Implementação Eficiente de Algoritmos para Teste de Primalidade

1

Abstract. *The development of cryptography, public key cryptography in particular, was a crucial factor for the growth and popularization of computer networks. It was responsible for demands of secure email communication, e-commerce, digital signatures and certification. The proper use of cryptographic techniques requires the development of efficient implementations capable of running in all kind of devices, which more and more are integrated into people's lives. The generation of cryptographic keys, for example, is not only a critical security operation but also a have high computational cost in the RSA algorithm. This work aims to study techniques for primality testing, the core element of the key generation process in this algorithm. Our focus is on the implementation, optimization and performance analysis of Simplified Frobenius Quadratic Test, a primality test from 2005 which did not receive enough attention in the literature. The achieved results are positive regarding the feasibility of reducing the computational cost of generating prime numbers.*

Resumo. *O desenvolvimento da criptografia, em especial a criptografia de chave assimétrica, foi fator determinante para o crescimento e popularização das redes de computadores. Foi responsável pela viabilização de demandas como comércio e correio eletrônicos, assinaturas e certificações digitais. O uso adequado de técnicas criptográficas requer o desenvolvimento de implementações eficientes que sejam capazes de serem executadas em diversos tipos de dispositivos, que cada vez mais se incorporam à vida das pessoas. A geração de chaves criptográficas, por exemplo, é uma operação não só crítica quanto à segurança, mas também de alto custo computacional no algoritmo RSA. Este trabalho tem como objetivo estudar técnicas para teste de primalidade, elemento que compõe o núcleo do processo de geração de chaves nesse algoritmo. É dado enfoque na implementação, otimização e análise de desempenho do Teste de Frobenius Quadrático Simplificado, um teste de primalidade de 2005 e pouco explorado. Os resultados atingidos são positivos quanto à viabilidade da redução do custo computacional da geração de números primos.*

1. Introdução

Após as recentes denúncias sobre os casos de vigilantismo global por parte do Governo norte-americano [Rezende 2014b, Rezende 2014a], a criptografia e segurança digital têm sido um tema constante na mídia mundial. No momento, há grande discussão a respeito da liberdade e privacidade individual e a criptografia é elemento fundamental para a garantia de tais direitos civis no cenário de telecomunicações globalizado.

O tamanho das chaves de sistemas criptográficos de chave pública como o RSA tendem a crescer consideravelmente nos próximos 10 anos. Em vários casos há exigência dessas chaves serem geradas em sigilo, então tais aplicações devem ser executadas também em ambientes seguros como *smartcards* ou dispositivos blindados.

Esses dispositivos normalmente possuem poder computacional muito limitado, tornando a geração de chaves de alto nível de segurança uma operação demorada e até mesmo com tempo de execução proibitivo.

Este trabalho crê na premissa de que a difusão de criptografia depende de soluções práticas e eficientes. Para isso, o seu objetivo geral é reduzir o custo computacional na geração de chaves criptográficas. Como objetivos específicos, tem-se a análise, implementação e otimização de algoritmos de teste de primalidade, que são responsáveis por maior porcentagem do tempo de geração de chaves no algoritmo RSA.

2. Teste de Primalidade

A fatoração de números grandes é tida como um problema intratável, porém percebeu-se que saber apenas quanto a primalidade de um número é uma matéria distinta. Teste de primalidade pode ser modelado como um problema de decisão em que tem resultado SIM para primo e NÃO para composto.

Os testes são divididos em duas categorias:

- Testes Determinísticos: são testes que asseguram certeza matemática para todas as respostas.
- Testes Probabilísticos: são testes que contém probabilidade de erro, de incerteza. Escolhe-se um candidato aleatório e aplica-se a ele critérios que possam refutar ou confirmar certa propriedade. A cada rodada do teste, o candidato adquire maior probabilidade para uma resposta.

Apesar de exister teste determinístico com tempo de execução polinomial como o AKS [Agrawal et al. 2004], os testes probabilísticos, com segura margem de erro delimitada em 2^{-80} ou 2^{-100} [ISO/IEC 18032 2005], ainda são mais eficientes.

