Cybersécurité
Introduction Crypto
TP1

Vasco VALADARES SEMANA et Guillaume BLAS

Donner votre avis en répondant aux questions suivantes :

Qu'est-ce qu'une donnée personnelle ? (donner des exemples)

Une donnée personnelle est toute information se rapportant à une personne physique identifiée ou identifiable. Par exemple, le nom, l'adresse e-mail, le numéro de téléphone, l'adresse IP, les données de localisation, les informations de santé, etc.

Que sont les données sensibles ? (donner des exemples)

Les données sensibles sont des données personnelles permettant d'identifier une personne physique de manière unique ou de révéler des informations intimes sur elle. Par exemple, les données de santé, les opinions politiques, les convictions religieuses, l'origine raciale ou ethnique, les données biométriques, etc.

Qu'est-ce que les métadonnées ? (donner des exemples)

Les métadonnées sont des données qui décrivent d'autres données. Elles fournissent des informations sur la structure, le contenu, la provenance ou l'utilisation d'une donnée. Par exemple, les métadonnées d'une photo peuvent inclure la date et l'heure où celle-ci a été prise, les coordonnées GPS, le modèle de l'appareil photo, etc.

Qu'est-ce que le cyberespace ?

Le cyberespace est un espace virtuel créé par les réseaux informatiques et les technologies de l'information. Il englobe l'ensemble des interactions, des communications et des activités qui se déroulent en ligne, que ce soit sur Internet, les réseaux sociaux, les applications mobiles, etc.

Pourquoi les données personnelles passionnent tant ?

Les données personnelles passionnent tant car elles sont au coeur de nombreuses activités en ligne. Ce sont des ressources préciseuses pour comprendre les comportements, les préférences et les besoins des utilisateurs, ce qui en fait un enjeu majeur pour la vie privée et la sécurité. Elles permettent aux entreprises d'appliquer des stratégies de marketing ciblé, aux gouvernements de surveiller les citoyens, et aux individus de partager des informations sur eux-mêmes.

Internet, est-il un cimetière de données personnelles ?

Il n'y a pas de bonne ou mauvaise réponse, cependant, nous pouvons considérer qu'Internet est un cimetière de données personnelles dans la mesure où de nombreuses informations personnelles y sont stockées, souvent sans le consentement explicite des utilisateurs, et ce, même après leur mort.

Si je n'utilise pas ou très peu Internet, suis-je concerné par la sécurité des données personnelles ? Pourquoi ?

Oui, même si on n'utilise pas ou très peu Internet, on est concerné par la sécurité des données personnelles, car de nombreuses données personnelles peuvent être collectées à notre sujet par d'autres moyens, tels que les transactions bancaires, les achats en magasin, les interactions avec les services publics, etc.

Le « sentiment de sécurité dans le cloud ».

Je pense que le sentiment de sécurité dans le cloud peut être trompeur, car nous ne savons pas toujours où nos données sont stockées, qui y a accès et comment elles sont protégées, on peut croire ou être persuadés que nos données sont en sécurité, alors qu'en réalité, elles peuvent être vulnérables à des attaques ou à des fuites.

« Pas grave si on prend mes données, je n'ai rien à cacher ».

Ce n'est pas une question d'avoir des choses à cacher, mais plutôt une question de respect de la vie privée et de contrôle sur ses propres informations. Même si on n'a rien à cacher, on peut ne pas vouloir que nos données soient utilisées à des fins commerciales, surveillées par des gouvernements ou exposées à des risques de sécurité. Ce n'est pas parce qu'on n'a rien à cacher que l'on a pas quelque chose à perdre.

Quels sont vos interrogations sur le monde digital ?

Comment puis-je protéger ma vie privée en ligne si tout ce que je fais est suivi sans que je n'en sois au courant ?

