Emanuel Rollin - 20106951

https://github.com/IFT3275-Securite-Informatique/devoir-1-cryptographie-sus

QUESTION 1

Idee Attaque 1 : Si N est petit, On peut factoriser N pour retrouver :

- p et q
- et phi tq phi = (p-1)(q-1)
- et d = inverse(e,phi)
- et m = pow(c,d,n)

On utiliserait la même méthode que dans le tuto https://www.youtube.com/watch?v=WvqoKl_LI4I&list=PLX3dA7a5RDPYzS7WUiuLktRJXPqaFHk58&index=3

```
def attaque1(n, e, c):
    dictionnary = factorint(n)    #sort of brut force
    attaque2(dictionnary[0], dictionnary[1], n, e, c)
```

ici N est grand, alors probablement impossible a factoriser dans le temps alloué pour la remise de ce tp

Idée Attaque 2 : Est-ce que la paire p,q existe dans les databases publiques tel que https://factordb.com/ ou https://www.alpertron.com.ar/ECM.HTM ?

On utilise la même méthode que l'attaque 1, mais on utilise une database qui contient déjà les facteurs pour le N specifé. On trouve un résultat sur alpertron, les valeurs de p et q étaient dans la database, voir code plus bas.

```
def attaque2(p, q, n, e, c):
    phi = (p-1)*(q-1)
    d = modinv(e, phi)
    m = modular_pow(c, d, n)
    print M(m)
```

```
def printM(m):
    binary = bin(m)[2:]
    padded_binary = binary.zfill((len(binary) + 7) // 8 * 8)
#le padding est essentiel à la lecture du message
    bytes = [padded_binary[i:i+8] for i in range(0,
len(padded_binary), 8)]
    message=""
    for byte in bytes:
        value = int(str(byte),2)
        letter = chr(value)
        message+=letter
    print(message)
```

Clé publique Question 1.1

N =
14351633690928181552910415014721024800278971276108690005970534210322078267404628923208243578956328373980574557987343284668088987010788191642824141952083164817391248643164035000086097393530005608928615873757935780597701932955798545493414628255058294246363124569770299851118078700702913956193343355024269304792444038855098349869008076488293410183490802531486146872625342555433476014692353040392452337247768666875256728706020140746463094321823613242377263667518297758570759601601155691750475913144416024025273328296953409286968533893124120478575051974850543903980111976204979608571910659156221711567923658 3

e = 3

Cryptogramme 1.1

C = 1101510739796100601351050380607502904616643795400781908795311659278941419415375

Via la DB alpertron on trouve

p=
10715086071862673209484250490600018105614048117055336074437503883703510511249361224931983788156958581275946729175531468251871452856923140435984577574698574803934567774824230985421074605062371141877954182153046474983581941267398767559165543946077062914571196477686542167660429831652624386837205668073457

q=
13393857589828341511855313113250022632017560146319170093046879854629388139061701531164979735196198226594933411469414335314839316071153925544980721968373218504918209718530288731776343256327963927347442727691308093729477426584248459448956929932596328643213995597108177709575537289565780483547741631653719

Tel que `attaque2(p, q, N, e, C)` retourne "Umberto Eco"

Clé publique Question 1.2

N =
17221960429113817863492498017665229760334765531330428007164641052386493920885554707849892294747594048776689469584811941601703

67844129458299713889703424997977808694983717968420001033168722360067307143390485095229367172423195469582545920975539060699530956357494837243598213416944408434967474317474605697904676813343577310719430442085422937057220239881971046349315235043163226355302567726074269720408051461805113819456513196492192727498270702594217800502904761235711809203123842506621973488494670663483187137290546241477681096402483981619592515049062514180404818608764516997842633077157249806627735448350463

e = 173

Cryptogramme 1.2

C =
25782248377669919648522417068734999301629843637773352461224686415010617355125387994732992745416621651531340476546870510355165303752005023118034265203513423674356501046415839977013701924329378846764632894673783199644549307465659236628983151796254371046814548224159604302737470578495440769408253954186605567492864292071545926487199114612586510433943420051864924177673243381681206265372333749354089535394870714730204499162577825526329944896454450322256563485123081116679246715959621569603725379746870623049834475932535184196208270713675357873579469122917915887954980541308199688932248258654715380981800909