Algoritmos probabilísticos, por sua vez, podem ser classificados em dois tipos:

- Algoritmos de Monte Carlo: podem produzir resultado incorreto.
- Algoritmos Las Vegas: podem causar falha ao tentar produzir um resultado.

Para problemas de decisão, algoritmos de Monte Carlo são usualmente classificados como:

- Monte Carlo com viés positivo: é sempre correto quando retorna SIM.
- Monte Carlo com viés negativo: é sempre correto quanto retorna NÃO.

2.1. Teste de Miller-Rabin

O teste de Miller-Rabin [Miller 1976, Rabin 1980] é um algoritmo de Monte Carlo com viés positivo para compositividade [Stinson 2006]. É o teste mais utilizado nas soluções de criptografia e um dos mais eficientes da atualidade [Dietzfelbinger 2004], portanto será usado como comparativo em seções posteriores.

2.2. Teste de Frobenius Quadrático Simplificado

O Teste de Frobenius Quadrático Simplificado (TFQS) [Seysen 2005] foi o teste proposto para implementação e análise neste trabalho. Nesta versão simplificada do teste de Frobenius Quadrático [Grantham 1998], o tempo de execução foi reduzido para duas rodadas de Miller-Rabin e probabilidade de erro, para o pior caso, de 2^{-12t}, sendo t o número de rodadas.

O TFQS é um algoritmo de Monte Carlo com viés positivo para compositividade, usa polinômios quadráticos e o automorfismo de Frobenius.

Seja $q = p^m$ a potência de um primo p e o corpo finito $F_q = G(q)$, G a extensão de Galois. O **automorfismo de Frobenius** ϕ_q é o mapeamento na bijeção

$$\begin{aligned} \phi : F &\to F \\ z &\mapsto z^q \end{aligned}$$

para todo $z \in F$. Para um n natural ímpar e c uma unidade módulo n, $R(n, c)$ denota o anel polinomial $\mathbb{Z}_n[x]/(f(x) = x^2 - c)$, e $R(n, c)^*$ o grupo multiplicativo em $R(n, c)$. Sendo $f(x)$ um polinômio mônico quadrático em $\mathbb{Z}_n$, se n é primo e $f(x)$ é irredutível em $\mathbb{Z}_n$, então este anel é equivalente ao corpo finito $G(n^2)$. Felizmente, esses polinômios são facilmente encontrados: para n primo, um polinômio quadrático em $\mathbb{Z}_n$ é irredutível se e somente se seu discriminante, Δ, é um resíduo não-quadrático módulo n, ou seja, para o polinômio mônico $x^2 - bx - c$, $\left(\frac{b^2+4c}{n}\right) = -1$.

$G(n^2)$ é cíclico de ordem $n^2 - 1$, então qualquer $z \in R(n, c)^*$ deve ter uma ordem dividindo $n^2 - 1$. Também possui um automorfismo natural, chamado "conjugado" em $R(n, c)$, que deve ser equivalente ao automorfismo de Frobenius $z \to z^n$ em $G(n^2)$ para n primo. Além disso, sendo n primo, $n^2 - 1$ é divisível por 24 [1] e também $G(n^2)$ tem um grupo cíclico de raízes 24 de unidade, gerado por uma raiz primitiva.

Todas q-ésimas raízes de unidade z, se existir, satisfazem $\Phi_q(z) = 0$ para o q-ésimo polinômio ciclotômico $\Phi - q$.

Para um polinômio $z = ax + b$, define-se o seguinte homomorfismo multiplicativo em $R(n, c)$:

$$\begin{array}{lcll} \bar{\cdot} & : & R(n,c) \to R(n,c), & \bar{z} = b - ax & \text{(conjugado);} \\ N(\cdot) & : & R(n,c) \to \mathbb{Z}_n, & N(z) = \bar{z} \cdot z = b^2 - ca^2 & \text{(norma).} \end{array}$$

A notação (a/n) se refere ao símbolo de Jacobi.

