Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Implementação Eficiente de Algoritmos para Teste de Primalidade

Bruno Cesar Dias Ribeiro

Brasília
2013

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Implementação Eficiente de Algoritmos para Teste de Primalidade

Bruno Cesar Dias Ribeiro

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Dr. Diego de Freitas Aranha

Brasília
2013

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof.ª Dr.ª Maristela Terto de Holanda

Banca examinadora composta por:

Prof. Dr. Diego de Freitas Aranha (Orientador) — CIC/UnB
Prof. João José Costa Gondim — CIC/UnB
Prof. Pedro Antônio Dourado de Rezende — CIC/UnB

CIP — Catalogação Internacional na Publicação

Ribeiro, Bruno Cesar Dias.

Implementação Eficiente de Algoritmos para Teste de Primalidade / Bruno Cesar Dias Ribeiro. Brasília : UnB, 2013.
143 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2013.

1. criptografia assimétrica, 2. primalidade, 3. chaves criptográficas

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Implementação Eficiente de Algoritmos para Teste de Primalidade

Bruno Cesar Dias Ribeiro

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dr. Diego de Freitas Aranha (Orientador)
CIC/UnB

Prof. João José Costa Gondim
CIC/UnB

Prof. Pedro Antônio Dourado de Rezende
CIC/UnB

Prof.ª Dr.ª Maristela Terto de Holanda
Coordenador do Bacharelado em Ciência da Computação

Brasília, 22 de Julho de 2013

Agradecimentos

Aos meus pais e irmãos, que nunca pouparam esforços para me apoiar e orientar em todo caminho que decidi seguir. Às Professoras Carla Koike, Fernanda Lima, Carla Castanho e a amigos por acreditarem em minha competência. Aos Professores Pedro Antônio Dourado de Rezende e Diego de Freitas Aranha, cujo conhecimento e entusiasmo foram definitivos para despertar curiosidade por uma área fascinante.

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 aplicações eficientes que sejam capazes de executar em diversos tipos de dispositivos que cada vez mais se incorporam à vida das pessoas. A geração de chaves criptográficas é uma operação não só crítica quanto à segurança, mas também de alto custo computacional. Este trabalho tem o intuito de estudar sistemas criptográficos, conceitos teóricos e teste de primalidade, elemento que compõe o núcleo do processo de geração de chaves. 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 dessas operações.

Palavras-chave: criptografia assimétrica, primalidade, chaves criptográficas

Abstract

The development of cryptography, public key cryptography in particular, was a crucial factor responsible for the growth and popularization of computer networks. It was responsible for appearance of demands as email, e-commerce, digitals signature and certification. The proper use of cryptographic techniques requires the development of efficient applications capable of running in all kind of devices witch more and more incorporates people's lives. The generation of cryptographic keys is not only a critical security operation but also a have high computational cost. This project has the goal of studying cryptosystems and their theoretical basis and primality test, the core element of the key generation process. It has an especial approach on implementing, optimizing and performance analysis of Simplified Frobenius Quadratic Test, a primality test from 2005 and under explored. The achieved results are positives about the feasibility of reducing the computational cost of operation.

Keywords: asymmetric cryptography, primality, cryptographic keys

Sumário

1 Introdução **1**
1.1 Criptografia . . . 1
1.1.1 Contexto Histórico . . . 1
1.1.2 Criptografia Moderna . . . 2
1.1.3 Segurança . . . 3
1.1.4 Princípios de Kerchoff . . . 3
1.2 Objetivos . . . 4

2 Sistema Criptográfico **5**
2.1 Sistema Criptográfico Simétrico . . . 7
2.1.1 Vantagens . . . 8
2.1.2 Desvantagens . . . 9
2.2 Sistema Criptográfico Assimétrico . . . 9
2.2.1 Funções Unidirecionais . . . 11
2.2.2 Vantagens . . . 12
2.2.3 Desvantagens . . . 12
2.3 Combinação dos Sistemas . . . 12

3 Criptografia de Chave Pública **14**
3.1 Complexidade Computacional . . . 14
3.1.1 Casos Melhor, Pior e Médio . . . 14
3.1.2 Limites Superior e Inferior . . . 15
3.2 Teoria dos Números . . . 16
3.2.1 Números Primos . . . 16
3.2.2 Teorema Fundamental da Aritmética . . . 18
3.2.3 Algoritmo Euclidiano . . . 19
3.2.4 Congruência . . . 20
3.2.5 Teorema Chinês do Resto . . . 21
3.2.6 Pequeno Teorema de Fermat . . . 21
3.2.7 Teorema de Euler . . . 22
3.2.8 Símbolos de Legendre e Jacobi . . . 22
3.3 Algoritmo RSA . . . 23
3.3.1 Esquema de Cifração . . . 25
3.3.2 Esquema de Assinatura Digital . . . 26
3.3.3 Funções de Resumo e Preenchimento . . . 28

4 Teste de Primalidade **30**
4.1 Importância dos Números Primos . . . 30
4.2 Fórmula para Primos . . . 31
4.2.1 Função de Mills . . . 31
4.3 Testes de Primalidade . . . 32
4.3.1 Crivo de Eratóstenes . . . 33
4.3.2 Teste de Fermat . . . 34
4.3.3 Teste de Solovay-Strassen . . . 35
4.3.4 Teste de Miller-Rabin . . . 36
4.3.5 Teste de Adleman-Huang . . . 37
4.3.6 Teste AKS . . . 37

5 Teste de Frobenius Quadrático Simplificado **40**
5.1 Implementação . . . 42
5.1.1 Relic . . . 43
5.1.2 Quadrado Perfeito . . . 44
5.1.3 Aritmética Básica . . . 44
5.1.4 Exponenciação Modular . . . 46
5.1.5 Probabilidade de Erro . . . 47
5.2 Otimização . . . 47
5.2.1 Crivo para Quadrados Perfeitos . . . 47
5.2.2 Multiplicação de Karatsuba . . . 47
5.2.3 Fórmula do Quadrado Complexo . . . 48
5.2.4 Janela Deslizante . . . 48
5.2.5 Redução de Montgomery . . . 49
5.2.6 Mudança Algébrica . . . 51
5.2.7 Redução Preguiçosa . . . 52
5.3 Testes . . . 52
5.4 Resultados . . . 53
5.4.1 Consumo de Tempo . . . 53
5.4.2 Teste de Primalidade . . . 56
5.4.3 Gerador de Primos . . . 57

6 Conclusão **60**

Referências **61**

Lista de Figuras

2.1 Modelo de comunicação utilizando criptografia. Adaptado de Menezes [29]. 7
2.2 Modelo de comunicação utilizando criptografia simétrica. Adaptado de Menezes [29]. 8
2.3 Modelo de comunicação utilizando criptografia assimétrica. Adaptado de Menezes [29]. 10
2.4 Diagrama de uma função dita uniderecional. Adaptado de Goldreich [17]. . 11

3.1 Comparação assintótica entre crescimento das funções. Retirado de Hein [19]. 16
3.2 Ilustração da dificuldade da fatoração de números grandes. Retirado de xkcd [50]. 24

5.1 Fluxograma das etapas do desenvolvimento do projeto. 43
5.2 Consumo de tempo (%) vs. Tamanho de n (em potência de 2) do TFQS versão 1. 54
5.3 Consumo de tempo (%) vs. Tamanho de n (em potência de 2) do TFQS versão 5. 55
5.4 Gráfico do desempenho entre TFQS versão 1 e versão 5. 55
5.5 Gráfico Rabin vs. TFQS para entrada primo. 57
5.6 Gráfico Rabin vs. TFQS para entrada composto. 57
5.7 Gerador de primo com semente fixa. Rabin vs. TFQS. 58
5.8 Gerador de primo com semente aleatória. Rabin vs. TFQS. 59

Lista de Tabelas

2.1 Tabela de terminologias 5

3.1 Algoritmo RSA. 24

5.1 Probabilidade de erro para o TFQS (*bits* k vs. número de rodadas t). Retirada de [43]. 47
5.2 Teste de assertivas em componentes. 53
5.3 Ambiente de testes. 53
5.4 Rodadas do Miller-Rabin ajustada para erro de 2^{-100}. 56
5.5 Comparação entre as operações de exponenciação modular simples e em anel polinomial. 59

Capítulo 1

Introdução

1.1 Criptografia

A criptografia, na sua definição clássica, é a técnica de transformar mensagens em um padrão não compreensível para que apenas o destinatário, através de um conhecimento privilegiado, possa retorná-la ao seu estado original, assim, protegendo-a de agentes não autorizados (adversários). Este conhecimento privilegiado é chamado de chave. É uma forma de escrita secreta, como pode ser observado na etimologia da palavra.

A criptoanálise refere-se ao conjunto de técnicas para analisar métodos criptográficos [4]. Um dos propósitos é decifrar criptogramas sem ter acesso à chave ou também sem conhecimento do sistema criptográfico utilizado. Para tal, deve-se buscar fraquezas que possam levar à descoberta do texto em claro ou da chave. Pode ser classificada como passiva, no qual a mensagem é interceptada sem que o destinatário saiba do ocorrido, e como criptoanálise ativa, que consiste em modificar, retransmitir ou substituir a mensagem em trânsito, algumas dessas ações podem ser detectadas pelas partes.

Criptologia é ciência que reúne a criptografia e criptoanálise e estuda suas interações [7].

1.1.1 Contexto Histórico

Por milhares de anos, reis, rainhas e generais confiavam o comando e governança de seus países a eficiente métodos de comunicação. Ao mesmo tempo, todos tinham consciência das consequências de suas mensagens cair em mãos erradas, não autorizadas, revelando segredos valiosos a nações rivais. Foi esse risco de interceptações inimigas que motivou o desenvolvimento de códigos e cifras. E do outro lado, os adversários desvelavam quebrar essas cifras com intuito de furtar os segredos. A história da criptografia é a história das batalhas entre criadores de código e quebradores de código, uma corrida armamentista intelectual que teve dramático impacto no curso da história [46].

Pouco se sabe sobre o início do uso da criptologia pelas sociedades antigas. Exatamente por sua própria natureza sigilosa, permanecia envolta em si mesma, a literatura clássica disponível é escassa, limitada e de difícil pesquisa. Há relatos de que profissionais responsáveis pela criptologia militar e diplomática eram obrigados a manter-se em anonimato e acatar a censura de suas publicações. David Kahn publicou *The Codebreakers* [22], um livro não técnico, mas que reúne inúmeros materiais históricos que exemplificam o uso

da criptografia, sua maior parte nos períodos das guerras, e que fez um ótimo serviço ao coletar dados históricos e trazer maior atenção para esta área.

Historicamente, o uso da criptografia para fins militares era crucial, desde a civilização egípcia, datando de 4000 A.C., passando pelo império Romano com clássico exemplo da cifra de César, até os tempos modernos, no qual teve papel fundamental nas duas grandes guerras. Kahn diz que graças ao uso da criptoanálise, economizou-se mais de um ano de duração da segunda grande guerra na região do pacífico. É evidente que conquista-se enorme vantagem ao desvendar manobras militares do seu adversário. Infelizmente, é durante esses períodos de conflito que ocorrem grandes avanços tecnológicos, com a criptografia não foi diferente. Tradicionalmente, era comum existir linguistas na área de criptoanálise, mas a Segunda Guerra Mundial trouxe definitivamente os matemáticos para a ponta [7]. As mentes mais brilhantes do mundo, desta área, estavam reunidas em comum esforços de construir novas cifras e novos ataques a cifras de rivais.

Durante todos esses anos, a criptologia era um domínio quase que exclusivo dos militares, mas foi no final dos anos 60 em diante, com o crescente uso das redes de computadores, massificação da Internet e redes de transmissão sem fios, que surgiu uma forte necessidade de segurança dessa informação, agora digital, e então passou a ser objeto de estudo público e da comunidade acadêmica. Hoje, o estado da arte da criptografia computacional acontece fora dos portões secretos dos militares e, até onde se imaginava, com pouca influência de governos [42].

Esta afirmação passou a ser questionada após as recentes denúncias e vazamento de documentos secretos da Agência de Segurança Nacional norte-americana (NSA) através de Edward Snowden. Foi revelada não só espionagem, mas também grande influência para a escolha de padrões criptográficos nacionais.

1.1.2 Criptografia Moderna

Por séculos, a criptografia se desenvolveu de arte, na qual poucos eram os que tinham o conhecimento, para ciência. Em *Encyclopedia of Cryptography and Security* [49], Tilborg cita o artigo de Shannon [44] de 1949, como base da criptografia moderna. Contudo, naquele tempo, grande parte dos pesquisadores ainda estavam restritos a agências governamentais e poder bélico. Situação que teve mudança gradativa com a expansão das indústrias de telecomunicações.

Dada esta nova ordem computacional e o importante marco da criação da criptografia assimétrica, tendo Diffie & Hellman [12] como pioneiros, a criptografia moderna passou a se ocupar muito menos com a obtenção de sigilo, seu uso clássico cuja etimologia era justificada plenamente, e alcançou novas propriedades da segurança da informação. Passou a ser o estudo de técnicas matemáticas relacionadas à segurança da informação como a confidencialidade, integridade, autenticação e irretratabilidade (não-repudiação). A criptografia, portanto, não é o único meio de garantir esta segurança, mas sim um conjunto de técnicas para tal [29].

Os principais objetivos da criptografia moderna são:

- Confidencialidade é a propriedade de manter o conteúdo da informação em sigilo de todos que não estejam autorizados a recebê-la. O sigilo e privacidade podem ser utilizados como sinônimos de confidencialidade. Existem diversas abordagens

para o fornecimento de confidencialidade, que vão desde a própria proteção física do conteúdo a algoritmos matemáticos que tornam os dados ininteligíveis.

- Integridade é uma propriedade de se resguardar da alteração não autorizada de dados. É preciso, para isto , ter a capacidade de detectar a manipulação da mensagem por partes não autorizadas. Esta abordagem pertence à criptoanálise ativa, na qual a manipulação inclui ações como inserção, exclusão ou substituição de dados.

- Autenticação é um serviço relacionado à identificação. Esta função se aplica tanto às entidades quanto a informação em si. Duas partes que buscam uma comunicação devem identificar uma à outra. Informações que trafeguem através de um canal devem ser autenticadas com origem, a data de origem, o conteúdo de dados, hora de envio, etc. Por estas razões, esta propriedade da criptografia geralmente é subdividida em duas classes: autenticação de entidades e autenticação de origem de dados.

- Irretratabilidade é um serviço que impede uma entidade de negar compromissos ou ações anteriores. Se surgem conflitos devido a uma entidade negar que certas ações foram realizadas, uma forma de solucionar a situação é necessária. Por exemplo, uma entidade pode autorizar a aquisição de propriedades por outra entidade e, posteriormente, negar que tal autorização foi concedida. É necessário, para sanar a disputa, que envolva uma terceira parte confiável.

1.1.3 Segurança

A segurança de primitivas e protocolos criptográficos pode ser avaliada sob vários modelos [29]. As métricas de segurança mais comuns são computacional e demonstrável (ou provável).

- Segurança incondicional: Assume-se que o adversário possui poder computacional ilimitado. A questão é apenas se há ou não informação suficiente disponível para tentar derrotar o sistema criptográfico. A cifra *One-Time Pad*, que utiliza uma chave aleatória de tamanho maior ou igual ao da mensagem, é um exemplo de cifra que atinge a segurança incondicional, apesar de ser vulnerável a ataques ativos.

- Segurança demonstrável: O sistema possui segurança demonstrável se a dificuldade de derrotá-lo pode ser reduzida a resolução de um problema conjecturado como difícil.

