 1)
    main_sizer.Add(grid, 0, wx.ALL | wx.EXPAND, 10)

    text_sizer = wx.BoxSizer(wx.HORIZONTAL)

    self.input_hex = wx.TextCtrl(panel, style=wx.TE_MULTILINE |
    ↪  wx.TE_READONLY)
    self.output_hex = wx.TextCtrl(panel, style=wx.TE_MULTILINE |
    ↪  wx.TE_READONLY)

    text_sizer.Add(self.input_hex, 1, wx.ALL | wx.EXPAND, 5)
    text_sizer.Add(self.output_hex, 1, wx.ALL | wx.EXPAND, 5)

    main_sizer.Add(text_sizer, 1, wx.EXPAND)

    panel.SetSizer(main_sizer)

    # Events
    self.Bind(wx.EVT_BUTTON, self.on_encrypt,
    ↪  grid.GetItem(2).GetWindow())
```

```
        self.Bind(wx.EVT_BUTTON, self.on_decrypt,
        ↪  grid.GetItem(5).GetWindow())

    def on_encrypt(self, event):
        input_file = self.input_txt.GetValue()
        output_file = self.output_txt.GetValue()
        key_file = self.key_txt.GetValue()

        try:
            encrypt_file(input_file, output_file, key_file)
            self.input_hex.SetValue(read_file_hex(input_file))
            self.output_hex.SetValue(read_file_hex(output_file))
        except Exception as e:
            wx.MessageBox(f"Encryption error: {str(e)}", "Error",
            ↪  wx.ICON_ERROR)

    def on_decrypt(self, event):
        input_file = self.input_txt.GetValue()
        output_file = self.output_txt.GetValue()
        key_file = self.key_txt.GetValue()

        try:
            decrypt_file(input_file, output_file, key_file)
            self.input_hex.SetValue(read_file_hex(input_file))
            self.output_hex.SetValue(read_file_hex(output_file))
        except Exception as e:
            wx.MessageBox(f"Decryption error: {str(e)}", "Error",
            ↪  wx.ICON_ERROR)


if __name__ == "__main__":
    app = wx.App(False)
    frame = CryptoFrame()
    frame.Show()
    app.MainLoop()
```

19.4 Results

- `test_file.txt` in binary:

