Advanced Security 1 Assignment 1

C21380786 Matthew Bradon

There are several ways to make your browser more secure. One way is to use Tor as your browser as your real IP address is hidden by sending your network traffic through many tor relays before reaching your target site. It even makes it difficult for your ISP to track you since they only see you accessing Tor. Easier ways of mitigating exposure of personally identifiable information is to use a password manager and use different passwords for each website you use. If one website gets a data breach many of your accounts could be compromised.

An ad blocker like uBlock Origin is a good way to get rid of ads and harmful scripts and trackers on websites. However due to Google's recent actions it's much more difficult to get an ad block on your chrome browser. This is a good reason to swap over to Tor or Firefox. Another useful extension is NoScript which allows you to control the execution of Javascript.

Ghostery, Disconnect and Privacy Browser are good tools which allow you to see who is tracking you and can block known trackers from tracking you. Other good ways to help stop tracking is to regularly clean your cookies in your browsers. Browsing in Incognito Mode is another useful way of preventing your browser from storing your history. However Google was hit with a lawsuit in regards to collecting data on those using Incognito Mode.

There are many reasons as people want access to your browsing activities and your information. Some organizations try to use it in forms of identity theft or trying to scam you maliciously using knowledge gathered from data breaches. Companies like Google and Meta want your browsing habits so they can sell it to advertisers that wealth of data also can be used in their own business ventures. Take for instance recent AI controversies using user data (i.e Github) to train their models like copilot.

