

**BAHRIA UNIVERSITY, (Karachi Campus)**

*Department of Computer Science*

**Operating System Lab CSL-320**

**PROJECT REPORT**

Title: Linux Security

Semester: Fall-2023

Class and Section: BSCS 5A

**Group Members:**

**Mania Imam (02-134212-013)**

**Tayyaba Imam (02-134202-056) LEAD**

**Shavana Yousuf (02-134212-022)**

*Under The Supervision Of*

**Ms. Mehwish Saleem**

**(Lab Engineer)**

# Contents

[Contents 2](#_Toc155464375)

[**Introduction:** 3](#_Toc155464376)

[Abstract: 3](#_Toc155464377)

[Motivations and Scope: 3](#_Toc155464378)

[Implementation Details and Technologies: 3](#_Toc155464379)

[**Project Description:** 4](#_Toc155464380)

[**Source Code:** 7](#_Toc155464381)

[**Output:** 9](#_Toc155464382)

[**Conclusion:** 11](#_Toc155464383)

# **Introduction:**

## Abstract:

In today's digital world, where sensitive information flows freely through networks, ensuring its security is paramount. Cryptography, the art of hiding messages from prying eyes, plays a crucial role in safeguarding our online existence. This project delves into the fascinating realm of encryption, exploring practical tools and techniques to transform plain text into garbled secrets.

Additionally , Linux systems, particularly Ubuntu, face continuous attack attempts that exploit vulnerabilities and threaten data integrity, confidentiality, and system availability. This project also tackles this challenge by proposing a comprehensive security system, designed to significantly enhance Ubuntu's security posture through a multi-layered approach.

## Motivations and Scope:

The ever-evolving threat landscape demands robust security solutions for Linux systems. Unmitigated attacks can lead to devastating consequences, including data breaches, system crippling, and financial losses. Ubuntu, a versatile and widely adopted operating system, requires robust security measures to protect the sensitive information and operations of millions of users. This project aims to fill this gap by designing and implementing a multi-layered security system addressing critical security concerns in Ubuntu environments.

## Implementation Details and Technologies:

The system will be built using a combination of open-source tools and custom scripts. Programming languages like Python or Bash will be employed. With the help of readily available libraries like tkinter and pyperclip, we build user-friendly interfaces for entering messages and selecting encryption methods. Our journey doesn't end with textual encryption. We venture into the realm of file protection, leveraging PyCrypto to encrypt and decrypt sensitive documents before sending them across communication channels. Password protection and MD5 hashing ensure secure storage and access control.

In order to securely encode or decode messages, a cryptosystem is a structure or scheme made up of a collection of algorithms that transform plaintext into ciphertext. System for computers that uses cryptography. In the presence of malevolent outsiders, or adversaries, cryptography enables secure communication. An algorithm and a key are used in encryption to convert an input (plaintext) into an encrypted output (i.e., ciphertext) Due to the complexity of modern cryptography and the usage of encryption to protect data, cryptographic systems have become a popular target for hackers.  Cryptosystems are used to send messages securely over the internet, including confidential information like credit card numbers. Another use of cryptography may be a secure electronic mail system that uses key management procedures, cryptographic hash functions, and digital signature mechanisms.

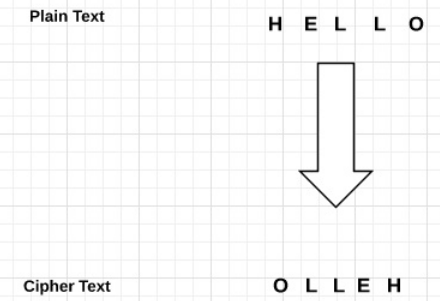
# **Project Description:**

To run, have a python environment setup with the listed dependencies properly installed:

* Pip
* Tkinter
* Pyperclip
* Setuptools

The message variable holds the plain text data, and the translated variable has the generated cypher text data. With the aid of an index number and a for loop, the length of plain text is computed. The characters are kept in the translated cypher text variable, which is printed in the final line.

We reverse a string of plain text to create cypher text using a technique called reverse cypher. Both the encryption and decryption processes are identical. The user only needs to reverse the cypher text to obtain the plain text in order to decrypt it. (Reverse cipher's main weakness is that it is quite weak.)

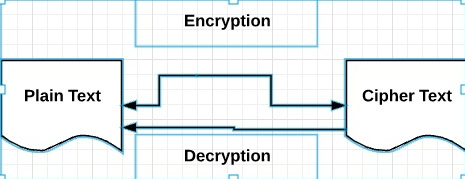


The monoalphabetic substitution cypher known as the plaintext and cypher texaffine uses a straightforward mathematical function to translate each letter of the alphabet to its numeric counterpart, encrypt it, and then convert it back to a letter.

