

Introduction to Cryptography

## **CSCI 462**

## **Project 1**

# **Authors:**

# *Giselle Creado (ghc4716), Christin Alex (caa8062), Eslam Tarrum (et9540)*

# **Professor:**

# *Dr. Wesam Almobaideen*

# **Date of submission:**

# *17th October 2022*

## Table of Contents

# **Section Name ………………………………………….. Page Number**

# [Introduction 3](#_Toc116640398)

# [Objectives 4](#_Toc116640399)

# [Description Of Algorithms 4](#_Toc116640400)

# [Development Of Encryption And Decryption Algorithm 5](#_Toc116640401)

# [Implementation Specifications 6](#_Toc116640402)

# [Summary 9](#_Toc116640403)

# [References 9](#_Toc116640404)

# [Appendix 10](#_Toc116640405)

## Introduction

How may cryptography be defined? Cryptography is a subfield of cryptology that analyzes different encryption schemes, such as cryptographic systems and ciphers. In today's world, digital businesses are booming, and the global society is becoming increasingly reliant on big data and secure transactions and communications [1]. Cryptography serves an important role in preserving data integrity, so that users of any platform can be certain that the information they receive is legitimate and has not been tampered with during transmission. In order to offer reliable and accessible information systems, cryptography assures data availability and non-repudiation of parties participating in the transaction, in addition to data integrity. Cryptographic keys, digital certificates, and cryptographic algorithms are a few examples of core elements in the cryptographic layer [2].

Two basic building blocks of classical encryption algorithms are substitution and transposition cipher techniques which are utilized to convert plaintext to ciphertext. In a nutshell, substitution involves replacing plaintext letters with other letters, whereas transposition requires rearranging plaintext letters in a different order [3]. Substitution and transposition ciphers can be monoalphabetic or polyalphabetic, indicating the use of one or several substitutions and transpositions.

Two classical encryption techniques, monoalphabetic cipher and Caesar cipher can be classified as substitution techniques. Caesar cipher or also known as a shift cipher is one of the most primitive examples of the substitution technique and involves substituting each letter with a letter that is three positions further down than the letter it is replacing. Monoalphabetic cipher involves a direct substitution of the plain text with another letter of the alphabet. It uses a constant key which consists of the randomized order of the 26 letters in the alphabet [4].

This project employs and implements the idea of product cipher, where two ciphers, monoalphabetic cipher and Caesar cipher will be used to encrypt and decrypt plaintext inputted from the user. For any inputted plaintext, all even indexed elements will be encrypted and decrypted using a monoalphabetic cipher and all odd indexed elements will be encrypted and decrypted using Caesar cipher. An amendment is made to the Caesar cipher, where the key (the shift) changes for each odd element, and the key value is determined by the position of the preceding even element.

## Objectives

The primary objective of this project is to develop a strong and efficient encryption and decryption algorithm for the product cipher of two substitution techniques, monoalphabetic cipher and Caesar cipher.

Other objectives include:

* Develop the algorithm in a systematic way that adopts good programming principles.
* Understand the underlaying methodology of integrating Caesar cipher and monoalphabetic cipher to encrypt plaintext
* Convey our understanding of the methodology used to the user, and display it while outputting the ciphertext and/or decrypted plaintext.
* Ensure our encryption and decryption algorithm works for complex inputs provided by the user.

## Description of algorithms

In our project, there are two main algorithms that we develop: encryption algorithm and decryption algorithm.

The encryption algorithm mainly comprises of integrating the two substitution techniques, monoalphabetic cipher and Caesar cipher. The algorithm first begins by prompting input from the user, which is then treated as plaintext. This plaintext is then divided into two halves, where even-indexed elements and odd-indexed elements are separated. The even-indexed elements get passed to the monoalphabetic cipher function, where each even-indexed element is substituted by the corresponding letter of the same position value in the key table. The odd-indexed elements get passed to the Caesar cipher function, along with the original array of the even-indexed elements. In this function, each odd-indexed element is shifted further down the alphabet list based on the value n, n being the position of the preceding even-indexed element in the alphabet list. Once new elements have been generated by both these functions, we concatenate the results and place the new elements in the respective positions of the elements they have substituted.