Avec facteurs issus de la db alpertron :

p=
10715086071862673209484250490600018105614048117055336074437503883703510511249361224931983788156958581275946729175531468251871452856923140435984577574698574803934567774824230985421074605062371141877954182153046474983581941267398767559165543946077062914571196477686542167660429831652624386837205668069673

q=
16072629107794009814226375735900027158421072175583004111656255825555265766874041837397975682235437871913920093763297202377807179285384710653976866362047862205901851662236346478131611907593556712816931273229569712475372911901098151338748315919115594371856794716529813251490644747478936580257043048672231

Tel que `attaque2(p, q, N, e, C)` retourne `"Marcel Proust"`

Pour rester de bonne foi et ne pas être pénalisé pour avoir trivialisé le problème, je vais implémenter une 3e attaque pour la Q1:

Attaque 3: small exponent attack (Source : https://www.youtube.com/watch?v=2QGDsDfNjWc&list=PLX3dA7a5RDPYzS7WUiuLktRJXPqaFHk58&index=6)

Si normalement, C = M^e % N mais que C=M^e < N, tq C=M^e, alors on peut simplement extraire le message en effectuant le nth root, qui fonctionne bien sur la Q1, car C est petit comparativement à N.

```
def attaque3(N, e, C): #small msg attack
    msg, reste = gmpy2.iroot(C, e) #deconstruction
    print M(msg)
```

Attaque 4: factorisation Fermat. Fonctionne en quelques itérations si p et q sont similaires et très proches de la racine de N. Ici notre script python teste 200 itérations avant d'abandonner. Fonctionne pour les valeurs de N qui sont peu sécuritaires, ne semble par s'appliquer à Q1.1 ou Q.1.2

```
def fermat_factorization(N):
    if N % 2 == 0:
        return [N // 2, 2]  # Fermat's factorization works
for odd numbers only
    #
    a = math.isqrt(N) + 1  # Start with the smallest
integer > sqrt(N)
    b2 = a * a - N         # Calculate b^2 = a^2 - N
    #
    it=0
    while not math.isqrt(b2) ** 2 == b2:  # Check if b2 is a
perfect square
        a += 1
        b2 = a * a - N
        it+=1
        if it>200:
            print("unlikely to factor")
            break
    #
    b = math.isqrt(b2)  # Now b is the integer square root
of b2
    return [a - b, a + b]  # Return factors (a - b) and (a +
b)

def attaque4(N,e,C):
   if(N%2==0):
      print("on ne peut pas utiliser cette attaque")
   tableau = fermat_factorization(N)
   p=tableau[0]
   q=tableau[1]
```

QUESTION 2

J'ignore s'il est nécessaire de discuter de la question 2 dans le rapport, mais ma stratégie est simple (et peu efficace, avec un score de 50%). On prend un grand volume de littérature française et on fait une analyse de fréquence des paires. On les trie en ordre descendant de fréquence et on les mappe directement avec la fréquence des entiers allant de 1 à 256 du ciphertext. On crée un pseudo plaintext qui a 50% de correspondance avec le vrai plaintext.

Je me suis interrogé sur la possibilité d'ajouter un 2e passage sur le pseudo ciphertext, pour analyser la présence de mots ou substituer une paire par une autre, mais le temps me manque et mon intuition ne suffit pas à trouver une solution de haute efficacité. Je n'ai pas vraiment de référence pour cette section, je me suis simplement inspiré du principe global d'analyse par fréquence, je peux joindre quelques githubs qui s'y sont attardé avec des analyses par cribs, notamment celui-ci : https://github.com/orgtre/frenchngrams/tree/master

Bilbiographie:

-Attaque par factorisation : https://www.youtube.com/watch?v=WvgoKl_LI4I&list=PLX3dA7a5RDPYzS7WUiuLktRJXPqaFHk58&index=3

-Attaque par factorisation de fermat : https://www.youtube.com/watch?v=-ShwJqAalOk

-Databases de primes https://factordb.com/ et https://www.alpertron.com.ar/ECM.HTM

-Analyse de fréquence en français https://github.com/orgtre/frenchngrams/tree/master