Laboratoire: Attaques cryptographiques

Introduction aux attaques cryptographiques

Daniel Migault

August 26, 2023

Contents

1 Matériel à rendre **3**

2 Code Cæsar **3**

3 Chiffrement par substitution **7**

List of Figures

Listings

1 Différentes représentation d'un octet 3
2 Différentes représentation d'un caractère 6

1 Matériel à rendre

Chaque étudiant remettra les fichiers suivants:

- un rapport PDF au nom de `report.pdf`
- les fichiers python `caesar.py` et `substitution.py` préalablement fournis et complétés par l'étudiant.

Les fichiers ne seront pas soumis sous forme d'archives zip ou tar.gz mais bien soumis individuellement. Il est également important de respecter le nom du fichier afin d'en permettre son évaluation.

2 Code Cæsar

Vous intercepter le message suivant:

$$intercepted\_message =' fytgpcdtepopdspcmczzvp'$$

Vous suspectez que la personne emploie un code de César. Le but est de trouver le message en clair associé.

On supposera que seulement des caractères ASCII minuscules a-z sont utilisés, les autres caractères comme la ponctuation, les espaces sont simplement inchangés lors du chiffrement.

Les caractères sont codés sur un octet.

Un octet est composé de 8 bits et ces bits peuvent avoir une représentation différente suivant que l'octet est interprété comme un caractère ou un chiffre. Pour représenter la valeur brute de cet octet on utilise la notation hexadécimale. La base décimale comprend les 10 chiffres suivants: 0, 1, 2, 3,4 ,5 ,6 ,7 , 8, 9. La base hexadécimale comprend les 16 chiffres suivants: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E et F. L'avantage d'avoir 16 chiffres est qu'ils se code facilement sur 4 bits ($2^4 = 16$). Comme 1 octet est comprend 8 bits, il peut être représente par 2 chiffres hexadécimaux.

De manière générale, la machine ne sait pas si cet octet représente un chiffre ou un caractère, c'est le programmeur qui indique à la machine comment interpréter ces bits. Le listing 1 illustre des représentations équivalentes d'un même octet. Un octet $\backslash x61$ peu soit représenter le caractère 'a' ou le chiffre '97'.

```
## conversion entre une représentation
## de type caractère et une représentation
## de type numérique
>>> ord('a')
97
## conversion entre une représentation
## de type numérique et une représentation
## de type binaire
>>> hex(97)
'0x61'
```

```
## conversion entre une représentation
## de type numérique et un représentation
## de type caractère
>>> chr(97)
'a'
```

Listing 1: Différentes représentation d'un octet

QUESTION 1

Le code de Cæsar consiste, étant donné la clé k, en une translation de k caractères.

Au vu du code du listing 1 expliquer comment la représentation numérique d'un caractère permet d'implémenter une translation de caractère.

Écrire le pseudo code d'une fonction qui prend comme paramètres d'entrée un caractère 'a', ..., 'z' et une clé k et retourne le caractère translaté. Vérifier le fonctionnement du pseudo code pour les caractères 'a' et 'z'.

Aucun code python n'est attendu et la description est attendue dans le rapport PDF. (2 points)

QUESTION 2

Compléter le fichier `caesar.py` et écrire une fonction `encrypt` qui prend comme entrées un texte (clair) composé d'une suite de caractères ASCII et d'une clé et qui retourne le texte chiffre par le code de Cæsar.

L'évaluation de la fonction `encrypt` évaluera le chiffrement des lettre individuelle 'a-Z', le chiffrement d'un mot et le chiffrement d'un paragraphe pour une clé choisit aléatoirement. On remarquera que le code Cæsar ne chiffre que les caractères a-z et que les caractères A-Z doivent subir une transformation pour ce ramener à l'espace des caractères chiffres a-z. Autrement dit, il faut convertir les caractères majuscule en minuscule.

L'évaluation se ferra à partir fichier `caesar.py` uniquement (3 points).

QUESTION 3

Expliquer la relation qui existe entre le chiffrement et le déchiffrement.