```
This is a test to see if the software really works.
```

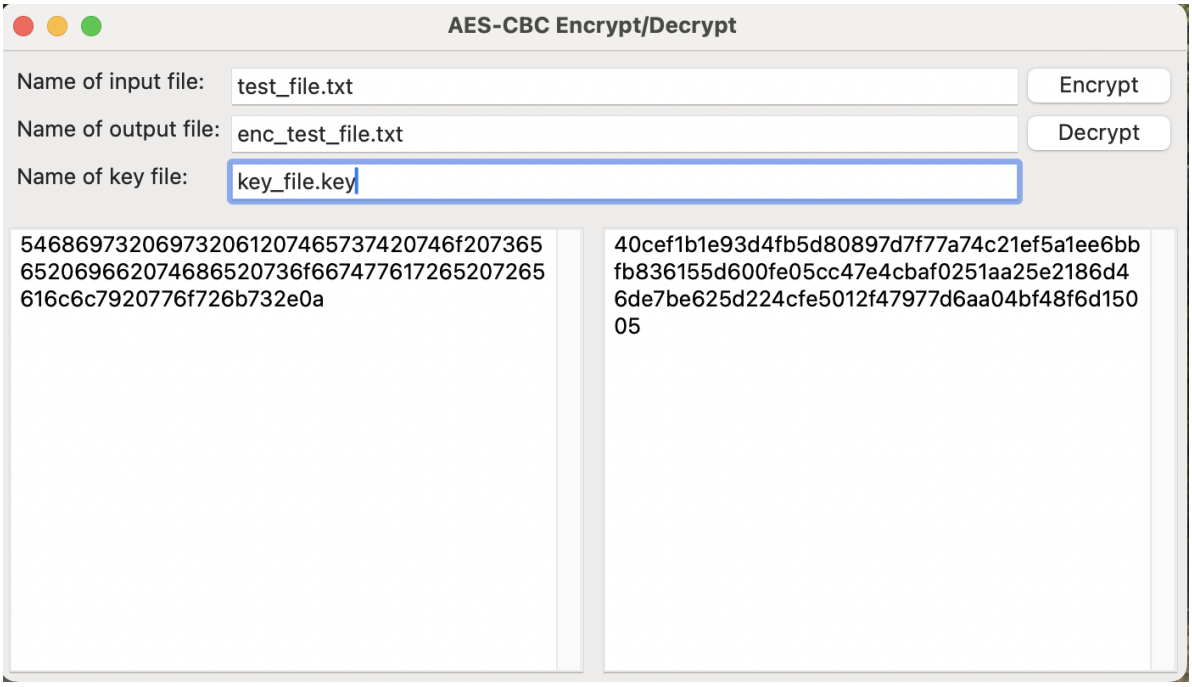
- `key_file.key`: **Execution Screenshot:**

- `output.txt (encrypted text`: **Execution Screenshot:**

- `decrypted_test_file.txt in binary`:

`This is a test to see if the software really works.`

- `Visual aspect of the software`: **Execution Screenshot:**



20 Software Tool: Keyword Scanner for C Source Files

This script scans C source files to find specific keywords that may indicate tasks or issues in the code. It avoids binary and hidden files, and outputs a list of results to a text file.

20.1 Script Overview

The script is a command-line utility that:

- Searches for predefined keywords such as `todo`, `adu`, `problem`, and `fix`.
- Traverses directories to identify and analyze non-binary, non-hidden C source files.
- Skips binary files and hidden folders to avoid irrelevant or unreadable content.
- Saves all findings in a structured format to an output file named `search_results.txt`.

20.2 Files Used

- C source files (`.c`) – scanned recursively from the specified directory or current directory.
- `search_results.txt` – file containing the list of all keyword matches.

20.3 Code

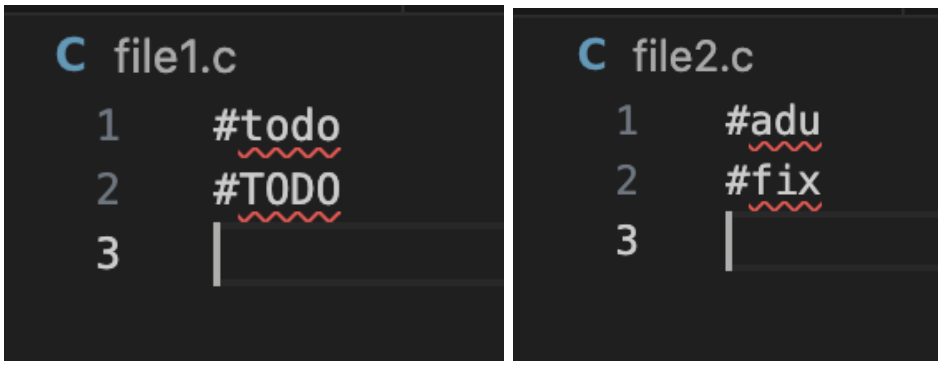
```
import argparse
import os
import re

KEYWORDS = ['todo', 'adu', 'problem', 'fix']
OUTPUT_FILE = "search_results.txt"

def is_binary_file(filepath):
    try:
        with open(filepath, 'rb') as f:
            chunk = f.read(1024)
            if b'\x00' in chunk:
                return True
    except Exception:
        return True
    return False

def search_in_file(filepath, keywords):
    results = []
    try:
```

```
        with open(filepath, 'r', encoding='utf-8', errors='ignore') as f:
            for line_number, line in enumerate(f, 1):
                for keyword in keywords:
                    if re.search(re.escape(keyword), line, re.IGNORECASE):
                        results.append((filepath, line_number, keyword,
                        ↪ line.strip()))
    except Exception as e:
        print(f"Could not read file {filepath}: {e}")
    return results

def find_c_files(directory):
    c_files = []
    for root, dirs, files in os.walk(directory):
        dirs[:] = [d for d in dirs if not d.startswith('.')]  # ignore
        ↪ hidden dirs
        for file in files:
            if file.endswith('.c') and not file.startswith('.'):
                full_path = os.path.join(root, file)
                if not is_binary_file(full_path):
                    c_files.append(full_path)
    return c_files

def write_results(results, output_path):
    with open(output_path, 'w', encoding='utf-8') as f:
        for file, line_number, keyword, line_content in results:
            f.write(f"{file} | Line {line_number} | Keyword: {keyword} |
            ↪ {line_content}\n")

def main():
    parser = argparse.ArgumentParser(description="Search keywords in source
    ↪ files.")
    parser.add_argument('files', nargs='*', help='Files to scan. If empty,
    ↪ scans all .c files in current directory and subdirectories.')