O Algoritmo 1 é um crivo inicial equivalente a uma rodada do teste de Miller-Rabin com uma base pequena. Retorna ou uma raiz oitava primitiva de unidade em uma extensão quadrática conveniente de $\mathbb{Z}_n$ ou uma prova de que n é composto.

Na verdade, o Algoritmo MR2 realiza uma rodada de Miller-Rabin com base 2 caso $n \neq 1 \pmod 8$ e com base c caso contrário. A relação $\epsilon^4 = -1 \pmod{R(n, c)}$ é evidente caso $n = 1 \pmod 8$ e facilmente verificável pelas representações padrões $(\pm 1 \pm \sqrt{-1})/\sqrt{2}$ das quatro raízes oitavas primitivas de unidade nos outros casos [Seysen 2005].

[1] prova trivial pelo Teorema Chinês do Resto

Algorithm 1 Teste de Miller-Rabin com base dois ou não-resíduo pequeno

Input: Inteiro ímpar n.

Output: Ou COMPOSTO ou um inteiro c tal que $(c/n) = -1$ e $\epsilon \in R(n, c)$ com $\epsilon^4 = -1$ (i.e. $\Phi_8(\epsilon) = 0$).

if $n = 3 \pmod 4$ **then**
  $\alpha \leftarrow 2^{\frac{n-3}{4}} \pmod n$
  if $2\alpha^2 \neq \pm 1 \pmod n$ **then return** COMPOSTO
  else return $c \leftarrow -1$, $\epsilon \leftarrow \alpha + \alpha x$
end if
if $n = 5 \pmod 8$ **then**
  $\alpha \leftarrow 2^{\frac{n-1}{4}} \pmod n$
  if $\alpha^2 \neq -1 \pmod n$ **then return** COMPOSTO
  else return $c \leftarrow 2$, $\epsilon \leftarrow \frac{1+\alpha}{2} x$
end if
if $n = 1 \pmod 8$ **then**
  if n é um quadrado perfeito **then**
    return COMPOSTO
  else
    $c \leftarrow$ valor aleatório pequeno tal que $(c/n) = -1$
    $\alpha \leftarrow c^{\frac{n-1}{8}} \pmod n$
    if $\alpha^4 \neq -1 \pmod n$ **then return** COMPOSTO
    else return c, $\epsilon \leftarrow \alpha$
  end if
end if

Como foi citado, o conjugado também é um automorfismo em $R(n, c)$. O comportamento de Frobenius sob o conjugado é similar ao comportamento do grupo de decomposição como um todo.

Suponha o grupo G. Dois elementos a e b são ditos conjugados se existe um elemento $g \in G$ tal que $gag^{-1} = b$. Isso mostra que o conjugado é uma relação de equivalência e toda conjugação é um automorfismo. Logo, a não verificação do automorfismo de Frobenius ($z^n \neq \bar{z}$) é um certificador da compositividade de n.

O Algoritmo 2 representa uma rodada do teste de Frobenius quadrático simplificado.

Algorithm 2 Rodada do TFQS

Input: Inteiro ímpar n, inteiro pequeno c tal que $(c, n) = -1$ e o polinômio $\epsilon \in R(n, c)$ com $\epsilon^4 = -1$.

Output: Ou PRIMO ou COMPOSTO

1: Escolha z aleatório $\in R(n, c)$ com $(N(z)/n) = -1$
2: **if** $z^n \neq \bar{z}$ **then**
3: **return** COMPOSTO
4: **end if**
5: **if** $z^{\frac{n^2-1}{8}} \notin \{\pm\epsilon, \pm\epsilon^3\}$ **then**
6: **return** COMPOSTO
7: **end if**
8: **return** PRIMO

3. Implementação

O planejamento da implementação foi realizado sobre a biblioteca criptográfica Relic [Aranha and Gouvêa]. Para uma implementação completa do teste, é necessário uma ferramenta que forneça aritmética multi-precisão, suporte à manipulação de números inteiros muito grandes que extrapolariam a precisão de qualquer tipo primitivo de variáveis.