De même compléter le fichier `caesar.py` et écrire la fonction `decrypt` qui prend comme entrées un texte (chiffré) composé d'une suite de caractères

ASCII et d'une clé et qui retourne le texte déchiffré par le code de Cæsar.

L'évaluation de la fonction `decrypt` évaluera le chiffrement des lettre individuelle 'a-Z', le chiffrement d'un mot et le chiffrement d'un paragraphe pour une clé choisit aléatoirement. L'évaluation vérifiera également que la combinaison `decrypt`(`encrypt`(clear_text, key)) retourne bien clear_text (en minuscule).

L'évaluation se ferra à partir fichier `caesar.py` uniquement. (2 points)

QUESTION 4

Expliquer et trouver deux clés différentes k et $k^{'}$ qui donnent un même chiffré. En déduire le nombre (minimal) de clés qui, pour un texte claire donné, fournissent des chiffrés différents.

Expliquer pourquoi le code de Cæsar se prête à une attaque de force brute et décrire comment vous procèderiez à une attaque de type brute force pour décoder un chiffré dont vous n'avez pas la clé k.

Aucun code python n'est attendu et la description est attendue dans le rapport PDF. (2 points)

QUESTION 5

Compléter le fichier `caesar.py` et écrire une fonction `brute_force` qui implémente une attaque de type force brute pour le message chiffré *intercepted_message*.

`brute_force` devra retourner un dictionnaire qui a chaque valeur de clé k possible associe un message clear_text dont le chiffre correspondrait a *intercepted_message*. Plus précisément, le dictionnaire sera de la forme $\{ k_0 : clear\_text_0, k_1 : clear\_text_1, ..., k_n : clear\_text_n \}$.

L'évaluation de la fonction `brute_force` évaluera les clés, valeurs du dictionnaire retourné a partir du fichier `caesar.py`.

Indiquez la valeur de clear_text et la clé k dans le rapport PDF que vous pensez la plus vraisemblable. (3 points)

Tous les caractères ASCII sont codes sur 7 bits ce qui permet de représenter $2^7 = 128$ caractères. Ce n'est clairement pas suffisant pour représenter l'ensemble des caractères. Unicode réunit l'ensemble des caractères disponibles et UTF-8 définit la manière de représenter ces caractères.

Le listing 2 illustre le représentation du caractère 'é' sur 2 octets.

La fonctionnalité des fonctions `hex` `chr` et `bytes` définies dans [1] `ord` est rappelée ci dessous.

ord(c): Given a string representing one Unicode character, return an integer representing the Unicode code point of that character. For example, ord('a') returns the integer 97 and ord('€') (Euro sign) returns 8364. This is the inverse of chr().

hex(x): Convert an integer number to a lowercase hexadecimal string prefixed with "0x". If x is not a Python int object, it has to define an __index__() method that returns an integer.

chr(i): Return the string representing a character whose Unicode code point is the integer i. For example, chr(97) returns the string 'a', while chr(8364) returns the string '€'. This is the inverse of ord(). The valid range for the argument is from 0 through 1,114,111 (0x10FFFF in base 16). ValueError will be raised if i is outside that range.

ass bytes([source[, encoding[, errors]])] Return a new "bytes" object, which is an immutable sequence of integers in the range 0 <= x < 256. bytes is an immutable version of bytearray – it has the same non-mutating methods and the same indexing and slicing behavior. Accordingly, constructor arguments are interpreted as for bytearray(). Bytes objects can also be created with literals, see String and Bytes literals.

```
>>> ord('é')
233
>>> hex(233)
'0xe9'
>>> bytes('é')
'\xc3\xa9'
```

Listing 2: Différentes représentation d'un caractère

Table 1: Exemple de représentation du caractère 'é'

Unicode point	character	UTF-8 (hex)	name
U+00E9	é	c3 a9	LATIN SMALL LETTER E WITHACUTE

Table 2: UTF8 représentation d'un code Unicode

Number of bytes	Bits for code point	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	7	U+0000	U+007F	0xxxxxxx			
2	11	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	16	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	21	U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

QUESTION 6

Au vue du code exécute dans le listing 2 et des tables 1 et 2, expliquer pour le caractère 'é' ayant un code de 233 qui est inférieure a 255 est codée sur deux octets et non 1 octet. Donner la valeur du code ASCII étendu associée au caractère 'é'.