1 - Code César

1.1 - Chiffrement et déchiffrement César

Nous avons implémenté deux fonctions : `caesar_encode` pour chiffrer un message en décalant chaque lettre de l'alphabet d'un nombre donné, et `caesar_decode` pour l'inverser. Les fonctions gèrent les majuscules, les minuscules, et préservent les ponctuations, espaces, etc..

```python
def caesar_encode(text, shift):
    result = ""
    for char in text:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            result += chr((ord(char) - base + shift) % 26 + base)
        else:
            result += char
    return result

def caesar_decode(text, shift):
    return caesar_encode(text, -shift)

encoded = caesar_encode("cybersecurite", 11)
print("Chiffrement de 'cybersecurite' avec clé 11 :", encoded)
print(f"Déchiffrement de '{encoded}' avec la clé 11 :", caesar_decode(encoded, 11))
```

✓ 0.0s

```
Chiffrement de 'cybersecurite' avec clé 11 : njmpcdpnfctep
Déchiffrement de 'njmpcdpnfctep' avec la clé 11 : cybersecurite
```

1.2 - Attaque par force brute

Nous avons implémenté une fonction de force brute (`brute_force_caesar`) qui teste les 26 clés possibles (0 à 25) pour déchiffrer un message César.

```python
def brute_force_caesar(cipher_text):
    for key in range(26):
        print(f"Clé {key}: {caesar_decode(cipher_text, key)}")

message1 = "WP NZOLRP PDE FY LCE"
message2 = "HA YKZWCA AOP QJ WNP"

print("============ Message 1 ============")
brute_force_caesar(message1)

print("\n============ Message 2 ============")
brute_force_caesar(message2)
```

✓ 0.0s

Extraits du brute force pour les messages 1 et 2 :

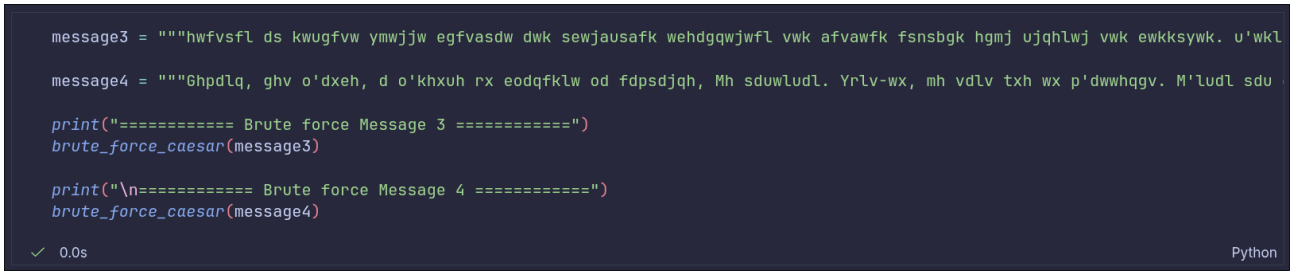
```
============ Message 1 ============
Clé  0: WP NZOLRP PDE FY LCE
Clé  1: VO MYNKQO OCD EX KBD
Clé  2: UN LXMJPN NBC DW JAC
  ...
Clé 10: MF DPEBHF FTU VO BSU
Clé 11: LE CODAGE EST UN ART
Clé 12: KD BNCZFD DRS TM ZQS
Clé 13: JC AMBYEC CQR SL YPR
Clé 14: IB ZLAXDB BPQ RK XOQ
Clé 15: HA YKZWCA AOP QJ WNP

============ Message 2 ============
Clé  0: HA YKZWCA AOP QJ WNP
Clé  1: GZ XJYVBZ ZNO PI VMO
Clé  2: FY WIXUAY YMN OH ULN
  ...
Clé 21: MF DPEBHF FTU VO BSU
Clé 22: LE CODAGE EST UN ART
Clé 23: KD BNCZFD DRS TM ZQS
Clé 24: JC AMBYEC CQR SL YPR
Clé 25: IB ZLAXDB BPQ RK XOQ
```

- Message 1 : « WP NZOLRP PDE FY LCE » → déchiffré avec la clé **11** : « LE CODAGE EST UN ART »
- Message 2 : « HA YKZWCA AOP QJ WNP » → déchiffré avec la clé **22** : « LE CODAGE EST UN ART »

On remarque que les deux messages donnent exactement le même texte en clair mais avec des clés différentes (11 et 22), ce qui montre qu'avec le chiffrement César, il existe plusieurs clés possibles pour obtenir le même résultat.

