

Projeto e Análise de Algoritmos

Prof. Diego Buchinger
diego.buchinger@outlook.com
diego.buchinger@udesc.br

Prof. Cristiano Damiani Vasconcellos
cristiano.vasconcellos@udesc.br

Um pouco de Teoria dos Números e seu uso na Criptografia

Teoria dos Números

Notação: $d \mid a \rightarrow d$ “divide” a

sendo que $d \geq 1$ e $d \leq |a|$

- Todo inteiro a é divisível pelos **divisores triviais** 1 e a
- Número primo: únicos divisores são 1 e a
- Todo número composto pode representado pela multiplicação de seus fatores primos ($42 = 2*3*7$)
- Qual a complexidade de tempo para descobrir se um número a qualquer (**int**) é primo?

Divisores Comuns

Um número é dito divisor comum se ele divide dois números:

$$d \mid a \text{ e } d \mid b \Rightarrow d \text{ é divisor comum de } a \text{ e } b$$

Propriedade dos divisores comuns:

$$a \mid b \quad \text{implica em} \quad |a| \leq |b|$$

$$a \mid b \text{ e } b \mid a \quad \text{implica em} \quad a = b$$

$$d \mid a \text{ e } d \mid b \quad \text{implica em} \quad d \mid (ax + by)$$

O máximo divisor comum entre dois números
 a e b é denotado por: $mdc(a, b)$

$$d \mid a \text{ e } d \mid b \quad \text{então} \quad d \mid mdc(a, b)$$

Divisores Comuns

Primos relativos ou Primos entre si:

Dois inteiros são chamados de primos relativos se o único inteiro positivo que divide os dois é 1: $\text{mdc}(a, b) = 1$.

Por exemplo, 49 e 15 são primos relativos:

$$49 \rightarrow 1, 7, 49$$

$$15 \rightarrow 1, 3, 5, 15$$

Propriedade

se $\text{mdc}(a, p) = 1$ e $\text{mdc}(b, p) = 1$ então $\text{mdc}(ab, p) = 1$

Testar: $a=49$, $b=15$, $p=7$

Fatoração

Fatoração Única

Um inteiro pode ser escrito como um produto da forma:

$$a = p_1^{e_1} \times p_2^{e_2} \times \dots \times p_r^{e_r}$$

Exemplo

$$6.000 = 2^4 \times 3^1 \times 5^3 = 16 \times 3 \times 125$$

Teste de primalidade: Dado um número n , determinar se n é primo (“fácil”)

Fatoração de inteiros: Dado um número n , representar n através de seus fatores primos (difícil – até o momento)

Divisores Comuns

Algoritmo de Euclides

EUCLID (a , b)

se $b = 0$

então retorne a

senão retorne EUCLID(b , $a \bmod b$)

Calcular:

EUCLID(2 , 0)

EUCLID(99 , 78)

Complexidade:

Números pequenos: $O(\log b)$

Números grandes (k bits): $O(k)$ (*mod)

Divisores Comuns

Algoritmo de Euclides Extendido

Adaptar o algoritmo anterior para calcular x e y em:

$$d = \text{mdc}(a, b) = ax + by$$

EXT-EUCLID (a , b)

se $b = 0$

então retorne (a , 1 , 0)

$(d', x', y') = \text{EXT-EUCLID}(b , a \bmod b)$

$(d , x , y) = (d' , y' , x' - a / b * y')$

retorne (d , x , y)

Calcular:

EXT-EUCLID(4 , 0)

EXT-EUCLID(99 , 78)

Divisão inteira

Aritmética Modular

É um sistema para manipular faixas restritas de números inteiros.

Relação de congruência:

$a \equiv b \pmod{n}$ se e somente se $a \bmod n = b \bmod n$.

$a \equiv b \pmod{n} \Leftrightarrow n \text{ divide } (a - b)$.