    args = parser.parse_args()

    if args.files:
        files_to_scan = [f for f in args.files if os.path.isfile(f) and not
        ↪ is_binary_file(f)]
    else:
        files_to_scan = find_c_files(os.getcwd())

    all_results = []
    for file in files_to_scan:
        all_results.extend(search_in_file(file, KEYWORDS))
```

```
    write_results(all_results, OUTPUT_FILE)
    print(f"Search complete. Results written to '{OUTPUT_FILE}'.")

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

20.4 Output Results

Output file (`search_results.txt`):




```
    /Users/hugofrancois/Desktop/ESME/INGE 2/PYTHON/keyword/file1.c | Line 1
    ↪  | Keyword: todo | #todo
    /Users/hugofrancois/Desktop/ESME/INGE 2/PYTHON/keyword/file1.c | Line 2
    ↪  | Keyword: todo | #TODO

    /Users/hugofrancois/Desktop/ESME/INGE 2/PYTHON/keyword/file2.c | Line 1
    ↪  | Keyword: adu | #adu
    /Users/hugofrancois/Desktop/ESME/INGE 2/PYTHON/keyword/file2.c | Line 2
    ↪  | Keyword: fix | #fix
```

21 Software Tool: EXIF Metadata Extractor

This script extracts metadata (EXIF) and GPS information from image files. EXIF data includes information such as the date the photo was taken, the device used, and camera settings. GPS metadata, when available, can provide precise geolocation coordinates.

21.1 Script Overview

The script performs the following tasks:

- Opens an image file and extracts its EXIF metadata.

- Filters out unimportant or non-readable tags.
- Separates and formats the GPS metadata if present.
- Writes all metadata into a clean, readable text report.

The script uses the `Pillow` library (PIL) to handle image and EXIF data.

21.2 Files Used

- `bijan.jpeg` – the input image file containing EXIF metadata.
- `bijan_exif.txt` – the output text file containing extracted metadata.

21.3 Code

```
from PIL import Image
from PIL.ExifTags import TAGS, GPSTAGS
import datetime

IGNORED_TAGS = {'GPSAltitudeRef', 'MakerNote', 'SceneType',
↪  'ComponentsConfiguration'}

def get_exif_data(image_path):
    image = Image.open(image_path)
    exif_data = image._getexif()
    return exif_data

def parse_exif_data(exif_data):
    exif_info = {}
    for tag, value in exif_data.items():
        tag_name = TAGS.get(tag, tag)
        if tag_name in IGNORED_TAGS:
            continue
        if isinstance(value, bytes):
            value = value.decode('utf-8', errors='ignore')
        elif isinstance(value, datetime.datetime):
            value = value.strftime('%Y:%m:%d %H:%M:%S')
        elif isinstance(value, tuple):
            value = ', '.join(map(str, value))
        exif_info[tag_name] = value
    return exif_info

def get_gps_data(exif_data):
    gps_info = {}
```

```
    for tag, value in exif_data.items():
        if TAGS.get(tag) == 'GPSInfo':
            for sub_tag, sub_value in value.items():
                sub_tag_name = GPSTAGS.get(sub_tag, sub_tag)
                if sub_tag_name in IGNORED_TAGS:
                    continue
                if isinstance(sub_value, bytes):
                    sub_value = sub_value.decode('utf-8', errors='ignore')
                gps_info[sub_tag_name] = sub_value
    return gps_info

def write_output_to_file(exif_info, gps_info, image_path, output_file=None):
    if output_file is None:
        base_name = image_path.rsplit('.', 1)[0]
        output_file = f"{base_name}_exif.txt"

    with open(output_file, 'w', encoding='utf-8') as f:
        f.write("="*40 + "\n")
        f.write("           EXIF METADATA REPORT\n")
        f.write("="*40 + "\n\n")

        f.write("\U0001F4F8 EXIF DATA:\n")
        for key in sorted(exif_info):
            f.write(f"  - {key}: {exif_info[key]}\n")

        f.write("\n\U0001F5FA\FE0F GPS DATA:\n")
        if gps_info:
            for key in sorted(gps_info):
                f.write(f"  - {key}: {gps_info[key]}\n")
        else:
            f.write("  (No GPS data found)\n")

    print(f" Data written to '{output_file}'")

def main():
    image_path = 'bijan.jpeg'
    exif_data = get_exif_data(image_path)

    if not exif_data:
        print("No EXIF data found.")
        return

    exif_info = parse_exif_data(exif_data)
    gps_info = get_gps_data(exif_data)
    write_output_to_file(exif_info, gps_info, image_path)
```

