Criptografia: Parte 12 (Método RSA-5)

Mário Leite

...

As partes **10** e **11** mostraram o algoritmo do método de criptografia RSA, baseado em chaves assimétricas. Este é um sistema de proteção de mensagens muito eficiente e empregado na maioria dos casos em que há necessidade de proteger dados e informações sigilosos. E conforme foi explicado, a segurança do Método RSA está na dificuldade do pretenso criptoanalista em fatorar um número **n** muito grande para descobrir **p** e **q** e depois o parâmetro **d** da chave privada (**d**,**n**) para tentar a decodificação da mensagem; mas, isto poderia levar até milhões de anos para ser conseguido, conforme já explicado. Este é o principal argumento a favor da segurança deste método de criptografia. Além do mais, a matemática requerida para calcular **d** é não trivial pois, envolve álgebra modular e congruências do tipo **a ≡ b (mod n)** e o cálculo de **d** em expressões do tipo ***inverso*[(a mod b)].** Entretanto, este rigor matemático pode ser driblado pelo programador com um pouco de raciocínio lógico através de simples *loop* que resolve satisfatoriamente o problema (como foi mostrado no programa "PotMod", apresentado na **Parte 11**). A função ***FunChavePri()*** do programa **"ProgCriptoRSA"** (referenciado mais abaixo) usa este código para resolver o problema de maneira bem simples e sem sofisticações. Neste programa acontece algo semelhante nas expressões requeridas do tipo $\mathbf{a^x \bmod n}$ (para **x** muito grande).

O programa **"ProgCriptoRSA"** é uma solução simples, porém, bem didática e correta de como *encriptar/decriptar* um texto através do método RSA. E por questões de praticidade, em vez de usar o sistema ASCII para definir os códigos dos caracteres da mensagem a ser encriptada, optou-se por numerar os caracteres de acordo com a **tabela RSAI.1** (mostrada no início deste método, na **Parte 08**), considerando só letras maiúsculas além do espaço cujo valor convencionado foi **55**. Isto torna as operações de potenciação mais "leves" nos cálculos e no processo computacional de um modo geral. O programa foi escrito modularmente com sete rotinas, além do programa principal:

- Função **FunCalcMDC(N1,N2)**: retorna o máximo divisor comum de dois números.
- Função **FunVerifPrimo(N)**: recebe um número e verifica se é primo, retornando *Verdadeiro* se for primo ou *Falso* se não for.
- Procedimento **ProPreCodifTexto(texto)**: faz a pré-codificação da mensagem de acordo com a tabela de caracteres.
- Procedimento **ProCodifTexto(c,n)**: recebe dois números inteiros (***c*** e ***n***) e gera o texto codificado no vetor VetCodif[ ].
- Função **FunChavePub(n)**: recebe um valor (**n=p*q**) e retorna o elemento ***c*** da chave púbica (**c**,**n**).
- Função **FunChavePri(c,y)**: recebe dois números inteiros [***c*** e **y=(p-1)*(q-1)**] e retorna o elemento ***d*** da chave privada (**d**,**n**).
- Procedimento **ProDeCodifTexto(d,n)**: recebe dois números inteiros (***d*** e ***n***) e gera o texto decodificado no vetor VetTexto[ ].

As **figuras RSAV.1**, **RSAV.2** e **RSAV.3** mostram um exemplo de encriptação/decriptação de uma mensagem,