- Segurança computacional: Avalia o recurso computacional necessário para tentar derrotar o sistema. Um método é dito computacionalmente seguro se o melhor ataque conhecido excede o poder computacional do adversário, com boa margem.

1.1.4 Princípios de Kerchoff

A chave criptográfica, ao contrário dos algoritmos, é o princípio permanente da criptografia. Foi declarado em 1883 por Auguste Kerckhoffs: a segurança do sistema criptográfico não deve depender do sigilo do algoritmo, mas sim de manter a chave em segredo.

Nenhum inconveniente deve ocorrer caso o sistema criptográfico caia nas mãos do inimigo [24, 46]. Kerckhoffs também enunciou, em sua publicação, seis propriedades que são tidas com grande relevância até os dias de hoje:

1. O sistema deve ser substancialmente, se não matematicamente, indecifrável.
2. Não deve ser exigido sigilo do sistema. Seu furto não deve causar problemas.
3. Deve ser fácil de se comunicar e de guardar a chave sem necessitar anotá-la, também deve ser fácil de alterar a chave entre as partes.
4. O sistema deve ser compatível com a comunicação via telégrafo.
5. O sistema deve ser portável e seu uso não deve necessitar mais de uma pessoa.
6. Por fim, dada as circunstâncias de sua aplicação, o sistema deve ser fácil de usar e não deve exigir esforço cognitivo nem conhecimento de longas regras [24].

1.2 Objetivos

Após as recentes denúncias sobre os casos de vigilantismo global por parte do Governo norte-americano [38, 39], 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.

Para isso, será realizado um estudo sobre sistemas criptográficos e uma base teórica de Teoria dos Números e Álgebra. Em seguida, o foco será no sistema de chave pública RSA e sua geração de chaves, algoritmos de teste de primalidade e escolha do Teste de Fobrenius Quadrático Simplificado para implementação, otimização e análise de desempenho.

Capítulo 2

Sistema Criptográfico

Para o início do estudo científico de técnicas criptográficas, primeiramente, necessitamos de elucidar a terminologia da área e expressar algumas definições formais.

A Tabela 2.1 contém alguns termos que serão usados constantemente.

Tabela 2.1: Tabela de terminologias

Participantes	São as partes envolvidas no processo ou protocolo de comunicação. Têm ações como enviar, receber ou manipular informação. Pode ser uma pessoa, um computador, terminal, etc.
Emissor	Em uma comunicação de duas partes, é a entidade legítima que envia a mensagem.
Receptor	Em uma comunicação de duas partes, é a entidade legítima de destino da mensagem.
Adversário	É a entidade que não é nem o emissor nem o receptor e tem o intuito de driblar o protocolo ou serviço de segurança de informação provido pelas partes. Pode ter outros sinônimos como inimigo, atacante, oponente, intruso. Um adversário frequentemente tenta se passar pelo emissor ou receptor legítimos. Precisamos enfatizar que apesar dos termos "ataque" e "inimigo", essas ações serão realizadas apenas com objetivos acadêmicos.
TTP	*Trusted Third Party* (Terceira Parte Confiável) É uma entidade confiável e de comum acordo, definida antecipadamente entre as partes envolvidas. Como exemplo, podemos citar os órgãos certificadores de chave pública.
Canal	Canal é o meio de transmissão da informação de uma entidade para outra.
Canal seguro	É um canal que não pode ser acessado pelo atacante.
Canal inseguro	É um canal que pode ser acessado por quaisquer partes, além do emissor e receptor, e que podem realizar ações como ler, inserir, excluir ou alterar as mensagens de forma indetectável.

A literatura usualmente exemplifica o desafio da comunicação utilizando as entidades Alice, Bob e Oscar, o que já se tornou um padrão. Alice e Bob desejam se comunicar

através de um canal inseguro de modo que um adversário, Oscar, não possa compreender o conteúdo da mensagem entre os dois primeiros. Este canal pode ser uma linha telefônica, uma rede de computador ou até mesmo por correio [48].

Um sistema criptográfico é uma quíntupla $(\mathcal{M}, \mathcal{C}, \mathcal{K}, E_e, D_d)$ que satisfaz as seguintes propriedades:

- $\mathcal{A}$ denota um conjunto finito chamado de alfabeto de definição, conjunto de símbolos. Por exemplo, $\mathcal{A} = \{a, b, c, ..., z\}$ é o alfabeto de uma linguagem como o Inglês. Outro exemplo seria o alfabeto binário: $\mathcal{A} = \{0, 1\}$. Este é frequentemente usado como alfabeto padrão, pois note que qualquer outro alfabeto pode facilmente ser codificado com o alfabeto binário. Uma conjunto de 5 *bits* pode gerar 32 cadeias binárias distintas, assim, pode-se atribuir cada letra do nosso alfabeto a uma única cadeia de *bits* de tamanho 5. Exemplos de padrões de codificação utilizados são o ASCII e Unicode, que possuem não só caracteres do alfabeto, imprimíveis, mas também caracteres especiais e de controle.

- $\mathcal{M}$ é definido como espaço de mensagens. Um elemento de $\mathcal{M}$ consiste em uma cadeia de símbolos de um alfabeto de definição. Um elemento de $\mathcal{M}$ é chamado de texto claro. Por exemplo, $\mathcal{M}$ é composto por uma cadeia de *bits*, um texto em português, código de uma linguagem de programação, etc.

- $\mathcal{C}$ define o conjunto de espaço de criptogramas. $\mathcal{C}$ consiste de cadeias de símbolos de um alfabeto de definição, mas que se difere do alfabeto de definição de $\mathcal{M}$. Um elemento de $\mathcal{C}$ é chamado de criptograma.

- $\mathcal{K}$ é o espaço de chaves. Um elemento de $\mathcal{K}$ é chamado de chave.

- Cada elemento $e \in \mathcal{K}$ determina uma bijeção de $\mathcal{M}$ em $\mathcal{C}$, definida por E_e.
  E_e é chamado de função de cifragem, ou algoritmo de cifragem. Note que E_e deve ser uma função bijetiva para que possua uma transformação inversa que retorne um único texto claro para cada criptograma distinto. Ou seja:

- Para cada $d \in \mathcal{K}$, D_d define a bijeção de $\mathcal{C}$ em $\mathcal{M}$. É chamado de função de decifragem ou algoritmo de decifragem.

- Portanto, seja $E_e : \mathcal{M} \to \mathcal{C}$

  - E_e é injetiva: Se $E_e(m_1) = c$ e $E_e(m_2) = c$, então $m_1 = m_2$ para qualquer $m_1, m_2 \in \mathcal{M}$
  - E_e é sobrejetora: $\forall\, c \in \mathcal{C}, \quad \exists!\, m \in \mathcal{M} : E_e(m) = c$

- Um sistema de cifragem então, Consiste em um conjunto $\{E_e : e \in \mathcal{K}\}$ de funções de cifração e um conjunto correspondente $\{D_d : d \in \mathcal{K}\}$ de decifração com a propriedade de que para cada $e \in \mathcal{K}$ existe um único $d \in \mathcal{K}$ tal que $D_d = E_e^{-1}$, consequentemente, $D_d(E_e(m)) = m$ para todo $m \in \mathcal{M}$. Ou seja,

$$\{\{E_e : e \in \mathcal{K}\}, \{D_d : d \in \mathcal{K}\}\} \mid \forall e \in \mathcal{K},\ \exists\, d \in \mathcal{K} \ :\ D_d = E_e^{-1} \text{ [4]}$$

- As chaves e e d, nas definições acima, são chamadas de par de chaves (e, d)

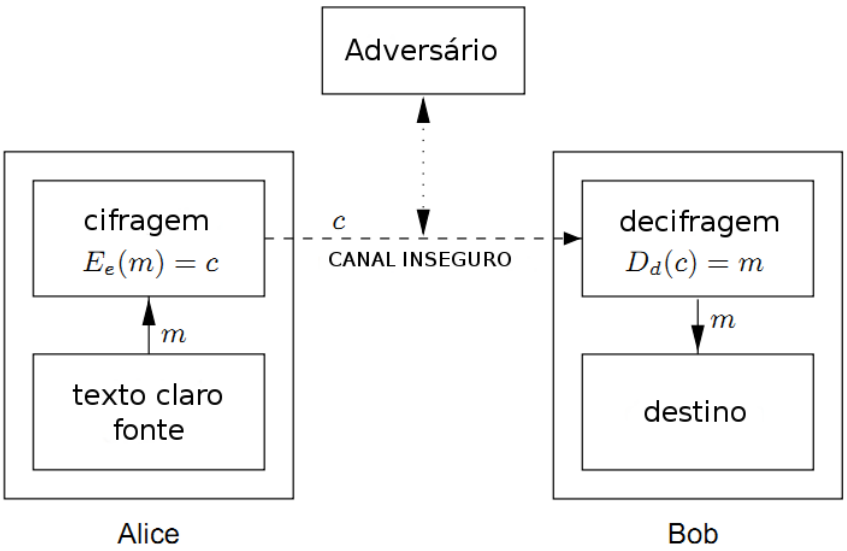

Figura 2.1: Modelo de comunicação utilizando criptografia. Adaptado de Menezes [29].

A Figura 2.1 exemplifica um modelo simplificado da comunicação entre duas entidades, utilizando criptografia.

O sistema utilizado na figura 2.1 tem o objetivo de fornecer sigilo à comunicação entre Alice e Bob. Primeiramente, Alice e Bob escolhem um par de chaves (e, d) através de um canal seguro. Após a definição das chaves entre as partes, quando Alice desejar enviar uma mensagem m para Bob, ela calcula o criptograma $c = E_e(m)$ e envia para Bob, que ao receber c, calcula o texto claro $m = D_d(c)$ para ler a mensagem de Alice [29].

Há dois tipos comuns de sistemas criptográficos: sistemas baseados em chaves simétricas e sistemas baseados em chaves assimétricas. Nas próximas seções iremos tratar da construção e utilidades de ambos.

2.1 Sistema Criptográfico Simétrico

O sistema criptográfico simétrico utiliza algoritmos simétricos para cifragem e decifragem. Algoritmos simétricos, as vezes chamados de algoritmos convencionais, são algoritmos nos quais a chave de cifragem pode ser calculada a partir da chave de decifragem e vice-versa, de forma trivial, em tempo polinomial (veremos os conceitos de complexidade e dificuldade computacional no próximo capítulo), ou seja, possuindo e pode-se facilmente obter d e vice-versa.

Na maioria destes algoritmos, entretanto, as chaves de cifragem e a chave de decifragem são as mesmas, $e = d$. Estes algoritmos, também chamados de algoritmos de chave secreta ou de chave única, requerem que o emissor e receptor escolham uma chave antes de iniciarem uma comunicação segura [42].

A segurança desse sistema está baseada no sigilo da chave. No momento em que a chave for revelada, qualquer entidade terceira pode cifrar e decifrar as mensagens em trânsito.

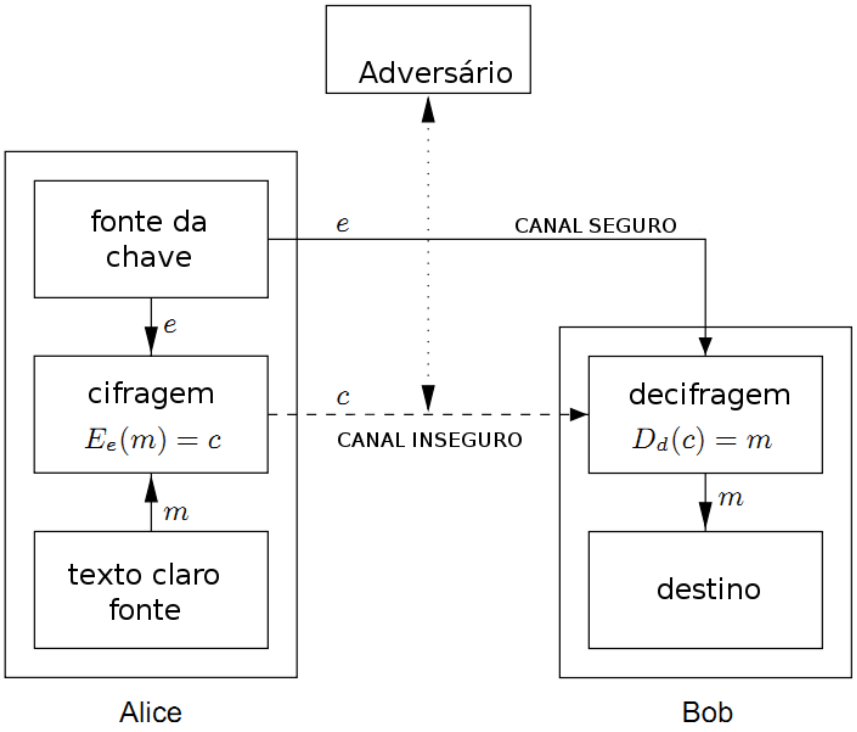

Figura 2.2: Modelo de comunicação utilizando criptografia simétrica. Adaptado de Menezes [29].

A Figura 2.2 mostra uma comunicação utilizando um sistema simétrico de criptografia. Em relação à Figura 2.1, notamos o surgimento de um canal seguro no qual a chave precisa ser distribuída entre os participantes, este canal é fundamental para que Alice e Bob escolham sua chave simétrica e é um dos maiores desafios desse sistema, conhecido como problema da distribuição de chaves [28].

2.1.1 Vantagens

Vantagens do sistema criptográfico simétrico [29]:

- Cifras de chave simétrica podem ser desenvolvidos para atingir altas taxas de rendimento ou processamento. Algumas implementações em *hardware* podem atingir taxas de cifragem que chegam à centenas de *megabytes* por segundo, enquanto a implementação em *software* atinge taxas de processamento de *megabytes* por segundo.
- As chaves dos sistemas criptográficos simétricos costumam ser relativamente pequenas em comparação com as chaves assimétricas.
- Sistemas criptográficos simétricos podem ser empregados pra construção de mecanismos criptográficos incluindo geradores de número pseudoaleatórios, funções de resumo criptográficos, verificação eficiente de integridade, etc.

- Sistemas criptográficos simétricos podem ser combinados para gerar uma cifra mais forte. Utiliza-se produtos de cifra e iterações com transformações simples para construir essas cifras resultantes que continuam simétricas.

- A Criptografia de chave simétrica tem uma extensa história, então acredita-se que muito de sua segurança já foi explorada, embora deva ser considerado que muito do conhecimento nessa área tem sido adquirido após a invenção do computador digital, e, em particular, da criação do DES na década de 1970.

2.1.2 Desvantagens

Desvantagens do sistema criptográfico simétrico [29]:

- Em um sistema de comunicação entre duas partes, a chave deve permanecer em sigilo pelas duas entidades.

- Em uma grande rede, há vários pares de chave para ser gerenciados. Portanto, necessita-se de eficientes métodos de gerência de chaves e o uso de TTP.

- Em um sistema de comunicação entre duas entidades Alice e Bob, a boa prática criptográfica determina que as chaves devem ser trocadas frequentemente, e, talvez, trocadas para cada sessão de comunicação.

- Mecanismos de assinatura digital que estão surgindo através de criptografia de chave simétrica usualmente requerem chaves muito grandes para funções de verificação pública ou o uso de TTP.

