Introduction

The Affine Cipher is a type of monoalphabetic substitution cipher that combines modular arithmetic with linear algebra to encrypt and decrypt text. Each character in the plaintext is mapped to its numeric equivalent, encrypted using a mathematical function, and then converted back to a letter. This cipher provides a basic method for secure communication, though it can be vulnerable to brute-force attacks if the key space is small.

Overview

This report presents the implementation of the Affine Cipher in Python, covering 4 phases:

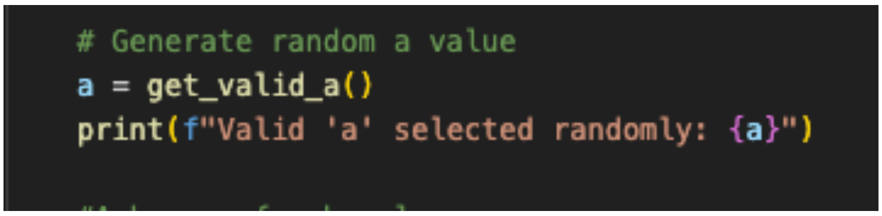
- Encryption: Transforming plain text into ciphertext using a mathematical formula.
- Decryption: Recovering the original plaintext from the ciphertext using an inverse function.
- Brute-Force Attack: Attempting to break the cipher by testing all possible keys (using a customized dictionary and calculating time)
- Display output

1. Data Encryption

The encryption process in the Affine Cipher follows the formula:

$$E(x)=(a \times x+b) \bmod 26$$

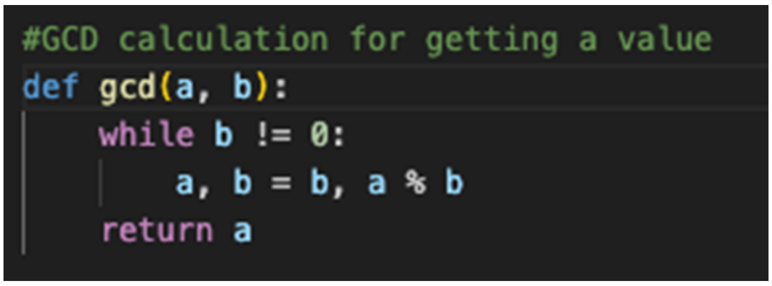
1.1 Generating a random “a” value



```python
    # Generate random a value
    a = get_valid_a()
    print(f"Valid 'a' selected randomly: {a}")
```

Figure 1 – Generating a random value

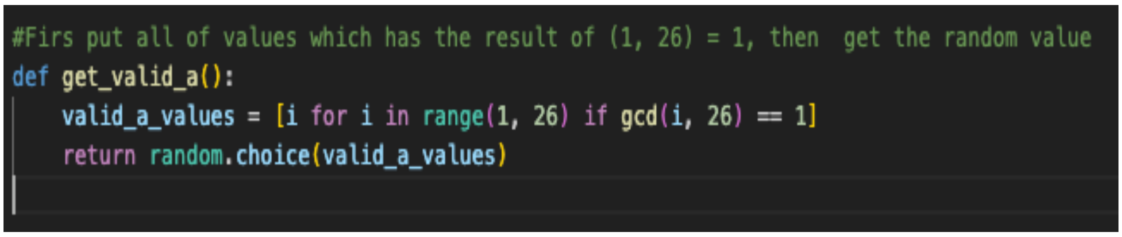
1.1.1 Adding the gcd function for getting the right value



```python
#GCD calculation for getting a value
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a
```

Figure 2 – Adding the gcd Function

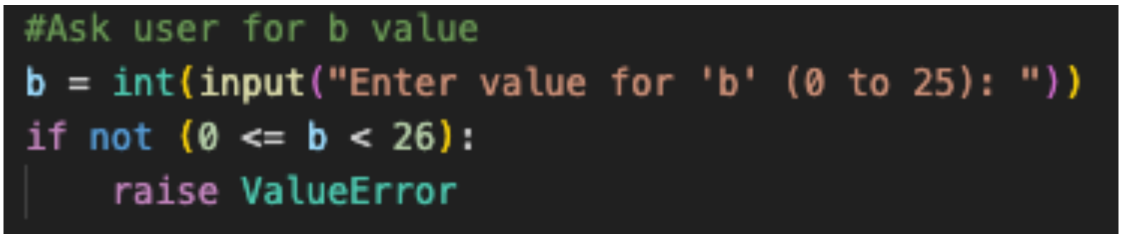
1.1.2 Random value



```python
#Firs put all of values which has the result of (1, 26) = 1, then  get the random value
def get_valid_a():
    valid_a_values = [i for i in range(1, 26) if gcd(i, 26) == 1]
    return random.choice(valid_a_values)
```

Figure 3 – Random Value

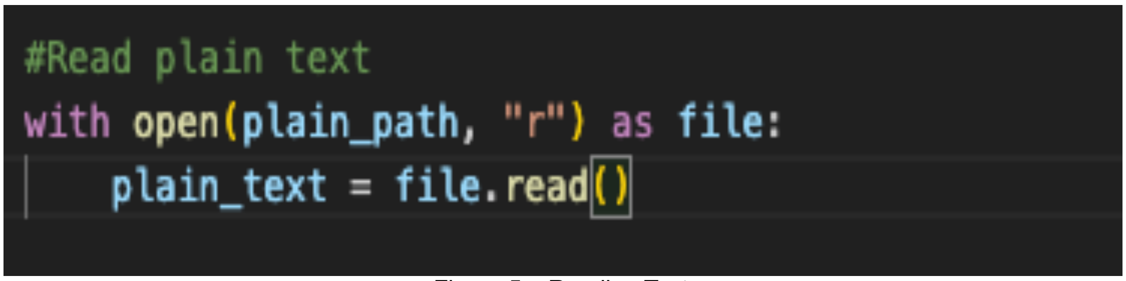
1.2 Asking User to select the b value between 0 and 25



```python
#Ask user for b value
b = int(input("Enter value for 'b' (0 to 25): "))
if not (0 <= b < 26):
    raise ValueError
```

Figure 4 – Selecting b Value

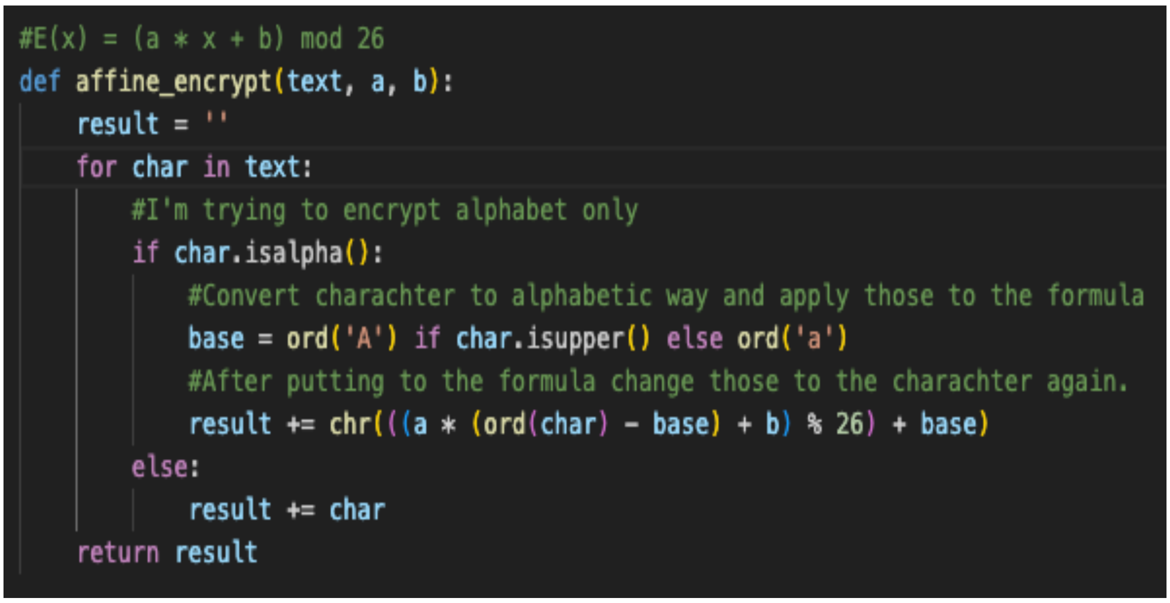
1.3 Reading Text



```python
#Read plain text
with open(plain_path, "r") as file:
    plain_text = file.read()
```