1.3 - Déchiffrement avec majuscules et minuscules

Comme on a déjà implémenté ces contraintes dans nos fonctions on passe à la brute-force, nous avons identifié les messages :



```
message3 = """hwfvsfl ds kwugfvw ymwjjw egfvasdw dwk sewjausafk wehdgqwjwfl vwk afvawfk fsnsbgk hgmj ujqhlwj vwk ewkksywk. u'wkl

message4 = """Ghpdlq, ghv o'dxeh, d o'khxuh rx eodqfklw od fdpsdjqh, Mh sduwludl. Yrlv-wx, mh vdlv txh wx p'dwwhqgv. M'ludl sdu

print("============ Brute force Message 3 ============")
brute_force_caesar(message3)

print("\n============ Brute force Message 4 ============")
brute_force_caesar(message4)
```

✓ 0.0s Python

```
============ Brute force Message 3 ============
  ...
Clé 18: pendant la seconde guerre mondiale les americains employerent des indiens navajos
pour crypter des messages. c'est l'un des rares codes de l'histoire a n'avoir jamais ete
brise. l'impenetrabilite du code navajo vient en particulier du fait que cette langue n'a
aucun lien avec une quelconque langue europeenne ou asiatique.
  ...

============ Brute force Message 4 ============
  ...
Clé  3: Demain, des l'aube, a l'heure ou blanchit la campagne, Je partirai. Vois-tu, je
sais que tu m'attends. J'irai par la foret, j'irai par la montagne. Je ne puis demeurer
loin de toi plus longtemps. Je marcherai les yeux fixes sur mes pensees, Sans rien voir au
dehors, sans entendre aucun bruit, Seul, inconnu, le dos courbe, les mains croisees,
Triste, et le jour pour moi sera comme la nuit. Je ne regarderai ni l'or du soir qui
tombe, Ni les voiles au loin descendant vers Harfleur, Et quand j'arriverai, je mettrai
sur ta tombe Un bouquet de houx vert et de bruyere en fleur. Victor Hugo - Les
Contemplations
  ...
```

La vérification par re-chiffrement confirme que nos fonctions sont correctes :

```
key3 = 18
key4 = 3

decoded3 = caesar_decode(message3, key3)
decoded4 = caesar_decode(message4, key4)

print(f"\n============ Message 3 déchiffré ============")
print(decoded3)

print(f"\n============ Message 4 déchiffré ============")
print(decoded4)

print("\n============ Vérification par re-chiffrement ============")
encoded3 = caesar_encode(decoded3, key3)
encoded4 = caesar_encode(decoded4, key4)

print("Message 3 re-chiffré == original =", encoded3 == message3, ", message :\n", encoded3)
print("Message 4 re-chiffré == original =", encoded4 == message4, ", message :\n", encoded4)
```

2] ✓ 0.0s Python

```
============ Message 3 déchiffré ============
pendant la seconde guerre mondiale les americains employerent des indiens navajos pour crypter des messages. c'est l'un des rares c

============ Message 4 déchiffré ============
Demain, des l'aube, a l'heure ou blanchit la campagne, Je partirai. Vois-tu, je sais que tu m'attends. J'irai par la foret, j'irai

============ Vérification par re-chiffrement ============
Message 3 re-chiffré == original = True , message :
 hwfvsfl ds kwugfvw ymwjjw egfvasdw dwk sewjausafk wehdgqwjwfl vwk afvawfk fsnsbgk hgmj ujqhlwj vwk ewkksywk. u'wkl d'mf vwk jsjwk
Message 4 re-chiffré == original = True , message :
 Ghpdlq, ghv o'dxeh, d o'khxuh rx eodqfklw od fdpsdjqh, Mh sduwludl. Yrlv-wx, mh vdlv txh wx p'dwwhqgv. M'ludl sdu od iruhw, m'ludl
```