2.2 Sistema Criptográfico Assimétrico

O sistema de criptografia assimétrica utiliza uma par de transformações associadas de cifragem/decifragem. Sendo $\{E_e : e \in \mathcal{K}\}$ o conjunto de transformações de cifragem e $\{D_d : d \in \mathcal{K}\}$ o conjunto de transformações de decifragem, em que $\mathcal{K}$ o espaço de chaves possíveis. E principalmente supõe que, para cada par, existe a propriedade de ser computacionalmente inviável calcular d tendo o conhecimento de e, onde $D_d = E_e^{-1}$.

Tendo este esquema em mente, consideremos a ilustração da comunicação entre Alice e Bob, Figura 2.3. Bob seleciona um par de chaves (e, d), envia a chave de cifragem e (chamada de chave pública) para Alice através de qualquer canal e mantém a chave de decifragem d (chamada de chave privada) em segredo. Em seguida, Alice pode se comunicar de forma segura com Bob aplicando a transformação da chave pública na sua mensagem m e obtendo o criptograma c tal que $c = E_e(m)$, supondo autêntica a chave pública recebida de Bob. Bob decifra o criptograma c aplicando a transformação inversa D_d.

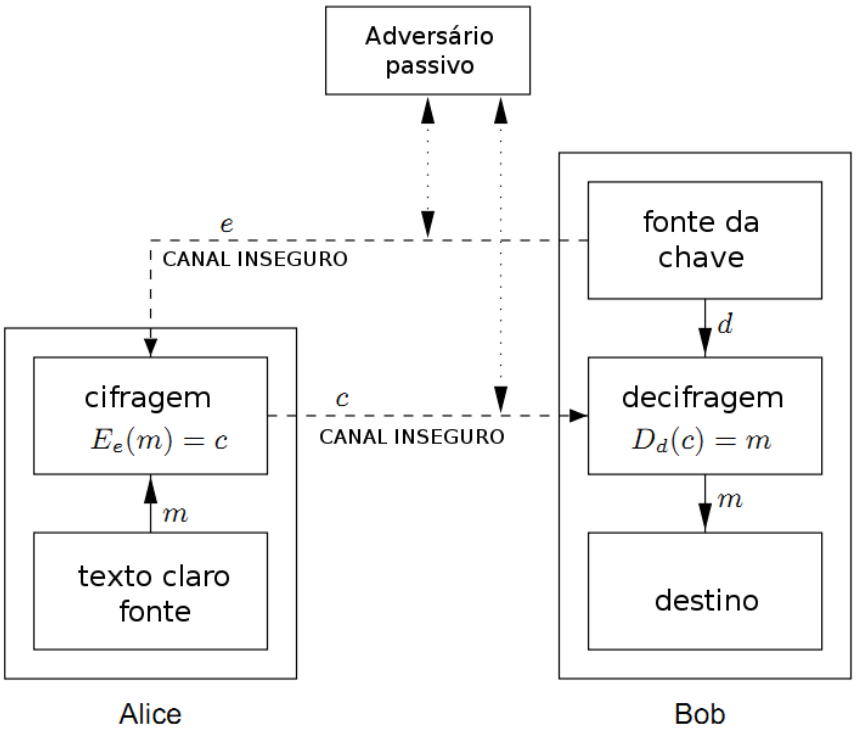

Figura 2.3: Modelo de comunicação utilizando criptografia assimétrica. Adaptado de Menezes [29].

Podemos notar como a Figura 2.3 difere da Figura 2.2 no sistema simétrico. Neste cenário assimétrico, a chave de cifração é transmitida para Alice através de um canal inseguro. Este canal pode ser o mesmo pelo qual o criptograma também será transmitido.

Uma vez que que a chave de cifragem não precisa ser mantida em segredo, qualquer outra entidade poderá, posteriormente, cifrar mensagens, para Bob, as quais apenas Bob, portador da chave privada, poderá decifrá-las, supondo autêntica a chave pública de Bob.

Como modelo mais didático, pode-se ver a criptografia de chave pública não como pares de chaves de Bob, mas sim uma chave mantida em segredo (chave privada) e um cadeado aberto (chave pública). O cadeado público pode ter quantas cópias seja necessário e pode ser analisado por qualquer entidade interessada. A cifragem seria o ato de colocar uma mensagem dentro de um baú seguro e trancá-lo com o cadeado de Bob, deste modo, apenas Bob possui a chave que abrirá o cadeado.

A criptografia de chave pública assume o conhecimento e autenticidade da chave pública e considera que a derivação da chave privada a partir da chave pública é um problema intratável, ou seja, assume a existência de funções unidirecionais e autenticação prévia de chaves públicas.

É importante resaltar que um sistema criptográfico de chave pública nunca poderá fornecer segurança incondicional. Isto ocorre pois caso o adversário tenha acesso a um criptograma y, ele pode cifrar exaustivamente todos os possíveis textos claros com a chave pública E_e até que encontre um único texto claro x tal que $E_e(x) = y$, então este x é a decifração de y. Portanto, estudamos a segurança computacional de criptografia de chave pública [48].

2.2.1 Funções Unidirecionais

Grosso modo, funções unidirecionais são funções que são fáceis de calcular, mas difíceis de inverter (de modo geral) [17].

Todo o sucesso da criptografia assimétrica está baseado na existência de funções unidirecionais, obter D_d a partir de E_e é considerado um problema intratável.

Uma função injetiva $f : X \to Y$ é dita estritamente unidirecional se é observado:

- Existe um método eficiente para calcular $f(x) \forall x \in X$, mas não há maneira eficiente de obter x da relação $y = f(x) \quad \forall y \in f(X)$ [7]

Usando maior rigor matemático, ainda não há prova de que funções unidirecionais existam, nem real evidência de que possam ser construídas, diz Schneier [42] sobre a publicação de Even [14]. Ainda assim, várias funções se comportam como unidirecionais, podem computá-las de forma rápida, mas, até então, não é conhecido maneira eficiente de obter sua inversa.

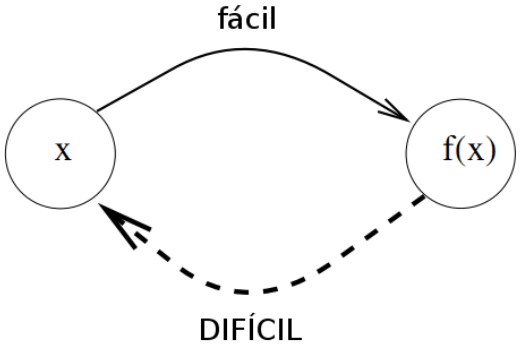

Figura 2.4: Diagrama de uma função dita uniderecional. Adaptado de Goldreich [17].

Uma das melhores formas de exemplificar funções unidirecionais é utilizar exemplos palpáveis do dia a dia.

Misturar as tintas amarela e azul para obter a tinta verde pode ser comparado a uma função unidirecional, pois o processo de misturar as tintas é fácil, entretanto, tentar separar as tintas é impossível ou intratável. A aritmética modular é uma área da matemática rica em funções unidirecionais [46].

Para sedimentar a compreensão do conceito de funções unidirecionais, podemos utilizar um exemplo mais prático proposto por Bauer [7]. Em uma lista telefônica, os dados são compostos pela dupla [Nome] - [número de telefone] em ordem alfabética. Dado uma lista telefônica com grande quantidade dados, tendo o nome de uma pessoa, é possível encontrar seu telefone em tempo hábil, mas a operação inversa, encontra o nome de uma pessoa tendo apenas o seu telefone, é uma tarefa difícil devido a desordem dos telefones. Assim, seria preciso comparar número por número até encontrar o seu proprietário.

O estudo dessas funções é fundamental para a segurança computacional, pois há a possibilidade de se descobrir novos métodos que solucionam esses problemas, que até então acredita-se ser unidirecional, em tempo polinomial.

2.2.2 Vantagens

Vantagens do sistema criptográfico assimétrico [29]:

- Apenas a chave privada deve ser mantida em segredo. Entretanto, a certificação de chaves públicas é necessária.
- A administração de chaves em uma rede escalável requer a presença de um TTP funcionalmente confiável ao contrário de um TTP incondicionalmente confiável. Dependendo do modo de operação, o TTP pode apenas ser exigido em uma forma *off-line*.
- Dependendo do modo de operação, o par (chave privada, chave pública) pode permanecer sem modificação por grande período de tempo, isto é, sendo usado por várias sessões e validade de vários anos, desde que seja preservado o sigilo da chave privada.
- Muitos esquemas de chave pública fornecem eficientes mecanismos de assinatura digital. A chave usada para descrever a função de verificação pública é geralmente muito menor do que a de criptografia simétrica para fins equivalentes.
- Em uma grande rede, o número de chaves necessárias é consideravelmente menor do que no cenário da criptografia simétrica.

2.2.3 Desvantagens

Desvantagens do sistema criptográfico assimétrico [29]:

- As taxas de processamento para os métodos criptográficos de chave pública são extremamente menores do que o melhor esquema de chave simétrica.
- Os tamanhos da chave são geralmente maiores do que os necessários para criptografia simétrica e o tamanho da assinatura de chave pública é maior, fornecendo autenticação de origem de dados por técnicas de chave simétrica.
- Ainda nenhum método de chave pública se provou seguro. O método de criptografia assimétrica mais difundido atualmente tem sua segurança baseada na dificuldade de um pequeno conjunto de problemas em Teoria dos Números, chamado de problema da fatoração de inteiros.
- Criptografia de chave pública tem uma história ainda recente em comparação com a Criptografia simétrica. O que nos faz acreditar que ainda houve pouca exploração a respeito de sua segurança.

2.3 Combinação dos Sistemas

Dada os cenários de vantagens e desvantagens das duas classes de sistemas criptográficos, podemos enumerar as vantagens que possam ser combinadas.

Técnicas de criptografia de chave pública podem ser usadas para estabelecer uma chave de sistema criptográfico simétrico a ser usado pelas entidades Alice e Bob. Neste cenário, Alice e Bob tiram proveito da validade longa das chaves pública/privada e da eficiência do esquema simétrico para cifrar grande quantidade de dados. A cifragem e decifragem dos dados são os processos que custam maior tempo e o esquema de chave pública para estabelecer a chave simétrica é apenas uma pequena fração do processo criptográfico. Os sistemas criptográficos atuais exploram os pontos fortes de ambos os sistemas:

- Criptografia de chave pública é a melhor maneira de se comunicar através de um canal inseguro e até então a única forma de fornecer um esquema para assinatura digital (particularmente a irretratabilidade).
- Criptografia de chave simétrica é muito eficiente para cifrar e garantir integridade de grandes volumes de dados.

Capítulo 3

Criptografia de Chave Pública

3.1 Complexidade Computacional

Um problema computacional pode ser visto como uma coleção infinita de instâncias parametrizada em conjunto com uma solução para cada instância. A sequência de entrada para um problema computacional é referida como uma instância do problema, e não deve ser confundido com o problema em si. Na teoria da complexidade computacional, um problema parametrizável se refere à questão abstrata para ser resolvido. Em contraste, uma instância deste problema é uma expressão concreta, que pode servir como entrada para um problema de decisão.

Para uma definição precisa do que significa resolver um problema utilizando uma determinada quantidade de tempo e espaço, um modelo computacional tal como a máquina de Turing determinística é utilizado. O tempo exigido por uma máquina de Turing determinística M na entrada x é o número total de transições de estado, ou etapas, que a máquina faz antes de parar e responder com a saída (SIM ou NÃO). Diz-se que a máquina de Turing M opera dentro do tempo $f(n)$, se o tempo exigido por M em cada entrada de comprimento n é no máximo $f(n)$. Um problema de decisão A pode ser resolvido em tempo $f(n)$ se existe uma operação da máquina de Turing em tempo $f(n)$ que resolve o problema. Como a teoria da complexidade está interessada em classificar problemas com base na sua dificuldade, definem-se conjuntos de problemas com base em alguns critérios.

Definições análogas podem ser feitas para os requisitos de espaço. Embora o tempo e o espaço sejam as mais conhecidas métricas de complexidade, qualquer medida de complexidade pode ser vista como um recurso computacional.

3.1.1 Casos Melhor, Pior e Médio

O melhor, o pior e o caso médio de complexidade referem-se a três maneiras diferentes de medir a complexidade de tempo (ou qualquer outra medida de complexidade) de entradas diferentes do mesmo tamanho. Uma vez que algumas entradas de tamanho n podem ser mais rápidas para resolver do que outras, definimos as seguintes complexidades:

- Complexidade no melhor caso: esta é a complexidade de resolver o problema para a melhor entrada de tamanho n.

- Complexidade no pior caso: esta é a complexidade de resolver o problema para a pior entrada de tamanho n.
- Complexidade no caso médio: esta é a complexidade de resolver o problema na média. Essa complexidade só é definida com relação a uma distribuição de probabilidade sobre as entradas. Por exemplo, se todas as entradas do mesmo tamanho são consideradas equiprováveis, a complexidade do caso médio pode ser definida com relação à distribuição uniforme sobre todas as entradas de tamanho n.

Por exemplo, considere o algoritmo de ordenação *QuickSort*, que resolve o problema de ordenar uma lista de inteiros que é dada como entrada. O pior caso é quando a entrada já está ordenada ou está em ordem inversa, e o algoritmo leva tempo $O(n^2)$ para este caso. Se assumirmos que todas as permutações possíveis da lista de entrada são igualmente prováveis, o tempo médio necessário para a ordenação é $O(n \log n)$. O melhor caso ocorre quando cada pivô divide a lista pela metade, também em tempo $O(n \log n)$.

3.1.2 Limites Superior e Inferior

Para classificar o tempo de computação (ou recursos semelhantes, como o consumo de espaço), é necessário provar os limites superiores e inferiores sobre a quantidade mínima de tempo exigida pelo algoritmo mais eficiente para resolver um determinado problema.

A complexidade de um algoritmo é geralmente entendida como a sua complexidade de pior caso, a menos que seja especificado o contrário. A análise de um determinado algoritmo cai sob o campo de análise de algoritmos. Esta análise é realizada de maneira assintótica, para entradas a partir de um determinado valor.

Para demonstrar que existe um limite superior para a função $g(n)$ é necessário mostrar que existe uma função $f(n)$ que domina assintoticamente $g(n)$. Assim, $f(n)$ domina $g(n)$ se existem as constantes positivas c e n_0 tais que para qualquer $n \geq n_0$, $g(n) \leq c \cdot f(n)$.

O limite superior é indicado pela notação O e desconsidera fatores constantes e termos menores. Isso faz com que os limites independam dos detalhes específicos do modelo computacional utilizado. Por exemplo, $T(n) = 7n^2 + 15n + 40$ tem limite superior, ou complexidade, $O(n^2)$.

Para demostrar um limite assintótico inferior Ω para a função $g(n)$, usa-se ideia análoga. Deve existir uma função $f(n)$ tal que para qualquer $n \geq n_0$, $0 \leq c \cdot f(n) \leq g(n)$. Provar Limites inferiores é mais difíceis, uma vez que limites inferiores fazem uma declaração sobre todos os possíveis algoritmos que resolvem um determinado problema, inclusive algoritmos ainda não conhecidos.

Limites inferiores geralmente têm maior aplicação quando usados como complemento para definição de limites restritos.

A figura 3.1 ilustra a comparação entre o crescimento assintótico de algumas funções comuns na análise de algoritmos.