Figure 5 – Reading Text

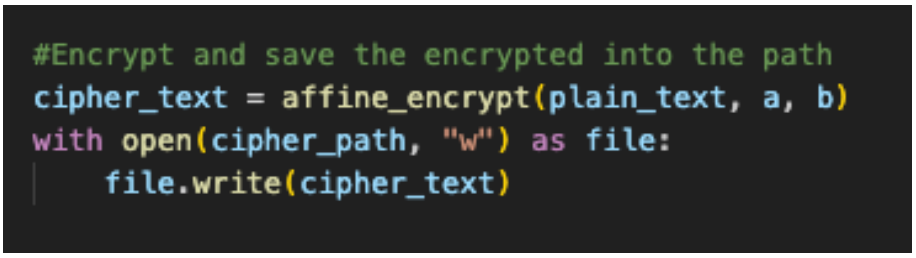
1.4 Encrypting the plain.txt (affine_encrypt) by Converting the alphabet to decimal, then adding those to the formula



```python
#E(x) = (a * x + b) mod 26
def affine_encrypt(text, a, b):
    result = ''
    for char in text:
        #I'm trying to encrypt alphabet only
        if char.isalpha():
            #Convert charachter to alphabetic way and apply those to the formula
            base = ord('A') if char.isupper() else ord('a')
            #After putting to the formula change those to the charachter again.
            result += chr(((a * (ord(char) - base) + b) % 26) + base)
        else:
            result += char
    return result
```

Figure 6 - Encrypting the Plain Text

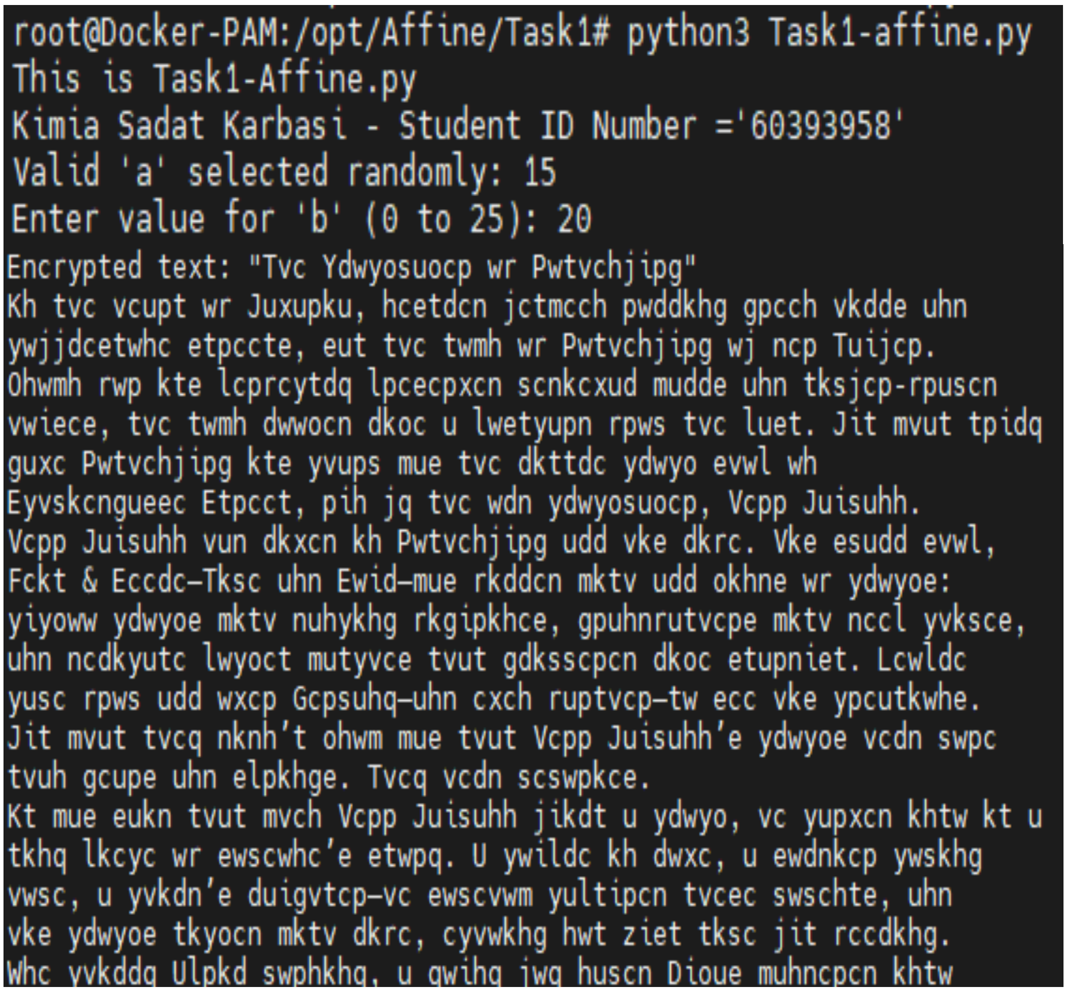
1.5 Adding the encrypted data in the cipher.txt



```python
#Encrypt and save the encrypted into the path
cipher_text = affine_encrypt(plain_text, a, b)
with open(cipher_path, "w") as file:
    file.write(cipher_text)
```

Figure 7- Adding the encrypted data

1.6 Code Execution



```
root@Docker-PAM:/opt/Affine/Task1# python3 Task1-affine.py
This is Task1-Affine.py
Kimia Sadat Karbasi - Student ID Number ='60393958'
Valid 'a' selected randomly: 15
Enter value for 'b' (0 to 25): 20
Encrypted text: "Tvc Ydwyosuocp wr Pwtvchjipg"
Kh tvc vcupt wr Juxupku, hcetdcn jctmcch pwddkhg gpcch vkdde uhn
ywjjdcetwhc etpccte, eut tvc twmh wr Pwtvchjipg wj ncp Tuijcp.
Ohwmh rwp kte lcprcytdq lpcecpxcn scnkcxud mudde uhn tksjcp-rpuscn
vwiece, tvc twmh dwwocn dkoc u lwetyupn rpws tvc luet. Jit mvut tpidq
guxc Pwtvchjipg kte yvups mue tvc dkttdc ydwyo evwl wh
Eyvskcngueec Etpcct, pih jq tvc wdn ydwyosuocp, Vcpp Juisuhh.
Vcpp Juisuhh vun dkxcn kh Pwtvchjipg udd vke dkrc. Vke esudd evwl,
Fckt & Eccdc-Tksc uhn Ewid-mue rkddcn mktv udd okhne wr ydwyoe:
yiyoww ydwyoe mktv nuhykhg rkgipkhce, gpuhnrutvcpe mktv nccl yvksce,
uhn ncdkyutc lwyoct mutyvce tvut gdksscpcn dkoc etupniet. Lcwldc
yusc rpws udd wxcp Gcpsuhq-uhn cxch ruptvcp-tw ecc vke ypcutkwhe.
Jit mvut tvcq nknh't ohwm mue tvut Vcpp Juisuhh'e ydwyoe vcdn swpc
tvuh gcupe uhn elpkhge. Tvcq vcdn scswpkce.
Kt mue eukn tvut mvch Vcpp Juisuhh jikdt u ydwyo, vc yupxcn khtw kt u
tkhq lkcyc wr ewscwhc'e etwpq. U ywildc kh dwxc, u ewdnkcp ywskhg
vwsc, u yvkdn'e duigvtcp-vc ewscvwm yultipcn tvcec swschte, uhn
vke ydwyoe tkyocn mktv dkrc, cyvwkhg hwt ziet tksc jit rccdkhg.
Whc yvkddq Ulpkd swphkhg, u gwihg jwg huscn Dioue muhncpcn khtw
```

