Jorge Ignacio Serrano Baez - 164467
Adrian Enrique Diaz Fornes - 171791.
Jonathan Eliasib Rosas Tlaczani - 168399.

HW No.1: Encryption and Decryption Problems Using Classical Ciphers.

Selected Topics 1 - LRT3062

May 6, 2025

Spring 2024

Robotics and Telecommunications Engineering.

Computation, Electronics and Mecatronics Department, Universidad de las Américas Puebla, Puebla, México 72810

Abstract

This assignment provides analytic and practical experience using encryption and decryption processes. Students will develop classical cipher using any programming language . The importance of this activity is to get trough two different encryption process to get familiarized the process of encryption and decryption in each one. Also to implement an attack in the Cesar's cipher to demonstrate the security of the cipher.

Introduction

The encryption and decryption play a fundamental role in information security by preserving the confidentiality and integrity of communication. Encryption transforms plaintext into unreadable ciphertext, providing a protective layer against unauthorized access, interpretation, or modification of sensitive data. This process is paramount in securing information during transmission or storage, serving as a cornerstone in contemporary cybersecurity practices.

Encryption & Decryption
Plaintext
Encryption
Ciphertext
Decryption
Plaintext

Figure 1: Encryption and Decryption

The concept of encryption dates back to ancient civilizations, notably used by the Egyptians and Romans. However, modern encryption techniques have significantly evolved, particularly in the mid-20th century with the rise of computers and electronic communication. The adoption of the Data Encryption Standard (DES) in the 1970s marked a milestone in encryption implementation.

Since then, encryption has continually evolved to address emerging security challenges, with the development of more sophisticated algorithms and protocols. In our digital age, where communication and data storage are prevalent, encryption remains a dynamic and essential component in ensuring secure digital interactions and protecting sensitive information.

Caesar Cipher: The Caesar Cipher represents one of the simplest encryption methods. It involves shifting each letter in the original message by a fixed number of positions in the alphabet. For instance, a shift of 3 would turn the letter 'A' into 'D'. Despite its simplicity, the Caesar Cipher is vulnerable to brute force attacks, where an adversary systematically tests all possible combinations to decipher the message.

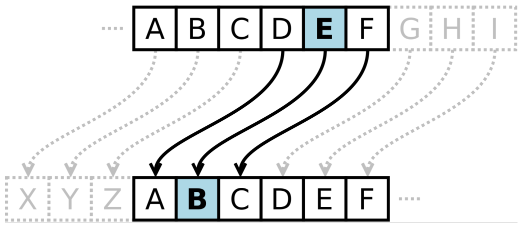


Figure 2: Cesar's cipher

Playfair Cipher: In contrast, the Playfair Cipher is more complex. It employs a key matrix to substitute pairs of letters in the original message, enhancing security. However, it is not immune to vulnerabilities. Analytical attacks may attempt to decrypt the message by identifying repetition patterns, frequencies, and the position of letters, necessitating a deeper analysis of language and letter frequency compared to brute force methods.

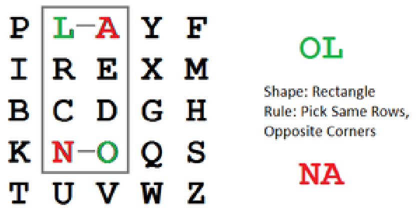


Figure 3: Cesar's cipher

In the context of encryption ciphers, an "attack" refers to a deliberate attempt to compromise the security of an encrypted message or system, often with the goal of deciphering the encrypted data without knowledge of the decryption key. These attacks aim to exploit weaknesses in the encryption algorithm, the key management, or the implementation of the cryptographic system. Here are some common types of attacks in the realm of encryption ciphers:

Figure 4: Cesar's cipher

Analytical Attack: Analytical attacks, particularly relevant to the Playfair Cipher, involve scrutinizing patterns, frequencies, and letter positions to infer the used key. This sophisticated approach requires a comprehensive understanding of language nuances and letter distribution. Brute Force Attack: On the other hand, brute force attacks involve trying all possible combinations until finding the correct one. While effective for simple ciphers like the Caesar Cipher, it becomes inefficient for more complex ciphers such as Playfair due to the vast potential key space.

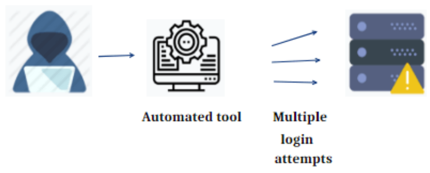


Figure 5: Brute Force attack

Methodology

In this assignment, in order to get familiar with procedure of encryption and decryption using classical ciphers, the following programming problems must be addressed:, each with its own steps and methodology:

Activity 1

For the first activity this steps need to be followed:

1. Develop a program in any language to the Cesar´s cipher for encryption and decryption that take a key between 0 to 25.
2. with the program finished, use this sentences as an example for the encryption with different keys.
   - k = 6 plaintext = ”Get me a vanilla ice cream, make it a double.”
   - k = 15 plaintext = ” I don't much care for Leonard Cohen.”
   - k = 16 plaintext = ”I like root beer floats.”
3. After this two steps verify that the program is working correctly for the encryption of the sentences, for a good communication.
4. Then with the decryption program in the same language use the cipher text given below for the decryption and obtain the plain text.
   - k = 12 ciphertext = 'nduzs ftq buzq oazqe.'
   - k = 3 ciphertext = ”fdhvdu qhhgv wr orvh zhljkw.”
   - k = 20 ciphertext = ”ufgihxm uly numnys.”
5. Verify that the keys are used in a correct form by obtaining the plaintext in the decryption program.
6. Subsequently develop a program that generate a Brute Force attack to the Cesar´s cipher, to obtain the plaintext and the different keys that are possible.
7. Also develop a program for the Playfair cipher for encryption and decryption with a key that is given by the user.
8. Then to implement a function for the Caesar cipher that performs a brute force attack on a ciphertext, showing all the possibles keys to decrypt the ciphertext. It should also take an optional parameter that takes a substring and only prints out potential plaintexts that contain that decryption trying this kewwords:
   - ciphertext = 'gryy gurz gb tb gb nzoebfr puncry.' keyword = 'chapel'
   - ciphertext = 'wziv kyv jyfk nyve kyv tpdsrcj tirjy.' keyword = 'cymbal'
   - ciphertext = 'baeeq klwosjl osk s esf ozg cfwo lgg emuz.' no keyword
9. verify that the cipehrtext is decrypted in a correct form for the understanding of the plaintext using the keywords previusly given.

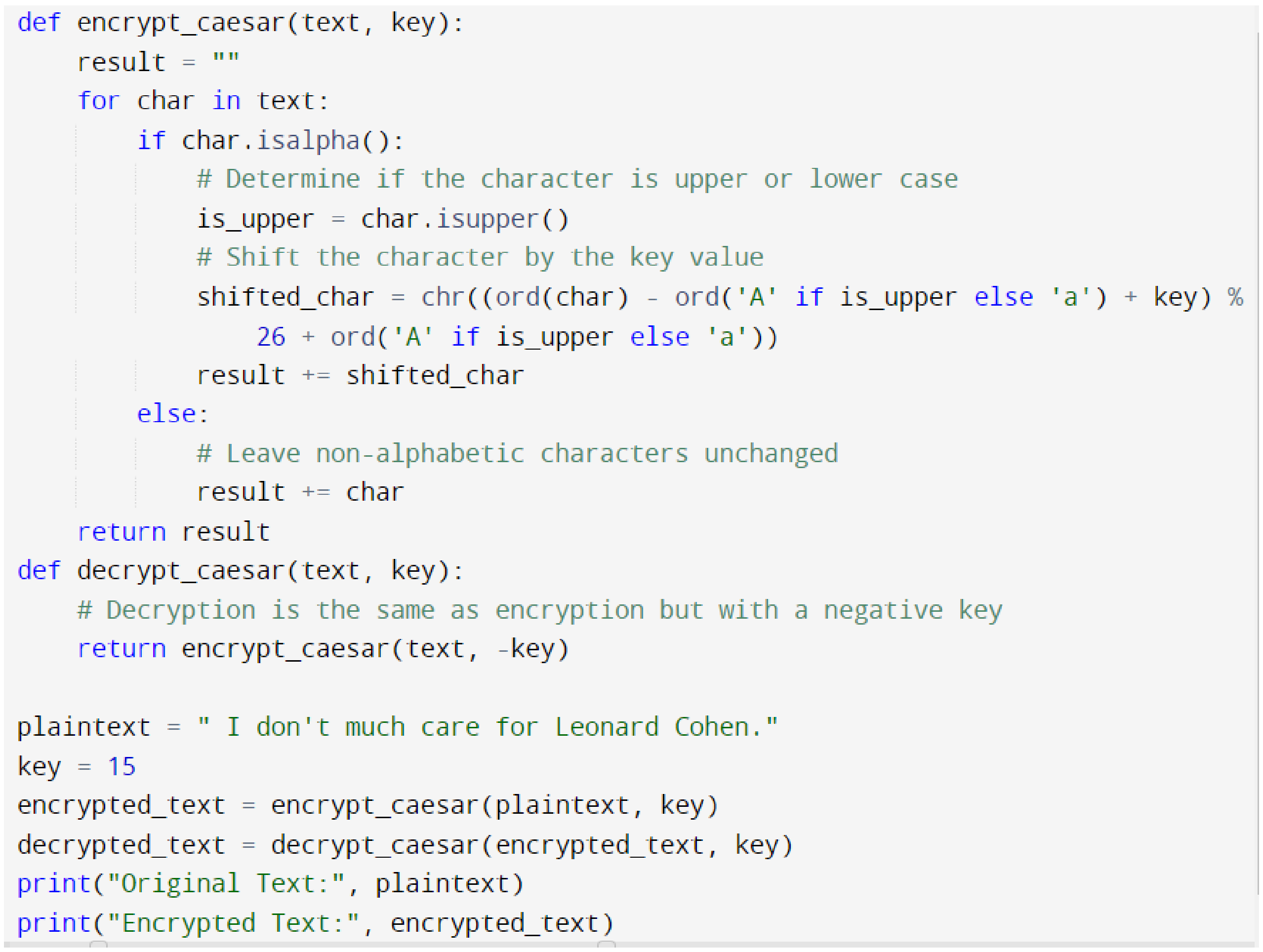
Activity 2

1. Finally, at the end of the activity its nedeed to implement the codes of the Cesar´s cipher in VHDL to simualte it.

Results and Discussions

For the firs activity as it was mentioned in the metodology, a programming language were selected Pyton in this case for develop all the programs that are requested in the activities.