The decryption algorithm is similar to the encryption algorithm but the steps implemented in the respective substitution techniques differ. Just like encryption, the decryption algorithm first begins by prompting input from the user but the input will be treated as ciphertext that has to be decrypted. The elements in the ciphertext are divided based on its odd and even index and are respectively sent to Caesar cipher and monoalphabetic cipher. The monoalphabetic cipher function in this algorithm searches for the provided element in the key table (symmetric key, the same key used in the encryption algorithm) and substitutes this element with the corresponding letter of the same position value in the alphabet list. The Caesar cipher in the decryption algorithm operates in a reverse manner to the one found in the encryption algorithm. Instead of shifting down the alphabet, we shift up the alphabet based on the value n, n being the alphabetical position of the previous even-indexed element. The results of these functions are then concatenated to produce the intended plaintext, where all the even and odd-indexed elements are placed in the respective positions of the elements they have substituted.

The user is expected to enter an additional argument while running the main programming file in order to show the steps of both the encryption and decryption algorithm. This will be further elucidated in the following sections.

## Development of encryption and decryption algorithm

The following pseudocode represents the general outline of the encryption and decryption algorithm that we implement.

***Encryption algorithm***

note index of spaces in text in list

note index of capitals in text in list

store plain text with spaces removed

call splitevenodd function to split text into 2 lists, even(has even index of string) and oddList(has odd index of string)

loop through each letter of plain text for alphabets, map them to another alphabet for substitution

iterate through transformed even letters along with the current odd letter list at the same time

add the value of the previous letter at index i to value of the current letter at index i for Caesar cipher shifting

use modulo operation on value by 26

map value back to corresponding alphabet index to make Caesar shifted string

final result string is created by concatenating letter by letter from substitution string and Caesar shifted string respectively in order

list of capital letter index is used on final result string to run uppercase operation on specified indexes present in list

list of spaces indexes are used to add spaces back to final result string

string is displayed as encrypted correctly

***Decryption algorithm***

note index of spaces in text in list

note index of capitals in text in list

store plain text with spaces removed

call splitevenodd function to split text into 2 lists, even(has even index of string) and oddList(has odd index of string)

loop through each letter of cipher text for alphabets, map them to original alphabet for substitution

iterate through transformed even letters along with current odd letter list at the same time

subtract the value of previous letter at index i to value of current letter at index i for Caesar cipher shifting to get back original letters for odd indexes

use modulo operation on value by 26

map value back to corresponding alphabet index to make caesar shifted string

final result string is created by concatenating letter by letter from 1) decrypted substitution string and 2) decrypted Caesar shifted string respectively in order

list of capital letter index is used on final result string to run uppercase operation on specified indexes present in list

list of spaces indexes are used to add spaces back to final result string

string is displayed as encrypted correctly

Each of the following subsections describes how the above pseudocode was implemented:-

***Encryption Algorithm***

In encryption, the plaintext is prepared for encrypting. First index lists are created for the space and capital letters of the plain text, then, the text is stripped. For the encryption, in the substitution of even letters, the alphabets of the plaintext are looked at table is looked at for each letter, and the original alphabet that is in the same index is what it was transformed from. Thus substituting back brings back the original text.  For caesar cipher decryption of odd letters, the reverted even letters are used as the key that performs a backwards shifting operation on the ciphertext through subtraction. Doing so reverts odd letters back to plaintext form

After encryption, we have 2 lists that are the result of 2 transformations based on different techniques that need to be combined. This is done so with a custom combine function that puts the text back in order in even and odd indexes. After encryption, and putting the text back together, it will still be in lowercase and without spaces. Thus using the index lists that were initially created, uppercase letters and spaces are added back before finally presenting to the user.

***Decryption Algorithm***

In decryption, the process is similar to encryption in that the ciphertext will be prepared for decrypting and eventually presenting in the same way as encryption: First index lists are created for the space and capital letters of the Cipher text, then, the text is stripped. After decryption, we have 2 lists that are the result of 2 separate decryption transformations that need to be combined which is done so with a custom combine function that puts the text back in order. After putting the text back together, it will still be in lowercase and without spaces. THus using the index lists that were initially created, uppercase letters and spaces are added back before finally presenting to the user.