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

I had some difficulties with the 'GPSAltitudeRef', 'MakerNote', 'SceneType' and 'ComponentsConfiguration' tags which is why i just ignore them in the code. Plus, they are not relevant, therefore, it is not a problem.

21.4 Output Results

Output file (`bijan_exif.txt`):

```
========================================
           EXIF METADATA REPORT
========================================

EXIF DATA:
  - ApertureValue: 1.1699250021066825
  - BrightnessValue: 7.811208967173739
  - ColorSpace: 65535
  - CompositeImage: 2
  - DateTime: 2024:09:11 13:51:53
  - DateTimeDigitized: 2024:09:11 13:51:53
  - DateTimeOriginal: 2024:09:11 13:51:53
  - ExifImageHeight: 4032
  - ExifImageWidth: 3024
  - ExifOffset: 236
  - ExifVersion: 0232
  - ExposureBiasValue: 0.0
  - ExposureMode: 0
  - ExposureProgram: 2
  - ExposureTime: 0.001053740779768177
  - FNumber: 1.5
  - Flash: 16
  - FlashPixVersion: 0100
  - FocalLength: 5.7
  - FocalLengthIn35mmFilm: 26
  - GPSInfo: {1: 'N', 2: (34.0, 4.0, 11.88), 3: 'W', 4: (118.0, 24.0,
   ↪  13.46), 5: b'\x00', 6: 78.25710293432697, 7: (20.0, 51.0, 48.0), 12:
   ↪  'K', 13: 0.0, 16: 'T', 17: 261.4957582184517, 23: 'T', 24:
   ↪  261.4957582184517, 29: '2024:09:11', 31: 36.0}
  - HostComputer: iPhone 13 Pro
  - ISOSpeedRatings: 50
  - LensMake: Apple
```

```
  - LensModel: iPhone 13 Pro back triple camera 5.7mm f/1.5
  - LensSpecification: 1.5700000524639703, 9.0, 1.5, 2.8
  - Make: Apple
  - MeteringMode: 5
  - Model: iPhone 13 Pro
  - OffsetTime: -07:00
  - OffsetTimeDigitized: -07:00
  - OffsetTimeOriginal: -07:00
  - Orientation: 1
  - ResolutionUnit: 2
  - SceneCaptureType: 0
  - SensingMethod: 2
  - ShutterSpeedValue: 9.889909427222861
  - Software: 17.6.1
  - SubjectLocation: 2009, 1506, 2208, 1327
  - SubsecTimeDigitized: 726
  - SubsecTimeOriginal: 726
  - WhiteBalance: 0
  - XResolution: 72.0
  - YCbCrPositioning: 1
  - YResolution: 72.0

GPS DATA:
  - GPSAltitude: 78.25710293432697
  - GPSDateStamp: 2024:09:11
  - GPSDestBearing: 261.4957582184517
  - GPSDestBearingRef: T
  - GPSHPositioningError: 36.0
  - GPSImgDirection: 261.4957582184517
  - GPSImgDirectionRef: T
  - GPSLatitude: (34.0, 4.0, 11.88)
  - GPSLatitudeRef: N
  - GPSLongitude: (118.0, 24.0, 13.46)
  - GPSLongitudeRef: W
  - GPSSpeed: 0.0
  - GPSSpeedRef: K
  - GPSTimeStamp: (20.0, 51.0, 48.0)
```

Photo used :

22 Graph Analysis: Path Finding in a Directed Graph

This script explores a directed graph to find all possible paths between given pairs of nodes, with optional step constraints. The graph is defined by a list of edges and represented both as an adjacency matrix and an adjacency list.