Exemplos:

$38 \equiv 14 \pmod{12}$, $38 \bmod 12 = 14 \bmod 12$

$-10 \equiv 38 \pmod{12}$, $-10 \bmod 12 = 38 \bmod 12$

Aritmética Modular

Soluções para a equação $ax \equiv b \pmod{n}$

Só há solução se: $\text{mdc}(a, n) \mid b$

$ax \equiv b \pmod{n}$ tem d soluções distintas

onde $d = \text{mdc}(a, n)$

MOD-LIN-SOLVER(a, b, n)

$(d, x', y') = \text{EXT-EUCLID}(a, n)$

se $d \mid b$

então $x_0 = x'(b/d) \pmod{n}$

para $i=0$ a $d-1$ faça

imprimir($x_0 + i(n/d) \pmod{n}$)

senão imprimir “nenhuma solução”

Calcular:

$14x \equiv 30 \pmod{100}$

MOD-LIN-SOLVER (14 , 30 , 100)

Inverso multiplicativo modular:

O inverso multiplicativo modular de um inteiro a no módulo m é um inteiro x tal que:

$$ax \equiv 1 \pmod{m}$$

* Existe se e somente se a e m são primos relativos.

$$17x \equiv 1 \pmod{120}$$

MOD-LIN-SOLVER (17 , 1 , 120)

Teorema de Fermat:

Se p é primo então:

$$a^{p-1} \equiv 1 \pmod{p}$$

Contudo, a^{p-1} pode ser um número relativamente grande, e realizar a operação de módulo pode ser um problema.

Calcular:

$$a=2 \quad / \quad p=5$$

$$a=5 \quad / \quad p=3$$

$$a=4 \quad / \quad p=7$$

Exponenciação Modular:

Realizar a operação de elevação ao quadrado repetida e realizar o módulo sempre possível.

Exemplo: $2^{53} \bmod 101$

O Caráter Primo

Densidade de números primos

Distribuição dos primos: $\pi(n) \rightarrow n^{\circ} \text{ de primos} \leq n$

Exemplo: $\pi(10) = 4 \rightarrow \{2, 3, 5, 7\}$

Teorema dos números primos: $\lim_{n \rightarrow \infty} \frac{\pi(n)}{n / \ln n} = 1$

Para n grande, $n / \ln n$ é uma boa aproximação para $\pi(n)$!

O Caráter Primo

Densidade de números primos

Com base no teorema apresentado podemos fazer uma estimativa de probabilidade para verificar se um número escolhido ao acaso é primo ou não como: $1 / \ln(n)$

Quantos números de 512 bits precisamos testar, em média, até encontrar um número primo?

$$\ln(2^{512}) \approx 355 \text{ números}$$

$$1 / 355 \approx 0,28\% \text{ (chance de encontrar um primo de } 1^{\text{a}})$$

O Caráter Primo

Densidade de números primos

Para testarmos o caráter primo de um número pequeno n , podemos testar verificando a divisibilidade por todos os números entre 2 e \sqrt{n}

Para inteiros pequenos: $\Theta(\sqrt{n})$

Para inteiros grandes com k bits: $\Theta(2^k)$

O crivo de eratóstenes (algoritmo) é um dos mais conhecidos para criar uma lista de primos até um dado n

(ver gif: https://pt.wikipedia.org/wiki/Crivo_de_Erat%C3%B3stenes)

Teste do caráter pseudoprimo

Considerando novamente a equação modular:

$$a^{n-1} \equiv 1 \pmod{n}$$

O teorema de Fermat nos diz que se n é primo, então n satisfaz esta equação para qualquer escolha de a ($a \in \mathbb{Z}_n^+$)

Se encontrarmos um a que não satisfaça a equação, então certamente n não é primo

Teste do caráter pseudoprimo

Ao testarmos se: $2^{n-1} \equiv 1 \pmod{n}$

caso falso: n certamente não é primo

caso verdade: ou n é primo, ou n é pseudoprimo de base 2

Mas, com que frequência há um falso positivo?

Raramente! Existem apenas 22 valores menores que 10.000: {341, 561, 645, 1105, ...}

Usando 512 bits a chance é de $1 / 10^{20}$

Usando 1024 bits a chance é de $1 / 10^{41}$

Teste do caráter pseudoprimo

Teste Aleatório do Caráter primo de Miller-Rabin

- ❖ Experimentar diversos valores como base:

Melhora a confiabilidade, mas existem números “traíçoeiros” e extremamente raros que dão falso positivo para diferentes bases (números de Carmichael)

- ❖ Observar raiz quadrada não trivial de 1 módulo n:

$x^2 \equiv 1 \pmod{n}$ e x não é 1 ou -1 (ex: $x=6, n=35$)

MILLER-RABIN(n, s)

para $j=1$ a s

$a = \text{RANDOM}()$

se (WITNESS(a, n))

então retorne falso

retorne verdade

[número composto, certamente]

[número quase certamente primo]

