# Lista 06

Vinícius de Oliveira Peixoto Rodrigues (245294)

Setembro de 2022

## Questão 1

A expansão ocorre porque, no RSA, o texto cifrado é calculado a partir da chave pública $(e, n)$ como:

$$c = m^e \mod n$$

Normalmente, o valor de $e$ é um número primo relativamente grande (costuma ser $2^{16} + 1$); isso significa que valores de $m$ pequenos (por exemplo, de 1 byte) têm chance de resultar em valores enormes de $c$ (possivelmente da ordem de $n$, que costuma ter centenas de bytes).

Essa expansão pode ser mitigada maximizando o tamanho de "bloco"do plaintext, isto é, cifrando chunks de tamanho comparável ao do valor $n$ (por exemplo, cifrando blocos de 128 bytes para $n$ com 1024 bits).

Para evitar que o arquivo decifrado aumente de tamanho, basta eliminar o padding gerado pela decifração (por exemplo, um caractere que era originalmente de 1 byte no RSA-1024, ao ser decifrado, vai ser um inteiro de 128 bytes, mas basta descartar o "excesso").

## Questão 2

Observando que $35 = 7 \cdot 5 \Rightarrow \phi(35) = (5-1)(7-1) = 24$, temos pelo teorema de Euler que se $\gcd(a, 35) = 1$:

$$a^{24} \equiv 1 \mod 35$$

Como $123 = 3 \cdot 41 \Rightarrow \gcd(123, 35) = 1$ e $145 = 6 \cdot 24 + 1$:

$$123^{145} \equiv \underbrace{\left(123^{24}\right)^6}_{\equiv 1 \mod 35} \cdot 123 \mod 35 \equiv 123^1 \mod 35 \equiv 18 \mod 35$$

## Questão 3

O valor 2 não seria uma escolha válida porque, por definição, para garantir a unicidade do inverso multiplicativo da chave pública ($ed \equiv \mod \phi(n)$) é condição necessária que $\gcd(e, \phi(n)) = 1$ (senão a chave pública $e$ não teria inverso único). Como $\phi(n) = (p-1)(q-1)$ é necessariamente par (visto que $p$ e $q$ são primos ímpares), a condição de unicidade do inverso não é satisfeita, de modo que a cifração deixa de ser injetiva (e por tanto se torna não-invertível).

## Questão 4

### 1.

A única forma de determinar com certeza absoluta é por meio de uma checagem direta, testando os divisores até $\sqrt{n}$. É comum que seja usado um algoritmo como o crivo de Eratóstenes para gerar uma "cache"de primos conhecidos até um certo valor.

### 2. e 3.

A checagem direta tem complexidade $O(\sqrt{n})$, enquanto o crivo de Eratóstenes tem complexidade temporal (em pior caso) de $O(n \log \log n)$ e espacial de $O(n)$. Esse resultaos tornam o método inviável para números muito grandes (como os usados no RSA).

## Questão 5

Não, pela mesma razão mencionada na questão 3: um número de chave pública par faria com que $\gcd(e, \phi(n)) \neq 1$, de modo que a função $f : m \to m^e \mod n$ deixaria de ser invertível.

## Questão 6

A chance de o número ser primo é dada por $1 - 0.25^n$, para $n$ testes que resultam em "inconclusivo". Dessa forma, queremos que $1 - 0.25^n = 0.99999 \Rightarrow n = log_{0.25}(1 - 0.99999) \Rightarrow n \approx 8.3$, de modo que bastam $\boxed{\text{9 testes}}$.

## Questão 7

Há $2^{1024}$ números de 1024 bits, indo de $2^{1023}$ a $2^{1024}-1$; a distância "média"esperada entre os primos nesse intervalo seria $\left(\ln(2^{1023}) + \ln(2^{1024})\right)/2 \approx 709.5$.

Dessa forma, é razoável esperar encontrar um número primo após testar em torno de 710 números consecutivos.

## Questão 8

Pelo pequeno teorema de Fermat:

$$a^p \equiv a \mod p \Rightarrow a^{p-2} \equiv a^{-1} \mod p$$

De modo que é possível encontrar o inverso multiplicativo calculando $a^{p-2} \mod p$. Para $a = 4$, $p = 11$, temos que

$$4^{-1} \equiv 4^9 \mod 11 \equiv 262144 \mod 11 = \boxed{3}$$

É fácil verificar observando que $4 \cdot 3 = 12 \equiv 1 \mod 11$.

## Questão 9

O programa em Python abaixo implementa o square-and-multiply:

```
from math import log, ceil

def sq_mult(m, e, n):
    s = ceil(log(e, 2))
    t = 1
    for i in reversed(range(0, s)):
        t *= t
        t %= n
        print(f"step {i}: squaring")
        if e & (1 << i):
            t *= m
            t %= n
            print(f"step {i}: multiplying")
    return t

def main():
    m = 42
    print("m = {m}, key = (31, 35)")
    sq_mult(m, 31, 35)
    print("-------------")
    print("m = {m}, key = (33, 35)")
    sq_mult(m, 33, 35)

if __name__ == "__main__":
    main()
```

No primeiro caso, foram realizadas 10 operações (5 squares, 5 multiplies); no segundo, apenas 8 (6 squares, 2 multiplies). Isso acontece porque há tantas operações de multiply quanto há bits 1 na chave pública $e$; como `31 = 0b11111` e `33 = 0b100001`, o volume de cálculos é menor (comparativamente ao tamanho de bits da chave pública) para a segunda.