---

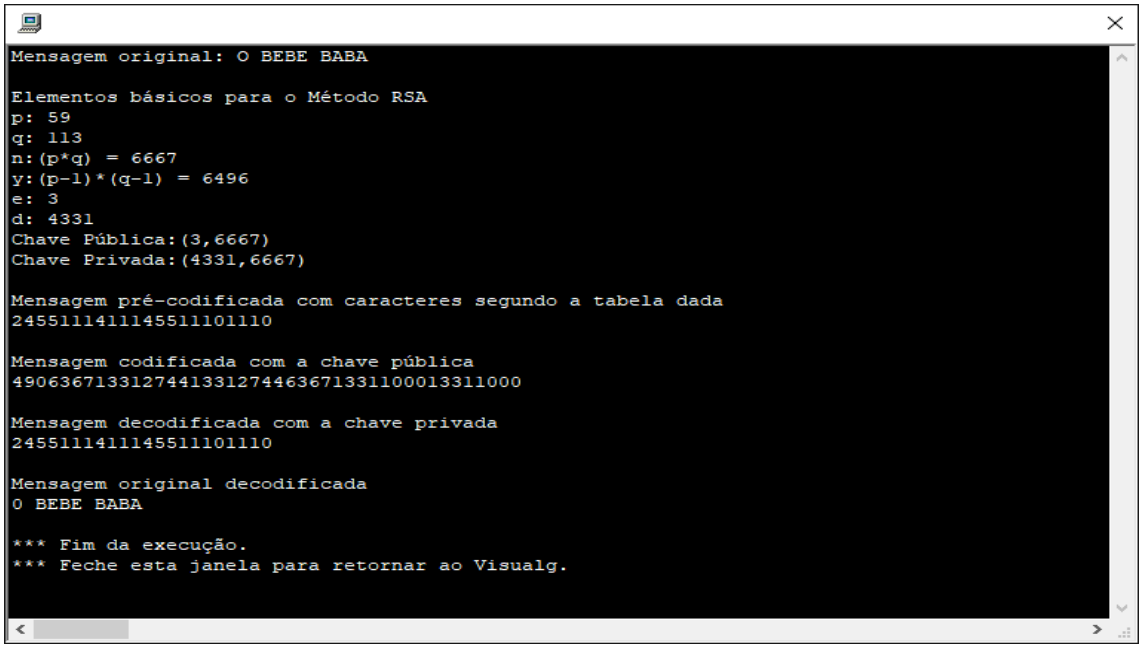


Figura RSAV.1 - Exemplo de criptografia com o programa "ProgCriptoRSA"

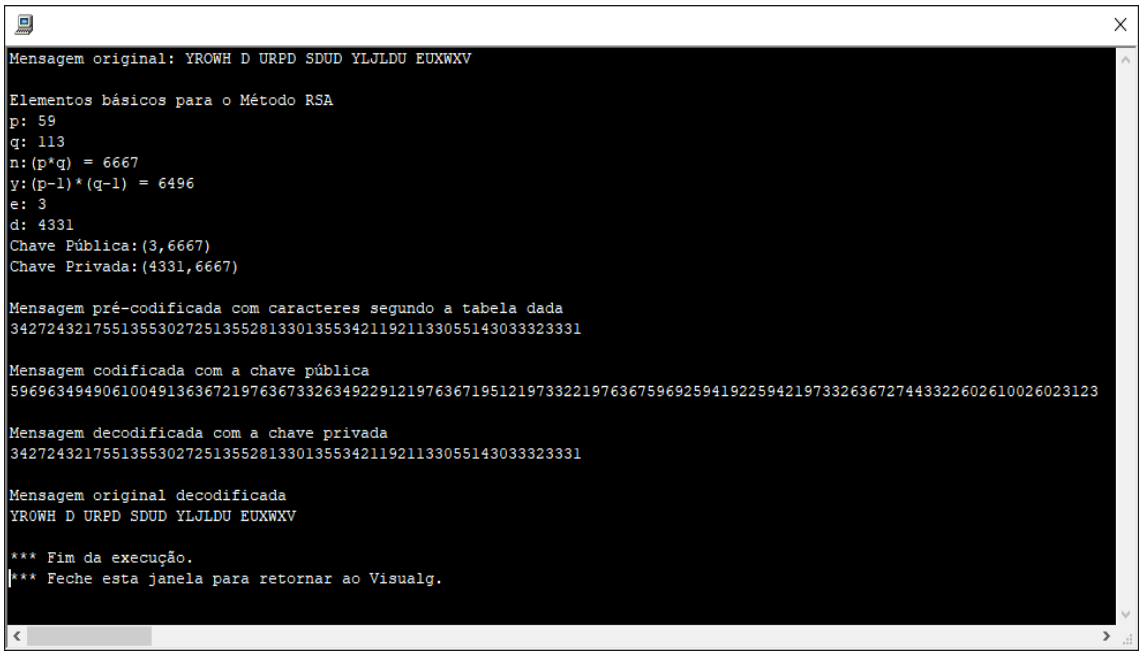


Figura RSAV.2 - Exemplo de criptografia com o programa "ProgCriptoRSA"