Aucun code python n'est attendu et la description est attendue dans le rapport PDF. (3 points)

3 Chiffrement par substitution

Dans cet exercice, on intercepte deux textes chiffrés que l'on cherchera à déchiffrer les textes chiffrés `cipher_text_1` et `cipher_text_2`

Les textes chiffrés sont indiqués dans le fichier `substitution.py`.

On soupçonne le texte encodé par la méthode de substitution où chaque caractère est code sur 2 octets.

La clé est alors constituée d'un dictionnaire qui à chaque lettre fait correspondre une valeur de 16 bits. Cette valeur est tirée au hasard pour chaque lettre et chaque lettre différente est codée par une valeur différente.

QUESTION 7

Soit n le nombre de lettres de l'alphabet considéré, évaluer le nombre de clés possibles.

Pour évaluer le nombre de clés possibles, on commencera par considérer un alphabet où seules les lettres a-z sont utilisées. On évaluera successivement la lettre 'a', puis 'b', 'c', puis... 'z'. Les codes correspondants à la lettre 'a' peut prendre toutes les valeurs possibles codées sur 16 bits. La lettre 'b' peut prendre toutes les valeurs possible codées sur 16 bit, sauf la valeur prise par 'a'. Idem pour 'c', et finalement 'z'. Évaluer le nombre de possibilités de chiffres et montrer qu'une attaque de type brute force est envisageable?

Aucun code python n'est attendu et la description est attendue dans le rapport PDF. (4 points)

Même si une attaque de type brute force est envisageable, on préfèrera effectuer une analyse cryptographique. Typiquement, on allons évaluer la fréquence des caractères dans un texte en claire - représentatif dans notre cas de la langue française - ainsi que sur un code chiffré. En supposant les fréquences des caractères semblables dans les deux textes, on "devinera" que le chiffré avec la plus haute fréquence correspondra à la lettre qui a la plus haute fréquence.

QUESTION 8

Compléter le fichier `substitution.py` et implémenter `freq_text`, une fonction qui prend comme paramètre d'entrée un texte - dans notre cas `text_hugo` - et qui retourne une liste dont chaque élément correspond à un caractère avec son occurrence associée. On ordonnera cette liste par ordre décroissant d'occurrence de manière a ce que les caractères les plus courants apparaissent en premier.

L'évaluation se ferra à partir fichier `substitution.py` uniquement. (4 points)

Cette liste de caractères sera considérée par la suite comme représentative d'un texte en langue Française. Typiquement, on considèrera qu'avec un forte probabilité, n'importe quel texte français présentera une distribution de caractères similaires.

Cette liste prendra typiquement la forme suivante $freq\_hugo = [('', 27), ('p', 24), ('v', 20), ('g', 19), ('q', 19)]$ caractères ordonnes selon leur occurrence. Dans l'exemple suivant, les trois caractères les plus fréquents seront 'é', 'p' et 'v'. Toujours d'après l'exemple, on considéra que n'importe quel texte de la même langue aura 'é' comme caractère qui apparait le plus grand nombre de fois, suivi de 'p' et puis 'v'.

Pour compter l'occurrence associée a chaque lettre, on constituera un dictionnaire qui associera a chaque lettre rencontrée dans le texte le nombre de fois ou ce caractère apparait. Ce dictionnaire se présentera sous la forme $freq = '' : 27, 'p' : 24, 'v' : 20, 'g' : 19, 'q' : 19$. Une fois le texte parcouru, on convertira le dictionnaire en liste ordonnée grâce a formule $sorted(freq.items(), key = lambdaitem : item[1], reverse = True)$.

QUESTION 9

Compléter le fichier `substitution.py` et implémenter `freq_cipher`, une fonction qui prend comme paramètre une séquence d'octets - dans notre cas `cipher_text_1` - et qui retourne une liste de "caractères" ordonnés selon leur occurrence.