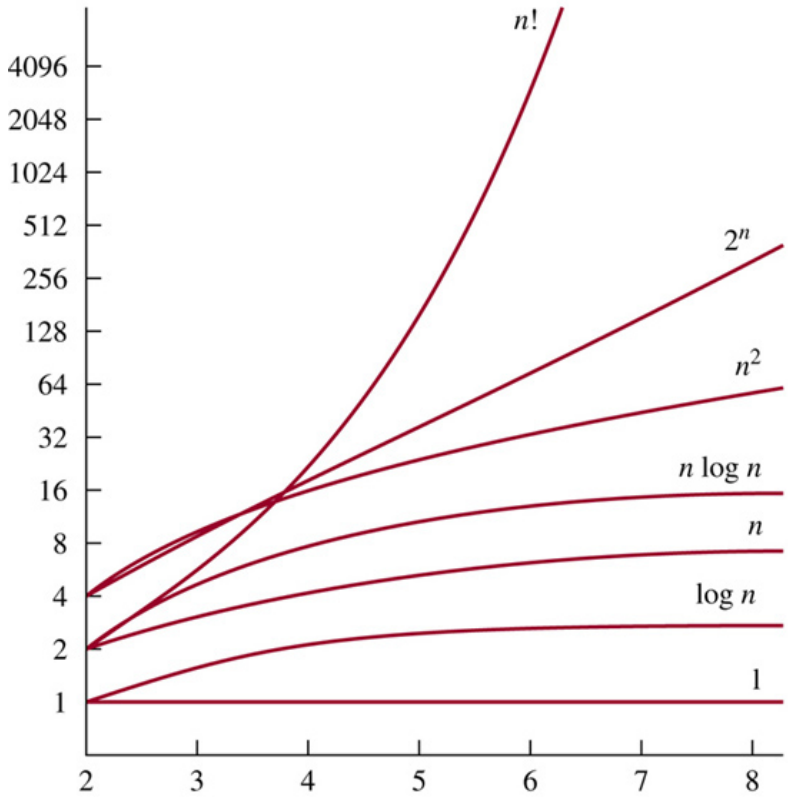

Figura 3.1: Comparação assintótica entre crescimento das funções. Retirado de Hein [19].

3.2 Teoria dos Números

A teoria dos números já foi vista como um tópico belo mas sem grande utilidade prática da matemática. Hoje, algoritmos de teoria dos números são amplamente utilizados devido a invenção de esquemas criptográficos baseados em números primos grandes. A viabilidade desses esquemas se torna possível através da habilidade de encontrar números primos grandes de maneira rápida e sua segurança repousa na intratabilidade de fatorar o produto desses primos [10]. Nesta seção expressaremos alguns conceitos da teoria dos números que serão necessário para a compreensão dos métodos criptográficos apresentados.

3.2.1 Números Primos

Existem algumas pistas de que os egípcios expressaram alguma atenção sobre os números primos, mas acredita-se que foram os antigos gregos (Euclides e Eratóstenes), por volta de 300 a.C., quem produziu estudos documentados.

Os números primos são os blocos de construção para todos os inteiros. Assim como os átomos são as estruturas elementares das moléculas e da matéria, os números primos formam uma espécie de tabela periódica da aritmética, este fato faz crer que eles também merecem um estudo particular.

Deste modo, precisa-se de maneiras de encontrar primos e métodos para construí-los, identificá-los e, se possível, quantificar suas ocorrências. Algumas destas missões levaram milhares de anos para se completar e algumas ainda estão em aberto [15].

Um número $p \in \mathbb{N}$ é denominado primo, se $p > 1$ e se seus únicos divisores são p e 1. Indicamos por

$$\mathbb{P} = \{p \in \mathbb{N} \ : p \text{ é primo}\}$$

o conjunto de todos os números primos. Podemos dizer então que

$$p \in \mathbb{P} \iff (\forall\, a, b \in \mathbb{N} \ : \ p = ab \Rightarrow a = p \text{ e } b = 1 \text{ ou } a = 1 \text{ e } b = p) \text{ [8]}.$$

Para um primo p, $\mathbb{Z}_p$ é o grupo aditivo módulo p, que consiste no conjunto $\{0, \ldots, p-1\}$ e a operação de adição módulo p. Neste grupo, todos os elementos, exceto a identidade 0, têm ordem p.

$\mathbb{Z}_p^*$ denota o grupo multiplicativo módulo p, consiste no conjunto $\{1, \ldots, p-1\}$ e a operação de multiplicação módulo p. Ambos são grupos fechados e cíclicos.

Um número $n > 1$ é dito **composto** se ele não é primo. Dois números $a, b \in \mathbb{Z}$ chamam-se **coprimos** (ou primos entre si) se e somente se $\text{mdc}(a, b) = 1$.

Lema de Euclides: sejam, a, b e $c \in \mathbb{Z}$. Se $a|bc$ e $\text{mdc}(a, b) = 1$, então $a|c$. Através deste Lema, obtemos o seguinte resultado fundamental dos números primos.

Seja $p \in \mathbb{P}$, então

$$\forall\, a, b \in \mathbb{N} \ : \ p|ab \Rightarrow p|a \text{ ou } p|b$$

Em palavras: um número primo divide um produto, somente se ele divide um de seus fatores.

Distribuição dos números primos

Existem infinitos números primos. Esta afirmação pode ser demonstrada por uma rápida redução ao absurdo. Suponha o contrário, suponha que existe um número finito de primos:

$$p_1,\ p_2,\ p_3,\ p_4,\ \cdots\ ,\ p_n$$

Esta é uma lista completa dos primos. Agora geramos um número C tal que seja o produto de todos os primos da lista, $C = p_1 \cdot p_2 \cdot p_3 \cdot \ldots \cdot p_n$. Pelo teorema fundamental da aritmética 3.2.2, sabemos que um número pode ser decomposto em fatores primos únicos, ou seja, os fatores de C são todos os primos da lista completa e esta expressão é única. Agora suponha $Q = C + 1$:

$$Q = C + 1 = p_1 \cdot p_2 \cdot p_3 \cdot \ \ldots \ \cdot p_n \ + 1$$

Então, Q pode ser enquadrado em apenas dois casos:

1. Q é primo. Então a lista de primos não é completa (absurdo).
2. Q é composto. Então Q deve ser divisível por todos seus fatores primos ($p_i \mid Q \quad \forall\, i \in [1, n]$), o que não é verdade, pois qualquer divisão por p_i deixa resto 1. Logo a lista não é completa (absurdo).

Há várias outras provas pra a infinidade de primos [40], mas o primeiro registro foi elucidado por Euclides [13] por volta de 300 a.C., utilizava a mesma ideia apresentada, mas em um tempo onde ainda não existia a Álgebra, toda a aritmética era realizada através da Geometria, segmentos de reta e circunferências. Um fato fascinante, pois a Matemática continua tão sólida quanto era há 2300 anos.

Seja $\pi(n)$ a função que retorne a quantidade de números primos no intervalo $[2, n]$. O **Teorema do Número Primo** mostra que

$$\pi(n) \sim \frac{n}{\ln n},$$

isto é, as funções possuem comportamento assintótico semelhante.

A partir deste resultado da função π, também podemos interpretar que o espaçamento médio entre os números primos é igual a $\ln n$. Bertrand e Tschebycheff foram mais além e apresentaram um limite superior para esse espaçamento entre os primos, provaram que para qualquer natural $n \geq 2$, existe pelo menos um primo p tal que $n < p < 2n$ [40].

Estes são ótimos resultado para o contexto da criptografia, pois a distribuição mostra que os números primos não só são infinitos, como também extremamente abundantes e regulares, não há grandes espaços (*gaps*) sem números primos, o que torna uma operação viável e desafiadora encontrar primos grandes.

Como exemplo figurado, um espaço numérico obtido com 512 *bits* contém aproximadamente 10^{151} números primos. O número de átomos no universo é de 10^{77}. Se cada átomo precisasse de 1 bilhão de números primos a cada micro-segundo desde o início do universo até os dias de hoje, precisaria de apenas 10^{109} primos; ainda assim restariam aproximadamente 10^{151} primos [42], ou seja, a quantidade de primos utilizados é insignificante, ínfima, em relação aos primos possíveis dentro de 512 *bits*.

3.2.2 Teorema Fundamental da Aritmética

1. Todo número $1 < n \in \mathbb{N}$ pode ser decomposto em fatores primos, quer dizer, existem $p_1, p_2, \ldots, p_3 \in \mathbb{P}$ tais que
$$n = p_1 \cdot p_2 \cdot \ldots \cdot p_r$$

2. Se $p_1 \cdot p_2 \cdot \ldots \cdot p_r = q_1 \cdot q_2 \cdot \ldots \cdot q_s$ com $p's$ e $q's \in \mathbb{P}$ e se $p_1 \leq p_2 \leq \ldots \leq p_r$ tal como $q_1 \leq q_2 \leq \ldots \leq q_r$, então
$$r = s \wedge p_1 = q_1, p_2 = q_2, \ldots, p_r = q_s$$

Demonstração:

1. Se $n = p$ é um número primo, a afirmação fica clara ($r = 1$).

   Mostramos a afirmação para n composto, supondo sua veracidade para todo $m \in \mathbb{N}$ com $1 < m < n$.

   Seja $S = \{t \in \mathbb{N} | 1 < t < n\}$.

   Como $n > 1$ sabemos que $n \in S$, isto é, $S \neq \emptyset$.

   Pelo princípio da indução, existe um $p_1 \in S$ minimal. É claro que p_1 é primo e temo $m \in \mathbb{N}$ com $n = p_1 \cdot m$.

Como $p_1 > 1$ e n não é primo, segue $1 < m < n$.

Como a afirmação já é válida para este m, existem $p + 2, p_3, \ldots, p_r \in \mathbb{P}$ tais que $m = p_2 \cdot p_3 \cdot \ldots \cdot p_r$.

Segue, como afirmado:

$$n = p_1 \cdot m = p_1 \cdot p_2 \cdot p_3 \cdot \ldots \cdot p_r$$

2. Suponha $p_1 \cdot p_2 \cdot \ldots \cdot p_r = q_1 \cdot q_2 \cdot \ldots \cdot q_s$ com $p's$ e $q's \in \mathbb{P}$ e se $p_1 \leq p_2 \leq \ldots \leq p_r$ tal como $q_1 \leq q_2 \leq \ldots \leq q_r$.

   Temos $p_1 | q_1 \cdot q_2 \ldots \cdot q_s$

   Aplicando-se repetidas vezes o resultado do Lema de Euclides, concluímos que p_1 tem que dividir algum dos fatores $q_1, q_2, \ldots, q_s$.

   Logo existe $k (k \leq 1 \leq s)$ com $p_1 | q_k$.

   Como p_1 e q_k são primos, temos $p_1 = q_k \geq q_1$.

   Da mesma forma, $q_1 | p_l$ com $(1 \leq l \leq r)$ e segue que $q_1 = p_l \geq p_1$

   Assim, $p_1 = q_1$. Agora de $p_1 \cdot p_2 \cdot \ldots \cdot p_r = q_1 \cdot q_2 \cdot \ldots \cdot q_s$ ocorre

   $$p_2 \cdot \ldots \cdot p_r = q_2 \cdot \ldots \cdot q_s$$

   Por indução concluímos $r - 1 = s - 1$, isto é, $r = s, p_2 = q_2, p_3 = q_3, \ldots, p_r = q_s$. Adicionando o resultado $p_1 = q_1$, alcançamos a afirmação inicial.

3.2.3 Algoritmo Euclidiano

Sejam $a, b \in \mathbb{N}$ dois números, pelo menos um deles diferente de zero, o máximo divisor comum entre a e b é o número natural

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

definido pelas duas propriedades:

1. $d|a$ e $d|b$ (i. e. d divide ambos a e b).
2. Se algum $c \in \mathbb{N}$ divide ambos a e b, então $c|d$.

Podemos definir d, utilizando uma equação diofantina, através do teorema [27]:
Sejam a e $b \in \mathbb{Z}$ não ambos igual a zero e seja $d = \text{mdc}(a, b)$. Então existem $x, y \in \mathbb{Z}$ tais que

$$ax + by = d.$$

O Algoritmo 1 mostra o método estendido de Euclides para obter o máximo divisor comum entre a e b.

Algorithm 1 Algoritmo euclidiano

Input: a e $b \in \mathbb{N}$

Output: (r, s, t) tais que $r = \text{mdc}(a, b)$ e $sa + tb = r$

$a_0 \leftarrow a$
$b_0 \leftarrow b$
$t_0 \leftarrow 0$
$t \leftarrow 1$
$s_0 \leftarrow 1$
$s \leftarrow 0$
$q \leftarrow \lfloor \frac{a_0}{b_0} \rfloor$
$r \leftarrow a_0 - qb_0$
while $r > 0$ **do**
    $temp \leftarrow (t_0 - qt) \bmod a$
    $t_0 \leftarrow t$
    $t \leftarrow temp$
    $temp \leftarrow s_0 - qs$
    $s_0 \leftarrow s$
    $s \leftarrow temp$
    $a_0 \leftarrow b_0$
    $b_0 \leftarrow r$
    $q \leftarrow \lfloor \frac{a_0}{b_0} \rfloor$
    $r \leftarrow a_0 - qb_0$
end while
return (r, s, t)

3.2.4 Congruência

Seja $n \in \mathbb{N}$ um número fixo. Dois números a, $b \in \mathbb{Z}$ são ditos congruentes módulo n, se $(a - b)$ é múltiplo de n.

Formalmente:

$$a \equiv b \bmod n.$$

Assim, $a \equiv b \bmod n \iff n|a - b$, ou mais, a divisão de n por a deixa resto b [27].

Propriedades fundamentais da congruência:

1. $a \equiv a \bmod n$
2. Se $a \equiv b \bmod n$, então $b \equiv a \bmod n$
3. Se $a \equiv b \bmod n$ e $b \equiv c \bmod n$, então $a \equiv c \bmod n$
4. Se $a \equiv b \bmod n$ e $c \equiv d \bmod n$, então $a + c \equiv b + d \bmod n$ e $ac \equiv bd \bmod n$

Se $\text{mdc}(a, n) = 1$, então existe um inteiro denotado por $a^{-1} \in \mathbb{Z}_n$ tal que

$$a \cdot a^{-1} \equiv 1 \bmod n.$$

O inteiro a^{-1} é chamado de inverso multiplicativo de a módulo n.

3.2.5 Teorema Chinês do Resto

O teorema chinês do resto é um método para solucionar certos sistemas de congruência. Suponha $m_1, m_2, \ldots, m_r$ coprimos entre si, ou seja, $\text{mdc}(m_i, m_j) = 1$ se $i \neq j$

Suponha $a_1, a_2, \ldots, a_r$ inteiros e considere o seguinte sistema de congruência:

$$\begin{aligned} x &\equiv a_1 \pmod{m_1} \\ x &\equiv a_2 \pmod{m_2} \\ &\vdots \\ x &\equiv a_r \pmod{m_r} \end{aligned}$$

O teorema garante que existe uma única solução módulo $M = m_1 \times m_2 \times \ldots \times m_r$ [48].

Algorithm 2 Algoritmo para obter o inverso multiplicativo (a, b). Também chamado de Algoritmo de Euclides Estendido.