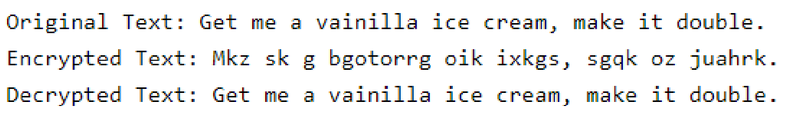
A program was developed for the Cesar´s cipher for encrypting a message that it is going to be send. the code its showed below:



```python
def encrypt_caesar(text, key):
    result = ""
    for char in text:
        if char.isalpha():
            # Determine if the character is upper or lower case
            is_upper = char.isupper()
            # Shift the character by the key value
            shifted_char = chr((ord(char) - ord('A' if is_upper else 'a') + key) %
                26 + ord('A' if is_upper else 'a'))
            result += shifted_char
        else:
            # Leave non-alphabetic characters unchanged
            result += char
    return result
def decrypt_caesar(text, key):
    # Decryption is the same as encryption but with a negative key
    return encrypt_caesar(text, -key)

plaintext = " I don't much care for Leonard Cohen."
key = 15
encrypted_text = encrypt_caesar(plaintext, key)
decrypted_text = decrypt_caesar(encrypted_text, key)
print("Original Text:", plaintext)
print("Encrypted Text:", encrypted_text)
```

Figure 6: Cesar´s cipher code

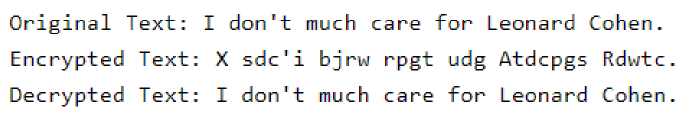
For the sentence given "Get me a vanilla ice cream, make it a double." with a key with the value 6, this ciphertext was obtained:



```
Original Text: Get me a vainilla ice cream, make it double.
Encrypted Text: Mkz sk g bgotorrg oik ixkgs, sgqk oz juahrk.
Decrypted Text: Get me a vainilla ice cream, make it double.
```

Figure 7: Cesar´s cipher 1

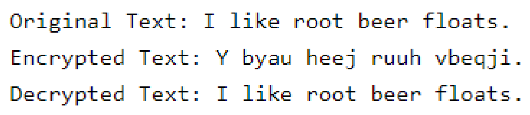
For the seconnd sentence "don't much care for Leonard Cohen" with the key value e qual to 15, the cipher texts is this:



```
Original Text: I don't much care for Leonard Cohen.
Encrypted Text: X sdc'i bjrw rpgt udg Atdcpgs Rdwtc.
Decrypted Text: I don't much care for Leonard Cohen.
```

Figure 8: Cesar´s cipher 2

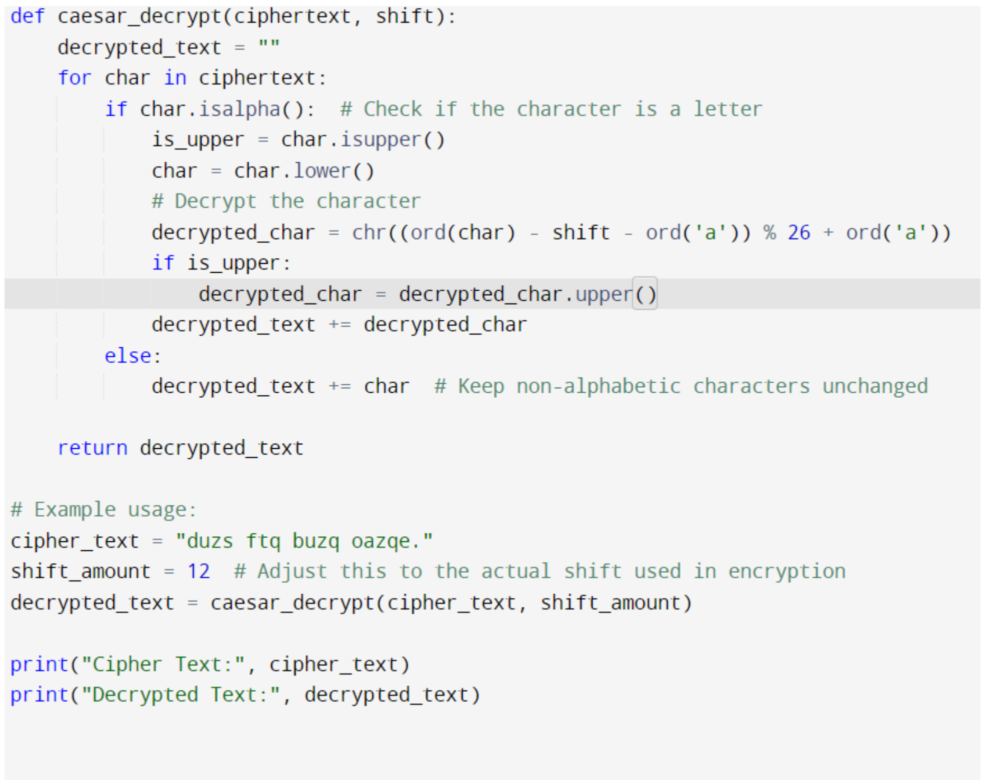
For the third one ”I like root beer floats.” with the key equal to 16, the cipher text is this:



```
Original Text: I like root beer floats.
Encrypted Text: Y byau heej ruuh vbeqji.
Decrypted Text: I like root beer floats.
```

Figure 9: Cesar´s cipher 3

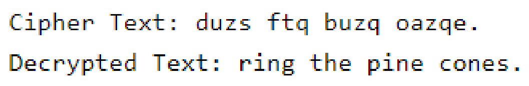
For the second part of the activity a drecyption fot the cesars cipher needs to be done, the next code show the process to decrypth a cyphertext having the key:



```python
def caesar_decrypt(ciphertext, shift):
    decrypted_text = ""
    for char in ciphertext:
        if char.isalpha():  # Check if the character is a letter
            is_upper = char.isupper()
            char = char.lower()
            # Decrypt the character
            decrypted_char = chr((ord(char) - shift - ord('a')) % 26 + ord('a'))
            if is_upper:
                decrypted_char = decrypted_char.upper()
            decrypted_text += decrypted_char
        else:
            decrypted_text += char  # Keep non-alphabetic characters unchanged

    return decrypted_text

# Example usage:
cipher_text = "duzs ftq buzq oazqe."
shift_amount = 12  # Adjust this to the actual shift used in encryption
decrypted_text = caesar_decrypt(cipher_text, shift_amount)

print("Cipher Text:", cipher_text)
print("Decrypted Text:", decrypted_text)
```

Figure 10: Decrypt Cesar´s cipher code

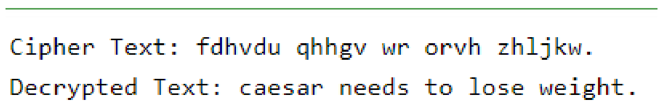
For the first ciphertext ”duzs ftq buzq oazqe.” with the key equal to 12, the plain text is the next one:



```
Cipher Text: duzs ftq buzq oazqe.
Decrypted Text: ring the pine cones.
```

Figure 11: Decrypt Cesar´s cipher 1

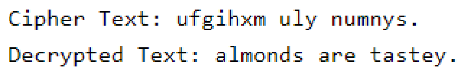
For the next one with the ciphertext ”fdhvdu qhhgv wr orvh zhljkw.” with the key equal to 3, the plaintext is the next one:



```
Cipher Text: fdhvdu qhhgv wr orvh zhljkw.
Decrypted Text: caesar needs to lose weight.
```