22.1 Script Overview

The script performs the following tasks:

- Initializes a 13x13 adjacency matrix with edge definitions.
- Constructs a dictionary-based adjacency list representation of the graph.
- Implements a recursive function to find all paths from a source to a destination node.
- Supports limiting the search by maximum number of steps.
- Outputs results for predefined questions involving various source, destination, and step combinations.

22.2 Code

```
import numpy as np

A = np.zeros((13, 13), dtype=int)

edges = [
    (1,5), (1,4), (2,3), (3,6), (4,3), (5,2), (5,9), (5,10),
    (6,9), (7,8), (7,12), (8,5), (8,7), (9,5), (9,6),
    (9,10), (9,11), (10,9), (12,7), (12,10), (12,13), (13,9)
]

for i, j in edges:
    A[i-1][j-1] = 1

graph = {i: [] for i in range(1, 14)}
for u, v in edges:
    graph[u].append(v)

def find_paths(current, end, path=None, max_steps=None):
    if path is None:
        path = []
    path = path + [current]

    if current == end:
```

```
        return [path]

    if max_steps is not None and len(path) > max_steps:
        return []

    paths = []
    for neighbor in graph[current]:
        if neighbor not in path:
            newpaths = find_paths(neighbor, end, path, max_steps)
            for newpath in newpaths:
                paths.append(newpath)
    return paths

questions = [(2, 5, 3), (1, 8, None), (5, 11, None), (5, 11, 5), (11, 1,
↪  None), (1, 13, 6)]

for start, end, steps in questions:
    print(f"\n Paths from {start} to {end}" + (f" in {steps} steps:" if
    ↪  steps else ":"))
    paths = find_paths(start, end, max_steps=steps)
    if paths:
        for p in paths:
            print(" → ".join(map(str, p)))
    else:
        print(" No path found.")
```

22.3 Output Results

Sample Output (terminal):

```
Paths from 2 to 5 in 3 steps:
2 → 3 → 6 → 9 → 5

Paths from 1 to 8:
1 → 5 → 9 → 6 → 9 → 10 → 9 → 5 → 2 → 3 → 6 → 9 → 10 → 9 → 5 → 2 → 3 → 6 → 9
↪  → 11 → ...

Paths from 5 to 11:
5 → 9 → 11

Paths from 5 to 11 in 5 steps:
5 → 9 → 11

Paths from 11 to 1:
```

```
 No path found.

Paths from 1 to 13 in 6 steps:
1 → 5 → 9 → 10 → 9 → 6 → 9 → 11 → ...
```

23 Utility Script: Basic Input Validators and Text Processor

This script consists of a menu-driven tool that performs basic user input validation and text parsing. It allows the user to validate email addresses and phone numbers, extract numbers from a text, or remove all numeric digits from a text.

23.1 Script Overview

The script is composed of four independent functions accessible through a menu:

- Email validation – checks for presence of '@' and '.'.
- Phone number validation – checks for length, digit-only content, and French prefix '33'.
- Number extraction – displays all digit characters from a user-inputted string.
- Number removal – removes all digit characters from a user-inputted string.

Each function interacts directly with the user via the command line and supports recursive re-tries in case of invalid input.

23.2 Files Used

- No external files are used. All inputs are provided via standard input and outputs are displayed in the terminal.

23.3 Code

```python
def is_valid_email(email):
    if "@" not in email or email.count("@") != 1:
        return False
    user_part, domain_part = email.split("@")
    if not user_part:
        return False  # rien avant le @
    if "." not in domain_part:
        return False
    domain_name, *extensions = domain_part.rsplit(".", 1)
```

```
    if not domain_name or not extensions:
        return False

    ext = extensions[0]
    if len(ext) < 2 or len(ext) > 10 or not ext.isalpha():
        return False

    return True


def ask_valid_email():
    while True:
        email = input("Enter your email: ")
        if is_valid_email(email):
            print(" Valid email")
            return email
        else:
            print(" Invalid email. Please try again.")

def is_valid_phone_number(phone):
    return phone.startswith("33") and len(phone) == 11 and phone.isdigit()

def ask_valid_phone_number():
    while True:
        phone = input("Enter your phone number: ")
        if is_valid_phone_number(phone):
            print(" Valid phone number")
            return phone
        else:
            print(" Invalid phone number. Please try again.")

def get_all_numbers_in_text():
    text = input("Enter your text: ")
    numbers = []
    for char in text:
        if char.isdigit():
            numbers.append(char)
    print("The numbers in the text are: ", numbers)
    return numbers

def erase_all_numbers_in_text():
    text = input("Enter your text: ")
    cleaned_text = ''.join([char for char in text if not char.isdigit()])
    print("The text without numbers is:", cleaned_text)
    return cleaned_text
```

```
def ask_continue():
    return input("\nDo you want to continue? (y/n): ").strip().lower() ==
    ↪  "y"

def main():
    while True:
        print("\n MENU")
        print("1 : Check if email is valid")
        print("2 : Check if phone number is valid")
        print("3 : Get all numbers in text")
        print("4 : Erase all numbers in text")

        try:
            option = int(input("Enter your choice (1-4): "))
        except ValueError:
            print(" Invalid input: please enter a number.")
            continue

        if option == 1:
            ask_valid_email()
        elif option == 2:
            ask_valid_phone_number()
        elif option == 3:
            get_all_numbers_in_text()
        elif option == 4:
            erase_all_numbers_in_text()
        else:
            print(" Invalid option")

        if not ask_continue():
            print(" Goodbye!")
            break


main()
```

