MINISTRY OF EDUCATION, CULTURE AND RESEARCH OF REPUBLIC OF MOLDOVA
TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND AUTOMATICS

Criptography and Security

Laboratory work 1: Caesar's Cipher

Elaborated:
st.gr. FAF-211 Nistor Stefan
Verified:
asist.univ. Cătălin Mîțu

Chișinău, 2023

Content

Objectives

1. To implement the Caesar algorithm for the English language alphabet in one of the programming languages. Use only the letter encoding as shown in Table 1 (do not use encodings specified in the programming language, such as ASCII or Unicode). The key values will be between 1 and 25 inclusive, and no other values are allowed. The values of the text characters are between 'A' and 'Z,' 'a' and 'z,' and no other values are allowed. If the user enters other values, they will be suggested the correct range. Before encryption, the text will be converted to uppercase, and spaces will be removed. The user will be able to choose the operation - encryption or decryption, input the key, message, or ciphertext, and obtain the respective ciphertext or decrypted message.
2. To implement the Caesar cipher algorithm with 2 keys, while maintaining the conditions expressed in Task 1. Additionally, key 2 must consist only of letters from the Latin alphabet and have a length of at least 7.

Caesar's Cipher

The Caesar cipher. In this cipher, each letter of the plaintext is replaced with a new letter obtained by a alphabetical shift. The secret key k, which is the same for both encryption and decryption, represents the number indicating the alphabetical shift, i.e., k 1, 2, 3,..., n–1, where n is the length of the alphabet. The encryption and decryption of the message using the Caesar cipher can be defined by the formulas:

$c = ek(x) = x + k \pmod{n}$,

$m = dk(y) = y - k \pmod{n}$,

where x and y represent the numerical representation of the respective characters in the plaintext m and the ciphertext c. The Modulo function (a mod b) returns the remainder of dividing the integer a by the integer b. This method of encryption is named after Julius Caesar, who used it to communicate with his generals, using the key k = 3 (Table 1).

For example, for k = 3, we have:

$ek(S) = 18 + 3 \pmod{26} = 21 = V$,

$dk(V) = 21 - 3 \pmod{26} = 18 = S$.

In this case, for m = 'caesar cipher,' we obtain c = 'fdhvdq flkha.

The Caesar cipher is very easy to break, making it a very weak cipher. As a result, a cryptanalyst can obtain the plaintext by trying all 25 possible keys. It is not known how useful the Caesar cipher was during the time it was used by the person from whom it gets its name, but it is likely that it was reasonably secure, as long as only a few of Caesar's enemies were capable of writing and reading, especially with knowledge of cryptanalysis concepts.

Double Caesar's Cipher

Considering the low cryptographic resistance of the Caesar cipher, mainly due to the small key space consisting of only 25 different keys for the Latin alphabet, it can be broken by systematically trying all the keys. If a message has been encrypted using the Caesar cipher, one of the keys will yield readable text in the language in which the message was written.

For example, if

m = BRUTE FORCE ATTACK

is a message written in English and was encrypted with the key k = 17, we obtain the ciphertext

c = SILKVWFITVRKKRTB

As you can see, only the text obtained with the key k=17 is meaningful in the English language, so the corresponding message for the ciphertext is

m = BRUTEFORCEATTACK.

To enhance the cryptographic resistance of the Caesar cipher, a permutation of the alphabet can be applied using a keyword (not to be confused with the basic cipher key). This keyword can be any sequence of letters from the alphabet, either a meaningful word or a nonsense one.

Let's say the second key is k2=cryptography. We apply this key to the alphabet as follows:

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

C R Y P T O G A H B D E F I J K M L N Q S U V W X Z

This new order is obtained by placing the letters of k2 at the beginning, followed by the remaining letters of the alphabet in their natural order. It's important to note that the letters are not repeated; if a letter occurs multiple times, it is placed only once.