Figure 8 – Executing the Code

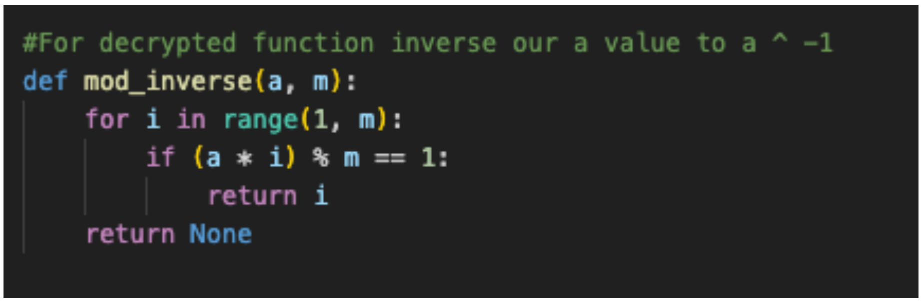
2. Data Decryption

The decryption process reverses the encryption formula:

D(y)=a−1×(y−b) mod26

2.1 Decrypting the cipher.txt

2.1.1 Inverse a value using the created function

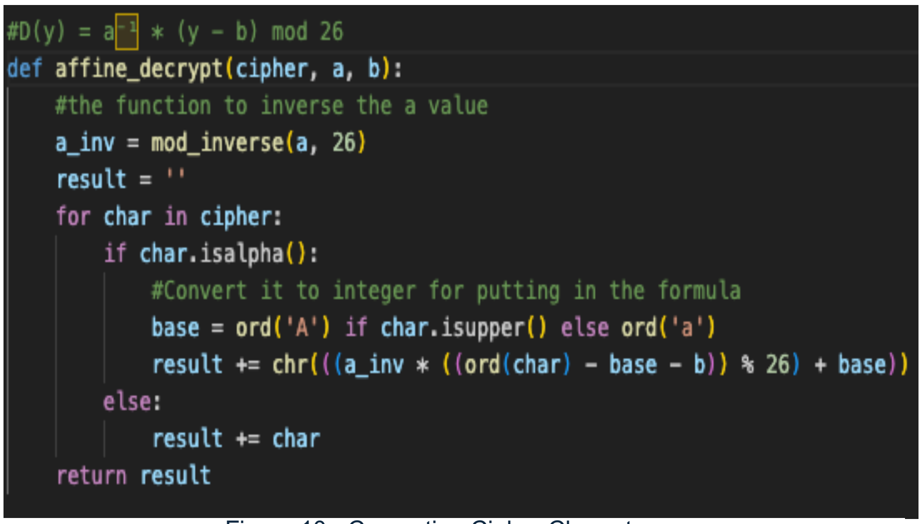


```python
#For decrypted function inverse our a value to a ^ -1
def mod_inverse(a, m):
    for i in range(1, m):
        if (a * i) % m == 1:
            return i
    return None
```

Figure 9 – Inverse a value

2.1.2 Reading cipher characters and converting them to decimal for importing into the formula:

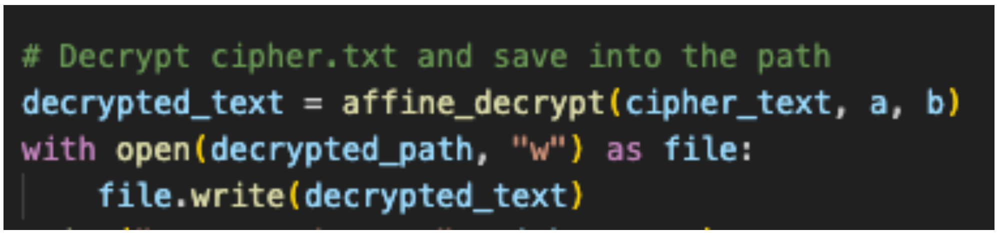
D(y)=a−1×(y−b) mod26



```python
#D(y) = a-1 * (y - b) mod 26
def affine_decrypt(cipher, a, b):
    #the function to inverse the a value
    a_inv = mod_inverse(a, 26)
    result = ''
    for char in cipher:
        if char.isalpha():
            #Convert it to integer for putting in the formula
            base = ord('A') if char.isupper() else ord('a')
            result += chr(((a_inv * ((ord(char) - base - b)) % 26) + base))
        else:
            result += char
    return result
```

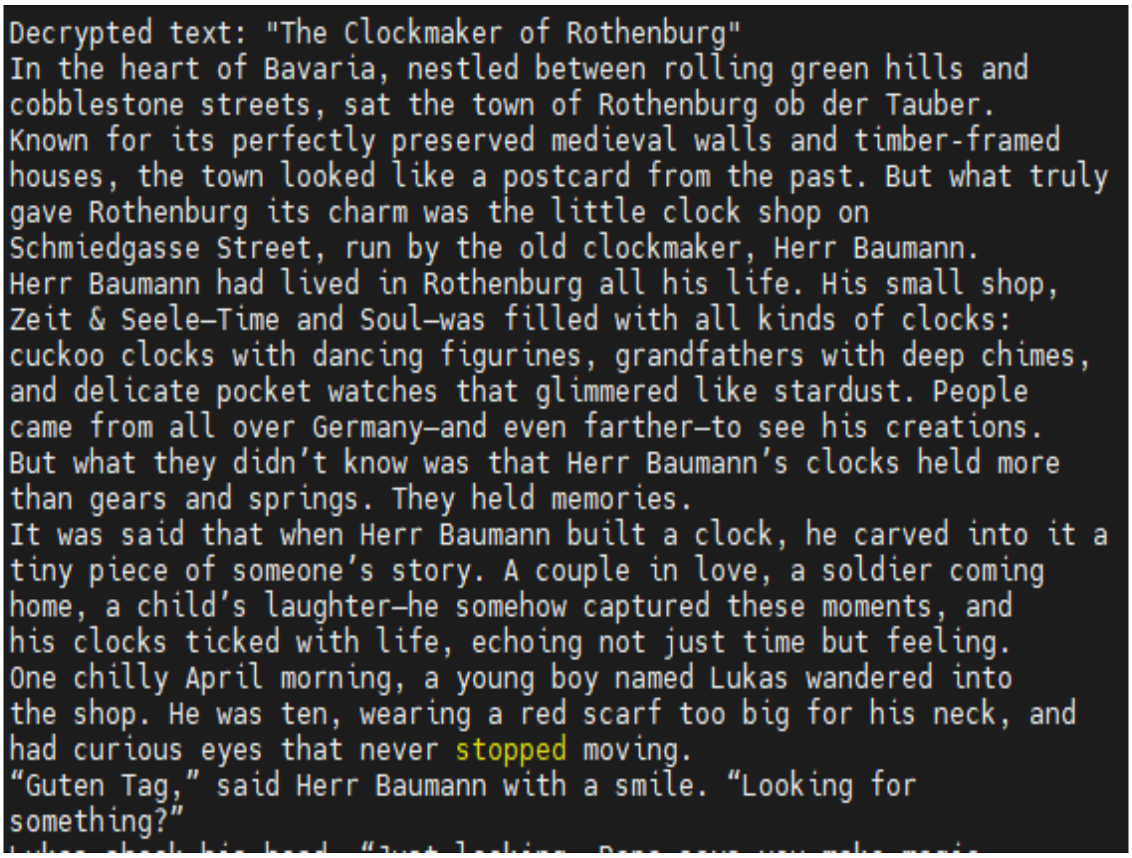
Figure 10 - Converting Cipher Characters

2.2 Adding the decrypted data in the decrypted.txt



```python
# Decrypt cipher.txt and save into the path
decrypted_text = affine_decrypt(cipher_text, a, b)
with open(decrypted_path, "w") as file:
    file.write(decrypted_text)
```