Em resolução com a Relic, a implementação foi executada utilizando a linguagem C padrão C99. Antes da implementação do teste em si, foi necessário elaborar funções específicas de utilidade comum e de operações em anel polinomial que não eram cobertas pela biblioteca. As principais delas serão discutidas nos tópicos a seguir.

3.1. Quadrado Perfeito

A raiz quadrada de um inteiro a com n *bits* deve estar contida no intervalo $(\ 2^{\lfloor \frac{n-1}{2} \rfloor},\ 2^{\lfloor \frac{n}{2} \rfloor})$. É possível, de forma semelhante a uma busca binária, encontrar a raiz inteira de a com o custo de $O(\log_2 n)$ multiplicações.

Tendo a raiz inteira c de a, basta verificar se $c^2 = a$ para determinar se a é quadrado perfeito.

3.2. Aritmética Básica

Adição e subtração de polinômios de mesmo grau é feita de maneira natural, respeitando a ordem de cada coeficiente e a redução módulo n.

A multiplicação de dois polinômios de graus m e n, resulta em um polinômio de grau $m + n$. Como a aritmética do teste é realizada no grupo multiplicativo $R(c, n)^*$, módulo $\mathbb{Z}_n$ e o polinômio quadrático $x^2 - c$, pode-se chegar a uma fórmula fechada para a multiplicação e redução: $(a_1b_0 + a_0b_1)x + a_0b_0 + a_1b_1c$.

O custo de processamento desses algoritmos é limitado pelo número de multiplicações, quadrados e reduções entre números grandes, portanto iremos desconsiderar o custo das somas e subtrações, que em geral são desprezíveis. Deste modo, uma multiplicação de dois polinômios tem o custo de quatro multiplicações e uma redução de coeficientes em $\mathbb{Z}_n$.

Para o quadrado de um polinômio em $R(n, c)^*$, basta usar o mesmo resultado de anterior para $a = b$, que resulta em $a^2 = 2a_1a_0x + a_0^2 + a_1^2c$ com o custo de uma multiplicação e dois quadrados.

3.3. Exponenciação Modular

A maneira mais ingênua, até mesmo para números de precisão simples, de calcular $c = a^k \bmod n$ consiste em realizar $k - 1$ multiplicações. Em aplicações criptográficas, a grandeza de k normalmente excede 2^{512} ou 2^{1024}. Computar tais multiplicações tomaria mais tempo do que a vida do Universo, o que torna ineficaz tal abordagem. Podemos descrever a exponenciação na seguinte forma recursiva:

$$a^k = \begin{cases} 1 & \text{se } k = 0 \\ a \cdot \left(a^{\frac{k-1}{2}}\right)^2 & \text{se } k \equiv 1 \pmod 2 \\ \left(a^{\frac{k}{2}}\right)^2 & \text{se } k \equiv 0 \pmod 2 \end{cases}$$

Utilizando essa ideia, podemos entender as sucessivas divisões por 2 como um *shift right* nos *bits* do expoente. O algoritmo implementado é o de exponenciação modular esquerda para direita, neste caso, a exponenciação modular é realizada com $O(\log_2\ n)$ quadrados e $o(\log_2\ n)$ multiplicações.

4. Otimização

4.1. Crivo para Quadrados Perfeitos

É possível chegar ao resultado de que todo quadrado perfeito par é múltiplo de 4, todo quadrado perfeito ímpar deixa resto 1 mod 4 e, por transitividade, 1 mod 8. Analisando o digito menos significativo, e o seu quadrado, dos múltiplos de 4 e dos ímpares que deixam resto 1 na divisão por 4, chega-se a conclusão que os quadrados perfeitos só podem ter os algarismos $0, 1, 4, 5, 6$ e 9 como digito menos significativo.

A partir dessa ideia, pode-se criar um crivo inicial que retorna FALSO para qualquer candidato a quadrado perfeito que termine em $2, 3, 7$ ou 8.