Figure 12: Decrypt Cesar´s cipher 2

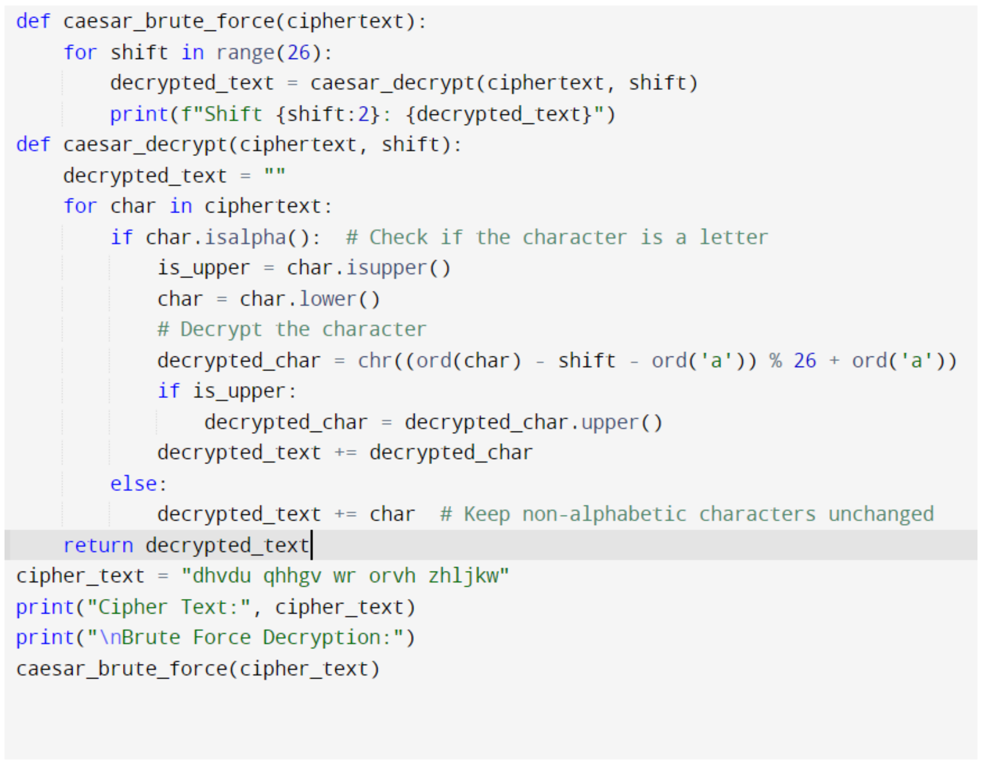
Finally, for the third ciphertext ”ufgihxm uly numnys.” with the key equal to 20, the plaintext is the nest one



```
Cipher Text: ufgihxm uly numnys.
Decrypted Text: almonds are tastey.
```

Figure 13: Decrypt Cesar´s cipher 3

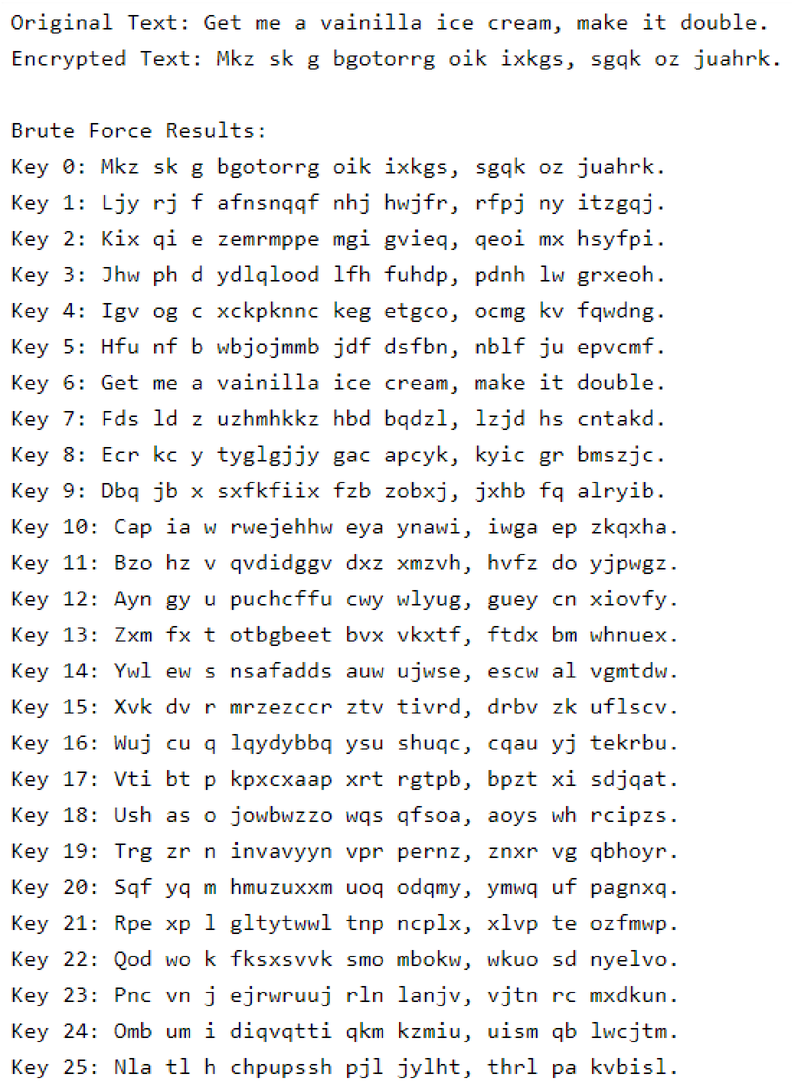
Also the development of a code to generate a brute force attack, trying all the 25 combination that are posible in the ciphertext, the code below shows how it is establish in pyton:



```
def caesar_brute_force(ciphertext):
    for shift in range(26):
        decrypted_text = caesar_decrypt(ciphertext, shift)
        print(f"Shift {shift:2}: {decrypted_text}")
def caesar_decrypt(ciphertext, shift):
    decrypted_text = ""
    for char in ciphertext:
        if char.isalpha():  # Check if the character is a letter
            is_upper = char.isupper()
            char = char.lower()
            # Decrypt the character
            decrypted_char = chr((ord(char) - shift - ord('a')) % 26 + ord('a'))
            if is_upper:
                decrypted_char = decrypted_char.upper()
            decrypted_text += decrypted_char
        else:
            decrypted_text += char  # Keep non-alphabetic characters unchanged
    return decrypted_text
cipher_text = "dhvdu qhhgv wr orvh zhljkw"
print("Cipher Text:", cipher_text)
print("\nBrute Force Decryption:")
caesar_brute_force(cipher_text)
```

Figure 14: Attack Cesar´s cipher code

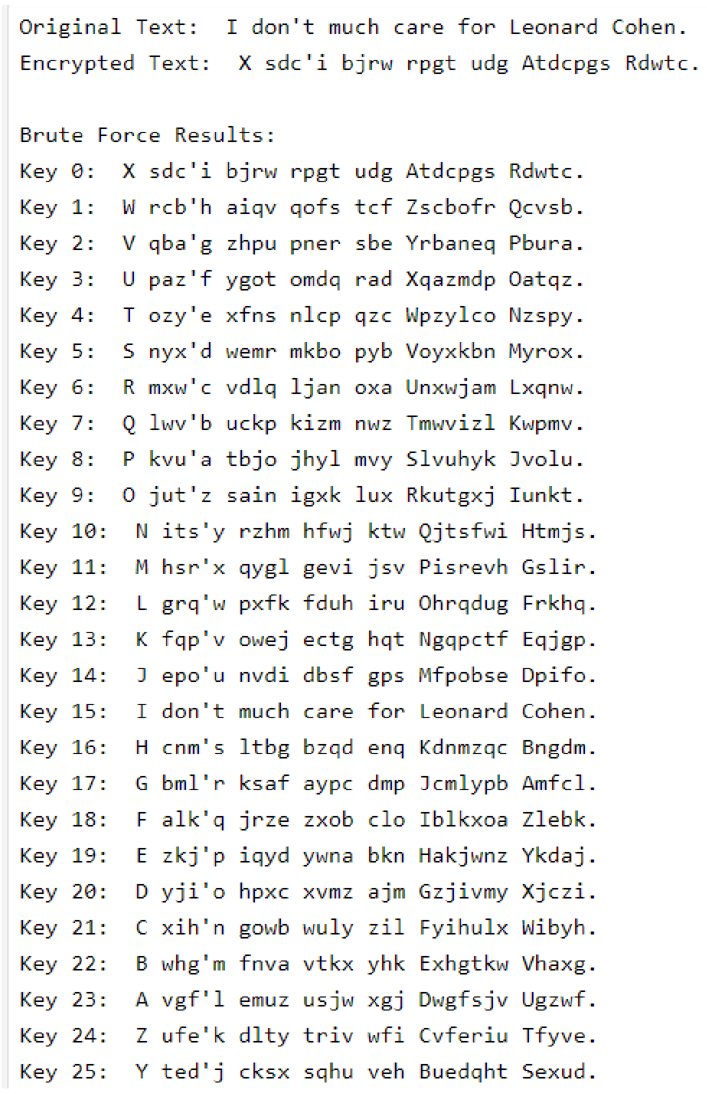
As it is sowed in the images below, the code generate a brute force attack were it try all th possible combination, in this case 25 to know the plaintext.



```
Original Text: Get me a vainilla ice cream, make it double.
Encrypted Text: Mkz sk g bgotorrg oik ixkgs, sgqk oz juahrk.

Brute Force Results:
Key 0: Mkz sk g bgotorrg oik ixkgs, sgqk oz juahrk.
Key 1: Ljy rj f afnsnqqf nhj hwjfr, rfpj ny itzgqj.
Key 2: Kix qi e zemrmppe mgi gvieq, qeoi mx hsyfpi.
Key 3: Jhw ph d ydlqlood lfh fuhdp, pdnh lw grxeoh.
Key 4: Igv og c xckpknnc keg etgco, ocmg kv fqwdng.
Key 5: Hfu nf b wbjojmmb jdf dsfbn, nblf ju epvcmf.
Key 6: Get me a vainilla ice cream, make it double.
Key 7: Fds ld z uzhmhkkz hbd bqdzl, lzjd hs cntakd.
Key 8: Ecr kc y tyglgjjy gac apcyk, kyic gr bmszjc.
Key 9: Dbq jb x sxfkfiix fzb zobxj, jxhb fq alryib.
Key 10: Cap ia w rwejehhw eya ynawi, iwga ep zkqxha.
Key 11: Bzo hz v qvdidggv dxz xmzvh, hvfz do yjpwgz.
Key 12: Ayn gy u puchcffu cwy wlyug, guey cn xiovfy.
Key 13: Zxm fx t otbgbeet bvx vkxtf, ftdx bm whnuex.
Key 14: Ywl ew s nsafadds auw ujwse, escw al vgmtdw.
Key 15: Xvk dv r mrzezccr ztv tivrd, drbv zk uflscv.
Key 16: Wuj cu q lqydybbq ysu shuqc, cqau yj tekrbu.
Key 17: Vti bt p kpxcxaap xrt rgtpb, bpzt xi sdjqat.
Key 18: Ush as o jowbwzzo wqs qfsoa, aoys wh rcipzs.
Key 19: Trg zr n invavyyn vpr pernz, znxr vg qbhoyr.
Key 20: Sqf yq m hmuzuxxm uoq odqmy, ymwq uf pagnxq.
Key 21: Rpe xp l gltytwwl tnp ncplx, xlvp te ozfmwp.
Key 22: Qod wo k fksxsvvk smo mbokw, wkuo sd nyelvo.
Key 23: Pnc vn j ejrwruuj rln lanjv, vjtn rc mxdkun.
Key 24: Omb um i diqvqtti qkm kzmiu, uism qb lwcjtm.
Key 25: Nla tl h chpupssh pjl jylht, thrl pa kvbisl.
```

