# **Decryption Report**

## **Objective**

The purpose of this report is to document the process undertaken to decode an intercepted message encrypted using a combination of custom encoding techniques. The provided encoding mechanism includes three reversible steps: character substitution, Base64 encoding, and a Caesar cipher with a shift of 4. However, the exact order of these encoding steps was randomized, making decryption non-trivial.

## **Code Implementation**

### **Decryption Functions**

1. **Reverse Step 1 – Character Substitution:** This function undoes the character substitution used during encoding. The substitution maps letters to their respective reverse positions in the alphabet.

python

Copier le code

import string

def reverse\_step1(s):

\_reverse\_step1 = str.maketrans(

"mlkjihgfedcbaMLKJIHGFEDCBAzyxwvutsrqponZYXWVUTSRQPON",

"zyxwvutsrqponZYXWVUTSRQPONmlkjihgfedcbaMLKJIHGFEDCBA"

)

return str.translate(s, \_reverse\_step1)

1. **Reverse Step 2 – Base64 Decoding:** This function decodes a Base64-encoded string back into its plaintext representation.

python

Copier le code

from base64 import b64decode

def reverse\_step2(s):

return b64decode(s).decode('utf-8')

1. **Reverse Step 3 – Caesar Cipher Decryption:** This function undoes the Caesar cipher applied during encoding by shifting the letters back by 4 positions.

python

Copier le code

def reverse\_step3(ciphertext, shift=4):

loweralpha = string.ascii\_lowercase

shifted\_string = loweralpha[-shift:] + loweralpha[:-shift]

converted = str.maketrans(loweralpha, shifted\_string)

return ciphertext.translate(converted)

### **Debugging and Testing Functions**

1. **Testing Individual Steps:** This function tests each decryption step individually to identify potential errors or patterns in the encoded message.

python

Copier le code

def test\_individual\_steps(encrypted\_message):

print("\n--- Testing reverse\_step1 (character substitution) ---")

try:

result\_step1 = reverse\_step1(encrypted\_message)

print(f"Result of reverse\_step1 (first 100 chars): {result\_step1[:100]}\n")

except Exception as e:

print(f"Error in reverse\_step1: {e}\n")

print("\n--- Testing reverse\_step3 (Caesar cipher decryption) ---")

try:

result\_step3 = reverse\_step3(encrypted\_message)

print(f"Result of reverse\_step3 (first 100 chars): {result\_step3[:100]}\n")

except Exception as e:

print(f"Error in reverse\_step3: {e}\n")

print("\n--- Testing reverse\_step2 (Base64 decoding) ---")

try:

result\_step2 = reverse\_step2(encrypted\_message)

print(f"Result of reverse\_step2 (first 100 chars): {result\_step2[:100]}\n")

except Exception as e:

print(f"Error in reverse\_step2: {e}\n")

1. **Testing All Permutations:** Since the encoding order is randomized, all six permutations of the decryption steps are tested to identify the correct decoding sequence.

python

Copier le code

from itertools import permutations

def decrypt\_message\_with\_debugging(encrypted\_message):

steps = [reverse\_step1, reverse\_step2, reverse\_step3]

for perm in permutations(steps):

print(f"\n--- Testing permutation: {', '.join([step.\_\_name\_\_ for step in perm[::-1]])} ---")

try:

result = encrypted\_message

for step in perm[::-1]: # Apply steps in reverse order

print(f"Applying {step.\_\_name\_\_} to: {result[:50]}...")

result = step(result)

print(f"\n\*\*\* Success! Decrypted result: {result[:500]} \*\*\*\n")

except Exception as e:

print(f"Error during {step.\_\_name\_\_}: {e}")

print(f"Intermediate result before failure: {result[:100]}\n")

## **Testing and Results**

1. **Input Message:** The intercepted message is stored in a file named intercepted.txt and is read as input by the program. A sample of the first 500 characters was used for debugging:

Copier le code

313312Mw16RXtNmlF2TVRmU1pIQxxmnTxtV1ZWV2JFOXBS...

1. **Step-by-Step Testing Results:**
   * **reverse\_step1:** Partial success. Some character substitutions were reversed successfully, but the output was not meaningful.
   * **reverse\_step2:** Failed due to an invalid Base64 format. This indicates the message was not encoded in Base64 at this stage.
   * **reverse\_step3:** Applied successfully but did not produce a meaningful plaintext result.
2. **Permutation Testing Results:**
   * All six permutations of the decryption steps were tested. While some combinations produced partial results, none successfully decoded the full message.
   * Errors included:
     + **Base64 decoding issues** when applied to non-Base64-encoded segments.
     + **Unintelligible intermediate outputs**, indicating that the decryption steps were applied in an incorrect order.

## **Conclusion**

Despite exhaustive testing and debugging, the correct permutation of decryption steps could not be identified. This suggests one of the following:

1. The intercepted message was encoded with additional steps or transformations not described in the provided encoding script.
2. The message format was altered or corrupted during interception.

## **Future Improvements**

1. **Automated Validation:**
   * Use dictionary-based validation to check for recognizable plaintext patterns during decryption.
   * Automate the identification of correct permutations based on pattern matching.
2. **Error Handling:**
   * Implement better error handling to skip invalid steps and log meaningful outputs for debugging.
3. **Additional Context:**
   * Investigate whether other encoding steps or transformations were applied during the original encryption process.