4.2. Multiplicação de Karatsuba

Anatolii A. Karatsuba, quando jovem aluno de graduação, propôs um eficiente algoritmo para multiplicação [Karatsuba and Ofman 1963] de dois números grandes que contradizia uma conjectura lançada por seu professor, A. Kolmogorov, a respeito de um limite inferior para essa multiplicação. A mesma ideia de Karatsuba pode ser aplicada na multiplicação de polinômios: $a \cdot b = \big[(a_1 + a_0)(b_1 + b_0) - a_1b_1 - a_0b_0\big]x + a_0b_0 + a_1b_1c$.

Expandindo essa expressão de Karatsuba, pode-se notar que retorna ao mesmo resultado da multiplicação em 3.2. Assim, reutilizando os produtos já computados, o custo da multiplicação de dois polinômios cai para três multiplicações e uma redução.

4.3. Fórmula do Quadrado Complexo

Também se tem uma expressão utilizada no quadrado de número complexos, $(a + bi)$, que pode ser usada no quadrado de polinômios quadráticos: $a^2 = 2a_0a_1x + (a_0 + a_1)(a_0 + a_1c) - a_0a_1 - a_0a_1c$. O custo é reduzido para apenas duas multiplicações: $(a_0 + a_1)(a_0 + a_1c)$ e a_0a_1.

4.4. Janela Deslizante

O método janela deslizante para exponenciação modular implementado é baseado em uma generalização do algoritmo de exponenciação esquerda para direita que possibilita o processamento de mais de um *bit* do expoente por iteração através do pré-cálculo de exponenciações em uma janela. A janela deslizante consegue reduzir o pré-cálculo e a quantidade de multiplicações [Menezes et al. 1996].

4.5. Redução de Montgomery

Durante a exponenciação modular, são realizadas sucessivas multiplicações e quadrados. As repetidas reduções em $\mathbb{Z}_n$ nessas operações é um fator limitante crítico do desempenho da função. Uma solução proposta por Montgomery é trocar as divisões por multiplicações. Contudo, há um detalhe importante: para ser realizada a redução de Montgomery é necessário o pré-cálculo de uma inversão em $\mathbb{Z}_n$, fato que deve ser considerado na análise do desempenho.

A otimização é alcançada quando é necessário uma repetição conhecida de operações que utilizam essa redução. Neste caso, a transformação na forma de Montgomery pode ser antecipada e a volta, atrasada para depois das repetições, aumentando o desempenho nas reduções.

4.6. Mudança Algébrica

4.6.1. Multiplicação por c pequeno

Analisando o algoritmo 1 (MR2), vê-se que o inteiro retornado c varia em apenas três casos: $c = -1 \equiv n - 1$, $c = 2$ e $c =$ valor aleatório pequeno tal que $(c/n) = -1$. As multiplicações de inteiros multi-precisão por c podem ser simplificadas para todos os casos.

No primeiro caso, podemos trocar uma multiplicação de $n - 1$ e uma redução módulo n por apenas uma subtração de n. No segundo caso, trocamos uma multiplicação e uma redução por um *shift left* e uma subtração. Para o terceiro caso, podemos trocar a busca aleatória por uma busca incremental com no máximo 3 cálculos de Jacobi [Seysen 2005], assim mantendo o valor de c restritamente pequeno de modo que possamos substituir uma multiplicação e uma redução por uma multiplicação por dígito simples e $O(1)$ subtrações.

4.6.2. Reuso de uma exponenciação modular

É possível reduzir a quantidade de exponenciações modulares necessárias nos cálculos de z^n e $z^{(n^2-1)/8}$ no algoritmo [Seysen 2005]. Tome $t = z^{\lfloor (n-1)/8 \rfloor} = z^{(n-1-\epsilon)/8}$, $\epsilon \in \mathbb{Z}$ e $0 \leq \epsilon < 8$. Computando t previamente e definindo ϵ, podemos calcular z^n com mais $O(1)$ multiplicações e quadrados. A partir de t, também podemos obter o valor de $z^{(n^2-1)/8}$ como segue: $z^{(n^2-1)/8} = z^{(n+1+\epsilon)(n-1-\epsilon)/8} \cdot z^{\epsilon(\epsilon+2)/8} = N(t) \cdot t^{\epsilon} \cdot z^{\epsilon(\epsilon+2)/8}$. Assim, de duas exponenciações modulares em anel polinomial é possível reduzir para apenas uma exponenciação e $O(1)$ multiplicações e quadrados.