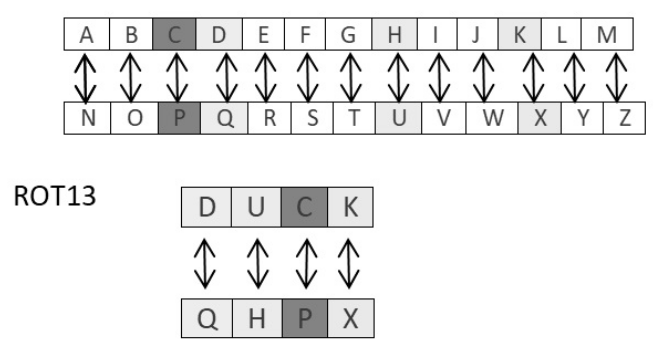
Diagram, text

Description automatically generated

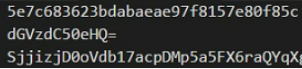
Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of [polyalphabetic substitution](https://en.wikipedia.org/wiki/Polyalphabetic_cipher). The **affine cipher** is a type of [monoalphabetic substitution cipher](https://en.wikipedia.org/wiki/Monoalphabetic_substitution_cipher), where each letter in an alphabet is mapped to its numeric equivalent, encrypted using a simple mathematical function, and converted back to a letter.



Rotate by 13 places is the acronym for the encryption ROT13. The shift in this unique Caesar Cipher scenario is always 13.



By arranging the characters in a column arrangement and reading them horizontally, the cypher text and the indicated key are the two values used as input parameters for the reverse decoding or decrypting procedure. Files can be encrypted and decrypted in Python before being sent across a communication channel. You must use the PyCrypto plugin for this. The values are kept in simply safe backup files in the Windows system and the passwords are created using the MD5 hash method. These values are shown below. Base64 encoding transforms binary data into text format so that it can be properly handled by a user across a communication channel. Electronic mail that is more private is sent using Base64 (PEM)



# **Source Code:**

import tkinter as tk

from tkinter import simpledialog

from itertools import starmap, cycle

title = "OS Encryption and Decryption Manager"

def main():

root = tk.Tk()

root.withdraw()

msg1 = "Welcome, please pick a cryptosystem (Encryption, Decryption)."

crypto\_choices = ["Encryption", "Decryption"]

crypto\_mode = simpledialog.askstring(title, msg1, initialvalue=crypto\_choices[0])

msg2 = "Please pick a crypto mode to use (Affine Cipher, Vigenere Cipher)."

cryptosystem\_choices = ["Affine Cipher", "Vigenere Cipher"]

cryptosystem\_mode = simpledialog.askstring(title, msg2, initialvalue=cryptosystem\_choices[0])

msg3 = "Would you like plaintext or cipher text?"

text\_choices = ["Plaintext", "Cipher text"]

text\_mode = simpledialog.askstring(title, msg3, initialvalue=text\_choices[0])

msg4 = "Please enter your text and key."

if cryptosystem\_mode == 'Affine Cipher':

field\_names = ["Text", "A", "B"]

elif cryptosystem\_mode == 'Vigenere Cipher':

field\_names = ["Text", "Key"]

field\_values = [simpledialog.askstring(title, f"Enter {field}", initialvalue="") for field in field\_names]

# make sure that none of the fields was left blank

while 1:

if field\_values is None:

break

errmsg = ""

for i in range(len(field\_names)):

if field\_values[i].strip() == "":

errmsg = errmsg + ('"%s" is a required field.\n\n' % field\_names[i])

if errmsg == "":

break # no problems found

field\_values = [simpledialog.askstring(title, f"Enter {field}", initialvalue=field\_values[i]) for i, field in enumerate(field\_names)]

my\_text = field\_values[0]

if cryptosystem\_mode == 'Affine Cipher':

a = field\_values[1]

b = field\_values[2]

print("[Affine Cipher]")

print('%s message' % (crypto\_mode.title()))

print(affine\_cipher(a, b, my\_text, crypto\_mode, text\_mode))

elif cryptosystem\_mode == 'Vigenere Cipher':

my\_key = field\_values[1]

print("[Vigenere Cipher]")

print('%s message' % (crypto\_mode.title()))

print(vigenere\_cipher(my\_key, my\_text, crypto\_mode, text\_mode))

root.mainloop()

def vigenere\_cipher(passed\_key, msg, crypto, text\_type):

text = msg

key = passed\_key

def encrypt(message, key):

message = filter(str.isalpha, message.upper())

def enc(c, k):

return chr(((ord(k) + ord(c) - 2 \* ord('A')) % 26) + ord('A'))

return ''.join(starmap(enc, zip(message, cycle(key))))

def decrypt(message, key):

def dec(c, k):

return chr(((ord(c) - ord(k) - 2 \* ord('A')) % 26) + ord('A'))

return ''.join(starmap(dec, zip(message, cycle(key))))

encr = encrypt(text, key)

decr = decrypt(encr, key)

if crypto == 'Encryption' and text\_type == 'Plaintext':

return 'Original: ' + text + '\nEncrypted: ' + encr

elif crypto == 'Decryption' and text\_type == 'Cipher text':

return 'Original: ' + text + '\nDecrypted: ' + decr

else:

return 'Please select the proper text type.'

