Emanuel Rollin - 20106951

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 bruteforce
    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)
    printM(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 =
1435163369092818155291041501472102480027897127610869000597053421032207826740462892320824357895632837398057455798734328466808898701078819164282414195208316481739124864316403500008609739353000560892861587375793578059770193295579854549341462825505829424636312456977029985111807870070291395619334335502426930479244403885509834986900807648829341018349080253148614687262534255543347601469235304039245233724776866687525672870602014074646309432182361324237726366751829775857075960160115569175047591314441602402527332829695340928696853389312412047857505197485054390398011197620497960857191065915622171156792365 83

e = 3

Cryptogramme 1.1

C = 110151073979610060135105038060750290461664379540078190879531165927894141941 5375

Via la DB alpertron on trouve

p=
107150860718626732094842504906000181056140481170553360744375038837035105112493612249319837881569585812759467291755314682518714528569231404359845775746985748039345677748242309854210746050623711418779541821530464749835819412673987675591655439460770629145711964776865421676604298316526243868372056680734 57

q=
133938575898283415118553131132500226320175601463191700930468798546293881390617015311649797351961982265949334114694143353148393160711539255449807219683732185049182097185302887317763432563279639273474427276913080937294774265842484594489569299325963286432139955971081777095755372895657804835477416316537 19

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

Clé publique Question 1.2

N =
17221960429113817863492498017665229760334765531330428007164641052386493920885554707849892294747594048776689469584811941601706784412945829971388970342499797780869498371796842000103316872236006730714339048509522936717242319546958254592097553906069953095635749483724359821341694440843496747431747460569790467681334357731071943044208542293705722023988197104634931523504316322635530256772607426972040805146180511381945651319649219272749827070259421780050290476123571180920312384250662197348849467066 34

83187137290546241477681096402483981619592515049062514180404818608764516997842633077157249806627735448350463

e = 173

Cryptogramme 1.2

C =
2578224837766991964852241706873499930162984363777335246122468641501061735512538799473299274541662165153134047654687051035516
5303752005023118034265203513423674356501046415839977013701924329378846764632894673783199644549307465659236628983151796254371
0468145482241596043027374705784954407694082539541866055674928642920715459264871991146125865104339434200518649241776732433816
8120626537233374935408953539487071473020449916257782552632994489645445032225656348512308111667924671595962156960372537974687
0623049834475932535184196208270713675357873579469122917915887954980541308199688932248258654715380981800909

Avec facteurs issus de la db alpertron :

p=
1071508607186267320948425049060001810561404811705533607443750388370351051124936122493198378815695858127594672917553146825187
1452856923140435984577574698574803934567774824230985421074605062371141877954182153046474983581941267398767559165543946077062
914571196477686542167660429831652624386837205668069673

q=
1607262910779400981422637573590002715842107217558300411165625582555526576687404183739797568223543787191392009376329720237780
7179285384710653976866362047862205901851662236346478131611907593556712816931273229569712475372911901098151338748315919115594
371856794716529813251490644747478936580257043048672231

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
    printM(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]
```

Bilbiographie:

-Attaque par factorisation : https://www.youtube.com/watch?

https://www.youtube.com/watch?v=WvqoKl_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