2 - Indice de Coïncidence

2.1 - Calcul de l'indice de coïncidence et détection de langue

Notre fonction `coincidence_index` calcule l'indice pour chaque message, puis `detect_language` compare la valeur obtenue avec les indices présents dans le tableau :

```python
def coincidence_index(text):
    freq = [0] * 26
    total_letters = 0
    for char in text:
        if char.isalpha():
            freq[ord(char.lower()) - ord('a')] += 1
            total_letters += 1
    if total_letters <= 1:
        return 0.0
    return sum(f * (f - 1) for f in freq) / (total_letters * (total_letters - 1))

INDICES = {
    "Suédois": 0.0644, "Serbe": 0.0643, "Russe": 0.0529, "Portugais": 0.0745,
    "Néerlandais": 0.0798, "Norvégien": 0.0694, "Malaysien": 0.0852,
    "Japonais": 0.0772, "Italien": 0.0738, "Hébreu": 0.0768, "Grec": 0.0691,
    "Français": 0.0778, "Finnois": 0.0737, "Esperanto": 0.069, "Espagnol": 0.077,
    "Danois": 0.0707, "Arabe": 0.0758, "Anglais": 0.0667, "Allemand": 0.0762,
}

def detect_language(text):
    ic = coincidence_index(text)
    closest = min(INDICES, key=lambda lang: abs(INDICES[lang] - ic))
    return closest, ic
```

✓ 0.0s

```python
for i, msg in [(3, message3), (4, message4), (5, message5), (6, message6)]:
    lang, ic = detect_language(msg)
    print(f"\nMessage {i}:")
    print(f"  Indice = {ic:.4f}")
    print(f"  Langue détectée : {lang} (Indice Théorique : {INDICES[lang]:.4f})")
```

✓ 0.0s

```
Message 3:
  Indice = 0.0776
  Langue détectée : Français (Indice Théorique : 0.0778)

Message 4:
  Indice = 0.0710
  Langue détectée : Danois (Indice Théorique : 0.0707)

Message 5:
  Indice = 0.0832
  Langue détectée : Malaysien (Indice Théorique : 0.0852)

Message 6:
  Indice = 0.0737
  Langue détectée : Finnois (Indice Théorique : 0.0737)
```

Le seul correct est le message 3 qui est bien en français.

3 - L'Analyse Fréquentielle

3.1 - Déchiffrement par analyse fréquentielle

Notre fonction `decrypt_by_frequency_analysis` combine deux techniques :