For the actual decryption algorithm, the same steps as encryption are performed but in reverse. For substitution of even letters, the mapped table is looked at for each letter, and the original alphabet that is in the same index is what it was transformed from. Thus substituting back brings back the original text.  For caesar cipher decryption of odd letters, the reverted even letters are used as the key that performs a backwards shifting operation on the ciphertext through subtraction. Doing so reverts odd letters back to plaintext form

***Console interface implemented to present encryption and decryption algorithm to the user***

console interface was created using argparse library which is recommended for use in all modern python CLIs. Program was name cryptoSystem with a version number of 1.0. 5 arguments were present in the program. Out of which 3 can be considered positional arguments which are necessary for the function of the program.

1. Type: type is used to indicate the type of cryptographic system that will be used to generate cipher texts. As of now the only option is the integer 0 which is the current project implementation
2. operation: operation is used to indicate whether we will perform either encryption or decryption using the selected cryptosystem.
3. name: Name is used as the argument where the actual plain text is inputted for processing and encryption/decryption

There are 2 other optional arguments which change the behavior of the program:

1. version: Used to indicate the version of the program currently
2. show: This is a switch that is toggled that is essentially a verbose command. It shows the step by step implementation of both encryption and decryption

## Implementation specifications

This section describes and highlights few methods or approaches that we have taken in order to meet the desired objectives and requirements of this project.

We place an argument: if(type == 0 and operation == 'encrypt') in the main.py file, this mainly is positioned to indicate which encryption technique is to be used. 0 indicates that the product cipher of monoalphabetic cipher and Caesar cipher is to be utilized. This was done to ensure that our code could be reusable in the future in case we decide to include other encryption techniques.

While using the terminal to run the main.py, the user will have to insert an additional argument “-show” in order to see the steps that occur in the encryption and decryption process. This was done to give the user the option to see the encryption and decryption results only or to see the working as well. Below are a few examples of how to run the main.py:

|  |  |
| --- | --- |
| Only output ciphertext generated | python .\main.py 0 encrypt ‘green car’ |
| Output ciphertext and steps involved | python .\main.py 0 decrypt ‘green car’ -show |
| Only output plaintext after decryption | python .\main.py 0 decrypt ‘bymju qhs’ |
| Output plaintext and steps involved | python .\main.py 0 decrypt ‘bymju qhs’ -show |

As seen in the appendix, there are several files and classes created. This was done to ensure that we enforce the principle of modularity and reusability and therefore meet our objective of abiding by the programming principles.

## Summary

We have implemented the cryptographic system in which even letters are substituted in a monoalphabetic table and odd letters use the substituted even letters as keys for their Caesar cipher operation for encryption and decryption. We have implemented these in the form of separate files that add together as modules that can be used independently. Classes are used to provide common structure and for easy retrieval of vital information. A CLI interface was implemented using the argparse library in python to implement encryption and decryption as choices for operation to perform on a plain text.

## References

[1] "What Is Cryptography and Why Is It Important? - Entrust Blog", *Entrust Blog*, 2022. [Online]. Available: https://www.entrust.com/blog/2021/06/why-is-cryptography-so-important-heres-what-you-need-to-know/. [Accessed: 17- Oct- 2022].

[2] N. Morgan, "The role of cryptography in information security", *Triskelelabs.com*, 2022. [Online]. Available: https://www.triskelelabs.com/blog/the-role-of-cryptography-in-information-security#:~:text=Cryptography%20ensures%20the%20integrity%20of,been%20tampered%20with%20during%20transmission. [Accessed: 17- Oct- 2022].

[3] C. Techniques, "Classical Encryption Techniques", *BrainKart*, 2022. [Online]. Available: https://www.brainkart.com/article/Classical-Encryption-Techniques\_8339/#:~:text=A%20substitution%20technique%20is%20one,with%20cipher%20text%20bit%20patterns. [Accessed: 17- Oct- 2022].

[4] "Mono-Alphabetic Substitution Cipher | 101 Computing", *101 Computing*, 2022. [Online]. Available: https://www.101computing.net/mono-alphabetic-substitution-cipher/. [Accessed: 17- Oct- 2022].