Input: a e $b \in \mathbb{N}$
Output: t tal que $t \cdot b = 1 \pmod{a}$

1: $a_0 \leftarrow a$
2: $b_0 \leftarrow b$
3: $t_0 \leftarrow 0$
4: $t \leftarrow 1$
5: $q \leftarrow \lfloor \frac{a_0}{b_0} \rfloor$
6: $r \leftarrow a_0 - qb_0$
7: **while** $r > 0$ **do**
8: $\quad temp \leftarrow (t_0 - qt) \bmod a$
9: $\quad t_0 \leftarrow t$
10: $\quad t \leftarrow temp$
11: $\quad a_0 \leftarrow b_0$
12: $\quad b_0 \leftarrow r$
13: $\quad q \leftarrow \lfloor \frac{a_0}{b_0} \rfloor$
14: $\quad r \leftarrow a_0 - qb_0$
15: **end while**
16: **if** $b_0 \neq 1$ **then**
17: $\quad$ **return** b não tem inverso mod a.
18: **else**
19: $\quad$ **return** t
20: **end if**

3.2.6 Pequeno Teorema de Fermat

O pequeno teorema de Fermat diz que se p é um número primo, então para qualquer inteiro a, $a^p - a$ é múltiplo de p. Utilizando congruência:

$$a^p \equiv a \pmod{p}.$$

Outro resultado conhecido para o Pequeno Teorema de Fermat pode ser trivialmente obtido dividindo a congruência por a:

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

3.2.7 Teorema de Euler

O Teorema de Euler é tido como uma generalização do pequeno teorema de Fermat e diz que se n e $a \in \mathbb{N}$ são coprimos, isto é, $\text{mdc}(a, n) = 1$, então

$$a^{\varphi(n)} \equiv 1 \pmod{n}.$$

A função $\varphi(n)$, também chamada de função totiente, denota a quantidade de inteiros positivos menores ou iguais a n que são coprimos com n. Formalmente, $\varphi(n) = |\{k \in \mathbb{N} : 1 \leq k \leq n \text{ e } \text{mdc}(k, n) = 1\}|$.

Esta função tem importante aplicação em Criptografia e também possui resultado simples para números primos. Sendo um primo p, ele é divisível apenas por 1 e por p, logo a quantidade de naturais coprimos e menores que p é $\varphi(p) = p - 1$.

Uma importante propriedade da função totiente é que $\varphi(a \cdot b) = \varphi(a) \cdot \varphi(b)$. Assim, como todo número composto pode ser fatorado em um produto único de primos, podemos calcular facilmente a função totiente de qualquer número, conhecendo a sua fatoração.

3.2.8 Símbolos de Legendre e Jacobi

O matemático francês Adrien Marie Legendre continuou o desenvolvimento nos estudos de Euler a respeito de resíduos quadráticos [8].

É dito que $a \in \mathbb{Z}_p$ é um **resíduo quadrático** módulo um p primo $\iff \exists\, y \in \mathbb{Z}_p$ tal que $y^2 \equiv a \pmod{p}$.

Critério de Euler

Partindo do pequeno teorema de Fermat, podemos chegar ao seguinte resultado quanto à residualidade de a, conhecido como critério de Euler.

Suponha que $a \equiv y^2 \pmod{p}$. Se p é primo, então $a^{p-1} \equiv 1 \pmod{p}$, $\forall a \not\equiv 0 \pmod{p}$. Assim,

$$a^{p-1} \equiv (y^2)^{p-1} \equiv 1 \pmod{p}.$$

Agora suponha que $a^{(p-1)/2} \equiv 1 \pmod{p}$. Seja b um elemento primitivo módulo p, então $a \equiv b^i \pmod{p}$ para $i \in \mathbb{Z}$. Logo,

$$a^{(p-1)/2} \equiv (b^i)^{(p-1)/2} \pmod{p}.$$

Como b tem ordem $p-1$, então $(p-1) \mid i(p-1)/2$. Portanto, i é par e as possíveis raízes quadradas de a são $\pm b^{i/2}$ mod p. Assim,

$$a \text{ é resíduo quadrático módulo } p \iff a^{(p-1)/2} \equiv 1 \pmod{p}.$$

Símbolo de Legendre

Para $a \in \mathbb{Z}$, p primo ímpar, o símbolo de Legendre $\left(\frac{a}{p}\right)$ é:

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{se } a \equiv 0 \pmod{p} \\ 1 & \text{se } a \text{ é resíduo quadrático módulo } p \\ -1 & \text{se } a \text{ é resíduo não-quadrático módulo } p \end{cases}$$

Pode-se encarar o símbolo de Legendre como o problema de decisão em relação a $\left(\frac{a}{p}\right)$ que tem-se afirmativo para resíduo quadrático e negativo para resíduo não-quadrático.

Símbolo de Jacobi

O símbolo de Jacobi é a generalização para o símbolo de Legendre. Tem importante aplicação em testes de primalidade e em fatoração de inteiros, portanto fundamental na criptografia.

Suponha $n \in \mathbb{N}$ com fatoração $n = \prod_{i=1}^{k} p_i^{e_i}$, com $p_i \in \mathbb{P}$ e $e_i \neq 0 \ \forall \ i$. Seja $a \in \mathbb{Z}$, o símbolo de Jacobi $\left(\frac{a}{n}\right)$ é definido por:

$$\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{e_i}$$

3.3 Algoritmo RSA

Criado por R.L. Rivest, A. Shamir e L. Adleman, o RSA é um método de Criptografia utilizado para garantir privacidade ou assinatura de documentos digitais [41]. Este método provê a primeira implementação de um Sistema Criptográfico de Chave Pública, conceito elegante projetado por Diffie e Hellman [12].

O sistema criptográfico de chave pública RSA está fortemente baseado na enorme diferença entre a facilidade de encontrar números primos grandes (100 a 200 dígitos) e na dificuldade de fatorar o produto desses primos [10]. Devemos, entretanto, fazer uma observação quanto esta afirmação. Dizer que a segurança do RSA baseia-se completamente no problema da fatoração de números grandes não é uma afirmação correta. Devemos afirmar, com maior rigor, que é conjecturado que sua segurança depende do problema da fatoração de números grandes, ou que o problema da fatoração fornece uma quota superior para o custo de ataque matemático ao RSA.

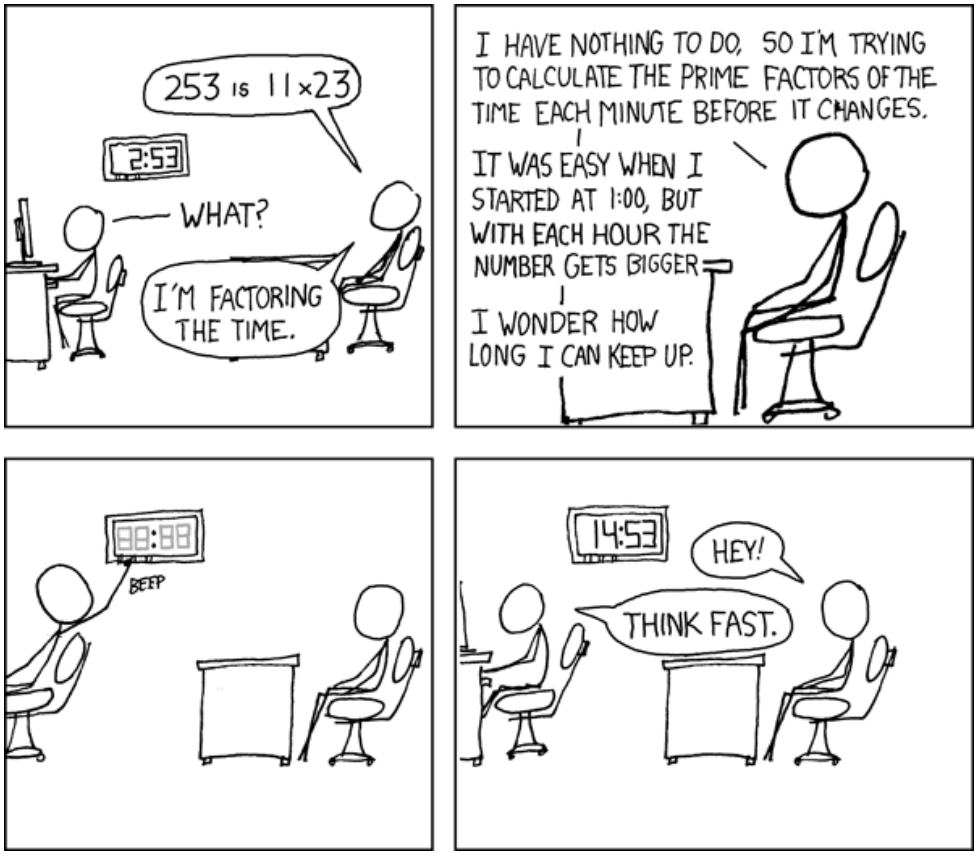

Figura 3.2: Ilustração da dificuldade da fatoração de números grandes. Retirado de xkcd [50].

Não há prova matemática de que a fatoração de n é a única abordagem para se derivar a chave privada na criptoanálise do RSA. Provar incondicionalmente que o RSA é seguro implica provar a existência de funções unidirecionais e, por consequência, estabelecer a relação $P \neq NP$.

O algoritmo em 3.1 exibido a seguir é chamado de RSA versão livro texto, pois o apresenta de forma simplificada e sem qualquer otimização, assim permitindo uma abordagem mais didática.

Tabela 3.1: Algoritmo RSA.

1. Gere dois primos grandes aleatórios e distintos p e q
2. $n \leftarrow pq$ e $\varphi(n) \leftarrow (p-1)(q-1)$
3. Escolha b aleatório com $(1 < b < \varphi(n))$ tal que $\text{mdc}(b, \varphi(n)) = 1$
4. $a \leftarrow b^{-1} \bmod \varphi(n)$

A chave pública (cifragem) é (b, n)
A chave privada (decifragem) é (a, n)

3.3.1 Esquema de Cifração

O esquema de criptografia de chave pública pode ser usado para cifrar mensagens enviadas entre duas partes que se comunicam de modo que um adversário que tem acesso ao tráfego dos criptogramas não possa decifrá-los [10]. O esquema de cifração, portanto, é utilizado para prover sigilo de conteúdo na comunicação. Para cifrar uma mensagem m com este método, usando a chave de cifração pública (e, n), proceda da seguinte forma.

Primeiramente, represente a mensagem como um inteiro com valor entre 0 e $n-1$. Seccione a mensagem em uma série de blocos representados também como inteiros, seguindo qualquer padrão de representação. Com este procedimento, obtém-se a mensagem numa forma numérica necessária para a cifragem.

Cifre elevando a mensagem à potência e-ézima módulo n. Ou seja, o resultado (o criptograma c) é o resto da divisão por n. Portanto, algoritmo de cifragem E:

$$E(m) = c = m^e \pmod{n} \tag{3.1}$$

Para decifrar o criptograma, eleve-o à potência d modulo n. Assim, o algoritmo de decifragem D é:

$$D(c) = m = c^d \pmod{n} \tag{3.2}$$

Note que a cifragem não aumenta o tamanho da mensagem, tanto o criptograma quanto a mensagem são inteiros e têm tamanho de 0 a $n-1$.

$$m, c \in \mathbb{Z}_n \tag{3.3}$$

A chave de cifragem (pública) é, então, o par de inteiros (e, n). Analogamente, a chave de decifragem (privada) é o par de inteiros (d, n).

Para a escolha do par de chaves, primeiro se calcula n como o produto de dois primos p e q.

$$n = p \cdot q. \tag{3.4}$$

Estes primos são grandes e aleatórios. Apesar de n passar a ser público, os fatores p e q se mantêm guardados devido a grande dificuldade de se fatorar n. Assim, também esconde o modo como d pode ser derivado de e [41].

Escolha um e inteiro aleatório que seja coprimo com $\varphi(n)$. Ou seja, que satisfaça:

$$\varphi(n) = (p-1) \cdot (q-1)$$
$$1 < e < \varphi(n) \ \mathrm{mdc}(e, \varphi(n)) = 1$$

Usualmente nas implementações mais conhecidas, utiliza-se, para o valor de e, primos padrões como 3, 5, 17, 257 ou 65537 [9]. Estes são chamados primos de Fermat, são primos que têm a forma:

$$F(n) = 2^{2^n} + 1. \tag{3.5}$$

O primo 65537 é o maior primo de Fermat conhecido, $F(n)$ para $n = 4$. Também é o mais utilizado como expoente e, por ser grande o bastante para evitar ataques pelos quais

expoentes pequenos tornam o RSA vulnerável (vulnerabilidades que serão discutidas mais a frente) e por ter baixo peso de Hamming, quantidade de *bits* 1 [25, 34].

O expoente privado d é calculado em relação a p, q e e de modo a ser o inverso multiplicativo módulo $\varphi(n)$. Portanto:

$$d \cdot e \equiv 1 \pmod{\varphi(n)}.$$

A partir dos passos descritos, podemos formalizar o algoritmo RSA no esquema de cifragem. Cada entidade deve gerar um par de chaves, pública e privada, correspondentes como definido no algoritmo 3.

Algorithm 3 Geração de chaves para RSA em esquema de cifração

1: Gere dois primos grandes aleatórios p e q, ambos com tamanhos próximos.
2: $n \leftarrow p \cdot q$
3: $\varphi(n) \leftarrow (p-1)(q-1)$
4: Escolha e de modo que $1 < e < \varphi(n)$ e $\text{mdc}(e, \varphi(n)) = 1$
5: Use o Algoritmo Euclidiano Estendido 1 para encontrar d tal que $1 < d < \varphi(n)$ e $e \cdot d \equiv 1 (\text{mod } \varphi(n))$
6: A chave pública (cifragem) é (b, n)
7: A chave privada (decifragem) é (a, n)

Os inteiros obtidos e e d também São chamados de expoente de cifragem e expoente de decifragem, respectivamente.

Suponha que Alice deseja cifrar uma mensagem m para Bob e em seguida Bob a decifra. Para isso, utiliza-se os passos do algoritmo 4

Algorithm 4 Cifragem e decifragem de uma mensagem utilizando RSA

Cifragem: Alice procede.

1: Obtém uma chave pública autêntica de Bob (e, n)
2: Representa m como inteiro $\in \mathbb{Z}_n$
3: $c \leftarrow m^e \pmod{n}$
4: Envia o criptograma c para Bob

Decifragem: para recuperar o texto claro m, Bob procede:

1: $m \leftarrow c^d \pmod{n}$

3.3.2 Esquema de Assinatura Digital

O esquema de criptografia de chave pública também permite atribuir uma assinatura digital a uma mensagem eletrônica. Essa assinatura é equiparável a uma versão eletrônica de uma assinatura de punho em um documento de papel, exceto quanto às premissas de confiança.

A assinatura digital provê uma autenticação irretratável de origem e integridade de conteúdo. Pode ser facilmente verificada e caso alterado qualquer *bit* da mensagem, perde sua validade.

É a ferramenta perfeita para assinar contratos de negócios eletronicamente, comércio eletrônico e qualquer outra comunicação que necessite ser autenticada [10], tendo satisfeitas as premissas de confiança.

No Brasil, as assinaturas digitais reguladas pelo ICP-Brasil têm valor jurídico. A partir da Medida Provisória 2.200-2 de 24 de outubro de 2001 foi criada a Infraestrutura de Chaves Públicas Brasileira (ICP-Brasil), uma cadeia hierárquica que viabilizaria a emissão de certificados digitais. Deste modo, as transações online entre os vários órgãos públicos e seus fornecedores têm valor legal. Este recurso foi logo expandido também para a iniciativa privada. A ICP-Brasil é mantida e auditada pelo Instituto Nacional de Tecnologia da Informação (ITI).

Para Bob enviar a Alice uma mensagem m assinada, ele deve calcular a assinatura s utilizando sua sua chave privada, no contexto do esquema de assinatura, que chamaremos de função S_B de Bob.

$$s = S_B(m) = m^d \pmod{n}$$

Bob envia a Alice o par (s, m). Caso ambas as partes queiram obter sigilo da mensagem e assinatura, basta Bob cifrar (s, m) utilizando a chave pública E_A de Alice.

