Cryptographie - EasyRSA - 100 points

Kévin DUVERGER

Table des matières

1 Résolution EasyRSA :

Dans ce challenge, 3 fichiers nous sont donnés :

- `CheckMyRoots.txt`
- `NotPrimes.txt`
- `SharedPrimes.txt`

Nous allons les résoudre un par un !

1.1 Check my roots :

Une première idée pour ce challenge pourrait être de tester si les premiers p et q sont proches. Vous pouvez le tenter mais vous verrez que vous n'aboutirez pas à grand chose. L'idée est plus de checker les racines du chiffré c, pour voir si l'on ne peut pas retrouver le message (car le chiffré dans RSA est obtenu avec $c = m^e \pmod n$).
La solution est donc de tenter de calculer la racine 17-ième de c (car $e = 17$) :

```
inputValueMin, inputValueMax = 1, 10 ** 40
while inputValueMax - inputValueMin > 10 :
    currentValue = (inputValueMin + inputValueMax) // 2
    temp = currentValue ** e
    if temp > c :
        inputValueMax = currentValue
    else :
        inputValueMin = currentValue
plaintext = 0
for i in range(inputValueMin, inputValueMax + 1) :
    if i ** e == c :
        plaintext = i
        break
```

L'idée derrière ce code est tout d'abord de faire une recherche par dichotomie, puis une fois un intervalle suffisamment petit trouvé, finir avec une recherche tout simple. Pour avoir le "vrai" texte du clair, il faut ensuite faire :

```
plaintext.to_bytes(7, byteorder="big")
```

1.2 Not primes :

L'idée ici est que les nombres p et q qui devraient être des nombres premiers ne le sont en fait pas (et en réalité leurs facteurs premiers ne sont que des petits premiers). Voici le code de factorisation :

```
def factor(value, smallPrimes) :
    factorList = [ 0 ] * len(smallPrimes)
    smallPrimeIndex = 0
    while value > 1 :
        while value % smallPrimes[smallPrimeIndex] == 0 :
            factorList[smallPrimeIndex] += 1
            value //= smallPrimes[smallPrimeIndex]
        smallPrimeIndex += 1
    return [ [ smallPrimes[i], factorList[i] ] for i in range(0, len(smallPrimes)) ]

smallPrimes = [ 2, 3, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43 ]
factorisation = factor(n, smallPrimes)
```

En ayant cette factorisation, nous pouvons maintenant calculer $\varphi(n)$:

```
def computePhi(factorisation) :
    toReturn = 1
    for i in range(0, len(factorisation)) :
        if factorisation[i][1] > 0 :
            toReturn *= (factorisation[i][0] - 1) * (factorisation[i][0] ** (factorisation[i][1] -
    1))
    return toReturn

phiValue = computePhi(factorisation)
```

Grâce à $\varphi(n)$, nous pouvons maintenant calculer $d = e^{-1} \pmod{\varphi(n)}$:

```
d = pow(e, -1, phiValue)
```

Et grâce à l'obtention de la clé secrète, nous pouvons obtenir le message :

```
plaintext = pow(c, d, n)
print(plaintext.to_bytes(8, byteorder="big"))
```

1.3 Shared primes :

Pour ce dernier fichier, on peut penser au fait que n et n' ont un premier partagé ce qui va nous permettre de les factoriser de trouver la clé secrète dans les 2 cas ! Pour trouver ce facteur commun, il faut faire le PGCD(n, n').
Voici le code de solution :

```
p = math.gcd(n, nPrime)
q = n // p
qPrime = nPrime // p
phiValue = (p - 1) * (q - 1)
phiValuePrime = (p - 1) * (qPrime - 1)
d = pow(e, -1, phiValue)
dPrime = pow(e, -1, phiValuePrime)
plaintext1 = pow(c, d, n)
plaintext2 = pow(cPrime, dPrime, nPrime)
print(plaintext1.to_bytes(9, byteorder="big"))
print(plaintext2.to_bytes(9, byteorder="big"))
```