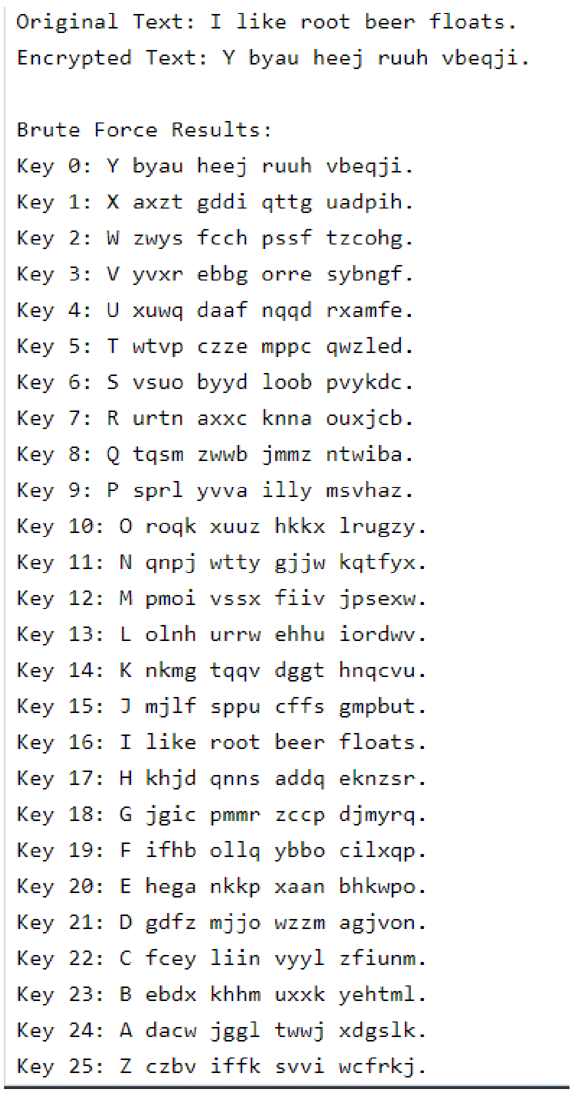
Figure 15: Attack Cesar´s cipher 2

For each key-plaintext pair, the function will try all possible keys (0 to 25) and print the associated decryptions. Since the function performs a brute force attack, it will try all possible keys and print the potential plaintexts.



```
Original Text:  I don't much care for Leonard Cohen.
Encrypted Text:  X sdc'i bjrw rpgt udg Atdcpgs Rdwtc.

Brute Force Results:
Key 0:  X sdc'i bjrw rpgt udg Atdcpgs Rdwtc.
Key 1:  W rcb'h aiqv qofs tcf Zscbofr Qcvsb.
Key 2:  V qba'g zhpu pner sbe Yrbaneq Pbura.
Key 3:  U paz'f ygot omdq rad Xqazmdp Oatqz.
Key 4:  T ozy'e xfns nlcp qzc Wpzylco Nzspy.
Key 5:  S nyx'd wemr mkbo pyb Voyxkbn Myrox.
Key 6:  R mxw'c vdlq ljan oxa Unxwjam Lxqnw.
Key 7:  Q lwv'b uckp kizm nwz Tmwvizl Kwpmv.
Key 8:  P kvu'a tbjo jhyl mvy Slvuhyk Jvolu.
Key 9:  O jut'z sain igxk lux Rkutgxj Iunkt.
Key 10:  N its'y rzhm hfwj ktw Qjtsfwi Htmjs.
Key 11:  M hsr'x qygl gevi jsv Pisrevh Gslir.
Key 12:  L grq'w pxfk fduh iru Ohrqdug Frkhq.
Key 13:  K fqp'v owej ectg hqt Ngqpctf Eqjgp.
Key 14:  J epo'u nvdi dbsf gps Mfpobse Dpifo.
Key 15:  I don't much care for Leonard Cohen.
Key 16:  H cnm's ltbg bzqd enq Kdnmzqc Bngdm.
Key 17:  G bml'r ksaf aypc dmp Jcmlypb Amfcl.
Key 18:  F alk'q jrze zxob clo Iblkxoa Zlebk.
Key 19:  E zkj'p iqyd ywna bkn Hakjwnz Ykdaj.
Key 20:  D yji'o hpxc xvmz ajm Gzjivmy Xjczi.
Key 21:  C xih'n gowb wuly zil Fyihulx Wibyh.
Key 22:  B whg'm fnva vtkx yhk Exhgtkw Vhaxg.
Key 23:  A vgf'l emuz usjw xgj Dwgfsjv Ugzwf.
Key 24:  Z ufe'k dlty triv wfi Cvferiu Tfyve.
Key 25:  Y ted'j cksx sqhu veh Buedqht Sexud.
```

Figure 16: Attack Cesar´s cipher 2

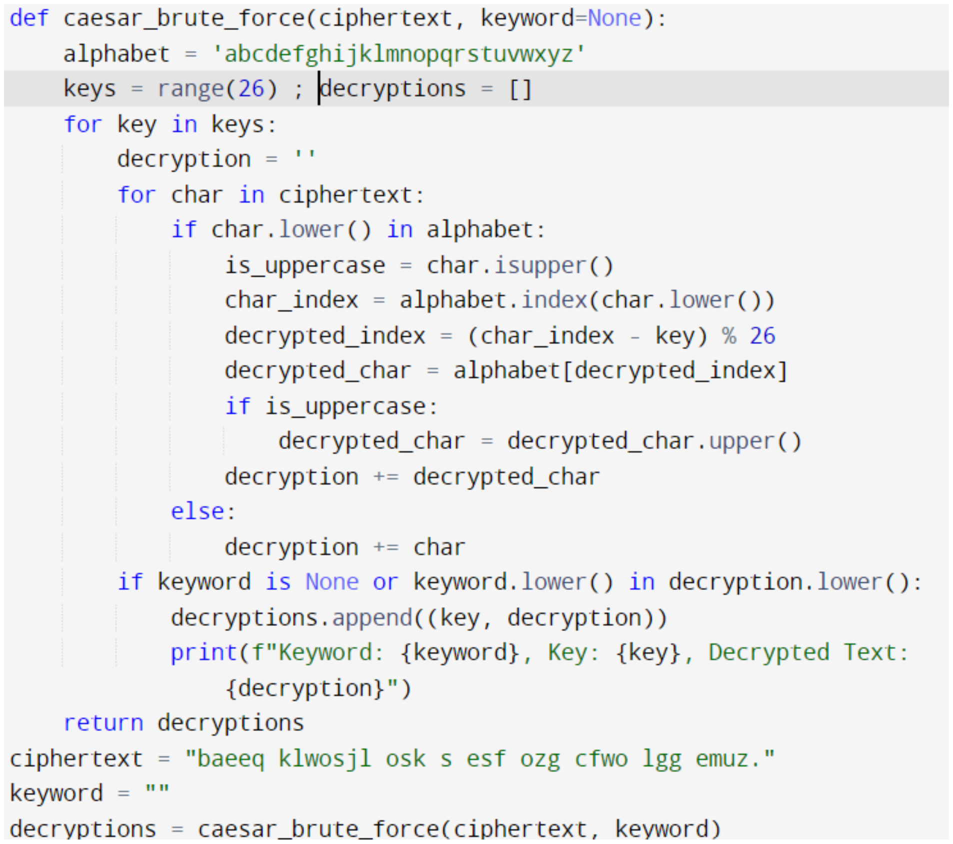


```
Original Text: I like root beer floats.
Encrypted Text: Y byau heej ruuh vbeqji.

Brute Force Results:
Key 0: Y byau heej ruuh vbeqji.
Key 1: X axzt gddi qttg uadpih.
Key 2: W zwys fcch pssf tzcohg.
Key 3: V yvxr ebbg orre sybngf.
Key 4: U xuwq daaf nqqd rxamfe.
Key 5: T wtvp czze mppc qwzled.
Key 6: S vsuo byyd loob pvykdc.
Key 7: R urtn axxc knna ouxjcb.
Key 8: Q tqsm zwwb jmmz ntwiba.
Key 9: P sprl yvva illy msvhaz.
Key 10: O roqk xuuz hkkx lrugzy.
Key 11: N qnpj wtty gjjw kqtfyx.
Key 12: M pmoi vssx fiiv jpsexw.
Key 13: L olnh urrw ehhu iordwv.
Key 14: K nkmg tqqv dggt hnqcvu.
Key 15: J mjlf sppu cffs gmpbut.
Key 16: I like root beer floats.
Key 17: H khjd qnns addq eknzsr.
Key 18: G jgic pmmr zccp djmyrq.
Key 19: F ifhb ollq ybbo cilxqp.
Key 20: E hega nkkp xaan bhkwpo.
Key 21: D gdfz mjjo wzzm agjvon.
Key 22: C fcey liin vyyl zfiunm.
Key 23: B ebdx khhm uxxk yehtml.
Key 24: A dacw jggl twwj xdgslk.
Key 25: Z czbv iffk svvi wcfrkj.
```

Figure 17: Attack Cesar´s cipher 3

It’s important to note that the Caesar cipher is a relatively weak encryption method and can be easily broken using a brute force attack. The small number of possible keys (26) allows an attacker to try all possible keys until the correct one is found. Overall, the output of the brute force attack function demonstrates the vulnerability of the Caesar cipher and the ease with which it can be decrypted using a brute force approach.