1. L'indice de coïncidence pour déterminer la langue probable du message
2. L'analyse des fréquences des lettres pour trouver la clé de déchiffrement

```
FRENCH_FREQ = {
    'A': 8.4, 'B': 1.1, 'C': 3.0, 'D': 4.2, 'E': 17.3, 'F': 1.1, 'G': 1.3,
    'H': 0.9, 'I': 7.3, 'J': 0.3, 'K': 0.1, 'L': 6.0, 'M': 3.0,
    'N': 7.1, 'O': 5.3, 'P': 3.0, 'Q': 1.0, 'R': 6.6, 'S': 8.1, 'T': 7.1,
    'U': 5.7, 'V': 1.3, 'W': 0.1, 'X': 0.4, 'Y': 0.3, 'Z': 0.1
}

ENGLISH_FREQ = {
    'A': 8.2, 'B': 1.5, 'C': 2.8, 'D': 4.3, 'E': 12.7, 'F': 2.2, 'G': 2.0,
    'H': 6.1, 'I': 7.0, 'J': 0.2, 'K': 0.8, 'L': 4.0, 'M': 2.4,
    'N': 6.7, 'O': 7.5, 'P': 1.9, 'Q': 0.1, 'R': 6.0, 'S': 6.3, 'T': 9.1,
    'U': 2.8, 'V': 1.0, 'W': 2.4, 'X': 0.2, 'Y': 2.0, 'Z': 0.1
}

ETAOIN_FR = 'EASITNRULODCPMVQFBGHJXYZWK'
ETAOIN_EN = 'ETAOINSHRDLCUMWFGYPBVKJXQZ'
```

```
def compute_letter_frequencies(text):
    freq = {}
    total = 0
    for char in text.upper():
        if char.isalpha():
            freq[char] = freq.get(char, 0) + 1
            total += 1
    for letter in freq:
        freq[letter] = (freq[letter] / total) * 100 if total > 0 else 0
    return freq

def sort_by_frequency(freq):
    return ''.join(sorted(freq, key=lambda x: freq[x], reverse=True))

def find_caesar_key_by_frequency(sorted_letters, language):
    if not sorted_letters:
        return 0
    most_frequent = sorted_letters[0]
    expected_most_frequent = ETAOIN_EN[0] if language == "Anglais" else ETAOIN_FR[0]
    return (ord(most_frequent) - ord(expected_most_frequent)) % 26

def decrypt_by_frequency_analysis(text):
    lang, ic = detect_language(text)
    freq = compute_letter_frequencies(text)
    sorted_letters = sort_by_frequency(freq)

    print(f"  Indice de coïncidence : {ic:.4f}")
    print(f"  Langue probable : {lang}")
    print(f"  Lettres par fréquence : {sorted_letters[:10]}...")
    print(f"  Top 5 lettres : ", end="")
    for letter in sorted_letters[:5]:
        print(f"{letter}({freq[letter]:.2f}%)", end=" ")
    print()

    key = find_caesar_key_by_frequency(sorted_letters, lang)
    decrypted = caesar_decode(text, key)
    return decrypted, key, lang
```

Résultats :

- **Message 4** :
  - Indice de coïncidence : 0.0710 (indice proche du Danois, mais c'est du français)
  - Lettre la plus fréquente : H (15.1%)
  - Clé trouvée : 3 (écart entre H et E)
  - Texte déchiffré : Demain, des l'aube, a l'heure ou blanchit la campagne, Je partirai. Vois-tu, je sais que tu m'attends. J'irai par la foret, j'irai par la montagne. Je ne puis demeurer loin de toi plus longtemps. Je marcherai les yeux fixes sur mes pensees, Sans rien voir au dehors, sans entendre aucun bruit, Seul, inconnu, le dos courbe, les mains croisees, Triste, et le jour pour moi sera comme la nuit. Je ne regarderai ni l'or du soir qui tombe, Ni les voiles au loin descendant vers Harfleur, Et quand j'arriverai, je mettrai sur ta tombe Un bouquet de houx vert et de bruyere en fleur. Victor Hugo - Les Contemplations
- **Message 5** :
  - Indice de coïncidence : 0.0832 (indice proche du Malaysien, mais c'est du français)
  - Lettre la plus fréquente : T (20.1%)
  - Clé trouvée : 15 (écart entre T et E)
  - Texte déchiffré : Julie et Jules ont invente un moyen de communication secret. l'expediteur ecrit le texte ligne par ligne dans un rectangle de 6 lignes et 3 colonnes. ensuite il le recopie colonne par colonne. Jules, pendant le controle de reseaux, a oublie sa calculette. il envoie le message suivant: BQTFZYLTKEDPCB ?PF. quelle doit etre la reponse de Julie?