[5] "Practical Cryptography", *Practicalcryptography.com*, 2022. [Online]. Available: http://practicalcryptography.com/ciphers/caesar-cipher/#:~:text=The%20Caesar%20cipher%20is%20one,become%20C%2C%20and%20so%20on. [Accessed: 17- Oct- 2022].

## 

## Appendix

**Main.py**

|  |
| --- |
| from secrets import choice |
|  | import monoAlphabeticCaesarsSubstitution |
|  | import argparse |
|  |  |
|  | def main(): |
|  | parser = argparse.ArgumentParser(prog='cryptoSystem', |
|  | prefix\_chars='-', |
|  | description='Encrypt using combination of caesars cipher and substitution') |
|  | parser.add\_argument('type', |
|  | metavar='type', |
|  | type=int, |
|  | help='Choosing type of cipher algorithm', choices=range(0,1)) |
|  | parser.add\_argument('operation', |
|  | metavar='operation', |
|  | type=str, |
|  | help='Whether to choose to perform decryption or encryption', choices=['encrypt', 'decrypt']) |
|  | parser.add\_argument('name', |
|  | metavar='name', |
|  | type=str, |
|  | help='plaintext to encrypt or decrypt') |
|  | parser.add\_argument('-version', '-v', |
|  | action='version', version='%(prog)s 1.0', help='Shows version') |
|  | #parser.add\_argument('-show',action='store\_True', help='Option that shows process of encryption and decryption') |
|  | args = parser.parse\_args() |
|  | type = args.type |
|  | #showCode = args.show |
|  | operation = args.operation |
|  | text = args.name |
|  | if(type == 0 and operation == 'encrypt'): |
|  | print(monoAlphabeticCaesarsSubstitution.encrypt(text)) |
|  | elif(type == 0 and operation == 'decrypt'): |
|  | print(monoAlphabeticCaesarsSubstitution.decrypt(text)) |
|  |  |
|  | if \_\_name\_\_ == "\_\_main\_\_": |
|  | main() |

## **caesar.py**

|  |
| --- |
| class caesarCipher: |
|  | alphabets = ['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'] |
|  | punctuationList = [" ", "-", "\_", ".", "&", "?", "!", "@", "#", "/"] |
|  | def \_\_init\_\_(self): |
|  | self.result = [] |
|  | def getResult(self): |
|  | return self.result |
|  | def getAlphabets(self): |
|  | return self.alphabets |
|  | def getPunctuationList(self): |
|  | return self.punctuationList |
|  | def clearResult(self): |
|  | self.result = [] |
|  | def encrypt(self, plain, shift): |
|  | """Performs a caesars cipher operation on two lists of characters of equal length translating second list of characters to a number |
|  | based on index found in list of alphabets so that a shift may be performed on the letters of list of characters of plaintext |
|  | :param plain: list of characters considered to comprise of plaintext |
|  | :type plain: list |
|  | :param shift: list of characters to be used for shifting |
|  | :type shift: list |
|  |  |
|  | :rtype: list |
|  | :return: list of characters after performing shift |
|  | """ |
|  | for i in range(len(plain)): |
|  | shiftChara = shift[i] #even |
|  | plainChara= plain[i] #odd |
|  | if (plainChara in self.punctuationList): |
|  | self.result.append(plainChara) |
|  | else: |
|  | #storing the index of previous letter which was substituted to use in shifting operation |
|  | prevLetter = self.alphabets.index(shiftChara)+1 |
|  | #storing index of current plain text lettet for shifting |
|  | currentLetter= self.alphabets.index(plainChara) |
|  | currentLetter = (currentLetter + prevLetter) % 26 |
|  | letter=self.alphabets[currentLetter] |
|  | self.result.append(letter) |
|  | return self.result |
|  |  |
|  | def decrypt(self, plain,shift): |
|  | """Performs a caesars cipher operation in reverse on two lists of characters of equal length translating second list of characters to a number |
|  | based on index found in list of alphabets so that a shift may be performed on the letters of list of characters of plaintext |
|  | :param plain: list of characters where operation will be performed |
|  | :type plain: list |
|  | :param shift: list of characters who will determine by how much the shift will occur |
|  | :type shift: list |
|  |  |
|  | :rtype: list |
|  | :return: list of characters that were there before performing a shift through subtraction |
|  | """ |
|  | for i in range(len(plain)): |
|  | shiftChara = shift[i] |
|  | plainChara = plain[i] |
|  | if (plainChara in self.punctuationList): |
|  | self.result.append(plainChara) |
|  | else: |
|  | #storing the index of previous letter which was substituted to use in shifting operation |
|  | prevLetter = self.alphabets.index(shiftChara) + 1 |
|  | #storing index of current plain text lettet for shifting |
|  | currentLetter = self.alphabets.index(plainChara) |
|  | currentLetter = (currentLetter - prevLetter ) % 26 |
|  | letter = self.alphabets[currentLetter] |
|  | self.result.append(letter) |
|  | return self.result |