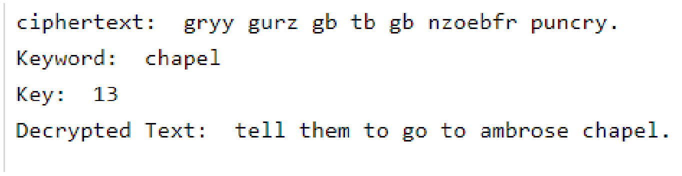
Fot the other part of the activity was generated a brute force attack with a keyword to decrypt the cipher text given in the activity. The code showed below its how it works for the process.



```
def caesar_brute_force(ciphertext, keyword=None):
    alphabet = 'abcdefghijklmnopqrstuvwxyz'
    keys = range(26) ; decryptions = []
    for key in keys:
        decryption = ''
        for char in ciphertext:
            if char.lower() in alphabet:
                is_uppercase = char.isupper()
                char_index = alphabet.index(char.lower())
                decrypted_index = (char_index - key) % 26
                decrypted_char = alphabet[decrypted_index]
                if is_uppercase:
                    decrypted_char = decrypted_char.upper()
                decryption += decrypted_char
            else:
                decryption += char
        if keyword is None or keyword.lower() in decryption.lower():
            decryptions.append((key, decryption))
            print(f"Keyword: {keyword}, Key: {key}, Decrypted Text:
                {decryption}")
    return decryptions
ciphertext = "baeeq klwosjl osk s esf ozg cfwo lgg emuz."
keyword = ""
decryptions = caesar_brute_force(ciphertext, keyword)
```

Figure 18: keyword code

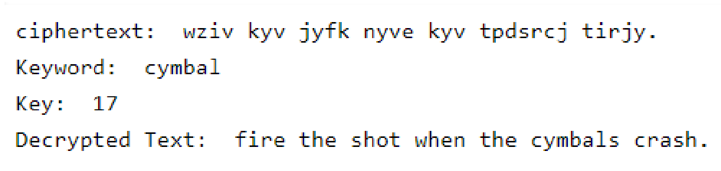
The first ciphertext given is 'gryy gurz gb tb gb nzoebfr puncry.' with the keyword 'chapel', applying in the code to this ciphertext is the next one:



```
ciphertext:  gryy gurz gb tb gb nzoebfr puncry.
Keyword:  chapel
Key:  13
Decrypted Text:  tell them to go to ambrose chapel.
```

Figure 19: decrypth ciphertext 1

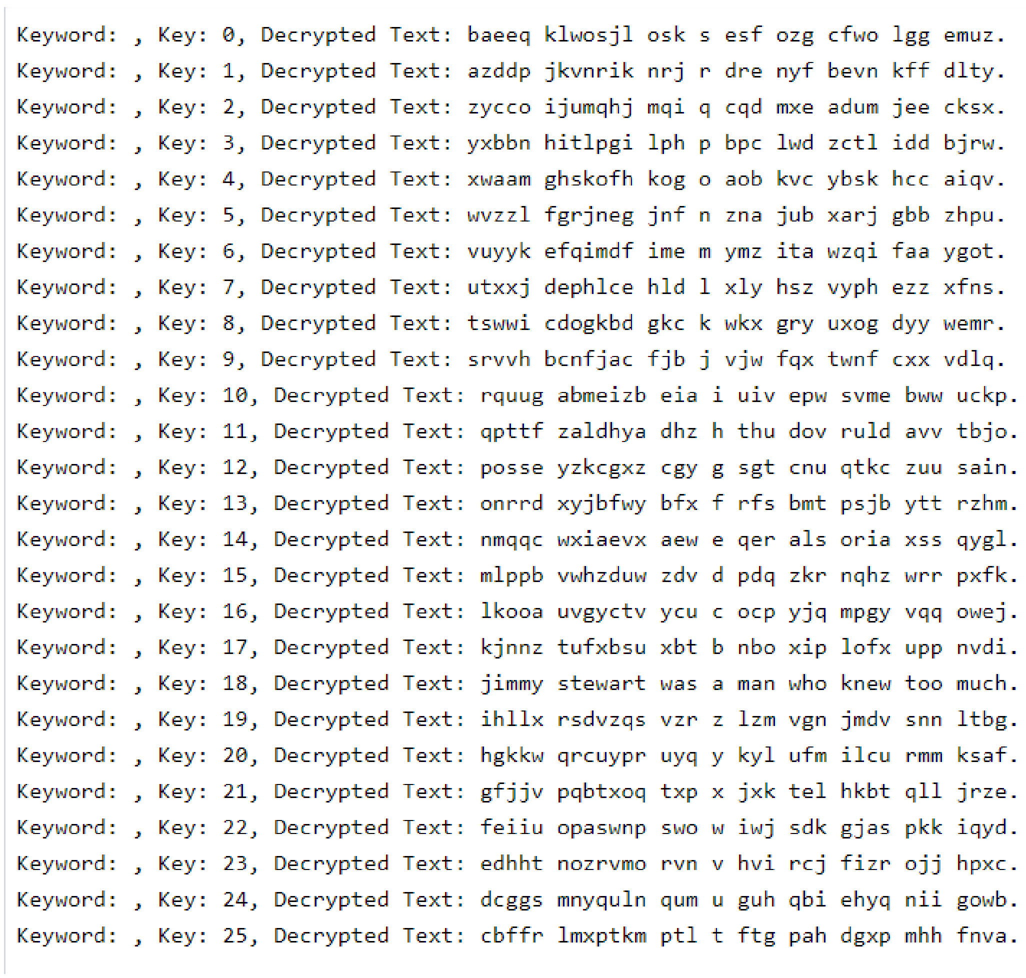
For the second ciphertext that is 'wziv kyv jyfk nyve kyv tpdsrcj tirjy.' with the keyword 'cymbal', the resultes are showed below:



```
ciphertext:  wziv kyv jyfk nyve kyv tpdsrcj tirjy.
Keyword:  cymbal
Key:  17
Decrypted Text:  fire the shot when the cymbals crash.
```

Figure 20: decrypth ciphertext 2

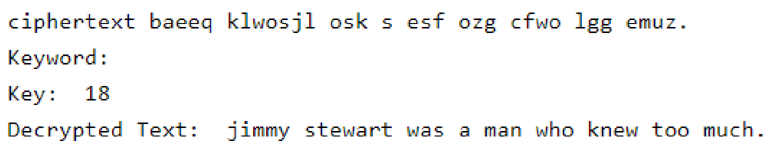
Finally for the third ciphertext that is 'baeeq klwosjl osk s esf ozg cfwo lgg emuz.' with no keyword, the sults are the next one:



```
Keyword: , Key: 0, Decrypted Text: baeeq klwosjl osk s esf ozg cfwo lgg emuz.
Keyword: , Key: 1, Decrypted Text: azddp jkvnrik nrj r dre nyf bevn kff dlty.
Keyword: , Key: 2, Decrypted Text: zycco ijumqhj mqi q cqd mxe adum jee cksx.
Keyword: , Key: 3, Decrypted Text: yxbbn hitlpgi lph p bpc lwd zctl idd bjrw.
Keyword: , Key: 4, Decrypted Text: xwaam ghskofh kog o aob kvc ybsk hcc aiqv.
Keyword: , Key: 5, Decrypted Text: wvzzl fgrjneg jnf n zna jub xarj gbb zhpu.
Keyword: , Key: 6, Decrypted Text: vuyyk efqimdf ime m ymz ita wzqi faa ygot.
Keyword: , Key: 7, Decrypted Text: utxxj dephlce hld l xly hsz vyph ezz xfns.
Keyword: , Key: 8, Decrypted Text: tswwi cdogkbd gkc k wkx gry uxog dyy wemr.
Keyword: , Key: 9, Decrypted Text: srvvh bcnfjac fjb j vjw fqx twnf cxx vdlq.
Keyword: , Key: 10, Decrypted Text: rquug abmeizb eia i uiv epw svme bww uckp.
Keyword: , Key: 11, Decrypted Text: qpttf zaldhya dhz h thu dov ruld avv tbjo.
Keyword: , Key: 12, Decrypted Text: posse yzkcgxz cgy g sgt cnu qtkc zuu sain.
Keyword: , Key: 13, Decrypted Text: onrrd xyjbfwy bfx f rfs bmt psjb ytt rzhm.
Keyword: , Key: 14, Decrypted Text: nmqqc wxiaevx aew e qer als oria xss qygl.
Keyword: , Key: 15, Decrypted Text: mlppb vwhzduw zdv d pdq zkr nqhz wrr pxfk.
Keyword: , Key: 16, Decrypted Text: lkooa uvgyctv ycu c ocp yjq mpgy vqq owej.
Keyword: , Key: 17, Decrypted Text: kjnnz tufxbsu xbt b nbo xip lofx upp nvdi.
Keyword: , Key: 18, Decrypted Text: jimmy stewart was a man who knew too much.
Keyword: , Key: 19, Decrypted Text: ihllx rsdvzqs vzr z lzm vgn jmdv snn ltbg.
Keyword: , Key: 20, Decrypted Text: hgkkw qrcuypr uyq y kyl ufm ilcu rmm ksaf.
Keyword: , Key: 21, Decrypted Text: gfjjv pqbtxoq txp x jxk tel hkbt qll jrze.
Keyword: , Key: 22, Decrypted Text: feiiu opaswnp swo w iwj sdk gjas pkk iqyd.
Keyword: , Key: 23, Decrypted Text: edhht nozrvmo rvn v hvi rcj fizr ojj hpxc.
Keyword: , Key: 24, Decrypted Text: dcggs mnyquln qum u guh qbi ehyq nii gowb.
Keyword: , Key: 25, Decrypted Text: cbffr lmxptkm ptl t ftg pah dgxp mhh fnva.
```