Para Alice verificar a autenticidade da assinatura de Bob basta aplicar a função de verificação de Bob V_B utilizando a chave pública de Bob.

$$m = V_B(s) = s^e \pmod{n}$$

Ao observar igualdade do resultado de $V_B(s)$ e m recebido, há garantia matemática de que apenas a chave privada correspondente a chave pública usada em V_B e sob a custódia de Bob, foi quem assinou a mensagem.

Algorithm 5 Assinatura de uma mensagem m utilizando RSA

1: Representa m como inteiro $\in \mathbb{Z}_n$
2: $s \leftarrow m^d \pmod{n}$
3: Envia o par (s, m)

Algorithm 6 Verificação de uma assinatura s utilizando RSA

1: Recebe o par (s, m)
2: $v \leftarrow s^e \pmod{n}$
3: **if** $m = v$ **then return true**
4: **else return false**

Demonstração da exatidão da decifragem

1. Para p e $q \in \mathbb{P}$ e por propriedades da função totient φ:

$$\varphi(n) = \varphi(p) \cdot \varphi(q)$$
$$\varphi(n) = (p-1)(q-1)$$

Como e é coprimo com $\varphi(n)$, ele tem um inverso multiplicativo d no anel de inteiros modulo $\varphi(n)$ $\mathbb{Z}_{\varphi(n)}$.

$$e \cdot d \equiv 1 \pmod{\varphi(n)}$$

Quer dizer que $e \cdot d$ dividido por $\varphi(n)$ deixa resto 1. Existe, então, um quociente inteiro k tal que:

$$ed = 1 + k \cdot \varphi(n). \tag{3.6}$$

Pelo terorema de Fermat, sabemos que:

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

Elevando os dois lados da congruência a $k \cdot (q-1)$ e depois multiplicados ambos lados por m obtemos:

$$\begin{aligned} m \cdot m^{k(p-1)(q-1)} &\equiv m \pmod{p} \\ m^{1+k(p-1)(q-1)} &\equiv m \pmod{p} \\ m^{1+k\varphi(n)} &\equiv m \pmod{p} \end{aligned} \tag{3.7}$$

Substituindo 3.6 em 3.7, temos:

$$m^{ed} \equiv m \pmod{p}.$$

Pela mesma justificativa, podemos chegar no resultado módulo q:

$$m^{de} \equiv m \pmod{q}.$$

Como p e q são primos distintos, a congruência vale para $p \cdot q$:

$$m^{ed} \equiv m \pmod{n}.$$

Lembrando que $c = (m^e)$, por fim, temos:

$$c^d \equiv (m^e)^d \equiv m \pmod{n}$$ [29].

3.3.3 Funções de Resumo e Preenchimento

Em um cenário real, o uso do RSA deve se atentar a diversos problemas não abordados no RSA livro texto 3.1.

Por motivos de segurança e também desempenho, as mensagens ou documentos digitais não são assinados diretamente. Caso o documento seja muito grande, haverá alto custo computacional para gerar uma assinatura de mesmo tamanho. Também podemos notar que tendo a posse de duas assinaturas distintas de Bob, pode-se forjar uma terceira assinatura a qual Bob não teria como retratá-la.

Imagine que Bob assinou as mensagens m_1 e m_2, logo temos s_1 e s_2 tal que:

$$s_1 = m_1^d \pmod{n}$$
$$s_2 = m_2^d \pmod{n}$$

Fazendo o produto das mensagens, criamos uma nova mensagem m_3:

$$m_3 = m_1 \cdot m_2$$

A assinatura de m_3, que desejamos obter sem o consentimento de Bob, seria:

$$s_3 = m_3^d \pmod{n}$$
$$s_3 = (m_1 m_2)^d \pmod{n}$$
$$s_3 = m_1^d m_2^d \pmod{n}$$

Assim, podemos facilmente fazer a forja de s_3 através do produto de s_1 e s_2:

$$s_3 = s_1 s_2$$
$$s_3 = m_1^d m_2^d \pmod{n}$$

Para obter resistência a esses ataques, não se utiliza o texto em claro para ser assinado, mas sim o seu *hash*, artefato de tamanho fixo resultante da função de resumo. Esta função de resumo criptográfico não é invertível, consequentemente o único método para encontrar uma mensagem m a partir do seu *hash* $h(m)$ é o da busca exaustiva (caso a função de *hash* tenha resistência a pré-imagem).

Outro fato não abordado no RSA livro texto é o caso das mensagens cifradas terem tamanho menor do que as chaves do RSA. Esta vulnerabilidade abre a oportunidade para vários tipos de ataques [32] utilizando ideia semelhante ao da forja de assinatura citada.

O preenchimento (*padding*) não é opcional, é uma primitiva criptográfica crítica. Deve-se realizar o *padding* com bits aleatórios nas mensagens a serem cifradas, seguindo o padrão descrito por *Public-Key Cryptography Standards* (PKCS#1). Padrão que já foi demonstrado inseguro [32] em sua versão 1.5 e hoje, sendo complementado pelo uso do *Optimal Asymmetric Encryption Padding* (OAEP), uma maneira mais segura de realizar o preenchimento que também faz uso de funções de resumo.

Capítulo 4

Teste de Primalidade

4.1 Importância dos Números Primos

A eficiência na geração de chaves e parâmetros para criptografia de chave pública é pré-requisito crucial para estes sistemas. Citando como exemplo, a necessidade de um número primo p para definir um corpo $\mathbb{Z}_p$ que satisfaça as propriedades no protocolo Diffie-Hellman [12] e também a necessidade de se gerar primos p e q para operações módulo $n = pq$ no algoritmo RSA, principal protocolo que teve enfoque neste documento. Neste caso, os primos devem ter tamanho grande (metade do tamanho de *bits* desejado para n), aleatórios, de forma que um adversário não obtenha vantagem com a probabilidade de escolha da chave e que a tentativa de fatorar n seja intratável.

A busca por números primos grandes é responsável por mais da metade do tempo de geração de chaves do RSA. Tal fato corrobora a estratégia de otimização adotada.

Dese-ja que estes números primos atenda a alguma propriedades adicionais a fim de tornar o sistema não passível a ataques especializados:

1. Resistente contra o algoritmo de fatoração por curvas elípticas: a diferença entre p e q não deve ser pequena. $p - q$ pequeno implica $p \approx q$, assim $p \approx \sqrt{n}$. Uma fatoração eficiente seria a busca de divisores ímpares próximos de $\sqrt{n}$.

2. O uso de *strong primes* (primos fortes) para os fatores p e q torna a fatoração de n mais custosa. Primos fortes têm definições distintas no campo da Criptografia e da Teoria dos Números. Em Criptografia, um primo p é dito forte se satisfaz as seguintes propriedades:

   - $p - 1$ deve ter fator primo grande p_1.
   - $p_1 - 1$ deve ter fator primo grande p_2.
   - $p + 1$ de ter fator primo grande p_3.

Dado o Teorema dos Números Primos, em um espaço de tamanho n, a probabilidade $P(n)$ de um número aleatório ser primo é $P(n) = \frac{1}{\ln n}$. Para criptografia, são usados apenas primos grandes, $p > 2$, portanto calcula-se apenas para n ímpares, o que aumenta a probabilidade para

$$P(n) = \frac{2}{\ln n}$$

4.2 Fórmula para Primos

Uma função ou polinômio gerador de primo é uma função $f(n)$ definida para todos os naturais $n \geq 1$ que tenha apenas primos em sua imagem.

Há muito tempo, matemáticos buscam uma função ou fórmula fechada que gere números números primos, sem exceções. Euler, em 1772, foi o primeiro a notar que o polinômio $P(n) = n^2 - n + 41$ produzia somente números primos como saída.

$$P(1) = 41, \quad P(2) = 43, \quad P(3) = 47, \quad P(4) = 53, \quad P(5) = 61, \quad P(6) = 71, \quad \ldots$$

Entretanto, o polinômio funcionava apenas para $1 \leq n < 41$, para $n = 41$, $P(41) = 1681 = 41 \cdot 41$, ela falha. Este foi um dos primeiros esforços realizados na área e serve apenas como curiosidade.

Goldbach, em correspondências trocadas com Euler por volta de 1750, desenvolveu uma prova de que não existe polinômio com coeficientes inteiros que resulte em primos para qualquer entrada inteira [33]. Também, nenhuma construção que seja eficiente para gerar primos é conhecida até o momento.

4.2.1 Função de Mills

Em 1947 foi demonstrada a primeira função geradora de primos por W. H. Mills [31], quem provou existir um número real A tal que, se $M(n)$ é a função geradora de Mills, então

$$M(n) = \lfloor A^{3^n} \rfloor$$

resulta um número primo para qualquer inteiro n. Os primos resultados de $M(n)$ são chamados de primos de Mills e A é chamada de constante de Mills.

$$A = 1,306377883863080690468614492 6\ldots \text{ [20].}$$

Um obstáculo evidente, que inviabiliza a utilização desta função, é o fato de seu crescimento ser exponencial, rapidamente os primos se tornam tão grandes que passa a ser intratável para os computadores.

$$\begin{array}{ll} n = 1, & M(n) = 2 \\ n = 2, & M(n) = 11 \\ n = 3, & M(n) = 1361 \\ n = 4, & M(n) = 2521008887 \\ n = 5, & M(n) = 16022362040098181318313201 83 \\ \vdots & \vdots \end{array}$$

Outra dificuldade é a de que quanto maior o valor de n, maior deve ser a precisão de A. E até então não há meio diferente de estender a precisão da constante senão pelo próprio valor do n-ésimo primo de Mills P_n, $A = P_n^{1/3^n}$, o que torna um argumento recursivo.

Pouco se sabe a respeito da constante de Mills, se é racional ou irracional [16] e se há outro meio de computá-la. Também existem outras funções desse tipo na literatura, contudo, devido os motivos citados, essas funções não têm aplicações práticas, mas exibem um problema ainda em aberto.

4.3 Testes de Primalidade

Como já foi dito, a fatoração de números grandes é um problema intratável para os computadores de hoje em dia. Mas e se apenas precisarmos saber se um número é primo ou é composto, independente de quais são os seus fatores? Em 1801, Gauss parecia já ter percebido que a fatoração e o teste de primalidade são dois problemas distintos, como pode ser visto na seguinte passagem [1]:

> "O problema de distinguir números primos de números compostos e de solucionar o último em seus fatores primos é conhecido por ser uma das questões mais importantes e úteis da aritmética. Ele tem aproximado a indústria e a sabedoria dos geômetras antigos e modernos de tal maneira que seria inútil discutir o problema por muito tempo... Além disso, a dignidade da própria Ciência parece exigir que todos os recursos possíveis sejam explorados para a solução de um problema tão elegante e tão celebrado."

Pode-se mencionar que Shor [45] demonstrou que o problema da fatoração se torna polinomial em computadores quânticos, computadores cujo cálculo pode percorrer por todos os estados possíveis ao mesmo tempo.

Como abordado na seção 3.1, um problema de decisão é uma questão formal cuja resposta seja ou SIM ou NÃO. O teste de primalidade é um problema de decisão no qual a reposta é SIM quando o número analisado é primo e NÃO caso seja composto. Não nos preocuparemos com a fatoração de números, mas apenas afirmar quanto a sua primalidade.

Dizer que um número é primo implica, evidentemente, negar sua compositividade e vice-versa:

$$\begin{aligned} \text{primo} &\iff \neg\text{composto} \\ \neg\text{primo} &\iff \text{composto} \end{aligned}$$

Logo, também podemos utilizar um teste de compositividade para provar a primalidade, ou não primalidade, de um número.

Trataremos a respeito de dois tipos de teste:

- Teste Determinístico: são testes que asseguram certeza matemática para todas as respostas.

- Teste 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.

  Como exemplo, imagine um jogo em que há dois copos opacos e uma bola vermelha sob um deles. A cada rodada do jogo os copos são "embaralhados" e você deve escolher um em que acredita conter a bola vermelha. O autor em nenhum momento mostra a existência da bola vermelha. A probabilidade $P(b)$ de você acertar o copo que possui a bola seria $P(b) = \frac{1}{2}$. Digamos que após 30 rodadas, você não acertou o

copo com a bola nenhuma vez, então a probabilidade de obter 30 fracassos sucessivos no jogo seria $2^{-30} \approx 0.0000001\%$, aproximadamente uma em um bilhão.

Tendo este resultado, podemos afirmar com alta probabilidade que o jogo é desonesto, mas nunca há certeza.

Os padrões de criptografia exigem uma probabilidade de erro ϵ de no máximo 2^{-80} ou 2^{-100} para gerar números primos [21], ou seja, é mais provável uma pessoa ganhar 3 vezes seguidas na Mega-Sena, o maior prêmio da loteria federal, do que um número composto ser dado como primo provável. Pode aparentar uma margem de segurança excessiva, mas aplicação de segurança devem ser excessivas.

Algoritmos probabilísticos 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.

4.3.1 Crivo de Eratóstenes

O crivo de Eratóstenes é um algoritmo determinístico de teste de primalidade.

No início da história desta disciplina, Eratóstenes desenvolveu um crivo elementar para listar números primos [15]. Para ilustrar este método, consideremos o problema de listar todos os primos menores que 50.

Primeiro, iremos ordenar todos os inteiros de 1 a 50 em uma grade.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	10
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Inicialmente, eliminamos o 1, seguimos para o 2 e retiramos da grade todos os múltiplos de 2. Agora selecionamos o próximo número que ainda permanece, o 3, e retiramos da grade todos os múltiplos de 3. Selecionamos o próximo, 5, e retiramos todos os seus múltiplos. Repita o processo até que não haja múltiplos menores que 50. Ao final os números restantes serão todos primos.

□	2	3	□	5	□	7	□	□	□
11	□	13	□	□	□	17	□	19	□
□	□	23	□	□	□	□	□	29	□
31	□	□	□	□	□	37	□	□	□
41	□	43	□	□	□	47	□	□	□

Note que a partir da retirada dos múltiplos de 7, todos os primos até 50 já estão definidos.

Pelo crivo de Erastótenes podemos concluir o seguinte resultado:

$$\text{Se } n \in \mathbb{N} \text{ não possui nenhum divisor } r \in \mathbb{N} \text{ tal que } r < \sqrt{n}, \text{ então } n \in \mathbb{P}$$

Esta abordagem, analisada no contexto criptográfico, pode ser considerada ingênua, pois o problema de eliminar todos os múltiplos da grade pode ser comparado como tão difícil quanto o problema da fatoração.

4.3.2 Teste de Fermat

O teorema de Fermat 3.2.6 garante que se um número p é primo e a é um inteiro tal que $1 \leq a \leq p-1$, então

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

Logo, definir um candidato a primo n e encontrar um a que não satisfaça o teorema de Fermat, $a^{n-1} \not\equiv 1 (\text{mod } p)$, é suficiente para provar a compositividade de n, então n não é primo. Por outro lado, se a validar a congruência de Fermat, agrega-se indício de que n pode ser primo.

Algorithm 7 Teste de primalidade de Fermat

Input: Inteiro ímpar $n \geq 3$ e o parâmetro de segurança $t \geq 1$.
Output: Ou PRIMO ou COMPOSTO.

1: **for** $i \leftarrow 1$ **to** t **do**
2: $a \leftarrow$ inteiro aleatório com $2 \geq a \geq n-2$
3: $r \leftarrow a^{n-1} \bmod n$
4: **if** $r \neq 1$ **then**
5: **return** COMPOSTO
6: **end if**
7: **end for**
8: **return** PRIMO

O parâmetro t é o número de rodadas a serem executadas. Quanto mais rodadas, maior a probabilidade de n ser primo.