L'évaluation se ferra à partir fichier `substitution.py` uniquement. (4 points)

On notera qu'ici le "texte" est représente par une série d'octets et chaque "caractère" est représente par deux octets. Il faudra donc lire les octets deux par deux. Une des possibilité est de parcourir chaque octet. Si l'index de l'octet associé est pair (0, 2, 4 ...) on considèrera l'octet a cet index ainsi qu'a l'index

suivant. Si l'index est impair on ne ferra rien. En effet l'index d'une liste commence a 0.

QUESTION 10

En comparant la liste ordonnée des caractères retournée par `freq_cipher` avec `clear_text_hugo` et celle obtenue avec `cipher_text_1` conjecturez le texte en claire qui correspond à `cipher_text_1`.

Aucun code python n'est attendu et la description est attendue dans le rapport PDF. (2 points)

Afin de vérifier la conjecture de la question précédente, nous allons definir une fonction `guess_clear_text` qui va nous permettre de tester des valeur potentielle de clés de déchiffrement. Plus spécifiquement, dans notre cas la clé de déchiffrement sera représentée par un dictionnaire qui établit la correspondance entre la valeur formée de deux octets et un caractère alphanumérique. Typiquement un tel dictionnaire aura la forme suivante $key = \{b'\backslash x9e' :' a', b'\backslash xec' :' o', ..., b'\backslash xc0' :' '\}$. Afin de permettre des tests, il est important de pouvoir tester différentes valeurs de clés de décryptions ainsi que des clés de décryptions partielles. Ce sera le propos de la fonction `guess_clear_text` qui prend en entrée le texte chiffré `encrypted_text` ainsi qu'une clé de déchiffrement `decryption_key` . La fonction `guess_clear_text` parcourt chaque caractère du texte chiffre (formée de deux octets). La fonction va alors créer, un texte `clear_text` qui correspond au texte partiellement déchiffre. Ce texte est initialisé comme un texte vide `""` . La fonction `guess_clear_text` va parcourir l'ensemble des caractères du texte chiffré (chaque caractère étant représente par deux octets). Pour chaque caractère, la fonction `guess_clear_text` vérifie si le caractère apparait dans la `decryption_key` . Si cela est le cas, le caractère associé par la clé de déchiffrement est ajoute au text `clear_text` . Si le caractère n'apparait pas dans la clé de déchiffrement `decryption_key` , ce dernier est ajouter tel quel au texte `clear_text` .

QUESTION 11

Implémenter `guess_clear_text` . Illustrer son utilisation on avec un clé de déchiffrement partielle qui sera construite en faisant correspondre les 15 caractères les plus fréquents du `cipher_text_1` avec ceux obtenu avec `clear_text_hugo`. Lorsqu'un caractère ne peut être déchiffré, le texte déchiffre présentera ce caractère sous sa forme encodée, soit `"%s"%encrypted_char`.

On implémentera `build_decryption_key` la fonction qui construira la clé de déchiffrement à partir des `key_size` premiers éléments la liste ordonnée des caractères obtenue pour `clear_text_hugo` et `cipher_text_1`. On pourra par exemple prendre `key_size` = 15.

L'évaluation se ferra à partir fichier `substitution.py` uniquement. (4 points)

QUESTION 12

Procéder de même avec `cipher_text_2` en construisant une cle de dechiffrement partielle des 15 caracteres les plus courant de `clear_text_hugo`. Cela permet-il de déchiffrer le texte. Y a-t-il des mots que l'on puisse reconnaitre ?

Aucun code python n'est attendu et la description est attendue dans le rapport PDF. (2 points)

On apprends que le texte commence par le prénom 'Anton Voyl n'arrivait'.

QUESTION 13

En déduire les 3 premières phrases du texte.

Aucun code python n'est attendu et la description est attendue dans le rapport PDF. (2 points)

References

[1] "Python3 built-in functions," https://docs.python.org/3.8/library/functions.html, 2020.