Criptografia: Parte 11 (Método RSA-4)

Mário Leite

...

Continuando sobre Criptografia RSA, um dos passos mais importantes deste método criptográfico é a formação da chave pública, que é a responsável pela codificação (encriptação) da mensagem. E um dos dois componentes desta chave é o parâmetro **e** que deve ser um dos coprimos da função de **Euler ϕ(n)** onde **n**, que no caso é **391 = 17*23**. Esta função, também conhecida como Função Totiente ou **Função ϕ()** - lê-se função **Fi** - de um número natural **n** é **ϕ(n)=y**; onde **y** é um número natural que representa a quantidade de números inteiros **xi [i=1,y-1]**, tal que **MDC(xi,y)=1**. Isto quer dizer que **y** é a quantidade de coprimos de **n** no intervalo **1<=i<y**. Por exemplo, para **n=9** tem-se **ϕ(9)=6** pois, de 1 até (9-1) existem os seguintes seis números coprimos de 9: **1 2 4 5 7 8**; e isto significa que:

- MDC(1,9) = 1
- MDC(2,9) = 1
- MDC(4,9) = 1
- MDC(5,9) = 1
- MDC(7,9) = 1
- MDC(8,9) = 1

O número **3** é um dos 352 coprimos de 391 [**ϕ(391)=352**]; e este número é que vai ser usado como componente da chave pública no exemplo a ser considerado aqui.
Entretanto, a maior dificuldade para definir a chave pública não é calcular **ϕ(p*q)**, e sim lidar com cálculos matemáticos exponenciais e modulares com números grandes. Na Parte III ficou destacado o cálculo de decriptação da letra **"E"**, do seguinte modo:

C5 389: (389^{235} mod 391)=($4.346666...^{608}$ mod 391) = 145

O resultado de **389^{235}** é um número assustador: **389^{235} =**
4346666415102061535075914826674517038967224543550104914409824154776502454l
80027962771411626354526318189023306753234478456909195243554904048711638323
71899033137592663296504819620664861151798108643255486737561790295320478414
41300866542280697181581568604592700008079489179994358740238839445215891833
29603891187465800380317075590247361057730368182688142078118124931754391236
62782700678903227042458153034065127151924987080418119236453179409356587395
6446582167304448985590049654037208022957837762478940710879305723358078522
84698056160571360623366969313046528093386844743568513564362796834226950956
93126825945757149

E mesmo que seja apenas um cálculo parcial dentro de uma expressão maior, como é o caso, isto pode se tornar inviável e até mesmo impossível de ser computado. Agora, imagine se fosse dois números muito grandes, como exige o método! Então, o tratamento automatizado deve ser muito bem planejado para diminuir o esforço computacional envolvido. Deste modo, a questão a ser discutida é a seguinte: *COMO TRABALHAR COM NÚMEROS TÃO GRANDES*?!
A resposta está na aritmética modular; um ramo da matemática tratado na Teoria dos Números. Entretanto, a literatura sobre este assunto mostra algumas propriedades que permitem calcular potências de números grandes sem muitas dificuldades. O programa **"PotMod"** (codificado em Python) mostra uma solução bem simples para efetuar o cálculo modular (**a^x mod n**) para **x** muito grande. E embora o tempo de processamento possa ser elevado para valores muito grandes, o *loop* utilizado resolve satisfatoriamente o problema. A **figura RSAIV.1** apresenta o código-fonte do programa e a **figura RSAIV.2** a saída para o cálculo de **389^{235} mod 391**, mostrando que o tempo necessário para fazer este cálculo modular não chegou nem a 1 segundo.

Continua com o "Método RSA-5 Parte 12)

```
#Programa "PotMod"
#Faz cálculo modular com potência de números grandes.
#Implementado em Python 3.6
#Autor: Mário Leite
#marleite@gmail.com
#------------------------------------------------------------
#Início
import time
endwhile = "endwhile"  #cria um falso terminador do loop while
x = 235
n = 391
a = 389
R = 1
j = 1
inicio = time.time()  #marca o início do loop
while(j <= x):
    R = (R * a) % n
    j = j + 1
endwhile
fim = time.time()  #marca o fim do loop
print("Tempo de processamento do loop: {} seg".format(fim-inicio))
print("Resultado de (389^235 mod 391): {}".format(R))
#FimPrograma------------------------------------------------------
```

Aqui onde faz todo o trabalho “pesado” do cálculo modular.

Figura RSAIV.1 - Código em Python para calcular 389^{235}

```
Tempo de processamento do loop: 0.0 seg
Resultado de (389^235 mod 391): 145

Process finished exit code 0
```

Figura RSAIV.2 - Saída do programa "PotMod"