3.2 - Validation avec le message 6

Pour valider notre méthode, nous l'appliquons au message 6 qui est en anglais :

- Indice de coïncidence : 0.0737 (plus proche du Finnois mais c'est de l'anglais…)
- Lettre la plus fréquente : L (13.1%)
- Clé trouvée : 7 (écart entre L et E)
- Texte déchiffré : William Blake The Tiger TIGER, tiger, burning bright In the forests of the night, What immortal hand or eye Could frame thy fearful symmetry? In what distant deeps or skies Burnt the fire of thine eyes? On what wings dare he aspire? What the hand dare seize the fire? And what shoulder and what art Could twist the sinews of thy heart? And when thy heart began to beat, What dread hand and what dread feet? What the hammer? what the chain? In what furnace was thy brain? What the anvil? What dread grasp Dare its deadly terrors clasp? When the stars threw down their spears, And water'd heaven with their tears, Did He smile His work to see? Did He who made the lamb make thee? Tiger, tiger, burning bright In the forests of the night, What immortal hand or eye Dare frame thy fearful symmetry ?

Cela confirme que notre méthode d'analyse fréquentielle fonctionne efficacement pour les textes en français comme en anglais. Même si l'indice de coïncidence n'est pas parfaitement aligné avec les valeurs de référence, l'analyse des fréquences permet de trouver la clé correcte et d'obtenir un texte en clair cohérent.

3.3 - Analyse par digrammes et trigrammes

Pour déterminer si un message est en clair ou chiffré, nous avons implémenté une analyse basée sur les digrammes (groupes de 2 lettres) et les trigrammes (groupes de 3 lettres).

Nous avons créé `count_digrams` et `count_trigrams` qui comptent les occurrences de chaque groupe de lettres dans un texte. La fonction `is_plaintext` compare les digrammes et trigrammes les plus fréquents du texte avec les références connues pour le français et l'anglais. Si un nombre suffisant de correspondances est trouvé, le texte est considéré comme étant en clair.

```
TOP_FRENCH_DIGRAMS = ['ES', 'DE', 'LE', 'EN', 'RE', 'NT', 'ON', 'ER', 'TE', 'EL',
                      'AN', 'SE', 'ET', 'LA', 'AI', 'IT', 'ME', 'OU', 'EM', 'IE']

TOP_FRENCH_TRIGRAMS = ['ENT', 'LES', 'EDE', 'DES', 'QUE', 'AIT', 'LLE', 'SDE', 'ION', 'EME',
                       'ELA', 'RES', 'MEN', 'ESE', 'DEL', 'ANT', 'TIO', 'PAR', 'ESD', 'TDE']

TOP_ENGLISH_DIGRAMS = ['TH', 'HE', 'IN', 'EN', 'NT', 'RE', 'ER', 'AN', 'TI', 'ON',
                       'AT', 'SE', 'ND', 'OR', 'AR', 'AL', 'TE', 'CO', 'DE', 'TO']

TOP_ENGLISH_TRIGRAMS = ['THE', 'AND', 'ING', 'HER', 'ERE', 'ENT', 'THA', 'NTH', 'WAS', 'ETH',
                        'FOR', 'DTH', 'HAT', 'STH', 'ITH', 'TER', 'EST', 'OFT', 'SAN', 'HIS']
```

```
def count_ngrams(text, n):
    cleaned = ''.join(c.upper() for c in text if c.isalpha())
    ngrams = {}
    for i in range(len(cleaned) - n + 1):
        gram = cleaned[i:i+n]
        ngrams[gram] = ngrams.get(gram, 0) + 1
    return dict(sorted(ngrams.items(), key=lambda x: x[1], reverse=True))


def count_digrams(text):
    return count_ngrams(text, 2)


def count_trigrams(text):
    return count_ngrams(text, 3)


def is_plaintext(text, language="french"):
    digrams = count_digrams(text)
    trigrams = count_trigrams(text)
    top_digrams = list(digrams.keys())[:20]
    top_trigrams = list(trigrams.keys())[:20]

    if language == "french":
        ref_digrams = TOP_FRENCH_DIGRAMS
        ref_trigrams = TOP_FRENCH_TRIGRAMS
    else:
        ref_digrams = TOP_ENGLISH_DIGRAMS
        ref_trigrams = TOP_ENGLISH_TRIGRAMS

    matching_digrams = sum(1 for d in top_digrams if d in ref_digrams)
    matching_trigrams = sum(1 for t in top_trigrams if t in ref_trigrams)

    score = matching_digrams + matching_trigrams
    threshold = 6

    return score >= threshold, score, matching_digrams, matching_trigrams
```