## **listOperations.py**

|  |
| --- |
| def splitevenodd(string): |
|  | """ Iterates through a string letter by letter, adds letters in even indexes to evenList, adds letters in odd indexes to oddList |
|  | :param string: |
|  | :type string: str |
|  |  |
|  | :rtype: list |
|  | :return: a 2D list that has 2 lists as elements: 1) a list of letters in even indexes 2) a list of letters in odd indexes |
|  | """ |
|  | evenList = [] |
|  | oddList = [] |
|  | for i in range(len(string)): |
|  | if (i % 2 == 0): |
|  | evenList.append(string[i]) |
|  | else: |
|  | oddList.append(string[i]) |
|  | return([evenList,oddList]) |
|  | def combine(sub,shift): |
|  | """Combines the lists made after substituting even letters of plain text and caesar shifting odd letters of plain texts into one list of characters |
|  | :param sub: list of transformed even letters of plaintext after substitution |
|  | :type sub: list |
|  | :param shift: list of transformed odd letters after caesar cipher shift operation |
|  | :type shift: list |
|  |  |
|  | :rtype: list |
|  | :return: list of combined characters from lists made substituting even letters of plain text and caesar shifting odd letters of plain texts |
|  | """ |
|  | combined=[] |
|  | length=max(len(sub),len(shift)) |
|  | try: |
|  | for a in range(length): |
|  | combined.append(sub[a]) |
|  | combined.append(shift[a]) |
|  | except Exception as e: |
|  | print("") |
|  | return combined |
|  | def createCapitalIndex(text): |
|  | """Goes through a list of characters or string and adds to list the indexes of those letters which are uppercase |
|  | :param text: list of characters |
|  | :type text: list |
|  |  |
|  | :rtype: list |
|  | :return: list of indexes of characters with uppercase |
|  | """ |
|  | capitalIndex=[] |
|  | for chara in text: |
|  | if(chara.isupper()): |
|  | capitalIndex.append(text.index(chara)) |
|  | return capitalIndex |

## **monoAlphabeticCaesarsSubstitution.py**

|  |
| --- |
| import listOperations |
|  | import substitution as sub |
|  | import caesar as caes |
|  | substitution = sub.substitution() |
|  | caesar = caes.caesarCipher() |
|  | def encrypt(plainText): |
|  | """Encrypts a plaintext to ciphertext after combining split even and odd letter list of plaintext that had substitution and |
|  | caesars cipher operation performed on them separately. Where caesar cipher operation depended on the value of elements of odd letters after substitution |
|  | :param plainText: Plaintext that will be encrypted to cipher text |
|  | :type plainText: str |
|  |  |
|  | :rtype: str |
|  | :return: A ciphertext |
|  | """ |
|  | #keep note of space so that it can be added back after encryption |
|  | spaceIndex = plainText.index(" ") |
|  | capitalIndex = listOperations.createCapitalIndex(plainText) |
|  | plainText=plainText.lower().replace(" ", "") |
|  |  |
|  | result=listOperations.splitevenodd(plainText) |
|  | evenList, oddList= result[0], result[1] |
|  | sub=substitution.encrypt(evenList) |
|  | shift=caesar.encrypt(oddList, evenList) |
|  |  |
|  | final\_result=listOperations.combine(sub,shift) |
|  | for i in capitalIndex: |
|  | final\_result[i] = final\_result[i].upper() |
|  | final\_result.insert(spaceIndex, " ") |
|  | final\_result=''.join(final\_result) |
|  |  |
|  | return final\_result |
|  |  |
|  | def decrypt(cipherText): |
|  | """Decrypts a cipherText by performing substitution in reverse on even letters and caesar shift in reverse on odd letters |
|  | :param cipherText: Cipher that will be decrypted to plain text |
|  | :type cipherText: str |
|  |  |
|  | :rtype: str |
|  | :return: A plaintext |
|  | """ |
|  | spaceIndex = cipherText.index(" ") |
|  | capitalIndex = listOperations.createCapitalIndex(cipherText) |
|  | cipherText=cipherText.lower().replace(" ", "") |
|  |  |
|  | result=listOperations.splitevenodd(cipherText) |
|  | evenList, oddList = result[0], result[1] |
|  | dec\_sub= substitution.decrypt(evenList) |
|  | dec\_shift= caesar.decrypt(oddList, dec\_sub) |
|  |  |
|  | final\_result = listOperations.combine(dec\_sub, dec\_shift) |
|  | for i in capitalIndex: |
|  | final\_result[i] = final\_result[i].upper() |
|  | final\_result.insert(spaceIndex, " ") |
|  | final\_result = ''.join(final\_result) |
|  |  |
|  | return final\_result |