Next, we apply the Caesar cipher, taking into account the new alphabet order:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

C R Y P T O G A H B D E F I J K M L N Q S U V W X Z

3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 0 1 2

P T O G A H B D E F I J K M L N Q S U V W X Z C R Y

Table 2. Caesar Cipher with key k1=3 and k2=cryptography

Since there are 26! = 403,291,461,126,605,635,584,000,000 possible permutations, the number of keys for this version of the algorithm will be: 26! * 25 = 10,082,286,528,165,140,889,600,000,000,

making it challenging to break via exhaustive methods, but it does not protect against frequency analysis attacks.

Implementation

Task 1.1

Implement the Caesar algorithm for the English language alphabet in one of the programming languages. Use only the letter encoding as shown in Table 1 (do not use the encodings specified in the programming language, such as ASCII or Unicode). The key values will be between 1 and 25 inclusive, and no other values are allowed. The values of the text characters are between 'A' and 'Z,' 'a' and 'z,' and no other values are allowed. If the user enters other values, they will be suggested the correct range. Before encryption, the text will be converted to uppercase, and spaces will be removed. The user will be able to choose the operation - encryption or decryption, input the key, message, or ciphertext, and obtain the respective ciphertext or decrypted message.

Task 1.2

To implement the Caesar cipher with 2 keys, while preserving the conditions expressed in Task 1.1. Additionally, key 2 must consist only of letters from the Latin alphabet and have a length of at least 7.

```
ALPHABET = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

def sanitize_input(text):
    return text.replace(" ", "").upper()

def validate_text(text):
    for char in text:
        if char not in ALPHABET:
            raise ValueError(f"Invalid character '{char}' in
            the text. Text must contain only Latin alphabet letters.")
    return text

def validate_key2(key2):
    if len(key2) < 7:
        raise ValueError("Key 2 must have a length of at least 7 characters.")
    for char in key2:
        if char not in ALPHABET:
            raise ValueError("Key 2 must contain only Latin alphabet letters.")
```

```
    return key2

def generate_shifted_alphabet(key2):
    unique_chars = ""
    for char in key2:
        if char not in unique_chars:
            unique_chars += char

    shifted = unique_chars
    for char in ALPHABET:
        if char not in unique_chars:
            shifted += char

    return shifted

def cezar_encrypt(text, key):
    source_alphabet = ALPHABET
    result = ""
    for char in text:
        idx = source_alphabet.index(char)
        result += source_alphabet[(idx + key) % 26]
    return result

def cezar_decrypt(text, key):
    source_alphabet = ALPHABET
    result = ""
    for char in text:
        idx = source_alphabet.index(char)
        result += source_alphabet[(idx - key) % 26]
    return result

def cezar_encrypt_2keys(text, key1, key2):
    encrypted_text = cezar_encrypt(text, key1)
    result = ""
```

```
    source_alphabet = ALPHABET
    key2_alphabet = generate_shifted_alphabet(key2)
    for char in encrypted_text:
        idx = source_alphabet.index(char)
        result += key2_alphabet[idx]
    return result

def cezar_decrypt_2keys(text, key1, key2):
    source_alphabet = ALPHABET
    key2_alphabet = generate_shifted_alphabet(key2)
    intermediate_text = ""
    for char in text:
        idx = key2_alphabet.index(char)
        intermediate_text += source_alphabet[idx]
    return cezar_decrypt(intermediate_text, key1)

def main():
    global ENCRYPTED_MESSAGE

    while True:
        print("Choose the operation:")
        print("1 - Encrypt")
        print("2 - Decrypt")
        print("3 - Encrypt with 2 keys")
        print("4 - Decrypt with 2 keys")
        print("0 - Exit")

        choice = input("Select option: ")

        if choice == '0':
            break

        if choice in ['1', '3']:
            message = input("Enter the message for encryption: ")
```

```
        sanitized_message = sanitize_input(message)

        try:
            validate_text(sanitized_message)
        except ValueError as e:
            print(e)
            continue

        key1 = int(input("Enter key 1 (between 1 and 25 inclusive): "))
        if not 1 <= key1 <= 25:
            print("Incorrect key 1. Enter a value between 1 and 25.")
            continue

        if choice == '1':
            ENCRYPTED_MESSAGE = cezar_encrypt(sanitized_message, key1)
            print("The encrypted message is:", ENCRYPTED_MESSAGE)
        else:
            key2 = input("Enter key 2 (minimum 7 characters): ")
            try:
                key2 = validate_key2(sanitize_input(key2))
            except ValueError as e:
                print(e)
                continue

            ENCRYPTED_MESSAGE = cezar_encrypt_2keys(sanitized_message, key1, key2)
            print("Shifted alphabet based on key2:", generate_shifted_alphabet(key2))
            print("The encrypted message with 2 keys is:", ENCRYPTED_MESSAGE)

    elif choice in ['2', '4']:
        encrypted_message = input("Enter the encrypted message for decryption: ")
        sanitized_message = sanitize_input(encrypted_message)

        try:
            validate_text(sanitized_message)
```

```
            except ValueError as e:
                print(e)
                continue

            key1 = int(input("Enter key 1 (between 1 and 25 inclusive): "))
            if not 1 <= key1 <= 25:
                print("Incorrect key 1. Enter a value between 1 and 25.")
                continue

            if choice == '2':
                decrypted_message = cezar_decrypt(sanitized_message, key1)
                print("The decrypted message is:", decrypted_message)
            else:
                key2 = input("Enter key 2: ")
                try:
                    key2 = validate_key2(sanitize_input(key2))
                except ValueError as e:
                    print(e)
                    continue

                decrypted_message = cezar_decrypt_2keys(sanitized_message, key1, key2)
                print("The decrypted message with 2 keys is:", decrypted_message)

        else:
            print("Invalid option. Try again.")

if __name__ == "__main__":
    ENCRYPTED_MESSAGE = ""
    main()
```

