Sécurité des communications

BARBIER J.M. - LGT Dumezil - NSI

5 novembre 2025

Table des matières

1 Cryptographie symétrique

1.1 Code César

1.1.1 Principe

- chiffrement par substitution monoalphabétique
- clef = 1 nombre
- chaque caractère est décalé de ce nombre
- historiquement : attribué à César
- exemple : ROT13
- attaques possibles
  - analyses fréquentielles
  - recherche de la valeur de décalage

1.1.2 Exercices (papier)

— Chiffrer votre nom et votre prénom en utilisant la méthode ROT13.
— Déchiffrer le message suivant (vous pouvez ne faire que les premiers mots et utiliser rot13.com pour le reste) :

`Tm bmzum « kpqnnzmumvb » mab cbqtqam lmxcqa tm fdqqm aqmktm liva tm amva lm kpqnnzmz cv umaaiom. T'wxmzibqwv qvdmzam, ycq acxxwam ycm t'wv kwvviqaam ti ktm, mab lwvk tm « lmkpqnnzmumvb ».`

1.1.3 Exercices (python)

— créer une fonction `decale` prenant comme arguments un caractère `lettre` et un nombre positif ou négatif `decalage`, qui renvoie le caractère `lettre` décalé de `decalage` dans l'alphabet. Seuls les ensembles de caractères A-Z et a-z sont pris en compte pour `lettre`, et le résultat renvoyé doit être dans l'intervalle A-Z. Le résultat pour un caractère n'appartenant pas à ces intervalles est la chaîne vide. Indication : utiliser la fonction `ord` pour obtenir le code ASCII d'un caractère, et `chr` pour obtenir le caractère correspondant à un code ASCII (RTFM si besoin !).

Votre code doit satisfaire aux assertions suivantes (à compléter vous-même)

```
assert decale(lettre="A", decalage=3) == "D"
assert decale(lettre="D", decalage=-3) == "A"
assert decale(lettre="a", decalage=5) == "F"
assert decale(lettre="f", decalage=-5) == "A"
assert decale(lettre="é", decalage=-5) == ""
assert decale(lettre=" ", decalage=5) == ""
#assert decale(lettre="A", decalage=30) == # à vous de compléter
#assert decale(lettre="D", decalage=-30) == # à vous de compléter
#assert decale(lettre="d", decalage=34) == # à vous de compléter
#assert decale(lettre="w", decalage=-34) == # à vous de compléter
# sinon, c'est pas plus cher de tout tester...
for l in "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ":
  for d in range(1, 50):
    assert decale(
      lettre=decale(lettre=l, decalage=d),
      decalage=-d)==l.upper(), f"Erreur sur {l} décalage {d}"
```

Créer ensuite une fonction `cesar` prenant comme argument un message `message` et un décalage `decalage` et renvoyant le message `message` chiffré en utilisant le décalage `decalage`. Ecrire une ou plusieurs assertions permettant de valider le fonctionnement de cette fonction. Comment peut-on utiliser cette fonction `cesar` pour déchiffrer un message ?

1.2 Code Vigenere

1.2.1 Principe

— chiffrement par substitution polyalphabétique
— clef = 1 phrase

- chaque caractère est décalé du nombre correspondant au caractère correspondant dans la clef répétée => le même caractère situé à différents endroits du message n'est plus chiffré de la même manière => l'analyse fréquentielle n'est plus possible.
- historiquement : utilisé par Enigma (= clef > 17000 lettres)
- attaques possibles
  - si la clef est trop courte / trop répétée, par analyse fréquentielle (+ complexe que monoalphabétique)
- sécurité : clef longue non répétée, pas prévisible

1.2.2 Exercice (papier)

Chiffrer à la main le message "BONJOURCECIESTUNMESSAGESECRET" en utilisant la clef "DUMEZIL"

1.2.3 Exercices (python, maison)

Créer une fonction `vigenere(message, clef)` permettant de chiffrer le message `message` en utilisant la clef `clef`. Ecrire une ou plusieurs assertions permettant de valider le fonctionnement de cette fonction.

Créer une fonction `devigenere(message, clef)` permettant de déchiffrer le message. Ecrire une ou plusieurs assertions permettant de valider le fonctionnement de cette fonction.

1.3 XOR

1.3.1 Principe

- chiffrement symétrique par clef
- xor entre chaque bit du message et chaque bit de la clef
- rappel : table de vérité de xor (fromage XOR dessert)
- parfaitement sûr pour une clef à usage unique, aléatoire et de même longueur que le message à crypter
- attaques possibles :
  - clef faibles ou réutilisée

1.3.2 Pratique

- Chiffrer à la main le message "VIVELANSI" en utilisant la clef 2024. Déchiffrer le résultat obtenu.

1.4 Faiblesses

1.4.1 Principe

La sécurité du message est basée principalement sur la **qualité de la clef** et sur la **fiablilité de sa transmission**.