Criptografia RSA

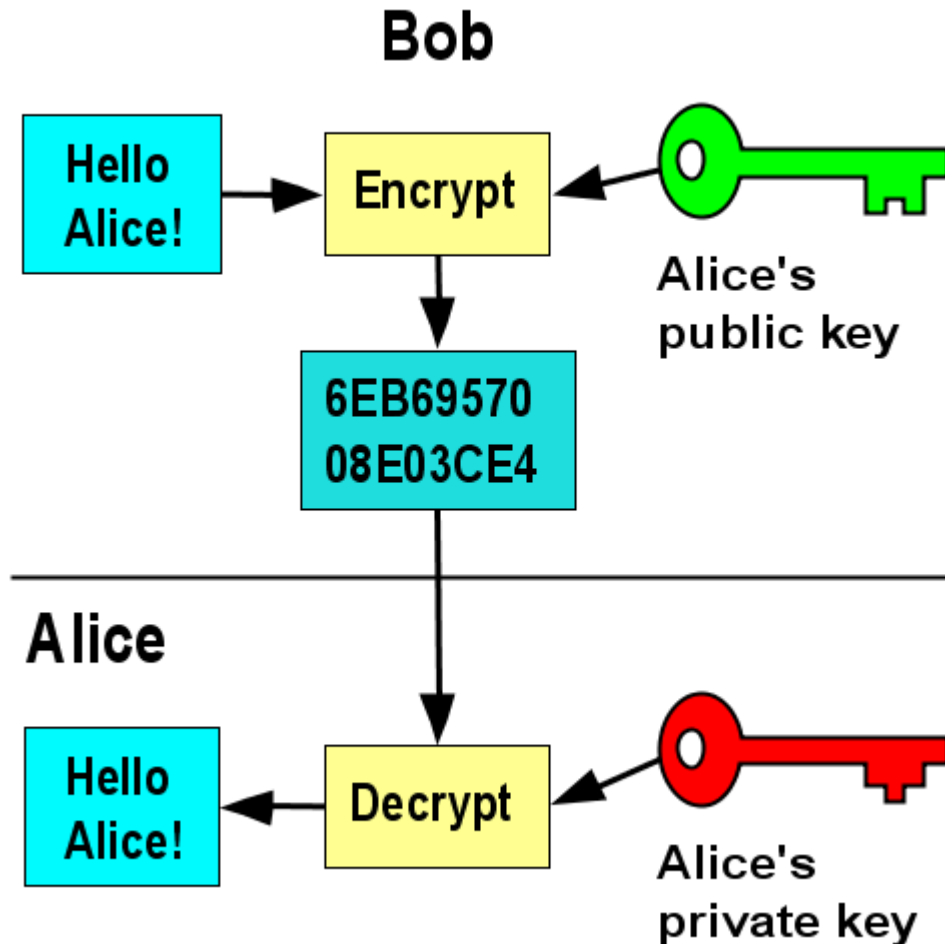
Ideia: Permitir que comunicação entre dois participantes sem que um intruso possa entender as mensagens trocadas.

Baseia-se na facilidade em se encontrar números primos grandes e na dificuldade em fatorar o produto entre dois números primos grandes.

Em um sistema de criptografia de chave pública, cada participante possui:

- + uma **chave pública** (pública);
- + uma **chave privada** (secreta);

Criptografia RSA



Como funciona:

- Bob obtém a chave pública de Alice;
- Bob usa a chave para codificar a mensagem M :
 $C = P_A(M)$ e envia C ;
- Alice recebe C e utiliza sua chave privada para recuperar a mensagem original M :
 $M = S_A(C)$

Algoritmo:

- Selecionar dois números primos grandes p e q , (512 bits, cada por exemplo) sendo $p \neq q$
- Calcular: $n = p * q$
- Selecionar um inteiro ímpar “pequeno” e tal que e seja primo relativo de $(p - 1)(q - 1)$ [número primo]:

$$\text{mdc}(e, (p - 1)(q - 1)) = 1$$

$$\textbf{Chave pública} = (e, n)$$

Criptografia RSA

Algoritmo:

- Calcular d como o inverso modular de e :

$$e * d \equiv 1 \text{ mod } ((p-1) (q-1))$$

Chave privada = (d, n)

Criptografia RSA

Transforma um inteiro M (que representa um bloco de dados da mensagem) em um inteiro C (que representa um bloco da mensagem criptografada), usando a seguinte função:

$$C = M^e \bmod n$$

Criptografia RSA

A transformação da mensagem criptografada C na mensagem original é executada através da formula:

$$M = C^d \bmod n$$

Cuidado!!

n deve ser maior do que M !