Figure 11 – Adding the decrypted data



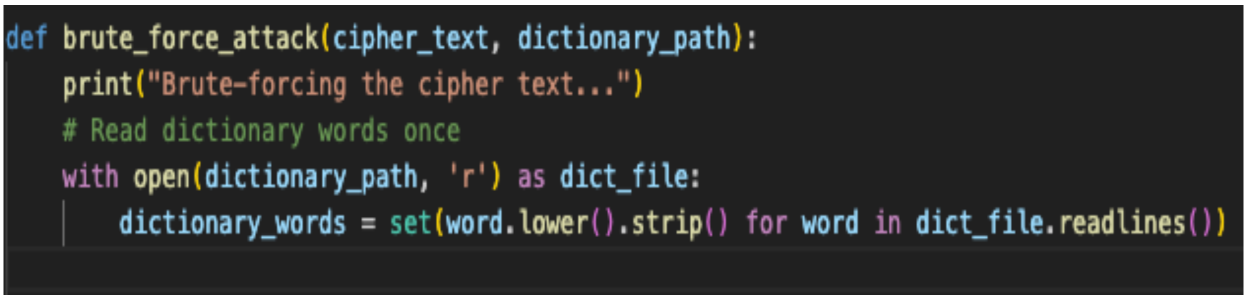
```
Decrypted text: "The Clockmaker of Rothenburg"
In the heart of Bavaria, nestled between rolling green hills and
cobblestone streets, sat the town of Rothenburg ob der Tauber.
Known for its perfectly preserved medieval walls and timber-framed
houses, the town looked like a postcard from the past. But what truly
gave Rothenburg its charm was the little clock shop on
Schmiedgasse Street, run by the old clockmaker, Herr Baumann.
Herr Baumann had lived in Rothenburg all his life. His small shop,
Zeit & Seele—Time and Soul—was filled with all kinds of clocks:
cuckoo clocks with dancing figurines, grandfathers with deep chimes,
and delicate pocket watches that glimmered like stardust. People
came from all over Germany—and even farther—to see his creations.
But what they didn't know was that Herr Baumann's clocks held more
than gears and springs. They held memories.
It was said that when Herr Baumann built a clock, he carved into it a
tiny piece of someone's story. A couple in love, a soldier coming
home, a child's laughter—he somehow captured these moments, and
his clocks ticked with life, echoing not just time but feeling.
One chilly April morning, a young boy named Lukas wandered into
the shop. He was ten, wearing a red scarf too big for his neck, and
had curious eyes that never stopped moving.
"Guten Tag," said Herr Baumann with a smile. "Looking for
something?"
```

Figure 12- Execution of Code

3. Brute-Force Attack

In this method, the hacker will use the default dictionaries on numerous websites or a custom-generated dictionary. For this implementation, I generate a basic dictionary containing frequently used verbs and words to test the possible decryption outcome. Then we are able to analyze the timing.

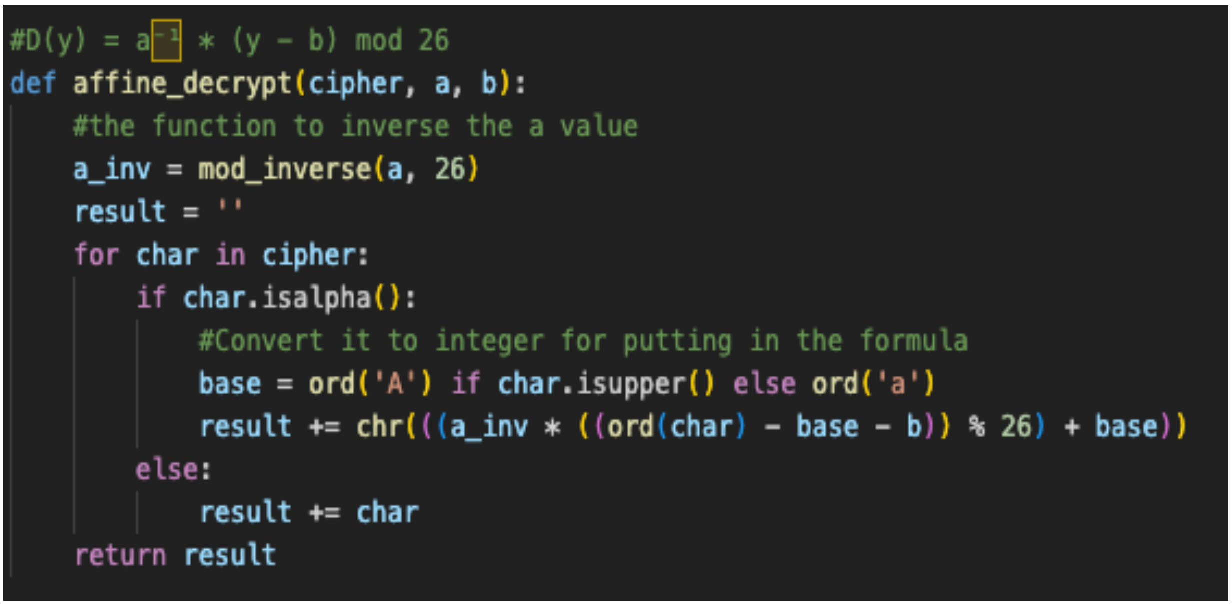
3.1 Reading a basic dictionary created by the hacker (no spaces + lowercase)



```python
def brute_force_attack(cipher_text, dictionary_path):
    print("Brute-forcing the cipher text...")
    # Read dictionary words once
    with open(dictionary_path, 'r') as dict_file:
        dictionary_words = set(word.lower().strip() for word in dict_file.readlines())
```

Figure 13 – Reading a basic Dictionary

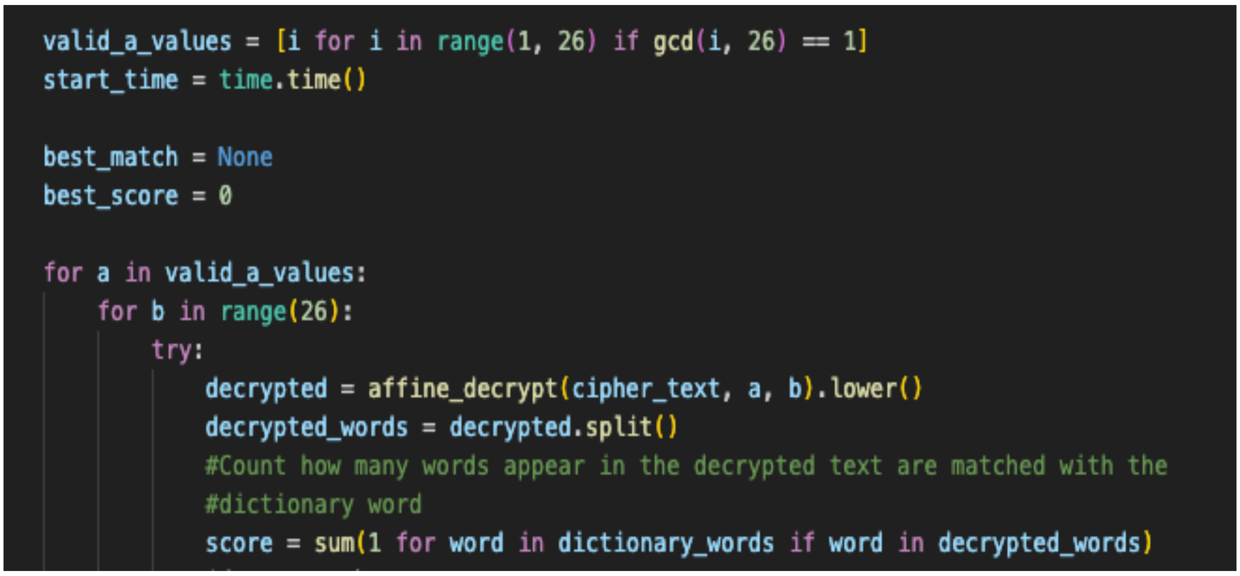
3.2 Using the affine decrypt function to put the values



```python
#D(y) = a⁻¹ * (y - b) mod 26
def affine_decrypt(cipher, a, b):
    #the function to inverse the a value
    a_inv = mod_inverse(a, 26)
    result = ''
    for char in cipher:
        if char.isalpha():
            #Convert it to integer for putting in the formula
            base = ord('A') if char.isupper() else ord('a')
            result += chr(((a_inv * ((ord(char) - base - b)) % 26) + base))
        else:
            result += char
    return result
```