Pseudoprimos

Um número composto que passa em um teste de primalidade é chamado de pseudoprimo. Existem números compostos com propriedades peculiares que passam na congruência do teorema para toda base. Neste caso, esses pseudoprimos são especialmente chamados de números de Carmichael.

O primeiro número de Carmichael é $561 = 3 \cdot 11 \cdot 17$, verifica-se que para todo $a \in \mathbb{Z}_{561}$, $a^{560} \equiv 1 \pmod{561}$ é verdadeiro. O terceiro número de Carmichael é 1729, também chamado de *taxicab number*, um número com uma história curiosa.

Assim como os primos, os números de Carmichael foram provados infinitos, como também que são infinitos os números de Carmichael formados pelo produto de 3 primos distintos [40]. Pinch [35] computou 20138200 números de Carmichael entre 1 e 10^{21}, o que dá aproximadamente $2 \cdot 10^{-14}$ as chances de encontrá-los. Mesmo aparentando ter uma baixa probabilidade de se encontrar um número de Carmichal, ela é muito maior do que a probabilidade de erro desejável e tal fato inviabiliza o teste.

O teste de Fermat não é usado verdadeiramente como teste de primalidade devido a ocorrência dos números de Carmichael, no entanto, é um ótimo exemplo didático para a compreensão de como funcionam os testes de primalidade probabilísticos e também mostra que é possível criar testes mais sofisticados, eficientes e com tempo de execução polinomial.

4.3.3 Teste de Solovay-Strassen

O teste de Solovay-Strassen [47] é um algoritmo de Mont Carlo com viés positivo para a compositividade de n e probabilidade de erro $\epsilon = \frac{1}{2}$. Teve um papel importante, pois foi o primeiro teste a ser popularizado, mostrando a real viabilidade da criptografia de chave pública, em particular o RSA, porém não é mais utilizado devido a criação de testes mais eficientes.

Baseia-se no critério de Euler e símbolos de Jacobi e Legendre 3.2.8.

$$n \text{ ímpar } \in \mathbb{P} \Rightarrow a^{\frac{n-1}{2}} \equiv \left(\frac{a}{n}\right) \pmod{n} \quad \forall a \in \mathbb{Z}_n^* \text{ tal que } \text{mdc}(a, n) = 1.$$

Caso $(\frac{a}{n}) = 0$ ou $a^{\frac{n-1}{2}} \not\equiv (\frac{a}{n}) (\text{mod } n)$, a é uma testemunha que prova que n é composto.

Caso contrário, $\text{mdc}(a, n) = 1$ e $a^{\frac{n-1}{2}} \equiv (\frac{a}{n}) (\text{mod } n)$, n se comporta como primo, pois satisfaz o critério de Euler para a base a. Contudo, seja n um ímpar composto, então existem no máximo $\frac{\varphi(n)}{2}$ bases a, com $1 \leq a \leq n-1$, que satisfazem o critério de Euler, mesmo com n composto [29].

Algorithm 8 Teste de primalidade de Solovay-Strassn

Input: Inteiro ímpar $n \geq 3$ e o parâmetro de segurança $t \geq 1$.
Output: Ou PRIMO ou COMPOSTO.

1: **for** $i \leftarrow 1$ **to** t **do**
2: $x \leftarrow (\frac{a}{n})$
3: **if** $x = 0$ **then**
4: **return** COMPOSTO
5: **end if**
6: $y \leftarrow a^{\frac{n-1}{2}} \bmod n$
7: **if** $x \neq y \bmod n$ **then**
8: **return** COMPOSTO
9: **end if**
10: **end for**
11: **return** PRIMO

4.3.4 Teste de Miller-Rabin

Miller [30], assumindo a Hipótese Estendida de Riemann (HER) como verdadeira, mostrou que a primalidade pode ser testada em tempo polinomial, isto é, o problema de decisão a respeito da primalidade de um número estar no conjunto de problemas P requer que seja possível encontrar um certificador de primalidade em tempo polinomial. Para isso, a demostração de Miller assumia a Hipótese Estendida de Riemann, um dos extraordinários problemas da Matemática ainda em aberto. A Hipótese de Riemann é um dos problemas do milênio e sua solução vale uma recompensa no valor de um milhão de dólares, que trata a respeito das raízes da função zeta de Riemann e também tem profunda ligação com a densidade dos números primos.

Rabin [37] aleatorizou a abordagem de Miller, conseguindo, deste modo, eliminar a dependência da HER e criar o teste probabilístico de Miller-Rabin, um dos testes mais eficientes da atualidade [11].

O teste de Miller-Rabin é um algoritmo de Monte Carlo com viés positivo para compositividade com erro $\epsilon = \frac{1}{4}$ e é o teste probabilístico mais utilizado na prática.

Sabemos que as raízes quadradas de 1 mod p com $p \in \mathbb{P}$, só podem ser 1 e -1.

Analisando o PTF, $a^{p-1} \equiv 1 (\text{mod } p)$ com $p \in \mathbb{P}$, ao extrair sucessivas raízes quadradas de a^{p-1}, devemos obter 1 ou -1.

Seja n um primo ímpar, $n-1$ é par, logo pode ser escrito na forma $n-1 = 2^k m$ em que a potência k é no mínimo 1 e m deve ser ímpar.

Seja a inteiro tal que $\text{mdc}(a, n) = 1$, então

$$a^m \equiv 1 (\text{mod } n)$$

ou

$$a^{2^r} m \equiv -1 (\text{mod } n), \text{ para algum } r, 0 \leq r \leq k-1$$

Consequentemente

- Se $a^m \not\equiv 1 (\text{mod} n)$ e se $a^{2^r} m \not\equiv -1 (\text{mod } n) \quad \forall r \in \digamma_k$, então a é uma forte testemunha de que n é composto.
- Caso contrário, se $a^m \equiv 1 (\text{mod} n)$ ou $a^{2^r} m \equiv -1 (\text{mod } n)$, para algum $r \in \digamma_k$, então a base a é uma testemunha da primalidade de n.

Assim, podemos formalizar o algoritmo de teste de primalidade 9.

Algorithm 9 Teste de primalidade de Miller-Rabin

Input: Inteiro ímpar $n \geq 3$ e o parâmetro de segurança $t \geq 1$.
Output: Ou PRIMO ou COMPOSTO.

Escreve $n-1$ na forma $2^k m$ {com k máximo e m ímpar}
for $i \leftarrow 1$ **to** t **do**
  $a \leftarrow$ inteiro aleatório tal que $1 \leq a \leq n-1$
  $b \leftarrow a^m \bmod n$
  if $b = 1$ **then**
    return PRIMO
  end if
  for $i \leftarrow 1$ **to** $k-1$ **do**
    if $b = -1$ **then**
      return PRIMO
    else
      $b \leftarrow b^2 \bmod n$
    end if
  end for
  return COMPOSTO
end for

4.3.5 Teste de Adleman-Huang

Adleman e Huang [2], em 1992, apresentaram uma variação de um teste de primalidade por curva elíptica. Esse teste é o oposto dos que foram abordados até então, é um algoritmo de Monte Carlo com viés positivo para primalidade, ou seja, para um número composto, o teste sempre vai dar como saída composto e para uma entrada primo, o teste vai dar na maioria das rodadas primo. Deste modo, se o saída é PRIMO, o número é com certeza primo e se a saída é COMPOSTO, o número é provavelmente composto.

Uma abordagem a se questionar seria a de intercalar um teste de viés positivo para compositividade com um teste de viés positivo para primalidade, assim o primeiro resultado não enviesado é dado como saída sempre correta. Infelizmente, após um número fixo de rodadas, há chances de nenhum dos dois casos acontecer para todas as rodadas [1].

4.3.6 Teste AKS

Em Agosto de 2002, o indiano N. Agrawal e seus slunos N. Kayal e N. Saxena apresentaram o algoritmo AKS, publicado no artigo "*PRIME is in P*" [3], nome que faz referência ao conjunto de problemas polinomiais. Foi a primeira prova de que existe um algoritmo determinístico com tempo de execução polinomial para o teste de primos, sem restrições ou premissas, demonstrando enfim que o problema da primalidade está em P. Este trabalho foi tão expressivo que teve reconhecimento da mídia internacional e também os autores foram prestigiados com os prêmios *Gödel* e *Fulkerson* em 2006.

A motivação inicial para este teste pode ser observada em uma propriedade do triângulo de Pascal.

$n = 0$ 1
$n = 1$ 1 1
$\boldsymbol{n = 2}$ **1 2 1**
$\boldsymbol{n = 3}$ **1 3 3 1**
$n = 4$ 1 4 6 4 1
$\boldsymbol{n = 5}$ **1 5 10 10 5 1**
$n = 6$ 1 6 15 20 15 6 1
$\boldsymbol{n = 7}$ **1 7 21 35 35 21 7 1**
$n = 8$ 1 8 28 56 70 56 28 8 1
$\vdots$ $\vdots$

Cada linha n contém os coeficientes da expansão do binômio $(x + y)^n$, que pode ser obtido pela combinação

$$\binom{n}{k} = \frac{n!}{k!(n-k)!},$$

para $n, k \in \mathbb{N}$ e $k \leq n$, ou pela relação de recorrência

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}.$$

Um interessante fato pode ser notado nas linhas primas do triângulo, todos os coeficientes diferentes de 1 são múltiplos de n quando n é primo. A partir do pequeno teorema de Fermat, chaga-se à seguinte expressão para polinômios: seja $n > 2$ e $a \in \mathbb{Z}_n$, então n é primo se e somente se

$$(x + a)^n = x^n + a \pmod{n}.$$

Como todos os coeficientes da expansão de $(x + a)^n$ são divisíveis por n, resta apenas os elementos das colunas 0 e n, x^n e a^n. Logo, se n é primo, então $\binom{n}{k} \equiv 0 \pmod{n}$ para qualquer $0 < k < n$.

Ainda assim, esta expressão resultada de Fermat também sofre com o problema dos pseudo-primos, números de Carmichael. Um outro problema é o alto custo computacional para calcular $(x + a)^n$, que tem complexidade $O(n)$, o que inviabilizaria o teste.

Algumas modificações foram realizadas para contornar esses obstáculos e reduzir sua complexidade. Para um valor pequeno r, escolhido adequadamente, então para todo a e r

$$(x + a)^n \equiv x^n + a \pmod{x^r - 1, n}.$$

A quantidade de valores de a que devem ser testados, assim como o valor de r são limitados por um polinômio em $\log n$, deste modo resultando em um teste de primalidade que é determinístico e com tempo de execução polinomial.

Algorithm 10 Teste de primalidade AKS

Input: Inteiro $n > 1$

Output: Ou PRIMO ou COMPOSTO.

1: **if** $n = a^b$ **then** // Se n é uma potência perfeita. $\exists\, a, b \in \mathbb{N} \ : n = a^b$
2: **return** COMPOSTO
3: **end if**
4: Encontre o menor r tal que $O_r(n) > \log^2 n$
5: **if** $1 < \text{mdc}(a, n) < n$ para algum $a \leq r$ **then**
6: **return** COMPOSTO.
7: **end if**
8: **if** $n \leq r$ **then**
9: **return** PRIMO
10: **end if**
11: **for** $a \leftarrow 1$ **to** $\lfloor \sqrt{\varphi(r)} \cdot \log n \rfloor$ **do**
12: **if** $(x + a)^n \neq x^n + a \pmod{x^r - 1, n}$ **then**
13: **return** COMPOSTO
14: **end if**
15: **end for**
16: **return** PRIMO

Nesta primeira versão apresentada pelos autores, foi provado que o algoritmo tem um limite superior assintótico $O(\log^{12} n)$. Logo após a publicação deste trabalho [3], vários matemáticos revisaram o artigo em busca de torná-lo mais eficiente, o que foi bastante positivo, pois logo nos meses seguintes havia a proposta de diferentes versões para o teste e com limites mais baixos.

A última versão do teste AKS proposta por Lenstra e Pomerance [26] conseguiu alcançar uma complexidade de $O(\log^6 n)$. É o limite teórico mais baixo para o teste que não depende de conjecturas ou possua exceções.

Apesar dos progressos realizados sob o algoritmo AKS, seu tempo de execução ainda é significativamente mais lento do que os testes probabilísticos, portanto ainda não é utilizado nas principais soluções para criptografia.

Capítulo 5

Teste de Frobenius Quadrático Simplificado

O teste escolhido para o desenvolvimento deste trabalho foi o Teste de Frobenius Quadrático Simplificado [43] (TFQS), baseado no Teste de Frobenius Quadrático proposto por Grantham [18]. Nesta versão simplificada, o tempo de execução foi reduzido de três rodadas do testes de Miller-Rabin para duas e a probabilidade de erro, para o pior caso, de 2^{-12t}, sendo t o número de rodadas.

A publicação de Grantham estimulou várias outras pesquisas e novas versões para o teste. O TFQS, a princípio, aparenta ser um teste muito promissor, pois, apesar de ter um tempo de execução maior, tem uma probabilidade de erro extremamente baixa, o que reduz bastante a quantidade de rodadas necessárias, t, para obter exigências de segurança e o torna um teste confiável. Também, ainda não se tem conhecimento de trabalhos de implementação ou análise de desempenho deste teste em comparação aos utilizados na indústria, o que reforça a motivação de examiná-lo.

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,

$$R(n, c) \text{ denota o anel polinomial } \mathbb{Z}_n[x]/(f(x) = x^2 - c),$$

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 $R(n,c)$:

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

Também será utilizada a notação (a/n) para o símbolo de Jacobi, tratado na seção 3.2.8.

O Algoritmo 11 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.

Algorithm 11 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. $\Phi_8(\epsilon) = 0$).

1: **if** $n = 3 \pmod 4$ **then**
2: $\quad \alpha \leftarrow 2^{\frac{n-3}{4}} \pmod n$
3: $\quad$ **if** $2\alpha^2 \neq \pm 1 \pmod n$ **then return** COMPOSTO
4: $\quad$ **else return** $c \leftarrow -1$, $\epsilon \leftarrow \alpha + \alpha x$
5: **end if**
6: **if** $n = 5 \pmod 8$ **then**
7: $\quad \alpha \leftarrow 2^{\frac{n-1}{4}} \pmod n$
8: $\quad$ **if** $\alpha^2 \neq -1 \pmod n$ **then return** COMPOSTO
9: $\quad$ **else return** $c \leftarrow 2$, $\epsilon \leftarrow \frac{1+\alpha}{2} x$
10: **end if**
11: **if** $n = 1 \pmod 8$ **then**
12: $\quad$ **if** n é um quadrado perfeito **then**
13: $\quad\quad$ **return** COMPOSTO
14: $\quad$ **else**
15: $\quad\quad$ $c \leftarrow$ valor aleatório pequeno tal que $(c/n) = -1$
16: $\quad\quad$ $\alpha \leftarrow c^{\frac{n-1}{8}} \pmod n$
17: $\quad\quad$ **if** $\alpha^4 \neq -1 \pmod n$ **then return** COMPOSTO
18: $\quad\quad$ **else return** c, $\epsilon \leftarrow \alpha$
19: $\quad$ **end if**
20: **end if**

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 [43].

[1] prova trivial pelo Teorema Chinês do Resto

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 12 representa uma rodada do teste de Frobenius quadrático simplificado.

Algorithm 12 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

5.1 Implementação

O planejamento da implementação foi realizado sobre a biblioteca criptográfica Relic [5]. 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. Deste modo, o desenvolvimento do projeto foi iniciado seguindo as etapas de Estudo, Implementação, Teste, Análise de desempenho, Otimização e Resultados apresentadas na figura 5.1.