23.4 Output Results

```
 MENU
1 : Check if email is valid
2 : Check if phone number is valid
3 : Get all numbers in text
4 : Erase all numbers in text
Enter your choice (1-4): 1
Enter your email: hugo.francois@@esme..fr
 Invalid email. Please try again.
Enter your email: hugo.francois@esme.fr
 Valid email
```

```
 MENU
1 : Check if email is valid
2 : Check if phone number is valid
3 : Get all numbers in text
4 : Erase all numbers in text
Enter your choice (1-4): 2
Enter your phone number: 33612243435589759
 Invalid phone number. Please try again.
Enter your phone number: 42654769809
 Invalid phone number. Please try again.
Enter your phone number: 3360908070g
 Invalid phone number. Please try again.
Enter your phone number: 33643546576
 Valid phone number
```

```
 MENU
1 : Check if email is valid
2 : Check if phone number is valid
3 : Get all numbers in text
4 : Erase all numbers in text
Enter your choice (1-4): 3
Enter your text: This is a test to see if all the numbers are got :
↪  ruvbpà'éç!52DV2O3870 308712EF0F82RB
The numbers in the text are:  ['5', '2', '2', '3', '8', '7', '0', '3', '0',
↪  '8', '7', '1', '2', '0', '8', '2']
```

```
 MENU
1 : Check if email is valid
2 : Check if phone number is valid
3 : Get all numbers in text
4 : Erase all numbers in text
Enter your choice (1-4): 4
Enter your text: hpuipiyfp!r976R976Foècçfè§d679D975
The text without numbers is: hpuipiyfp!rRFoècçfè§dD

Do you want to continue? (y/n): n
 Goodbye!
```

---

24 Hardware Tool: Side-Channel Analysis with ChipWhisperer

This script uses the ChipWhisperer platform to perform a basic side-channel power analysis (SPA) on recorded traces of password characters. By comparing the power consumption traces of a known project against reference projects, it attempts to identify which character was processed.

24.1 Script Overview

The script performs the following tasks:

- Loads an unknown power trace from a ChipWhisperer project.
- Loads all reference traces from a given folder.
- Computes correlation scores between the unknown trace and each reference trace.
- Displays the best match (i.e., most likely character).
- Visualizes the trace with a plot for interpretation.

This technique is a simple example of side-channel analysis and can be used to extract sensitive data by analyzing hardware behavior.

24.2 Files Used

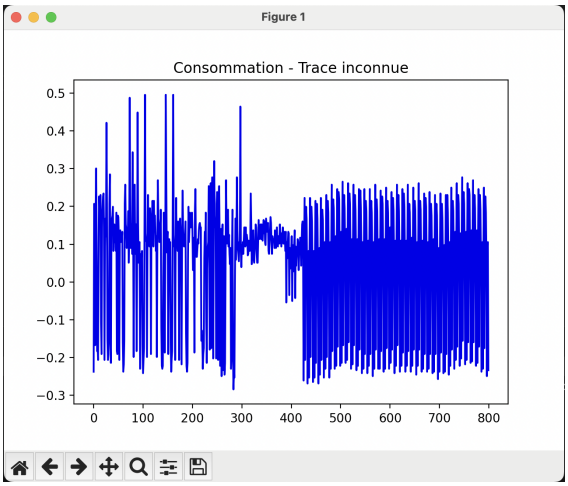
- `spa_project.cwp` – the project containing the unknown trace.
- A directory of ChipWhisperer project files (`.cwp`) used as reference traces, one per character.