Figure 14 – Using Affine decrypt function

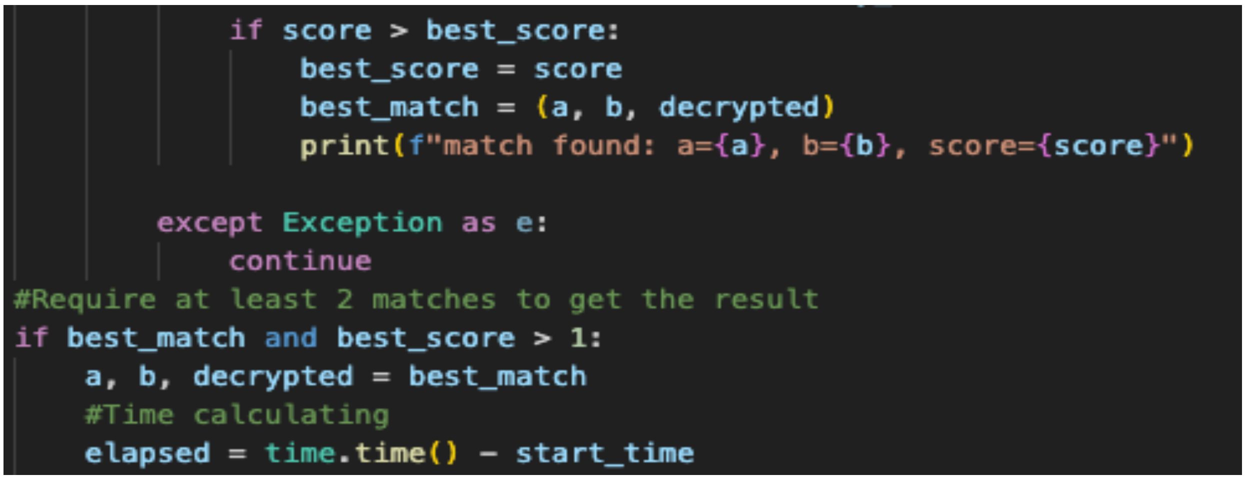
3.3 Counting how many words appear in the decrypted text matches the created dictionary



```python
valid_a_values = [i for i in range(1, 26) if gcd(i, 26) == 1]
start_time = time.time()

best_match = None
best_score = 0

for a in valid_a_values:
    for b in range(26):
        try:
            decrypted = affine_decrypt(cipher_text, a, b).lower()
            decrypted_words = decrypted.split()
            #Count how many words appear in the decrypted text are matched with the
            #dictionary word
            score = sum(1 for word in dictionary_words if word in decrypted_words)
```

Figure 15 – Counting words

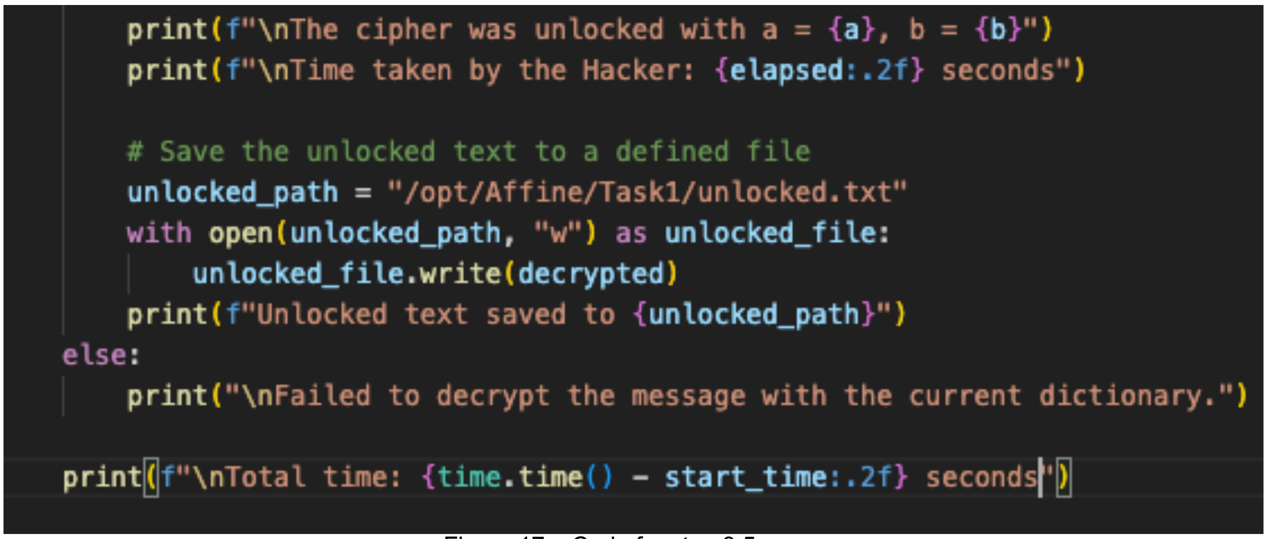
3.4 Finding the correct best_match and best_score



```
                if score > best_score:
                    best_score = score
                    best_match = (a, b, decrypted)
                    print(f"match found: a={a}, b={b}, score={score}")

            except Exception as e:
                continue
#Require at least 2 matches to get the result
if best_match and best_score > 1:
    a, b, decrypted = best_match
    #Time calculating
    elapsed = time.time() - start_time
```

Figure 16 – Finding the correct best_match and best_score

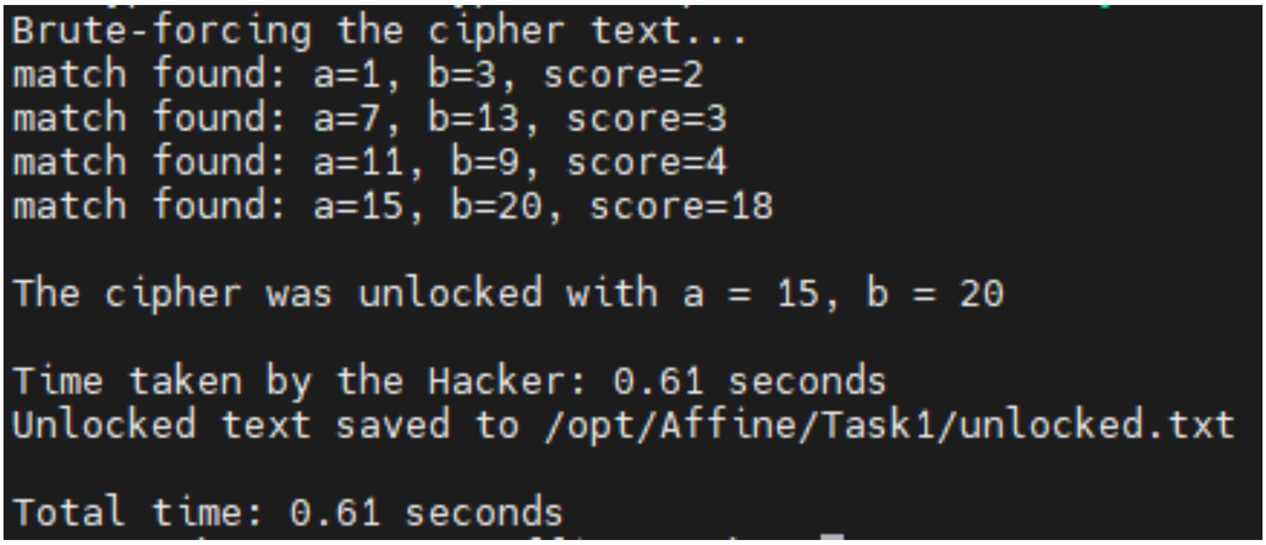
3.5 Writing the unlocked text in the defined path and printing the unlocked text and the time to brute force



```
    print(f"\nThe cipher was unlocked with a = {a}, b = {b}")
    print(f"\nTime taken by the Hacker: {elapsed:.2f} seconds")

    # Save the unlocked text to a defined file
    unlocked_path = "/opt/Affine/Task1/unlocked.txt"
    with open(unlocked_path, "w") as unlocked_file:
        unlocked_file.write(decrypted)
    print(f"Unlocked text saved to {unlocked_path}")
else:
    print("\nFailed to decrypt the message with the current dictionary.")

print(f"\nTotal time: {time.time() - start_time:.2f} seconds")
```