Figure 21: brute force attack 3

for this case the keys is the number 18 as it is show in the figure below:



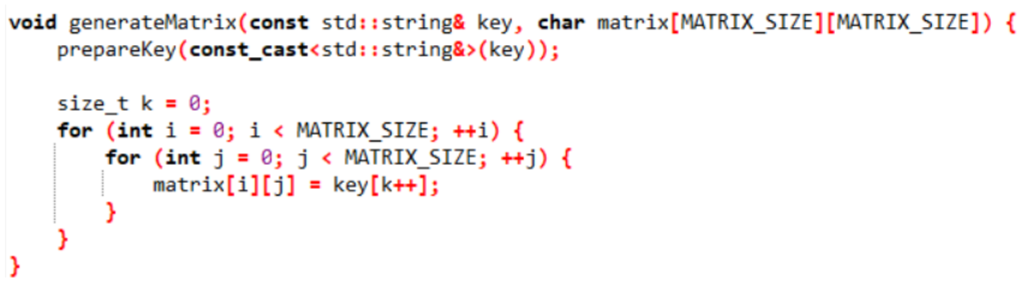
```
ciphertext baeeq klwosjl osk s esf ozg cfwo lgg emuz.
Keyword:
Key:  18
Decrypted Text:  jimmy stewart was a man who knew too much.
```

Figure 22: decrypth ciphertext 3

Exercise B

Continuing with part b, we are asked to make code to implement the encryption and decryption function for the Playfair cipher, which requires a key and a character string. It should only work with capital letters A to Z.

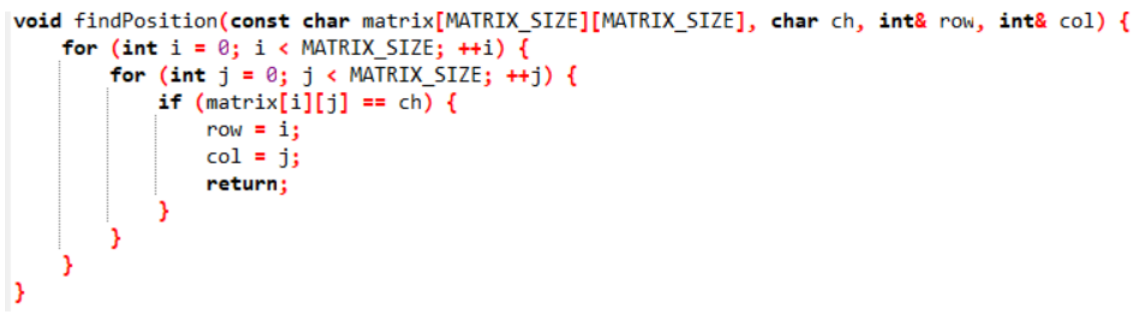
This function prepares the key for use in Playfair encryption. Converts the key to uppercase, removes 'J' and duplicate characters, and fills the key with remaining unique characters.



```
void generateMatrix(const std::string& key, char matrix[MATRIX_SIZE][MATRIX_SIZE]) {
    prepareKey(const_cast<std::string&>(key));

    size_t k = 0;
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        for (int j = 0; j < MATRIX_SIZE; ++j) {
            matrix[i][j] = key[k++];
        }
    }
}
```

Figure 23: Playfair code. Second part

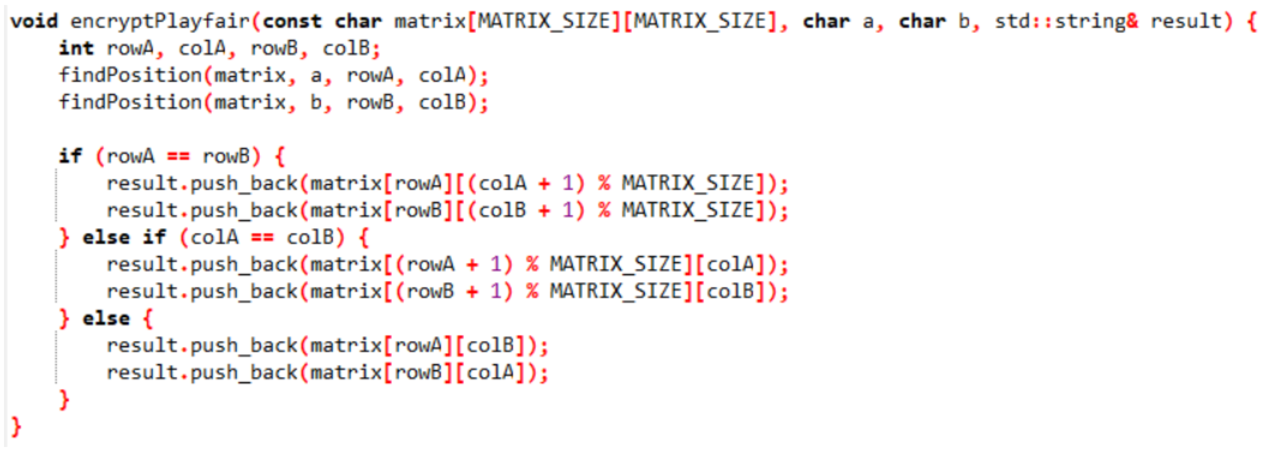
This function generates the Playfair matrix based on the prepared key. The matrix is a 5x5 grid of characters used for encryption and decryption.



```
void findPosition(const char matrix[MATRIX_SIZE][MATRIX_SIZE], char ch, int& row, int& col) {
    for (int i = 0; i < MATRIX_SIZE; ++i) {
        for (int j = 0; j < MATRIX_SIZE; ++j) {
            if (matrix[i][j] == ch) {
                row = i;
                col = j;
                return;
            }
        }
    }
}
```

Figure 24: Playfair code. Third part

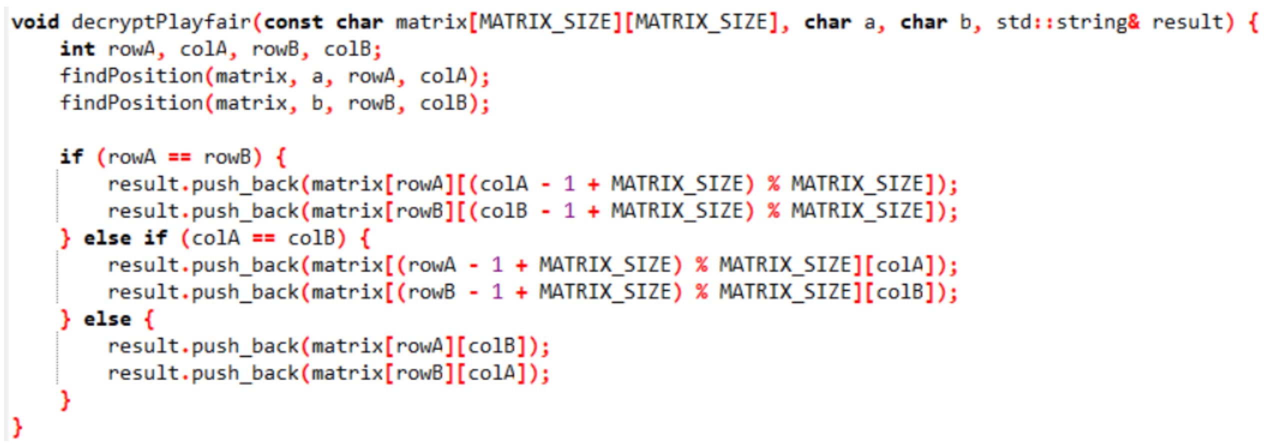
This code segment is a function called findPosition. This function is responsible for finding the position (row and column) of a given character within a 2D matrix.



```
void encryptPlayfair(const char matrix[MATRIX_SIZE][MATRIX_SIZE], char a, char b, std::string& result) {
    int rowA, colA, rowB, colB;
    findPosition(matrix, a, rowA, colA);
    findPosition(matrix, b, rowB, colB);

    if (rowA == rowB) {
        result.push_back(matrix[rowA][(colA + 1) % MATRIX_SIZE]);
        result.push_back(matrix[rowB][(colB + 1) % MATRIX_SIZE]);
    } else if (colA == colB) {
        result.push_back(matrix[(rowA + 1) % MATRIX_SIZE][colA]);
        result.push_back(matrix[(rowB + 1) % MATRIX_SIZE][colB]);
    } else {
        result.push_back(matrix[rowA][colB]);
        result.push_back(matrix[rowB][colA]);
    }
}
```