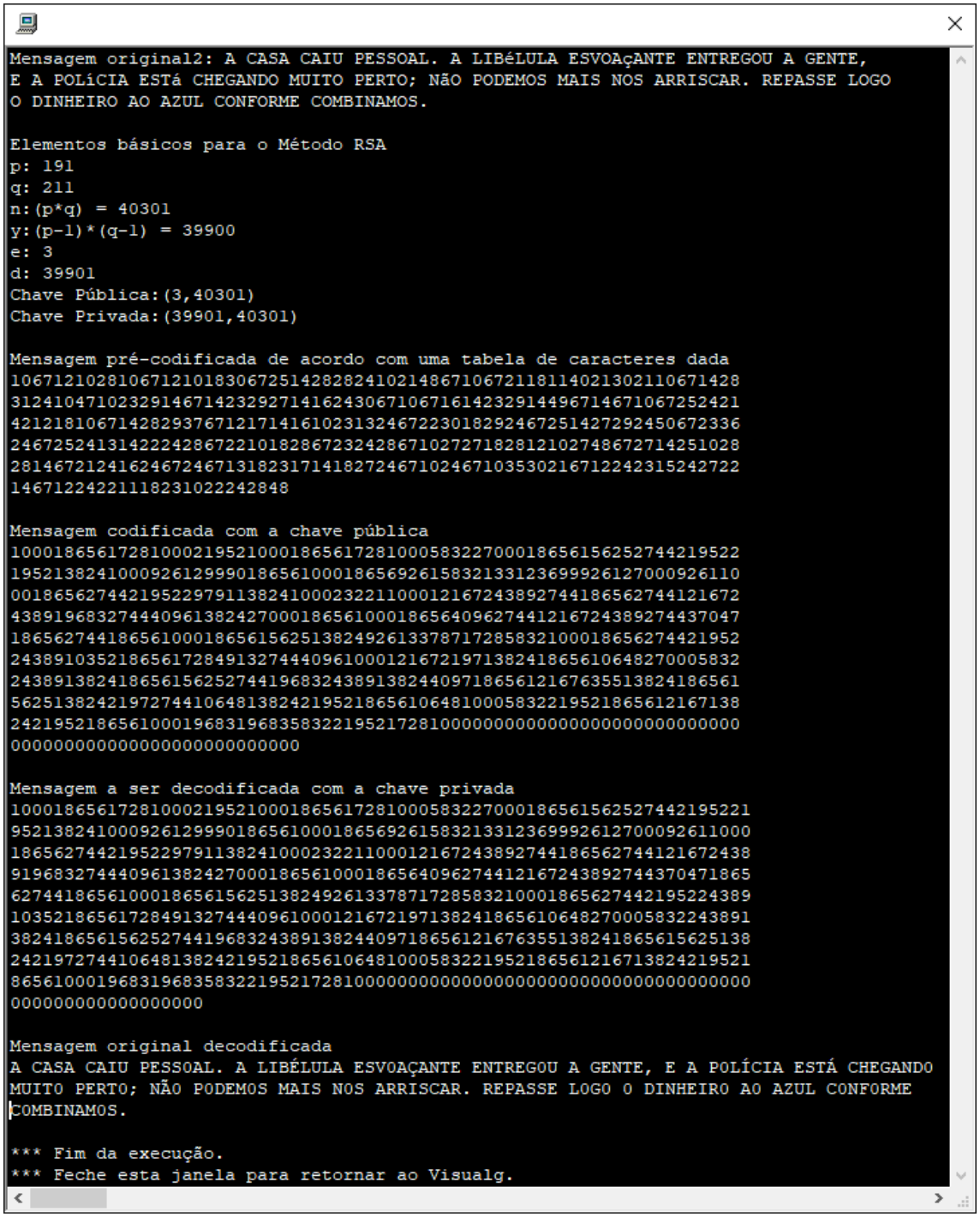
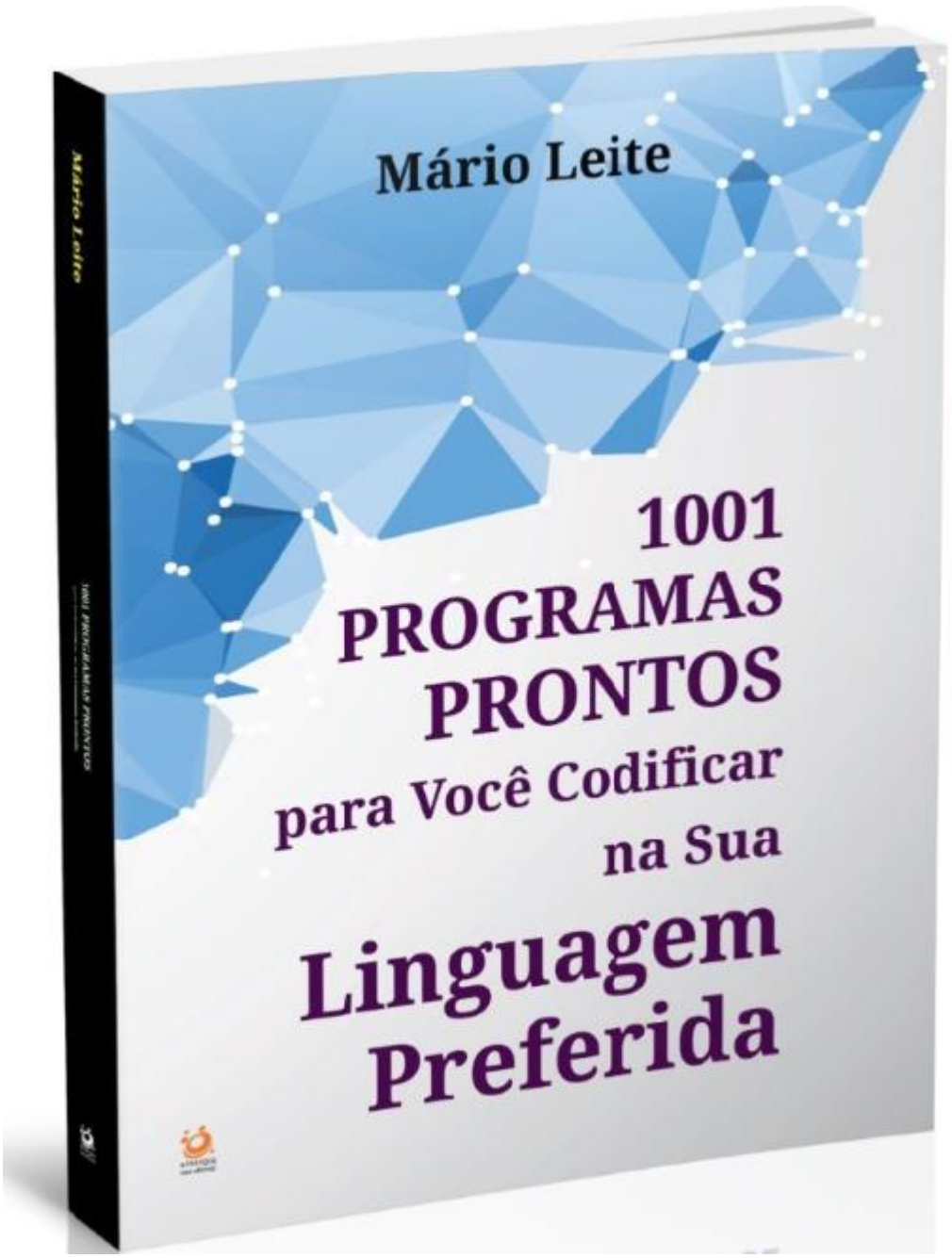


Figura RSAV.3 - Exemplo de criptografia com o programa "ProgCriptoRSA"

Nota: O código-fonte completo (em pseudocódigo) do programa "ProgCriptoRSA" e de mais outros métodos de criptografia, estão no meu livro ***"1001 Programas Prontos Para Você Codificar Na Sua Linguagem Preferida"***, que pode ser adquirido nas Livrarias *Amazon e Clube de Autores* (em *e-book*) ou em *pdf*, diretamente com o autor pelo *e-mail*: marleite@gmail.com; neste caso, o código-fonte (em Visualg) também será disponibilizado.


Mário Leite
1001
PROGRAMAS
PRONTOS
para Você Codificar
na Sua
Linguagem
Preferida