Caso contrário existem múltiplas interpretações para a mensagem codificada.

Se M for maior do que n , deve-se dividir a mensagem em blocos

Criptografia RSA

(Exemplo)

Mensagem: “ola” \Rightarrow 111 105 97 \Rightarrow 01101111 01101001 01100001

A princípio nossa mensagem M teria o valor decimal: 7.301.473

Vamos utilizar **p = 521** e **q = 383**,

$$\text{logo } n = p * q = 199.543 \quad \text{e} \quad (p-1) * (q-1) = 198.640$$

Como $M > n$, devemos dividir a mensagem M em blocos.

Vamos usar blocos de dois caracteres!

Escolher arbitrariamente um valor para e [primo relativo a $(p-1)(q-1)$, ou simplesmente um número primo] $\rightarrow e = 227$ [primo “pequeno”]

note que: $\text{mdc}(227, 198.640) = 1$

Chave pública = (227, 199.543)

Criptografia RSA

(Exemplo)

$p = 521$ e $q = 383$ | $n = 199.543$ | $e = 227$

Para gerar a chave privada precisamos calcular d como o inverso modular de e :

$$e * d \equiv 1 \pmod{(p-1)(q-1)} = 227 * d \equiv 1 \pmod{198.640}$$

Usamos Euclides Estendido $(227, 198.640) \Rightarrow \text{mdc} = 1$ / $x = -92.757$ / $y = 106$

E assim a única solução válida para a equação modular é:

$$d = x (1 / d) \bmod n = -92.757 \bmod 198.640 = 105.883$$

Chave privada = (105.883, 199.543).

Mensagem M: 01101111 01101001 01100001 00000000

$$M_1 = 57.193 \quad / \quad M_2 = 24832$$

Codificando

$$C_1 = 57.193^{227} \bmod 199.543 = 34.997$$

$$C_2 = 24.832^{227} \bmod 199.543 = 61.019$$

Decodificando

$$M_1 = 34.997^{105883} \bmod 199.543 = 57.193$$

$$M_2 = 61.019^{105883} \bmod 199.543 = 24.832$$

Ataque Força Bruta ao RSA

Considerando que as mensagens criptografadas forem capturadas por terceiros, um ataque de **força bruta** é um ataque em que testa-se uma a uma todas as combinações possíveis para se quebrar (descobrir) uma chave privada.

No caso do RSA o “atacante” irá usar o valor n da chave pública para fatorar os valores de p e q .

Uma vez descoberto p e q basta calcular a chave privada e transformar a mensagem criptografada.

Trabalho RSA

- Escreva um programa que realize:
 - ❖ codificação (criptografia RSA) de uma mensagem (String) em blocos de 4 caracteres (32 bits) $\Rightarrow n > 4.294.967.295$;
 - ❖ decodificação de mensagem criptografada com chave privada
 - ❖ decodificação de mensagem criptografada através de um ataque de força bruta usando a chave pública;
- Quantificar o tempo para criar uma mensagem codificada
- Quantificar e comparar o tempo para decodificar uma mensagem usando a chave privada e força bruta com diferentes tamanhos para os números primos (começando com **p** e **q** de 32 bits)
- Pode-se usar as bibliotecas (importadas ou da linguagem) para manipular números grandes (BigInt) – JAVA ou C/C++

Trabalho RSA

- Funções a serem implementadas:

geraPrimoPequeno() : int

crivoEratóstenes ()

primoProvavel(bits : int) : BigInt

testePrimalidade(n : BigInt) : boolean

inversoModular(a : BigInt , b : BigInt) : BigInt

gcdExt (a : BigInt , b : BigInt) : Trio

expModular(num : BigInt, exp : int, n : BigInt) : BigInt

ataqueForcaBruta(msg : String, e : int, n : BigInt) : String

Referências

Algoritmos. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Campus.

Algorithms. Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani. McGraw Hill.

Concrete Mathematics: A Foundation for Computer Science (2nd Edition). Ronald L. Graham, Donald E. Knuth, Oren Patashnik. Addison Wesley.

M. R. Garey and D. S. Johnson. 1978. *“Strong” NP-Completeness Results: Motivation, Examples, and Implications*. J. ACM 25, 3 (July 1978)