4.7. Redução Preguiçosa

Entre as três multiplicações na implementação de Karatsuba é necessária uma redução modular extra para que não seja excedida a precisão da biblioteca, no caso dos operandos terem precisão máxima.

A técnica da redução preguiçosa consiste em atrasar esta redução intermediária para o final da operação:

- Elimina-se a redução intermediária e define um fator de redução $r = n \cdot 2^{|n|}$.
- Quando necessária uma soma ou subtração de um produto com a precisão acumulada, realiza-se uma soma ou subtração de r como redução.

O custo final da multiplicação de Karatsuba em $I(n, c)$ é de três multiplicações e duas reduções. Na operação de quadrado não há como usar a técnica da redução preguiçosa, pois o cenário ja é ótimo, duas multiplicações e duas reduções modulares.

5. Resultados

O ambiente de testes do projeto tem a seguinte configuração:

Tabela 1. Ambiente de testes.

Computador	Asus Q550L. Intel Core i7-4500U. 8GB DDR3.
OS / kernel	Ubuntu 13.10 x86˙64 / 3.11.0-15-generic
Compilador	gcc 4.8.1
Relic	relic-0.3.5
GMP	2:5.1.2+dfsg amd64. *Multiprecision arithmetic library*

O uso da biblioteca GMP otimiza consideravelmente as funções da Relic.

A cada passo de implementação e otimização dos algoritmos, foi realizado testes para garantir a corretude dos componentes e uma coleta de dados consistente. O artefato de testes foi desenvolvido em um ambiente fornecido pela própria biblioteca Relic e verificava asssertivas sobre propriedades de anéis de polinômio.

Após a implementação otimizada do TFQS, chegou-se à comparação com o teste de Miller-Rabin, algoritmo padrão da Relic. Nesta análise, foram necessários alguns ajustes para igualar o *setup* dos algoritmos. Inicialmente, as rodadas do teste de Miller-Rabin estavam configuradas para uma probabilidade de erro de 2^{-80}, sendo necessário um ajuste no número de rodadas para se equiparar ao erro do TQFS, 2^{-100}.

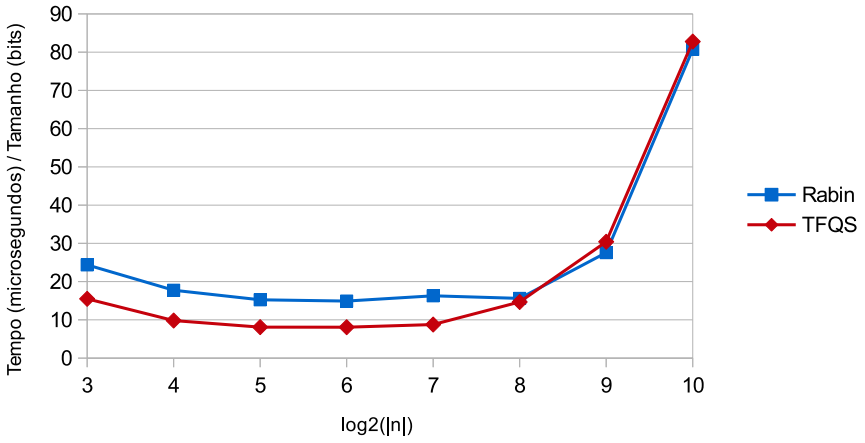

Figura 1. Gerador de primo com teste Rabin vs. TFQS.

Na comparação dos resultados obtidos (Figura 1), nota-se que o TFQS é mais veloz para teste de inteiros de até 256 *bits*. Neste intervalo, obteve-se uma redução de cerca de 30% no tempo de execução. A partir de 512 *bits*, o Miller-Rabin passa a ser mais eficiente.

