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 Grama Alexandru
Verified:
asist.univ. Mitu Catalin

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 (mod n), m = dk(y) = y – k (mod 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 (mod 26) = 21 = V, dk(V) = 21 – 3 (mod 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"
ALPHABET_LOW = "abcdefghijklmnopqrstuvwxyz"

def sanitize_input(text):
    return text.replace(" ", "").upper() if text.isupper() else text.replace(" ", "").lowe

def validate_key2(key2):
    """Validare pentru cheia 2."""
    if len(key2) < 7:
        raise ValueError("Cheia 2 trebuie să aibă o lungime de cel puțin 7 caractere.")
    for char in key2:
        if not ('a' <= char <= 'z') and not ('A' <= char <= 'Z'):
            raise ValueError("Cheia 2 trebuie să conțină doar litere ale alfabetului latin
    return key2


def cezar_encrypt(text, key):
    if text.isupper():
        source_alphabet = ALPHABET
```

```
    else:
        source_alphabet = ALPHABET_LOW
    result = ""
    for char in text:
        idx = source_alphabet.index(char)
        result += source_alphabet[(idx + key) % 26]
    return result


def cezar_decrypt(text, key):
    if text.isupper():
        source_alphabet = ALPHABET
    else:
        source_alphabet = ALPHABET_LOW
    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 = ""

    if text.isupper():
        source_alphabet = ALPHABET
        key2_alphabet = ALPHABET if key2.isupper() else ALPHABET_LOW
    else:
        source_alphabet = ALPHABET_LOW
        key2_alphabet = ALPHABET_LOW if key2.islower() else ALPHABET

    for i, char in enumerate(encrypted_text):
        shift = key2_alphabet.index(key2[i % len(key2)])
```

```
        idx = source_alphabet.index(char)
        result += source_alphabet[(idx + shift) % 26]

    return result

def cezar_decrypt_2keys(text, key1, key2):
    decrypted_text = ""

    if text.isupper():
        source_alphabet = ALPHABET
        key2_alphabet = ALPHABET if key2.isupper() else ALPHABET_LOW
    else:
        source_alphabet = ALPHABET_LOW
        key2_alphabet = ALPHABET_LOW if key2.islower() else ALPHABET

    for i, char in enumerate(text):
        shift = key2_alphabet.index(key2[i % len(key2)])
        idx = source_alphabet.index(char)
        decrypted_text += source_alphabet[(idx - shift) % 26]

    return cezar_decrypt(decrypted_text, key1)



def main():
    global ENCRYPTED_MESSAGE

    while True:
        print("Alegeti operația:")
        print("1 - Criptare")
        print("2 - Decriptare")
        print("3 - Criptare cu 2 chei")
        print("4 - Decriptare cu 2 chei")
        print("0 - Ieșire")
```

```
choice = input(": ")

if choice == '0':
    break

if choice in ['1', '3']:
    message = input("Introduceți mesajul pentru criptare: ")
    sanitized_message = sanitize_input(message)
    key1 = int(input("Introduceți cheia 1 (între 1 și 25 inclusiv): "))
    if not 1 <= key1 <= 25:
        print("Cheie 1 incorectă. Introduceți o valoare între 1 și 25.")
        continue

    if choice == '1':
        ENCRYPTED_MESSAGE = cezar_encrypt(sanitized_message, key1)
        print("Criptograma este:", ENCRYPTED_MESSAGE)
    else:
        key2 = input("Introduceți cheia 2 (minim 7 caractere): ")
        try:
            key2 = validate_key2(sanitize_input(key2))
        except ValueError as e:
            print(e)
            continue
        ENCRYPTED_MESSAGE = cezar_encrypt_2keys(sanitized_message, key1, key2)
        print("Criptograma cu 2 chei este:", ENCRYPTED_MESSAGE)

elif choice in ['2', '4']:
    encrypted_message = input("Introduceți criptograma pentru decriptare: ")
    sanitized_message = sanitize_input(encrypted_message)
    key1 = int(input("Introduceți cheia 1 (între 1 și 25 inclusiv): "))
    if not 1 <= key1 <= 25:
        print("Cheie 1 incorectă. Introduceți o valoare între 1 și 25.")
        continue
```

```
            if choice == '2':
                decrypted_message = cezar_decrypt(sanitized_message, key1)
                print("Mesajul decriptat este:", decrypted_message)
            else:
                key2 = input("Introduceți cheia 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("Mesajul decriptat cu 2 chei este:", decrypted_message)

        else:
            print("Opțiune invalidă. Încercați din nou.")


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

Screenshot:

Task 1.1

```
C:\Users\snist\AppData\Local\Programs\Python\Python310\python.exe "C:/Users/snist/PycharmProjects/lab criptografie/main.py"
Alegeti operația:
1 - Criptare
2 - Decriptare
3 - Criptare cu 2 chei
4 - Decriptare cu 2 chei
0 - Ieșire
: 1
Introduceți mesajul pentru criptare: DOG
Introduceți cheia 1 (între 1 și 25 inclusiv): 6
Criptograma este: JUM
Alegeti operația:
1 - Criptare
2 - Decriptare
3 - Criptare cu 2 chei
4 - Decriptare cu 2 chei
0 - Ieșire
```

Task 1.2

```
: 3
Introduceți mesajul pentru criptare: DOG
Introduceți cheia 1 (între 1 și 25 inclusiv): 6
Introduceți cheia 2 (minim 7 caractere): DOBERMAN
Criptograma cu 2 chei este: MIN
Alegeti operația:
1 - Criptare
2 - Decriptare
3 - Criptare cu 2 chei
4 - Decriptare cu 2 chei
0 - Ieșire
:
```

Conclusion

In conclusion, our laboratory experiment involved the implementation and exploration of Caesar's algorithm and the Double Caesar's algorithm for encryption and decryption. These classical encryption techniques, while simple in concept, provided valuable insights into the fundamentals of cryptography and the importance of key management.

In the first part of our experiment, we successfully implemented Caesar's algorithm, adhering to the specified constraints such as using only the letter encoding shown in the provided table and ensuring that the key values fell within the range of 1 to 25 inclusive. We also handled input validation to ensure that the text contained only valid characters ('A' to 'Z' and 'a' to 'z') and transformed the text to uppercase while removing spaces. This implementation allowed us to both encrypt and decrypt messages with ease.

Moving on to the Double Caesar's algorithm, we extended our understanding of classical cryptography. We introduced the concept of using two keys, with key 2 consisting of letters from the Latin alphabet and having a minimum length of 7 characters. This added layer of complexity enhanced the security of the encryption and decryption process. The permutation of the alphabet based on the second key brought an additional level of protection against brute force attacks.

Through this lab, we not only gained practical experience in implementing cryptographic algorithms but also recognized the limitations of classical ciphers like the Caesar cipher. While these ciphers are instructive and historically significant, their low cryptographic resistance makes them unsuitable for modern secure communication. Nevertheless, they serve as an excellent starting point for learning about encryption techniques and their vulnerabilities.

In the real world, modern cryptographic systems rely on much more sophisticated algorithms and robust key management to ensure the confidentiality and integrity of sensitive information. This laboratory experiment was an essential step in our journey to understand the foundations of cryptography and laid the groundwork for further exploration into advanced encryption methods.

Github: https://github.com/AlexGrama22/CS_Labs