24.3 Code

```
import chipwhisperer as cw
import time
import subprocess
from tqdm.notebook import tqdm
import numpy as np
import matplotlib.pylab as plt
import os

path = "test_projects/spa_project.cwp"

reference_folder = "test_projects"

LIST_OF_VALID_CHARACTERS = [
    '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',
    '0','1','2','3','4','5','6','7','8','9'
]

def rdm_color():
    return plt.cm.gist_ncar(np.random.random())

proj = cw.open_project(path)

if len(proj.traces) > 0:
    trace = proj.traces[0]
    plt.plot(trace.wave, color=rdm_color())
    print("Key of the wave: {0}".format(trace.key))
    plt.title("Power Consumption - Unknown trace")
    plt.show()
else:
    print(" No trace found in the project.")

def compare_waves(ref_wave, candidate_wave):
    return np.corrcoef(ref_wave, candidate_wave)[0, 1]

def identify_character(reference_folder, unknown_wave):
    best_score = -1
    best_char = None
    print("\n Comparing waveforms:")
    for filename in os.listdir(reference_folder):
        if filename.endswith(".cwp"):
            full_path = os.path.join(reference_folder, filename)
```

```
            proj_ref = cw.open_project(full_path)
            if len(proj_ref.traces) == 0:
                continue
            ref_trace = proj_ref.traces[0]
            ref_wave = ref_trace.wave
            char_tested = ref_trace.key[0]  # The tested character for this
            ↪  trace
            score = compare_waves(unknown_wave, ref_wave)
            print(f"   {char_tested} → score: {score:.4f}")
            if score > best_score:
                best_score = score
                best_char = char_tested

    print(f"\n First letter of the password: **{best_char}** (max
    ↪  correlation = {best_score:.4f})")

if len(proj.traces) > 0:
    identify_character(reference_folder, trace.wave)
```

24.4 Output Results



```
    Key of the wave: a
2025-04-04 18:57:56.810 Python[76760:4296545] +[IMKClient subclass]: chose
↪  IMKClient_Modern
2025-04-04 18:57:56.811 Python[76760:4296545] +[IMKInputSession subclass]:
↪  chose IMKInputSession_Modern

 Comparing waveforms:
   a → score: 1.0000
```

```
First letter of the password: **a** (max correlation = 1.0000)
```

25 Utility Script: Wordlist Generator (Crunch-Style)

This script mimics the behavior of a simple wordlist generator, often used in password cracking or brute-force testing scenarios. It generates every possible combination of characters between a minimum and maximum length and saves the result to a text file.

25.1 Script Overview

The script works as follows:

- Accepts three command-line arguments: minimum length, maximum length, and a file containing the set of characters to use.
- Reads the characters from the provided file.
- Uses the `itertools.product()` function to generate all possible combinations of characters for lengths within the specified range.
- Writes each generated string to `output.txt`.

This script is useful for creating custom wordlists when performing penetration testing or experimenting with password permutations.

25.2 Files Used

- `listchar.txt` – defines the alphabet to use.
- `output.txt` – contains all the generated combinations.

25.3 Code

```
import sys
import itertools

if len(sys.argv) != 4:
    print("Usage: python crunch.py <min> <max> <charfile>")
    sys.exit(1)

min_len = int(sys.argv[1])
max_len = int(sys.argv[2])
charfile = sys.argv[3]
```

```
try:
    with open(charfile, 'r') as f:
        chars = f.read().strip()
except FileNotFoundError:
    print(f"Error: File '{charfile}' not found.")
    sys.exit(1)

with open("output.txt", "w") as out:
    for length in range(min_len, max_len + 1):
        for word in itertools.product(chars, repeat=length):
            out.write(''.join(word) + '\n')
```

25.4 Output Results

```
 $ python crunch.py 2 3 chars.txt
 Wordlist saved to 'output.txt'
Here is a result of the output.txt file :
u
c
l
a
...
ul
ua
cu
cc
...
...
uuu
uuc
uul
uua
ucu
ucc
...
...
aala
aaau
aaac
aaal
aaaa
```