def affine\_cipher(a, b, msg, crypto, text\_type):

text = msg

KEY = (int(a), int(b), 55)

DIE = 128

def encrypt\_char(char):

K1, K2, kI = KEY

return chr((K1 \* ord(char) + K2) % DIE)

def encrypt(string):

return "".join(map(encrypt\_char, string))

def decrypt\_char(char):

K1, K2, KI = KEY

return chr(KI \* (ord(char) - K2) % DIE)

def decrypt(string):

return "".join(map(decrypt\_char, string))

encr = encrypt(text)

decr = decrypt(text)

if crypto == 'Encryption' and text\_type == 'Plaintext':

return 'Original: ' + text + '\nEncrypted: ' + encr

elif crypto == 'Decryption' and text\_type == 'Cipher text':

return 'Original: ' + text + '\nDecrypted: ' + decr

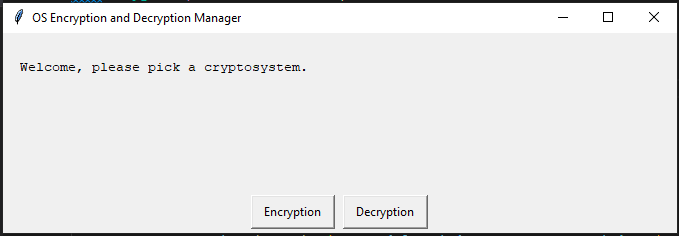
else:

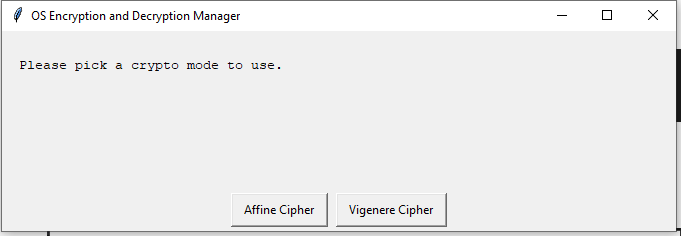
return 'Please select the proper text type.'

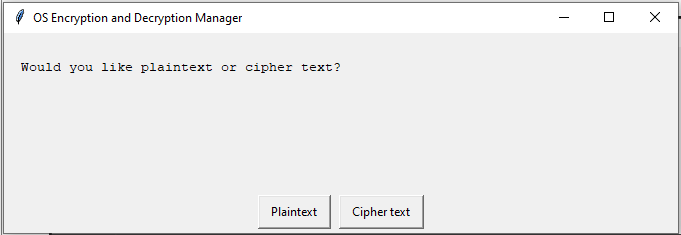
if \_\_name\_\_ == '\_\_main\_\_':

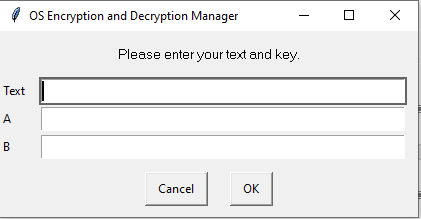
main()

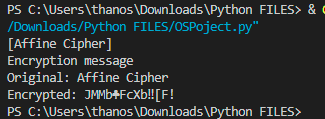
# **Output:**

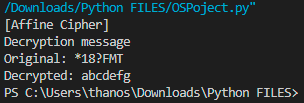












# **Conclusion:**

Cryptosystems use the algorithms for key creation, encryption, and decryption to help protect data. A cryptographic key, a string of bits utilized by a cryptographic algorithm to convert plain text into ciphertext or the other way around, lies at the core of all cryptographic activities. To carry out this kind of action, a cryptographic algorithm requires the key as one of the variables in the data it receives as input. The integrity of the keys used determines how secure the cryptographic technique is.

This comprehensive security system represents a significant advancement in securing Linux Ubuntu environments. By implementing robust multi-layered security mechanisms, it addresses critical security concerns and enhances data integrity, confidentiality, and system availability. The user-friendly tools and detailed documentation simplify security administration, empowering users to manage system security effectively. This project demonstrates a deep understanding of security concepts and proficient implementation skills, showcasing the team's capabilities in tackling contemporary security challenges.