Em uma visão panorâmica, o TFQS pode ser delimitado pelo custo de uma exponenciação modular em anel de polinômio por rodada. De maneira semelhante, o teste de Miller-Rabin pode ser delimitado por uma exponenciação modular simples por rodada.

6. Conclusão

Neste trabalho foi apresentado métodos de aritmética em anel de polinômio e também técnicas de otimização que culminaram na produção de um artefato eficaz. Foi apresentado um teste de primalidade ainda novo que obteve resultado satisfatório para inteiros de até 256 *bits*. No seu cenário ótimo, o TFQS se mostrou, em média, 30% mais veloz que o teste de Miller-Rabin. Conclui-se que o TFQS possui iterações mais longas, no entanto, executa menos rodadas devido sua baixa probabilidade de erro. O teste de Miller-Rabin possui iterações mais rápidas, devido aritmética mais simples, mas executa maior número de rodadas. À medida que a precisão do inteiro testado aumenta, essa vantagem do TFQS diminui, tornando nula quando o número de rodadas do Miller-Rabin iguala ao custo de uma exponenciação modular em anel de polinômio em relação ao custo de uma exponenciação modular simples.

O escopo principal do trabalho foi atingido em parte, houve a implementação de um teste mais eficiente, porém com restrição de magnitude. O problema de gerar chaves com alto nível de segurança em dispositivos com poder computacional limitado em tempo hábil ainda persiste.

Como trabalhos futuros, propõe-se o estudo de outros testes presentes na indústria como o teste de Lucas, Baillie-PSW [Baillie and Wagstaff 1980] e Miller-Rabin combinado ao teste de Lucas-Selfridge [Pomerance et al. 1980] que é baseado no teste Baillie-PWS e aperfeiçoado por Selfridge.

Por fim, todo código produzido nesta obra está disponível no repositório online (url) com um arquivo *bash* que gerencia a configuração, compilação e execução de testes, *benchmarks*, produção de gráfico de desempenho, entre outros.

Referências

Agrawal, M., Kayal, N., and Saxena, N. (2004). PRIMES is in P. *Annals of Mathematics*, 160(2):781–793.

Aranha, D. F. and Gouvêa, C. P. L. RELIC is an Efficient Library for Cryptography. `http://code.google.com/p/relic-toolkit/`.

Baillie, R. and Wagstaff, S. (1980). Lucas pseudoprimes. *Mathematics of Computation*, 35(152):1391–1417.

Dietzfelbinger, M. (2004). *Primality Testing in Polynomial Time*. Lecture Notes in Computer Science. Springer.

Grantham, J. (1998). A probable prime test with high confidence. *J. Number Theory*, 72(1):32–47.

ISO/IEC 18032 (2005). Information technology - Security techniques - Prime number generation.

Karatsuba, A. and Ofman, Y. (1963). Multiplication of many-digital numbers by automatic computers. *Physics-Doklady*, 7:595–596.

Menezes, A. J., van Oorschot, P. C., and Vanstone, S. A. (1996). *Handbook of Applied Cryptography*. CRC Press.

Miller, G. L. (1976). Riemann's hypothesis and tests for primality. *Journal of Computer and Systems Sciences*, 13(3):300–317.

Pomerance, C., Selfridge, J. L., and Wagstaff, S. (1980). The pseudoprimes to $25 \cdot 10^9$. *Mathematics of Computation*, 35(151):1003–1026.

Rabin, M. O. (1980). Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128–138.

Rezende, P. A. D. (2014a). Como Controlar o Vigilantismo Global? `http://www.cic.unb.br/docentes/pedro/trabs/operamundi.html`.

Rezende, P. A. D. (2014b). O que aprendemos com Edward Snowden? `http://www.cic.unb.br/docentes/pedro/trabs/aprendercsnowden.html`.

Seysen, M. (2005). A simplified quadratic frobenius primality test.

Stinson, D. R. (2006). *Cryptography: Theory and Practice*. Chapman and Hall/CRC, Florida.