Figure 17 – Code for step 3.5

3.6 Execution Code



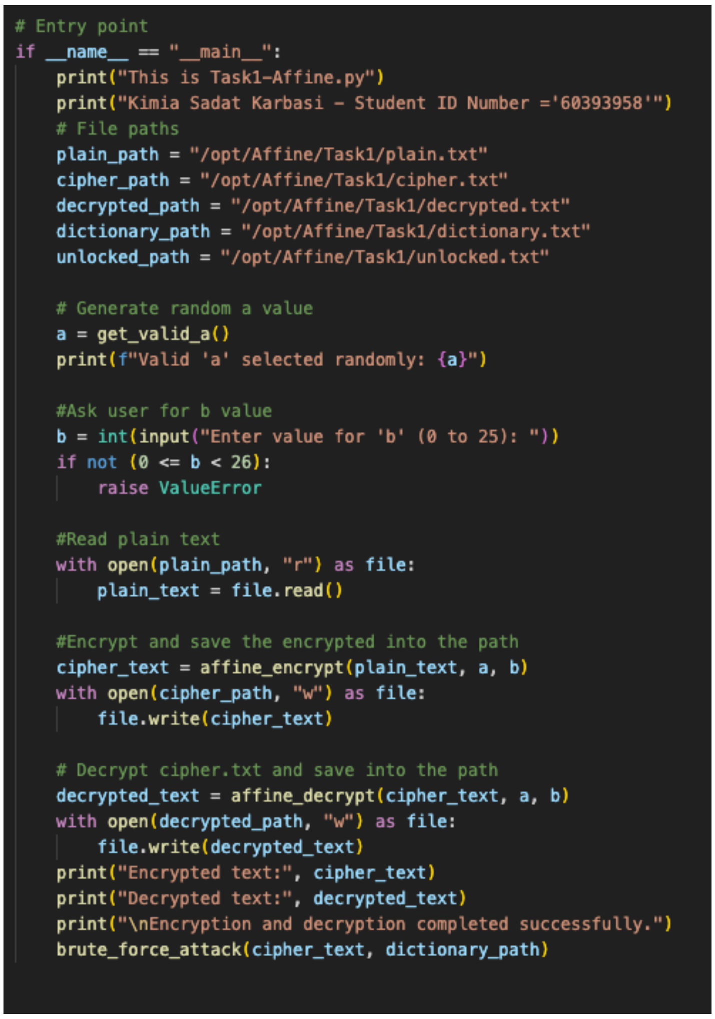
```
Brute-forcing the cipher text...
match found: a=1, b=3, score=2
match found: a=7, b=13, score=3
match found: a=11, b=9, score=4
match found: a=15, b=20, score=18

The cipher was unlocked with a = 15, b = 20

Time taken by the Hacker: 0.61 seconds
Unlocked text saved to /opt/Affine/Task1/unlocked.txt

Total time: 0.61 seconds
```

Figure 18 - Executing the Code

4. Display Output

4.1 Adding the main functions for executing

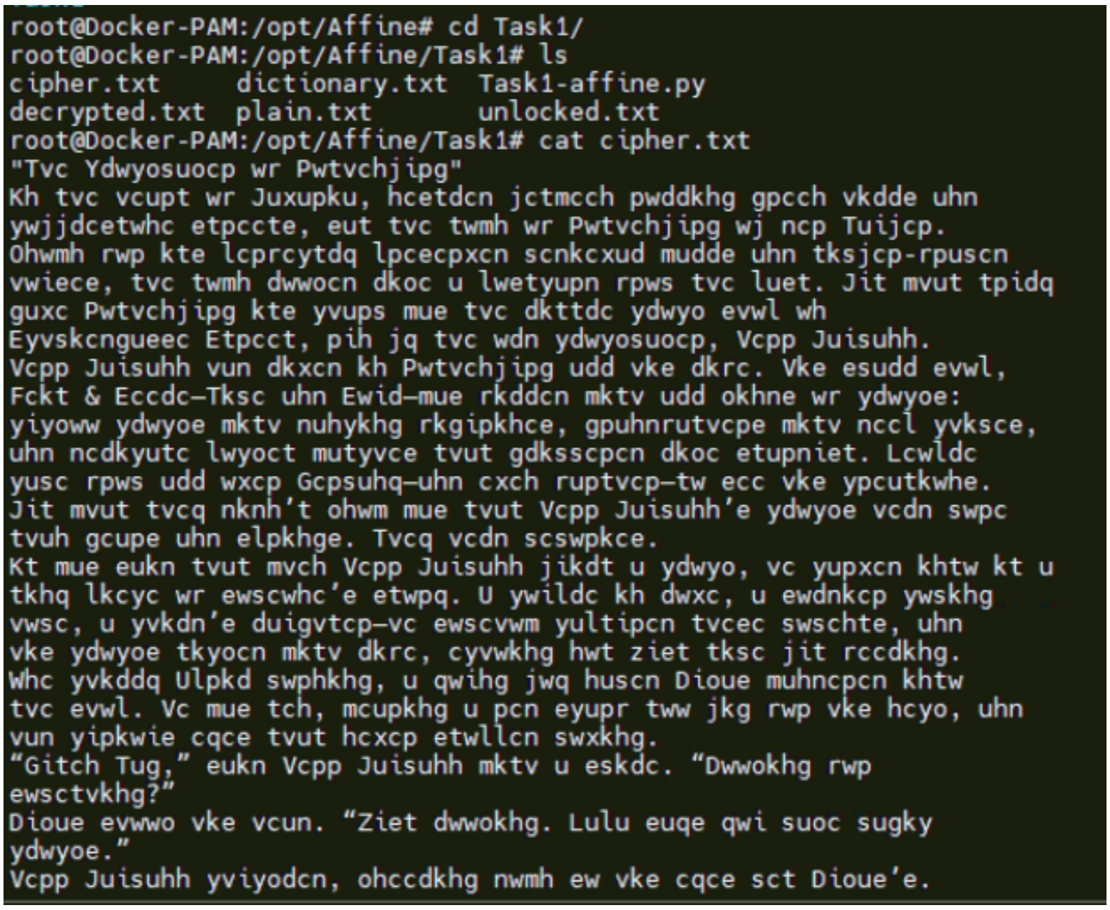
This part involves defining paths, retrieving the values of “a” and “b”, and reading plaintext to execute the data encryption function. Once the data is encrypted, the decrypted result is saved to the correct path. Finally, the ciphertext is used to perform a brute-force attack.



```python
# Entry point
if __name__ == "__main__":
    print("This is Task1-Affine.py")
    print("Kimia Sadat Karbasi - Student ID Number ='60393958'")
    # File paths
    plain_path = "/opt/Affine/Task1/plain.txt"
    cipher_path = "/opt/Affine/Task1/cipher.txt"
    decrypted_path = "/opt/Affine/Task1/decrypted.txt"
    dictionary_path = "/opt/Affine/Task1/dictionary.txt"
    unlocked_path = "/opt/Affine/Task1/unlocked.txt"

    # Generate random a value
    a = get_valid_a()
    print(f"Valid 'a' selected randomly: {a}")

    #Ask user for b value
    b = int(input("Enter value for 'b' (0 to 25): "))
    if not (0 <= b < 26):
        raise ValueError

    #Read plain text
    with open(plain_path, "r") as file:
        plain_text = file.read()

    #Encrypt and save the encrypted into the path
    cipher_text = affine_encrypt(plain_text, a, b)
    with open(cipher_path, "w") as file:
        file.write(cipher_text)

    # Decrypt cipher.txt and save into the path
    decrypted_text = affine_decrypt(cipher_text, a, b)
    with open(decrypted_path, "w") as file:
        file.write(decrypted_text)
    print("Encrypted text:", cipher_text)
    print("Decrypted text:", decrypted_text)
    print("\nEncryption and decryption completed successfully.")
    brute_force_attack(cipher_text, dictionary_path)
```