- clef aussi longue que le message
- clef aléatoire
- clef non réutilisée

2 Cryptographie asymétrique

2.1 Chiffrement à clef publique

2.1.1 Principe

Principe : la sécurité est basée sur l'existence de deux clefs de chiffrement. Un message chiffré avec une des deux clefs n'est déchiffrable qu'avec la deuxième clef.

Deux utilisations :

- chiffrement d'un message : l'expéditeur utilise la clef publique du destinataire pour coder son message. Le destinataire utilise sa clef privée pour décoder le message de l'expéditeur, garantissant la confidentialité du contenu
- authentification de l'expéditeur : l'expéditeur utilise sa clef privée pour coder un message que le destinataire peut décoder avec la clef publique de l'expéditeur ; c'est le mécanisme utilisé par la signature numérique pour authentifier l'auteur d'un message.

2.2 RSA

2.2.1 Principe

C'est la première méthode de chiffrement asymétrique. Inventé en 1977 par Ronald Rivest, Adi Shamir, Léonard Adleman (=> R.S.A.)

Il est basé sur un problème facile à résoudre dans un sens, et très difficile à résoudre dans l'autre sens : la factorisation d'un nombre premier.

Exemple :

- A=1123123123123151
- B=312341988223271
- AxB=350798509295814574987017046921 (facile)
- 350798509295814574987017046921=?x? (très difficile)

Plus d'informations : https ://fr.wikipedia.org/wiki/Chiffrement_RSA

Attaques possibles :

- brute force : factorisation de la clef (record : 795 bits en 2019)
- failles sur la génération des nombres premiers p et q

2.2.2 Pratique

Python : créer une fonction `factorise(n)` qui renvoie la décomposition en facteurs premiers du nombre n (en utilisant l'algorithme super naif suivant : diviser n par tous les nombres `i > 1` ; si le reste de la division `n%i == 0`, alors enregistrer `i` et recommencer avec `n=n/i`, tant que `i <= sqrt(n)`)

2.3 PGP

2.3.1 Principe

Chiffrement mixte :

- authentification : l'expéditeur crée un "condensat" (hash) de son message, le chiffre avec la clef publique de l'expéditeur et l'ajoute en en-tête du message
- confidentialité : génération d'une clef secrète pour chaque message, chiffrement symétrique du message compressé avec cette clef. La clef est chiffrée avec la clef publique de l'expéditeur et ajoutée à l'en-tête du message.

La compression rend plus difficile la cryptanalyse du message.

2.3.2 Exercices

En utilisant la commande `gpg`

- créer une paire de clefs
- chiffrer un message
- signer un message
- déchiffrer un message
- vérifier l'authenticité d'un message

2.4 HTTPS / TLS

2.4.1 Principe

protocole HTTP plus TLS (Transport Layer Sécurity)

- le client contacte le serveur en demandant une connexion sécurisée (propose des algos)
- le serveur répond en choissant une méthode et en présentant un certificat, délivré par une **autorité de confiance**
  - le certificat contient des informations sur le serveur (nom de domaine, date d'expiration, identité de l'organisation) et une clef publique du serveur permettant le chiffrement
  - il est signé par la clef secrète de l'autorité de confiance
- le client vérifie la signature du certificat en utilisant la clef publique de l'autorité de confiance, et vérifie la validité du certificat (nom de domaine, date de validité)
- si le certificat est valide, le client crée une clef symétrique pour un chiffrement symétrique, la chiffre avec la clef publique du serveur et lui envoie (=> le serveur peut maintenant chiffrer/déchiffrer les message)

— le reste des échanges est fait en utilisant cette clef

2.4.2 Illustration

3 élèves pour 3 rôles

— autorité de certification, disposant d'un papier clef privée et d'un papier clef publique
— serveur, disposant d'un papier clef privée et d'un papier clef publique
— client, disposant d'une copie du papier clef publique de l'autorité de certification

Chaque clef est représentée par un symbole (carré, rond, ovale, étoile, …), et on représente sur papier le chiffrage d'un message en entourant le message par le symbole correspondant à la clef utiliée.

Etape 1 : CSR

— le serveur crée un papier avec sa clef publique et son nom de domaine
— il donne ce papier à l'autorité de certification
— l'autorité de certification vérifie que le serveur est bien en charge du nom de domaine
— l'autorité de certification ajoute une date de validité et signe avec sa clef privée le papier, et le renvoie au serveur

Etape 2 : établissement de la connection HTTPS

— le client demande au serveur une connection HTTPS
— le serveur recopie son certificat et l'envoie au client
— le client vérifie que la signature de l'autorité de certification est bonne, que le domaine est bon et que la date de validité est bonne, et récupère la clef publique du serveur
— il crée alors une clef symétrique de qualité en lançant un dé une dizaine de fois, chiffre cette clef avec la clef publique du serveur
— il envoie cette clef chiffrée au serveur
— le serveur la décrypte

Etape 3 : communication

— le serveur et le client communiquent en chiffrant leurs messages par la clef symétrique