## **substitution.py**

|  |
| --- |
| class substitution: |
|  | alphabets = ['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', " ", "-", "\_", ".", "&", "?", "!", "@", "#", "/"] |
|  | sub\_alphabets = ['h', 'i', 'l', 'w', 'm', 'k', 'b', 'd', 'p', 'c', 'v', 'a', 'z', 'u', 's', 'j', 'g', 'r', 'y', 'n', 'q', 'x', 'o', 'f', 't', 'e', " ", "-", "\_", ".", "&", "?", "!", "@", "#", "/"] |
|  | def \_\_init\_\_(self): |
|  | self.resultStr = [] |
|  | def getResultStr(self): |
|  | return self.resultStr |
|  | def getAlphabets(self): |
|  | return self.alphabets |
|  | def getSubAlphabets(self): |
|  | return self.sub\_alphabets |
|  | def clearResultStr(self): |
|  | self.resultStr = [] |
|  | def encrypt(self, plain): |
|  | """Substitutes characters in plaintext that have a match in alphabet with a letter in sub\_alphabets of the same index |
|  | :param plain: list of characters from plaintext string that were in even indexes |
|  | :type plain: list |
|  |  |
|  | :rtype: list |
|  | :return: list of characters substituted in plaintext between alphabets and sub\_alphabets |
|  | """ |
|  | length = len(plain) |
|  | alphaLength=len(self.alphabets) |
|  | for i in range(length): |
|  | for j in range (alphaLength): |
|  | if (plain[i] == self.alphabets[j]): |
|  | #if plain text character is upper, append uppercase version of mapping |
|  | if(plain[i].isupper()): |
|  | self.resultStr.append(self.sub\_alphabets[j].upper()) |
|  | else: |
|  | self.resultStr.append(self.sub\_alphabets[j]) |
|  | return self.resultStr |
|  |  |
|  | def decrypt(self, cipher): |
|  | """Performs opposite operation of encrypt. Here the substituted characters of plaintext are matched back to original alphabets index |
|  | :param cipher: list of characters from ciphertext string that were in even indexes |
|  | :type cipher: list |
|  |  |
|  | :rtype: list |
|  | :return: list of reverted characters from |
|  | """ |
|  | length = len(cipher) |
|  | alphalength = len(self.alphabets) |
|  | for i in range(length): |
|  | for j in range(alphalength): |
|  | if (cipher[i] == self.sub\_alphabets[j]): |
|  | #if plain text character is upper, append uppercase version of mapping |
|  | if(cipher[i].isupper()): |
|  | self.resultStr.append(self.alphabets[j].upper()) |
|  | else: |
|  | self.resultStr.append(self.alphabets[j]) |
|  | return self.resultStr |

Provided below is the link to the GitHub repository where our code is present:

[ghc4716/Cryptography: Cryptography Project (github.com)](https://github.com/ghc4716/Cryptography)