Figure 19 – Adding the main functions to the code

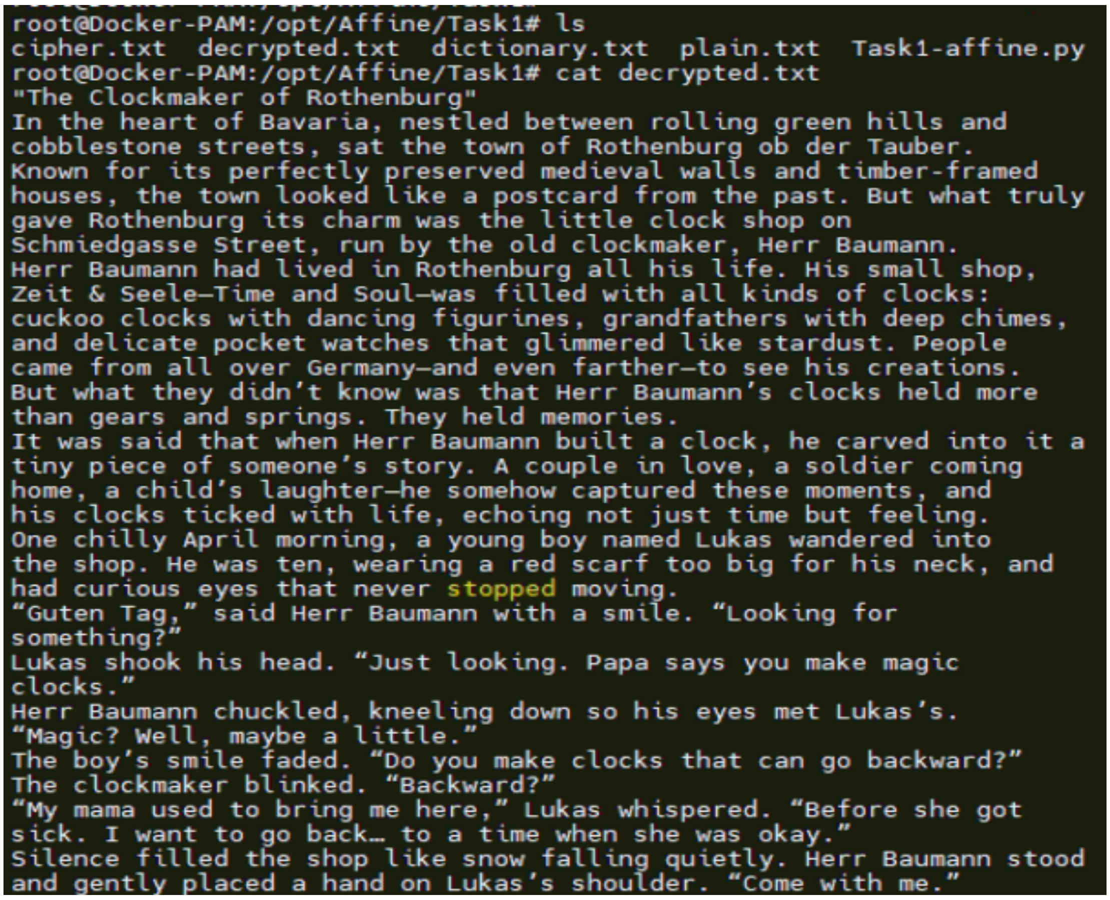
4.2 Display cipher.txt



```
root@Docker-PAM:/opt/Affine# cd Task1/
root@Docker-PAM:/opt/Affine/Task1# ls
cipher.txt     dictionary.txt  Task1-affine.py
decrypted.txt  plain.txt       unlocked.txt
root@Docker-PAM:/opt/Affine/Task1# cat cipher.txt
"Tvc Ydwyosuocp wr Pwtvchjipg"
Kh tvc vcupt wr Juxupku, hcetdcn jctmcch pwddkhg gpcch vkdde uhn
ywjjdcetwhc etpccte, eut tvc twmh wr Pwtvchjipg wj ncp Tuijcp.
Ohwmh rwp kte lcprcytdq lpcecpxcn scnkcxud mudde uhn tksjcp-rpuscn
vwiece, tvc twmh dwwocn dkoc u lwetyupn rpws tvc luet. Jit mvut tpidq
guxc Pwtvchjipg kte yvups mue tvc dkttdc ydwyo evwl wh
Eyvskcngueec Etpcct, pih jq tvc wdn ydwyosuocp, Vcpp Juisuhh.
Vcpp Juisuhh vun dkxcn kh Pwtvchjipg udd vke dkrc. Vke esudd evwl,
Fckt & Eccdc–Tksc uhn Ewid–mue rkddcn mktv udd okhne wr ydwyoe:
yiyoww ydwyoe mktv nuhykhg rkgipkhce, gpuhnrutvcpe mktv nccl yvksce,
uhn ncdkyutc lwyoct mutyvce tvut gdksscpcn dkoc etupniet. Lcwldc
yusc rpws udd wxcp Gcpsuhq–uhn cxch ruptvcp–tw ecc vke ypcutkwhe.
Jit mvut tvcq nknh't ohwm mue tvut Vcpp Juisuhh'e ydwyoe vcdn swpc
tvuh gcupe uhn elpkhge. Tvcq vcdn scswpkce.
Kt mue eukn tvut mvch Vcpp Juisuhh jikdt u ydwyo, vc yupxcn khtw kt u
tkhq lkcyc wr ewscwhc'e etwpq. U ywildc kh dwxc, u ewdnkcp ywskhg
vwsc, u yvkdn'e duigvtcp–vc ewscvwm yultipcn tvcec swschte, uhn
vke ydwyoe tkyocn mktv dkrc, cyvwkhg hwt ziet tksc jit rccdkhg.
Whc yvkddq Ulpkd swphkhg, u qwihg jwq huscn Dioue muhncpcn khtw
tvc evwl. Vc mue tch, mcupkhg u pcn eyupr tww jkg rwp vke hcyo, uhn
vun yipkwie cqce tvut hcxcp etwllcn swxkhg.
"Gitch Tug," eukn Vcpp Juisuhh mktv u eskdc. "Dwwokhg rwp
ewsctvkhg?"
Dioue evwwo vke vcun. "Ziet dwwokhg. Lulu euqe qwi suoc sugky
ydwyoe."
Vcpp Juisuhh yviyodcn, ohccdkhg nwmh ew vke cqce sct Dioue'e.
```