Résultats sur les messages chiffrés :

- Les 4 messages chiffrés sont correctement identifiés comme « Texte chiffre » avec des scores de 0 ou 1 (aucun digramme/trigramme courant détecté). Voici notre output console :

```
Message 3 (chiffre): Texte chiffre (score=0, digrammes=0, trigrammes=0, top digrammes:
['WF', 'WK', 'VW', 'MW', 'WD'])
Message 4 (chiffre): Texte chiffre (score=0, digrammes=0, trigrammes=0, top digrammes:
['HV', 'DL', 'HQ', 'GH', 'XU'])
Message 5 (chiffre): Texte chiffre (score=1, digrammes=1, trigrammes=0, top digrammes:
['AT', 'TH', 'DC', 'IT', 'TR'])
Message 6 (chiffre): Texte chiffre (score=1, digrammes=1, trigrammes=0, top digrammes:
['AO', 'OL', 'OH', 'DO', 'HA'])
```

Résultats sur les messages déchiffrés :

- Message 3 (français) : texte en clair, score = 15 (12 digrammes + 3 trigrammes correspondants)
- Message 4 (français) : texte en clair, score = 14 (12 digrammes + 2 trigrammes correspondants)
- Message 5 (français) : texte en clair, score = 15 (12 digrammes + 3 trigrammes correspondants)
- Message 6 (anglais) : texte en clair, score = 18 (12 digrammes + 6 trigrammes correspondants)

3.4 - (BONUS) Lettres répétées consécutivement

Notre fonction `count_repeated_letters` parcourt le texte et identifie toutes les lettres qui apparaissent consécutivement un nombre donné de fois (2 ou 3 fois).

```python
def count_repeated_letters(text, repeat_count=2):
    cleaned = text.upper()
    results = {}
    i = 0
    while i ⩽ len(cleaned) - repeat_count:
        char = cleaned[i]
        if char.isalpha():
            count = 1
            while i + count < len(cleaned) and cleaned[i + count] == char:
                count += 1
            if count ⩾ repeat_count:
                pattern = char * repeat_count
                results[pattern] = results.get(pattern, 0) + 1
                i += count
                continue
        i += 1
    return dict(sorted(results.items(), key=lambda x: x[1], reverse=True))


for name, msg in [("Message 3 (dechiffre)", decoded3), ("Message 4 (dechiffre)", decoded4),
                  ("Message 5 (dechiffre)", decoded5), ("Message 6 (dechiffre)", decoded6)]:
    doubles = count_repeated_letters(msg, 2)
    triples = count_repeated_letters(msg, 3)
    print(f"{name}:")
    print(f"  Lettres doublees : {doubles if doubles else 'Aucune'}")
    print(f"  Lettres triplees : {triples if triples else 'Aucune'}")
    print()
```

✓ 0.0s

```
Message 3 (dechiffre):
  Lettres doublees : {'RR': 1, 'SS': 1, 'TT': 1, 'EE': 1, 'NN': 1}
  Lettres triplees : Aucune

Message 4 (dechiffre):
  Lettres doublees : {'TT': 2, 'EE': 2, 'NN': 1, 'MM': 1, 'RR': 1}
  Lettres triplees : Aucune

Message 5 (dechiffre):
  Lettres doublees : {'NN': 3, 'MM': 1, 'TT': 1, 'SS': 1, 'LL': 1}
  Lettres triplees : Aucune

Message 6 (dechiffre):
  Lettres doublees : {'MM': 5, 'EE': 4, 'LL': 1, 'RR': 1}
  Lettres triplees : Aucune
```