Figure 25: Playfair code. Fourth part

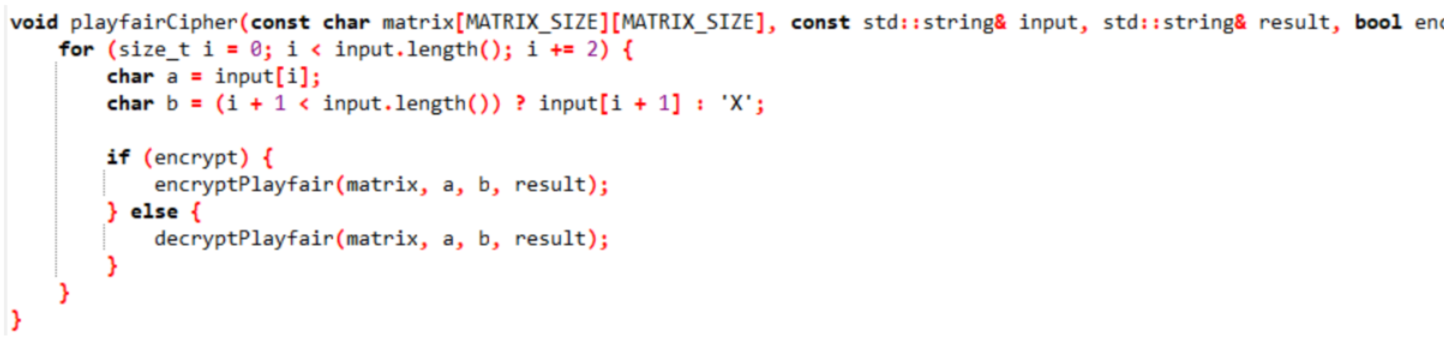
This function encrypts the characters using the Playfair cipher based on their positions in the array.



```
void decryptPlayfair(const char matrix[MATRIX_SIZE][MATRIX_SIZE], char a, char b, std::string& result) {
    int rowA, colA, rowB, colB;
    findPosition(matrix, a, rowA, colA);
    findPosition(matrix, b, rowB, colB);

    if (rowA == rowB) {
        result.push_back(matrix[rowA][(colA - 1 + MATRIX_SIZE) % MATRIX_SIZE]);
        result.push_back(matrix[rowB][(colB - 1 + MATRIX_SIZE) % MATRIX_SIZE]);
    } else if (colA == colB) {
        result.push_back(matrix[(rowA - 1 + MATRIX_SIZE) % MATRIX_SIZE][colA]);
        result.push_back(matrix[(rowB - 1 + MATRIX_SIZE) % MATRIX_SIZE][colB]);
    } else {
        result.push_back(matrix[rowA][colB]);
        result.push_back(matrix[rowB][colA]);
    }
}
```

Figure 26: Playfair code. Fifth part

This function decrypts the characters using the Playfair cipher based on their positions in the array.



```
void playfairCipher(const char matrix[MATRIX_SIZE][MATRIX_SIZE], const std::string& input, std::string& result, bool enc
    for (size_t i = 0; i < input.length(); i += 2) {
        char a = input[i];
        char b = (i + 1 < input.length()) ? input[i + 1] : 'X';

        if (encrypt) {
            encryptPlayfair(matrix, a, b, result);
        } else {
            decryptPlayfair(matrix, a, b, result);
        }
    }
}
```

Figure 27: Playfair code. Sixth part

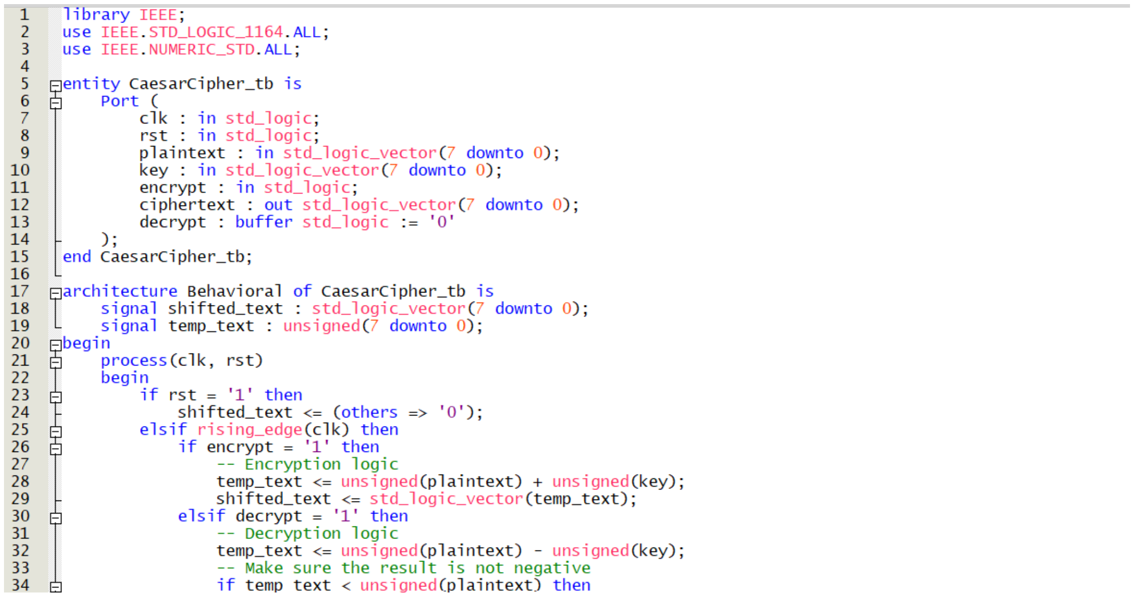
This function is responsible for processing the input text and performing the encryption or decryption using the Playfair cipher. The playfairCipher function takes the Playfair matrix (matrix), the input text (input), a reference to the result string (result), and a boolean flag (encrypt) indicating whether to perform encryption or decryption.

For the second activity we implemented the Ceasar´s Cipher Code in VHDL.

The VHDL code provides an implementation of a Caesar cipher in a ModelSim simulation architecture. The CaesarCiphertb module (testbench) accepts input signals such as plaintext, the encryption/decryption key, control signals for encryption and decryption, as well as a clock signal (clk) and a reset signal (rst).

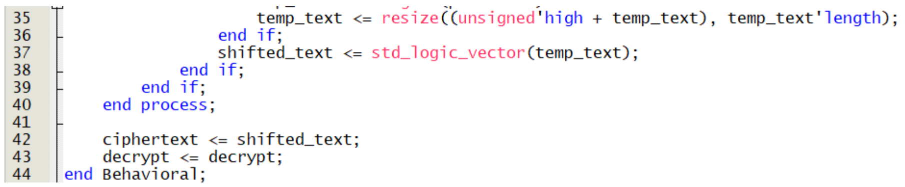
The Behavioral process handles the encryption and decryption logic. When the encryption signal is activated (encrypt = ´1´), it adds the key to the plaintext to encrypt it. When the decryption signal is activated (decrypt = ´1´), it subtracts the key from the plaintext to decrypt it. It ensures that the result is not negative. The results are stored in shiftedtext.

ciphertext is the output of the encrypted/decrypted text.



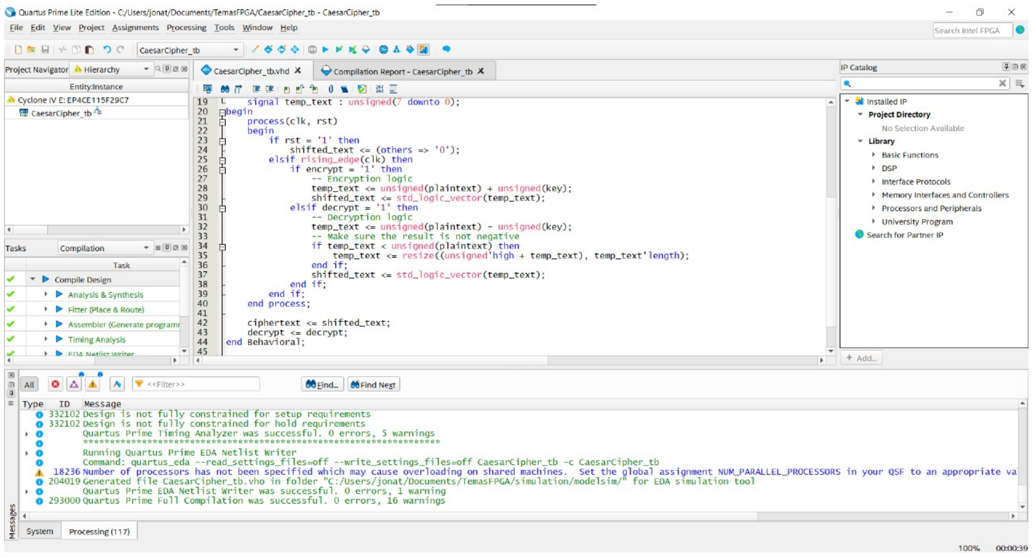
```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity CaesarCipher_tb is
    Port (
        clk : in std_logic;
        rst : in std_logic;
        plaintext : in std_logic_vector(7 downto 0);
        key : in std_logic_vector(7 downto 0);
        encrypt : in std_logic;
        ciphertext : out std_logic_vector(7 downto 0);
        decrypt : buffer std_logic := '0'
    );
end CaesarCipher_tb;

architecture Behavioral of CaesarCipher_tb is
    signal shifted_text : std_logic_vector(7 downto 0);
    signal temp_text : unsigned(7 downto 0);
begin
    process(clk, rst)
    begin
        if rst = '1' then
            shifted_text <= (others => '0');
        elsif rising_edge(clk) then
            if encrypt = '1' then
                -- Encryption logic
                temp_text <= unsigned(plaintext) + unsigned(key);
                shifted_text <= std_logic_vector(temp_text);
            elsif decrypt = '1' then
                -- Decryption logic
                temp_text <= unsigned(plaintext) - unsigned(key);
                -- Make sure the result is not negative
                if temp_text < unsigned(plaintext) then
```