While there are ways to mitigate tracking it is extremely difficult to prevent it. Even if you block cookies, more advanced methods like browser fingerprinting can still identify. Many websites rely on third-party services for analytics and embedded content. Many websites share user data to advertisers. Even VPNs which market themselves on privacy collect and store your browsing habits

```
CeaserCipherText = """RQH YDULDWLRQ WR WKH VWDQGDUG FDHVDU FLSKHU LV
ZKHQ WKH DOSKDEHW LV "NHBHG" EB XVLQJ D ZRUG. LQ WKH WUDGLWLRQDO
YDULHWB, RQH FRXOG ZULWH WKH DOSKDEHW RQ WZR VWULSV DQG MXVW
PDWFK XS WKH VWULSV DIWHU VOLGLQJ WKH ERWWRP VWULS WR WKH OHIW RU
ULJKW. WR HQFRGH, BRX ZRXOG ILQG D OHWWHU LQ WKH WRS URZ DQG
VXEVWLWXWH LW IRU WKH OHWWHU LQ WKH ERWWRP URZ. IRU D NHBHG
YHUVLRQ, RQH ZRXOG QRW XVH D VWDQGDUG DOSKDEHW, EXW ZRXOG ILUVW
ZULWH D ZRUG (RPLWWLQJ GXSOLFDWHG OHWWHUV) DQG WKHQ ZULWH WKH
UHPDLQLQJ OHWWHUV RI WKH DOSKDEHW. IRU WKH HADPSOH EHORZ, L XVHG D
NHB RI "UXPNLQ.FRP" DQG BRX ZLOO VHH WKDW WKH SHULRG LV UHPRYHG
EHFDXVH LW LV QRW D OHWWHU. BRX ZLOO DOVR QRWLFH WKH VHFRQG "P" LV
QRW LQFOXGHG EHFDXVH WKHUH ZDV DQ P DOUHDGB DQG BRX FDQ'W KDYH
GXSOLFDWHV."""

vigenereKey = "leg"
vigenerePlainText = "explanation"

# Part 2 - Implement a program to decrypt the Ceaser Cipher
def encryptCeaserCipher(plaintext, shift):
        cipherText = ""

        for char in plaintext:
        if char.isalpha():
        # Convert the character to ascii value
        asciiValue = ord(char)

        cipherText += chr((asciiValue + shift - 97) % 26 + 97)
        else:
        cipherText += char
        return cipherText

def decryptCeaserCipher(cipherText, shift):
        decryptedText = ""
        cipherText = cipherText.lower()

        for char in cipherText:
        if char.isalpha():
        # Convert the character to ascii value
        asciiValue = ord(char)

        decryptedText += chr((asciiValue - shift - 97) % 26 + 97)
        else:
```

```
        decryptedText += char

        return decryptedText


# Brute force check all 25 keys
for shift in range(1,26):
        print("Shift: ", shift)
        print(decryptCeaserCipher(CeaserCipherText, shift))
        print("\n")

ceaserPlainText = "hello"
ceaserShift = 3
print("-Ceaser Cipher-")
ceaserCipherText = encryptCeaserCipher(ceaserPlainText, ceaserShift)
print("Encrypted Text: ", ceaserCipherText)
print("Decrypted Text: ", decryptCeaserCipher(ceaserCipherText, ceaserShift))

# 4 - Implement Vigenere Cipher
def encryptVignere(plaintext, key):
        encryptedText = ""
        keyIndex = 0

        for char in plaintext:
        if char.isalpha():
        keyChar = key[keyIndex % len(key)]
        keyIndex += 1

        shift = ord(keyChar) - 97
        asciiValue = ord(char)

        if char.isupper():
                encryptedText += chr((asciiValue + shift - 65) % 26 + 65)
        elif char.islower():
                encryptedText += chr((asciiValue + shift - 97) % 26 + 97)
        else:
        encryptedText += char

        return encryptedText

def decryptVigenere(cipherText, key):
        decryptedText = ""
        keyIndex = 0
```

```
        for char in cipherText:
        if char.isalpha():
        keyChar = key[keyIndex % len(key)]
        keyIndex += 1

        shift = ord(keyChar) - 97
        asciiValue = ord(char)

        if char.isupper():
                decryptedText += chr((asciiValue - shift - 65) % 26 + 65)
        elif char.islower():
                decryptedText += chr((asciiValue - shift - 97) % 26 + 97)
        else:
        decryptedText += char

        return decryptedText

print("-Vigenere Cipher-")
vigenereCipherText = encryptVignere(vigenerePlainText, vigenereKey)
print("Encrypted Text: ", vigenereCipherText)
print("Decrypted Text: ", decryptVigenere(vigenereCipherText, vigenereKey))




# 5 Encrypt and Decrypt using 2 x 2 Hill Cipher
hillKey = [[1, 12], [5, 23]]
hillPlainText = "enigmamachine"

def char_to_num(c):
        return ord(c.upper()) - ord('A')

# Convert number to character (0=A, 1=B, ..., 25=Z)
def num_to_char(n):
        return chr((n % 26) + ord('A'))

def determinant_2x2(matrix):
        return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]

# Function to find the modular inverse of a number mod
def mod_inverse(a, m):
        a = a % m
        for x in range(1, m):
        if (a * x) % m == 0:
```

```
        return -1  # Inverse doesn't exist
        if (a * x) % m == 1:
        return x
        return -1

def inverse_matrix_2x2(matrix):
        det = determinant_2x2(matrix) % 26
        det_inv = mod_inverse(det, 26)

        if det_inv == -1:
        raise ValueError("Matrix is not invertible in mod 26.")

        # Inverse matrix formula for 2x2 matrix:
        # [a, b] ->   [ d, -b]
        # [c, d] ->   [-c,  a]
        inverse = [[matrix[1][1] * det_inv % 26, -matrix[0][1] * det_inv % 26],
        [-matrix[1][0] * det_inv % 26, matrix[0][0] * det_inv % 26]]

        # Mod 26 can't handle negatives directly, so we handle it manually
        for i in range(2):
        for j in range(2):
        inverse[i][j] = inverse[i][j] % 26

        return inverse


# Encrypt using Hill cipher with a 2x2 matrix
def encryptHillCipher(plaintext, key_matrix):
        # Make sure the plaintext length
        if len(plaintext) % 2 != 0:
        plaintext += 'X'  # Padding with 'X

        # Convert plaintext to number pairs
        plaintext_pairs = [char_to_num(c) for c in plaintext]

        # Split plaintext into 2-letter blocks
        ciphertext = ''
        for i in range(0, len(plaintext_pairs), 2):
        x1 = plaintext_pairs[i]
        x2 = plaintext_pairs[i + 1]

        # Apply matrix multiplication and mod 26
        c1 = (key_matrix[0][0] * x1 + key_matrix[0][1] * x2) % 26
        c2 = (key_matrix[1][0] * x1 + key_matrix[1][1] * x2) % 26
```

```
        # Convert back to characters
        ciphertext += num_to_char(c1)
        ciphertext += num_to_char(c2)

        return ciphertext

# Decrypt using Hill cipher with a 2x2 matrix
def decryptHillCipher(ciphertext, key_matrix):
        # Find the inverse of the key matrix mod 26
        key_matrix_inv = inverse_matrix_2x2(key_matrix)

        # Convert ciphertext to number pairs
        ciphertext_pairs = [char_to_num(c) for c in ciphertext]

        # Split ciphertext into 2-letter blocks
        plaintext = ''
        for i in range(0, len(ciphertext_pairs), 2):
        c1 = ciphertext_pairs[i]
        c2 = ciphertext_pairs[i + 1]

        # Apply matrix multiplication with the inverse key matrix and mod 26
        p1 = (key_matrix_inv[0][0] * c1 + key_matrix_inv[0][1] * c2) % 26
        p2 = (key_matrix_inv[1][0] * c1 + key_matrix_inv[1][1] * c2) % 26

        # Convert back to characters
        plaintext += num_to_char(p1)
        plaintext += num_to_char(p2)

        return plaintext



print("-Hill Cipher-")
hillCipherText = encryptHillCipher(hillPlainText, hillKey)
print("Encrypted Text: ", hillCipherText)
print("Decrypted Text: ", decryptHillCipher(hillCipherText, hillKey))
```