Screenshot:

Task 1.1.1

```
Select option: 1
Enter the message for encryption: big black car
Enter key 1 (between 1 and 25 inclusive): 6
The encrypted message is: HOMHRGIQIGX
```

Task 1.1.2

```
Select option: 2
Enter the encrypted message for decryption: HOMHRGIQIGX
Enter key 1 (between 1 and 25 inclusive): 6
The decrypted message is: BIGBLACKCAR
```

Task 1.2.1

```
Select option: 3
Enter the message for encryption: big black car
Enter key 1 (between 1 and 25 inclusive): 6
Enter key 2 (minimum 7 characters): naughty
Shifted alphabet based on key2: NAUGHTYBCDEFIJKLMOPQRSVWXZ
The encrypted message with 2 keys is: BKIBOYCMCYW
```

Task 1.2.2


Select option: 4
Enter the encrypted message for decryption: BKIBOYCMCYW
Enter key 1 (between 1 and 25 inclusive): 6
Enter key 2: naughty
The decrypted message with 2 keys is: BIGBLACKCAR

Conclusion

In summary, during my lab work, I delved into both Caesar's and Double Caesar's encryption and decryption methods. While these methods might seem straightforward, they offered me a profound understanding of the basic principles of cryptography and the crucial role of managing keys.

In the initial phase, I applied Caesar's method, adhering to specific guidelines, such as using a designated letter encoding and ensuring the keys were between 1 to 25. I made sure to accept only valid input characters, converting the text to uppercase and eliminating any spaces. This approach made the process of encryption and decryption quite straightforward for me.

When tackling the Double Caesar's method, I deepened my understanding by incorporating a dual-key system. The second key utilized letters from the Latin alphabet and required a minimum of 7 characters, significantly enhancing the security of the encryption. This added step provided a stronger defense against potential brute force attempts.

This lab experience gave me hands-on exposure to cryptographic techniques and highlighted the limitations of traditional methods like the Caesar cipher. While they are immensely educational and historically relevant, their limited security capabilities render them unsuitable for modern secure communications. However, they served as an invaluable foundation for my journey into understanding the complexities of encryption.

In the real world, encryption depends heavily on sophisticated algorithms and robust key management to safeguard critical information. This lab proved to be a crucial stepping stone in my journey to grasp the fundamentals of cryptography, preparing me for further exploration into more advanced encryption methodologies.