Figure 20 – Display Cipher Text

4.3 Display decrypted.txt

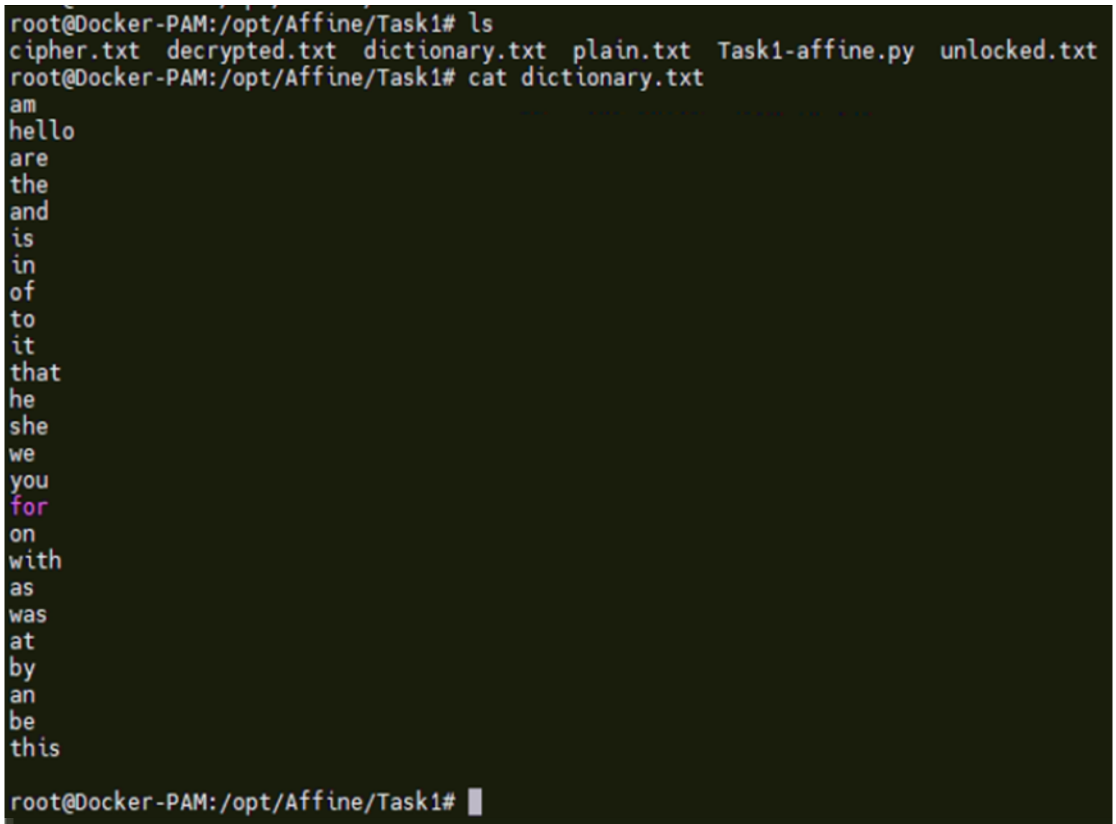


```
root@Docker-PAM:/opt/Affine/Task1# ls
cipher.txt  decrypted.txt  dictionary.txt  plain.txt  Task1-affine.py
root@Docker-PAM:/opt/Affine/Task1# cat decrypted.txt
"The Clockmaker of Rothenburg"
In the heart of Bavaria, nestled between rolling green hills and
cobblestone streets, sat the town of Rothenburg ob der Tauber.
Known for its perfectly preserved medieval walls and timber-framed
houses, the town looked like a postcard from the past. But what truly
gave Rothenburg its charm was the little clock shop on
Schmiedgasse Street, run by the old clockmaker, Herr Baumann.
Herr Baumann had lived in Rothenburg all his life. His small shop,
Zeit & Seele–Time and Soul–was filled with all kinds of clocks:
cuckoo clocks with dancing figurines, grandfathers with deep chimes,
and delicate pocket watches that glimmered like stardust. People
came from all over Germany–and even farther–to see his creations.
But what they didn't know was that Herr Baumann's clocks held more
than gears and springs. They held memories.
It was said that when Herr Baumann built a clock, he carved into it a
tiny piece of someone's story. A couple in love, a soldier coming
home, a child's laughter–he somehow captured these moments, and
his clocks ticked with life, echoing not just time but feeling.
One chilly April morning, a young boy named Lukas wandered into
the shop. He was ten, wearing a red scarf too big for his neck, and
had curious eyes that never stopped moving.
"Guten Tag," said Herr Baumann with a smile. "Looking for
something?"
Lukas shook his head. "Just looking. Papa says you make magic
clocks."
Herr Baumann chuckled, kneeling down so his eyes met Lukas's.
"Magic? Well, maybe a little."
The boy's smile faded. "Do you make clocks that can go backward?"
The clockmaker blinked. "Backward?"
"My mama used to bring me here," Lukas whispered. "Before she got
sick. I want to go back… to a time when she was okay."
Silence filled the shop like snow falling quietly. Herr Baumann stood
and gently placed a hand on Lukas's shoulder. "Come with me."
```

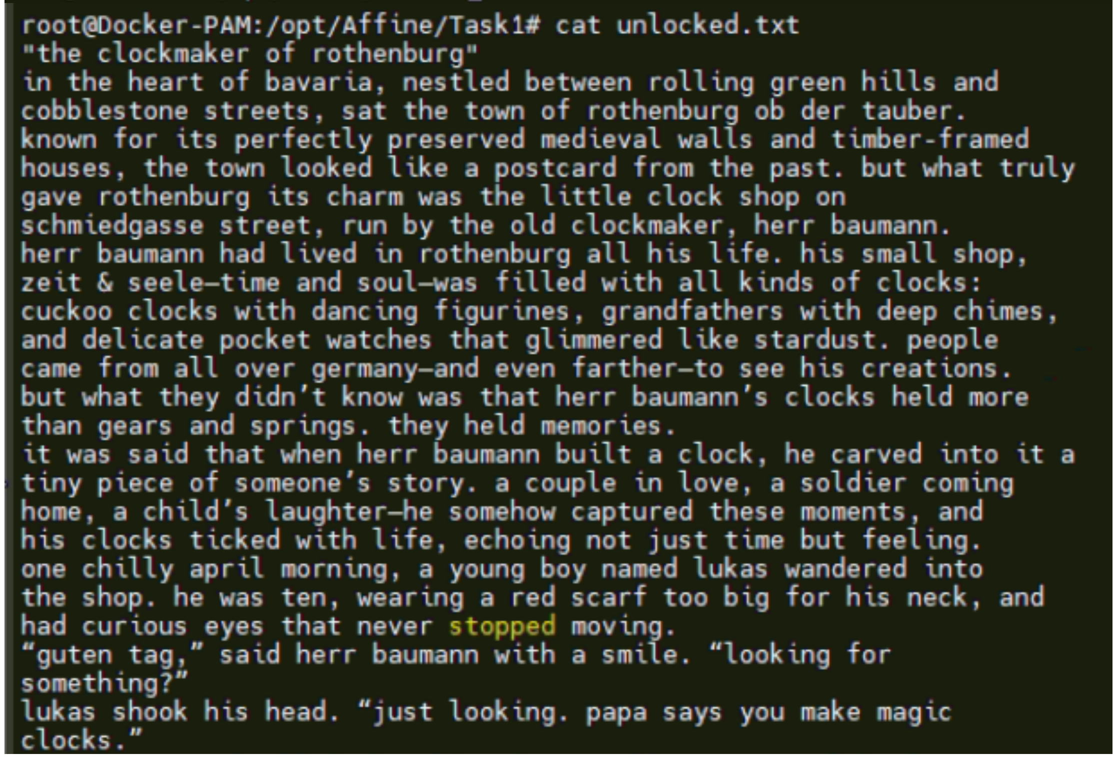
Figure 21- Decrypted Text

4.4 Display a customized dictionary and Brute-Force Attack (unlocked.txt)

We can use numerous dictionaries on websites and use tools for creating a dictionary list. But in this case, we used our own dictionary to get a better outcome.



```
root@Docker-PAM:/opt/Affine/Task1# ls
cipher.txt  decrypted.txt  dictionary.txt  plain.txt  Task1-affine.py  unlocked.txt
root@Docker-PAM:/opt/Affine/Task1# cat dictionary.txt
am
hello
are
the
and
is
in
of
to
it
that
he
she
we
you
for
on
with
as
was
at
by
an
be
this

root@Docker-PAM:/opt/Affine/Task1# 
```



```
root@Docker-PAM:/opt/Affine/Task1# cat unlocked.txt
"the clockmaker of rothenburg"
in the heart of bavaria, nestled between rolling green hills and
cobblestone streets, sat the town of rothenburg ob der tauber.
known for its perfectly preserved medieval walls and timber-framed
houses, the town looked like a postcard from the past. but what truly
gave rothenburg its charm was the little clock shop on
schmiedgasse street, run by the old clockmaker, herr baumann.
herr baumann had lived in rothenburg all his life. his small shop,
zeit & seele—time and soul—was filled with all kinds of clocks:
cuckoo clocks with dancing figurines, grandfathers with deep chimes,
and delicate pocket watches that glimmered like stardust. people
came from all over germany—and even farther—to see his creations.
but what they didn't know was that herr baumann's clocks held more
than gears and springs. they held memories.
it was said that when herr baumann built a clock, he carved into it a
tiny piece of someone's story. a couple in love, a soldier coming
home, a child's laughter—he somehow captured these moments, and
his clocks ticked with life, echoing not just time but feeling.
one chilly april morning, a young boy named lukas wandered into
the shop. he was ten, wearing a red scarf too big for his neck, and
had curious eyes that never stopped moving.
"guten tag," said herr baumann with a smile. "looking for
something?"
lukas shook his head. "just looking. papa says you make magic
clocks."
```

Figure 22 – Code Outcome

Conclusion

This report demonstrated the implementation of the Affine Cipher in Python, covering encryption, decryption, brute-force attack simulation, and output display. The encryption process transformed plaintext into ciphertext using modular arithmetic, while decryption recovered the original message using the inverse key operation. The brute-force attack phase tested the cipher’s vulnerability by systematically checking all possible keys with a created dictionary to measure the execution time.