Figure 28: Ceasar´s Cipher first part of the code



```vhdl
                    temp_text <= resize((unsigned'high + temp_text), temp_text'length);
                end if;
                shifted_text <= std_logic_vector(temp_text);
            end if;
        end if;
    end process;

    ciphertext <= shifted_text;
    decrypt <= decrypt;
end Behavioral;
```

Figure 29: Ceasar´s Cipher second part of the code

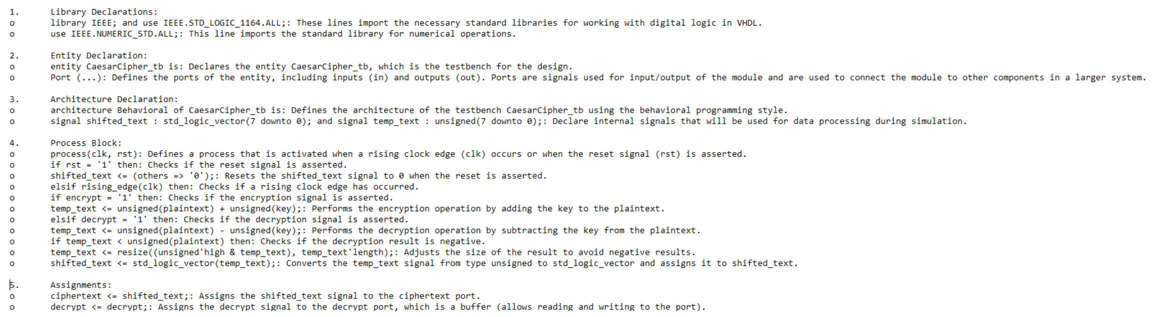


Quartus Prime Lite Edition - C:/Users/jonat/Documents/TemasFPGA/CaesarCipher_tb - CaesarCipher_tb

```vhdl
    signal temp_text : unsigned(7 downto 0);
begin
    process(clk, rst)
    begin
        if rst = '1' then
            shifted_text <= (others => '0');
        elsif rising_edge(clk) then
            if encrypt = '1' then
                -- Encryption logic
                temp_text <= unsigned(plaintext) + unsigned(key);
                shifted_text <= std_logic_vector(temp_text);
            elsif decrypt = '1' then
                -- Decryption logic
                temp_text <= unsigned(plaintext) - unsigned(key);
                -- Make sure the result is not negative
                if temp_text < unsigned(plaintext) then
                    temp_text <= resize((unsigned'high + temp_text), temp_text'length);
                end if;
                shifted_text <= std_logic_vector(temp_text);
            end if;
        end if;
    end process;

    ciphertext <= shifted_text;
    decrypt <= decrypt;
end Behavioral;
```

```
Type  ID     Message
      332102 Design is not fully constrained for setup requirements
      332102 Design is not fully constrained for hold requirements
             Quartus Prime Timing Analyzer was successful. 0 errors, 5 warnings
             ************************************************************
             Running Quartus Prime EDA Netlist Writer
             Command: quartus_eda --read_settings_files=off --write_settings_files=off CaesarCipher_tb -c CaesarCipher_tb
      18236  Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an appropriate va
      204019 Generated file CaesarCipher_tb.vho in folder "C:/Users/jonat/Documents/TemasFPGA/simulation/modelsim/" for EDA simulation tool
             Quartus Prime EDA Netlist Writer was successful. 0 errors, 1 warning
      293000 Quartus Prime Full Compilation was successful. 0 errors, 16 warnings
```

Figure 30: Ceasar´s Cipher Code Compiled



```
1.      Library Declarations:
o       library IEEE; and use IEEE.STD_LOGIC_1164.ALL;: These lines import the necessary standard libraries for working with digital logic in VHDL.
o       use IEEE.NUMERIC_STD.ALL;: This line imports the standard library for numerical operations.

2.      Entity Declaration:
o       entity CaesarCipher_tb is: Declares the entity CaesarCipher_tb, which is the testbench for the design.
o       Port (...): Defines the ports of the entity, including inputs (in) and outputs (out). Ports are signals used for input/output of the module and are used to connect the module to other components in a larger system.

3.      Architecture Declaration:
o       architecture Behavioral of CaesarCipher_tb is: Defines the architecture of the testbench CaesarCipher_tb using the behavioral programming style.
o       signal shifted_text : std_logic_vector(7 downto 0); and signal temp_text : unsigned(7 downto 0);: Declare internal signals that will be used for data processing during simulation.

4.      Process Block:
o       process(clk, rst): Defines a process that is activated when a rising clock edge (clk) occurs or when the reset signal (rst) is asserted.
o       if rst = '1' then: Checks if the reset signal is asserted.
o       shifted_text <= (others => '0');: Resets the shifted_text signal to 0 when the reset is asserted.
o       elsif rising_edge(clk) then: Checks if a rising clock edge has occurred.
o       if encrypt = '1' then: Checks if the encryption signal is asserted.
o       temp_text <= unsigned(plaintext) + unsigned(key);: Performs the encryption operation by adding the key to the plaintext.
o       elsif decrypt = '1' then: Checks if the decryption signal is asserted.
o       temp_text <= unsigned(plaintext) - unsigned(key);: Performs the decryption operation by subtracting the key from the plaintext.
o       if temp_text < unsigned(plaintext) then: Checks if the decryption result is negative.
o       temp_text <= resize((unsigned'high & temp_text), temp_text'length);: Adjusts the size of the result to avoid negative results.
o       shifted_text <= std_logic_vector(temp_text);: Converts the temp_text signal from type unsigned to std_logic_vector and assigns it to shifted_text.

5.      Assignments:
o       ciphertext <= shifted_text;: Assigns the shifted_text signal to the ciphertext port.
o       decrypt <= decrypt;: Assigns the decrypt signal to the decrypt port, which is a buffer (allows reading and writing to the port).
```

Figure 31: Description of the code

The simulation will demonstrate how plaintext is transformed into ciphertext and vice versa using the implemented Caesar cipher in the code. Changes in the encrypted/decrypted text can be observed based on the provided key.

The simulation results will help verify the correct implementation of the Caesar cipher encryption and decryption in the VHDL design, as well as understanding how the key affects the plaintext.

Output for encryption:
Encrypting "HELLO" with key = 3:
Plaintext: 01001000 01000101 01001100 01001100 01001111
Key: 00000000 00000000 00000000 00000000 00000011 (3 in binary)
Ciphertext: 01010000 01001110 01010111 01010111 01010000
(Resulting characters after shifting each character by 3 positions)

Output for decryption:
Decrypting "PNWWP" with key = 3:
Ciphertext: 01010000 01001110 01010111 01010111 01010000
Key: 00000000 00000000 00000000 00000000 00000011 (3 in binary)
Plaintext: 01001000 01000101 01001100 01001100 01001111
(Resulting characters after shifting each character back by 3 positions)

Conclusions

With this homework, the classical ciphers has provided a comprehensive understanding of encryption and decryption processes using classical ciphers. Through the implementation of encryption/decryption functions for the Caesar and Playfair ciphers, as well as a brute force attack function for the Caesar Cipher, we have gained practical experience in classical cryptography.

The successful implementation of the Caesar Cipher encryption and decryption functions allowed us to operate on the characters 'a' to 'z' (both upper and lower case) while leaving any other characters unchanged. Additionally, the implementation of the Playfair cipher encryption/decryption functions provided insight into the encryption process using a keyword and the creation of the Playfair matrix. Furthermore, the VHDL implementation of the Caesar cipher and the subsequent simulation of the encryption and decryption process in a hardware description language has enhanced our understanding of the practical application of classical ciphers in a hardware environment.

In conclusion, this lab has not only deepened our knowledge of encryption techniques but has also provided valuable hands-on experience in classical cryptography, thereby enhancing our understanding of encryption and decryption processes.

Bibliography

Ashutosh Kumar, A. (2016, junio 2). Caesar cipher in cryptography. GeeksforGeeks. https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/

Jim Kurose, Keith Ross, Computer Networking: A Top-Down Approach Featuring the Internet,, 7/e Pearson Prentice Hall, 2016.

Playfair. (s/f). Rumkin.com. Recuperado el 13 de febrero de 2024, de https://rumkin.com/tools/cipher/playfair/

What is a Brute Force Attack? (s/f). Fortinet. Recuperado el 13 de febrero de 2024, de https://www.fortinet.com/resources/cyberglossary/brute-force-attack

What is Cryptography? Definition, Importance, Types. (s/f). Fortinet. Recuperado el 13 de febrero de 2024, de https://www.fortinet.com/resources/cyberglossary/what-is-cryptography

William Stallings, Cryptography and Network Security: Principles and Practices, 7th Edition, Pearson Prentice Hall, 2016.