```python
from random import *

# ANNEXE :
#-----------------------------------------------------------------------------------------------
def pgcd(a, b):
    while b:
        a, b = b, a % b
    return a

def euclide_etendu(a, b):
    if b == 0:
        return a, 1, 0
    else:
        d, u, v = euclide_etendu(b, a % b)
        return d, v, u - (a // b) * v

def cle_e(phi_n):
    e = 2
    while pgcd(e, phi_n) != 1:
        e += 1
    return e

def cle_d(e, phi_n):
    u, d, v = euclide_etendu(e, phi_n)
    return d % phi_n
#-----------------------------------------------------------------------------------------------



# EXERCICE 1 :
# 1)

# chiffrement RSA avec les clés publiques (n, e) et un message claire m

def E(a, b, n):
    d = 1
    beta = bin(b)
    for i in range(2, len(beta)):
        d = (d**2) % n
        if beta[i] == "1":
            d = (d*a) % n
    return d

# dechiffrement RSA avec les clés privées (n, d) et un message chiffré c

def D(c, n, d):
    beta = bin(d)
    m = 1
    for i in range(2, len(beta)):
        m = (m**2) % n
        if beta[i] == "1":
            m = (m*c) % n
    return m


# 2)

# a) clé publique k = (8633,17) et le message en clair m = 1111

# i) longueur de n = 8633 = 4 octets = 32 bits

# ii)
```

```python
rep1 = E(1111, 17, 8633)
print("a) ii) ", rep1)

# iii) Vérifiez que la  clé k est bien construite et déduisez-en la clé privée k'
# 8633 = 89 * 97
# phi(8633) = (89-1)*(97-1) = 8640
# 17 * d = 8453
# d = 8453 / 17 = 497


# iv)
d = 497
rep2 = D(rep1, 8633, d)
print("a) iv) ", rep2)

# b) clé privée k' = (6557, 67) et le message chiffré c = 1234
# retrouvez le message en clair m
# k' = (n, d) = (6557, 67)
# k = (n, e) = (6557, ?)
# phi_n = (89 - 1) * (73 - 1) = 6336
# e * 67 ≡ 1 (mod 6336)
# e = 6337 / 67 = 95
e = 95
rep3 = D(1234, 6557, e)
print("b) ", rep3)


# EXERCICE 2 :
# 1)
def est_pseudo_premier(n, a):
    if n < 2:
        return 0
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return 0
    if pow(a, n-1, n) == 1:
        return 1
    else:
        return 0

# 2)
a = 2
def generer_nombre_premier(B):
    while True:
        n = randint(2**(B-1), 2**B - 1)

        if est_pseudo_premier(n, a):
            return n

# 3)
B = 1000
n = generer_nombre_premier(B)
print("n = ", n)
# la probabilité est de  1 / (1000 * ln(2))



# EXERCICE 3 :
# 1)
def Cles_RSA(B):
    p = generer_nombre_premier(B)
    q = generer_nombre_premier(B)
```

```python
    while p == q:
        q = generer_nombre_premier(B)

    N = p * q
    phi_N = (p - 1) * (q - 1)

    e = cle_e(phi_N)

    d = cle_d(e, phi_N)

    cle_publique = (N, e)
    cle_privee = (N, d)

    return cle_publique, cle_privee


# 2)
B = 32
cles_publique, cles_privee = Cles_RSA(B)
print("Clé publique (N, e):", cles_publique)
print("Clé privée (N, d):", cles_privee)

message = 111222333444555666777888999

# Chiffrer le message
N, e = cles_publique
rep4 = E(message, e, N)
print("Message clair:", message)
print("Message chiffré:", rep4)

# Déchiffrer le message
N, d = cles_privee
rep5 = D(rep4, N, d)
print("Message déchiffré:", rep5)

# pour 1000 bits c'est un peu long car il faut générer 2 nombres premiers de 1000 bits chacun
#  ce qui est très long
```