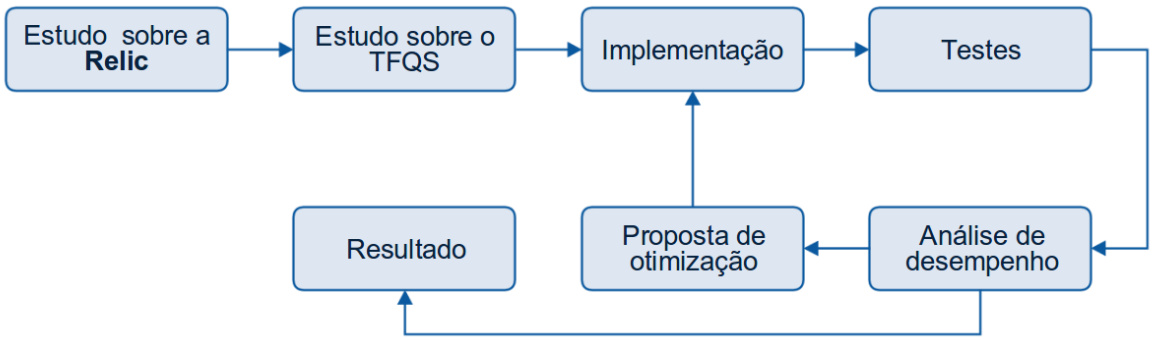

Figura 5.1: Fluxograma das etapas do desenvolvimento do projeto.

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.

5.1.1 Relic

A Relic é uma moderna biblioteca criptográfica com ênfase em eficiência e flexibilidade. Além de possuir várias opções de protocolos criptográficos, oferece suporte para construir *toolkits* criptográficos próprios, adaptáveis a diferentes níveis de segurança e também a escolha de algoritmos. É um projeto em andamento que prima pela fácil portabilidade e independência de arquitetura.

O elemento básico da biblioteca é o inteiro multi-precisão *BigNumber*, representado pelo tipo `bn_t`.

Listing 5.1: Tipo inteiro multi-precisão.

```
1  /**
2   * Represents a multiple precision integer.
3   *
4   * The field dp points to a vector of digits. These digits are organized
5   * in little-endian format, that is, the least significant digits are
6   * stored in the first positions of the vector.
7   */
8  typedef struct {
9          /** The number of digits allocated to this multiple precision integer. */
10         int alloc;
11         /** The number of digits actually used. */
12         int used;
13         /** The sign of this multiple precision integer. */
14         int sign;
15         /** The sequence of contiguous digits that forms this integer. */
16         dig_t *dp;
17 } bn_st;
18
19 /**
```

```
20  * Pointer to a multiple precision integer structure.
21  */
22 typedef bn_st *bn_t;
```

Para a implementação da aritmética no anel $R(n, c)$, os polinômios serão representados por um vetor de *BigNumbers* `bn_t*`, no qual o índice representa a ordem, e o valor, o coeficiente.

5.1.2 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.

Listing 5.2: Funções de quadrado perfeito e raiz.

```
1  /**
2   * Tests if a multiple precision integer is a perfect square.
3   *
4   * @param[in] a    - the multiple precision integer to test.
5   * @return 1 if the argument is a perfect square, 0 otherwise.
6   */
7  int bn_is_perfect_sqr(bn_t a);
8
9  /**
10  * Computes the integer floor square root of a multiple precision integer
11  *
12  * @param[out] c   - the result.
13  * @param[in] a    - the multiple precision integer to square root.
14  */
15 void bn_sqt(bn_t c, bn_t a);
```

5.1.3 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.

Sejam $a = a_1x + a_0$ e $b = b_1x + b_0$ polinômios de grau 1, então

$$a \cdot b = (a_1x + a_0)(b_1x + b_0)$$
$$a \cdot b = a_1b_1x^2 + (a_1b_0 + a_0b_1)x + a_0b_0$$

$$\begin{array}{rr|l} a_1b_1x^2 + (a_1b_0 + a_0b_1)x + a_0b_0 & & x^2 - c \\ \cline{3-3} -a_1b_1x^2 & + a_1b_1c & a_1b_1 \\ \cline{1-2} (a_1b_0 + a_0b_1)x + a_0b_0 + a_1b_1c & & \end{array} \tag{5.1}$$

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 muito inferiores. Deste modo, uma multiplicação de dois polinômios custa 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 5.1 com $a = b$, que resulta em

$$a^2 = 2a_1a_0x + a_0^2 + a_1^2c$$

com o custo de uma multiplicação e dois quadrados. As multiplicações por potências de 2 são feitas de forma hábil apenas com operações de *shift*.

Listing 5.3: Funções de aritmética básica.

```
1  /**
2   * Adds two multiple precision polynoms in a polynomial ring. c = a + b.
3   *
4   * @param[out] c   - the result.
5   * @param[in] a    - the first multiple precision polynom to add.
6   * @param[in] b    - the second multiple precision polynom to add.
7   * @param[in] n    - the modulus of coefficients.
8   */
9  void bn_pol_add(bn_t *c, bn_t *a, bn_t *b, bn_t n);
10
11 /**
12  * Subtracts two multiple precision polynoms in a polynomial ring. c = a - b.
13  *
14  * @param[out] c   - the result.
15  * @param[in] a    - the first multiple precision polynom to add.
16  * @param[in] b    - the second multiple precision polynom to add.
17  * @param[in] n    - the modulus of coefficients.
18  */
19 void bn_pol_sub(bn_t *c, bn_t *a, bn_t *b, bn_t n);
20
21 /**
22  * Multiplies two multiple precision polynoms in a polynomial ring.
23  *
24  * @param[out] c   - the result.
25  * @param[in] a    - the first multiple precision polynom to multiply.
26  * @param[in] b    - the second multiple precision polynom to multiply.
27  * @param[in] m    - the polynomial modulus.
28  * @param[in] n    - the modulus of coefficients.
29  */
30 void bn_pol_mul_basic(bn_t *c, bn_t *a, bn_t *b, bn_t *m, bn_t n);
31
32 /**
33  * Computes the square of a multiple precision polynom in a polynomial ring.
34  *
35  * @param[out] c   - the result.
36  * @param[in] a    - the multiple precision polynom to square.
37  * @param[in] m    - the polynomial modulus.
```

```
38  * @param[in] n    - the modulus of coefficients.
39  */
40 void bn_pol_sqr_basic(bn_t *c, bn_t *a, bn_t *m, bn_t n);
```

5.1.4 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. Assim, o Algoritmo 13 faz um percorrimento da esquerda para a direita no expoente e realiza quadrados ou multiplicações.

Algorithm 13 Exponenciação modular binária esquerda para direita

Input: Polinômios a, f. Inteiros e, n.
Output: $c = a^e \bmod (n, f)$.

1: $c \leftarrow 1$
2: **for** $i \leftarrow |e| - 1$ **downto** 0 **do**
3: $\quad c \leftarrow c^2 \bmod (n, f)$
4: $\quad$ **if** $e_i = 1$ **then**
5: $\quad\quad c \leftarrow a \cdot c \bmod (n, f)$
6: $\quad$ **end if**
7: **end for**
8: **return** c

Neste caso, a exponenciação modular é realizada com $O(\log_2 n)$ quadrados e $o(\log_2 n)$ multiplicações.

Listing 5.4: Função de exponenciação modular esquerda para direita.

```
1  /**
2   * Exponentiates a multiple precision polynom modulo a polynomial modulus
3   * using the binary method.
4   *
5   * @param[out] c   - the result.
6   * @param[in] a    - the polynomial basis.
7   * @param[in] b    - the exponent.
8   * @param[in] m    - the polynomial modulus.
9   * @param[in] n    - the modulus of coefficients.
10  */
```

```
11 void bn_pol_mxp_ltr(bn_t *c, bn_t *a, bn_t b, bn_t *m, bn_t n);
```

5.1.5 Probabilidade de Erro

O número de rodadas do teste assegura um limite inferior de probabilidade de erro de 2^{-100}, seguindo recomendação [21]. Esta implementação adota os valores da Tabela 5.1 como referência.

Tabela 5.1: Probabilidade de erro para o TFQS (*bits* k vs. número de rodadas t). Retirada de [43].

$k \backslash t$	1	2	3	4	5
300	48	100	124	143	160
400	57	118	146	169	189
500	65	134	166	192	214
600	72	148	184	212	237
1000	96	197	243	281	314

5.2 Otimização

5.2.1 Crivo para Quadrados Perfeitos

Seja a um inteiro par e b um inteiro ímpar, então eles terão a forma $a = 2n$ e $b = 2m+1$, para $n, m \in \mathbb{N}$. Assim,

$$a^2 = (2n)^2 = 4n^2$$
$$b^2 = (2m+1)^2 = 4m^2 + 4m + 1 = 4\underbrace{(m + m^2)}_{k} + 1 = 4k + 1 \ .$$

Disto, tira-se 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.

5.2.2 Multiplicação de Karatsuba

Anatolii Alexeevitch Karatsuba, quando jovem aluno de graduação, propôs um eficiente algoritmo para multiplicação [23] 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.

$$a \cdot b = \big[(a_1 + a_0)(b_1 + b_0) - a_1 b_1 - a_0 b_0\big]x + a_0 b_0 + a_1 b_1 c$$

Expandindo essa expressão de Karatsuba, pode-se notar que retorna ao mesmo resultado da equação 5.1. Assim, reutilizando os produtos já computados, o custo da multiplicação de dois polinômios cai para três multiplicações e uma redução.

Listing 5.5: Função multiplicação de Karatsuba.

```
/**
 * Multiplies two multiple precision polynoms using Karatsuba multiplication
 * in a polynomial ring.
 *
 * @param[out] c  - the result.
 * @param[in] a   - the first multiple precision polynom to multiply.
 * @param[in] b   - the second multiple precision polynom to multiply.
 * @param[in] m   - the polynomial modulus.
 * @param[in] n   - the modulus of coefficients.
 */
void bn_pol_mul_karat(bn_t *c, bn_t *a, bn_t *b, bn_t *m, bn_t n);
```

5.2.3 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.

$$a^2 = 2a_0 a_1 x + (a_0 + a_1)(a_0 + a_1 c) - a_0 a_1 - a_0 a_1 c$$

O custo é reduzido para apenas duas multiplicações: $(a_0 + a_1)(a_0 + a_1 c)$ e $a_0 a_1$.

Listing 5.6: Função de quadrado utilizando forma do quadrado complexo.

```
/**
 * Computes de squareof a multiple precision polynoms using complex square
 * form in a polynomial ring.
 *
 * @param[out] c  - the result.
 * @param[in] a   - the multiple precision polynom to square.
 * @param[in] m   - the polynomial modulus.
 * @param[in] n   - the modulus of coefficients.
 */
void bn_pol_sqr_complex(bn_t *c, bn_t *a, bn_t *m, bn_t n);
```

5.2.4 Janela Deslizante

O método janela deslizante para exponenciação modular é baseado em uma generalização do Algoritmo 13 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 [29].

Para o Algoritmo 14, w é chamado de "tamanho da janela" e $w \geq 1$. e tem forma binária $e = (e_t e_{t-1} \dots e_1 e_0)_2$ com $e_t = 1$, logo $t = |e|$.

Algorithm 14 Exponenciação modular janela deslizante

Input: Polinômios a, f. Inteiros e, n, w.
Output: $c = a^e \bmod (n, f)$.

Pré-cálculo:
  $a_1 \leftarrow a$, $a_2 \leftarrow a^2$
  for $i \leftarrow 1$ **to** $(2^{w-1} - 1)$ **do** $a_{2i+1} \leftarrow a_{2i-1} \cdot a_2$
$c \leftarrow 1$, $i \leftarrow t$
while $i \geq 0$ **do**
  if $e_i = 0$ **then**
    $c \leftarrow c^2$, $i \leftarrow i - 1$
  else
    Encontre a maior sequência de *bits* $e_i e_{i-1} \ldots e_l$ tal que $e_l = 1$ e $(i - l + 1) \leq w$
    $c \leftarrow c^{2^{i-l+1}} \cdot a(e_i e_{i-1} \ldots e_l)_2$, $i \leftarrow l - 1$
  end if
end while
return c

Listing 5.7: Função de exponenciação modular janela deslizante.

```
/**
 * Exponentiates a multiple precision polynom modulo a polynomial modulus
 * using the sliding window method.
 *
 * @param[out] c   - the result.
 * @param[in] a    - the polynomial basis.
 * @param[in] b    - the exponent.
 * @param[in] m    - the polynomial modulus.
 * @param[in] n    - the modulus of coefficients.
 */
void bn_pol_mxp_slide(bn_t *c, bn_t *a, bn_t b, bn_t *m, bn_t n);
```

5.2.5 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 do desempenho da função.

A divisão em multi-precisão, como também em precisão simples, tem alto custo computacional. Uma solução proposta por Montgomery é trocar as divisões por multiplicações.

Seja $r = a \cdot b \pmod{n}$. Para n ímpar, escolhe-se $V = 2^{|n|}$. O $\mathrm{mdc}(V, n) = 1$, portanto V tem inverso módulo n.

Ao invés de utilizar os operandos a e b, usaremos os resíduos $\bar{a}$ e $\bar{b}$. Esta transformação é chamada de forma de Montgomery:

$$\bar{a} = aV \pmod{n}, \qquad \bar{b} = bV \pmod{n}.$$

Assim, podemos multiplicar $\bar{a}$ e $\bar{b}$ mantendo a forma de Montgomery:

$$\begin{aligned}\bar{r} &= \bar{a} \cdot \bar{b} \cdot V^{-1} \pmod{n} \\ &= aV \cdot bV \cdot V^{-1} \pmod{n} \\ &= a \cdot b \cdot V \pmod{n}\end{aligned}$$

Deste modo, podemos então calcular r em termos de $\bar{r}$:

$$r = \bar{r} \cdot V^{-1} \pmod{n}.$$

Deste modo, trocou-se uma divisão por quatro multiplicações.

Também há um detalhe importante: para ser realizada a redução de Montgomery é necessário o pré-cálculo de V^{-1}, ou seja, calcular 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.

Listing 5.8: Função multiplicação e quadrado com redução de Montgomery.

```
1  /**
2   * Multiplies two multiple precision polynoms using Karatsuba multiplication
3   * in a polynomial ring using Montgomery reduction.
4   *
5   * @param[out] c   - the result.
6   * @param[in] a    - the first multiple precision polynom to multiply.
7   * @param[in] b    - the second multiple precision polynom to multiply.
8   * @param[in] m    - the polynomial modulus.
9   * @param[in] n    - the modulus of coefficients.
10  * @param[in] u    - the reciprocal of the modulus.
11  */
12 void bn_pol_mul_karat_u(bn_t *c, bn_t *a, bn_t *b, bn_t *m, bn_t n, bn_t u);
13
14 /**
15  * Computes de square of a multiple precision polynoms using complex square
16  * form in a polynomial ring using Montgomery reduction.
17  *
18  * @param[out] c   - the result.
19  * @param[in] a    - the multiple precision polynom to square.
20  * @param[in] m    - the polynomial modulus.
21  * @param[in] n    - the modulus of coefficients.
22  * @param[in] u    - the reciprocal of the modulus.
23  */
24 void bn_pol_sqr_complex_u(bn_t *c, bn_t *a, bn_t *m, bn_t n, bn_t u);
```

5.2.6 Mudança Algébrica

Multiplicação por c pequeno

Analisando o algoritmo 11 (MR2), vê-se que o inteiro retornado c varia em apenas três casos:

1. $c = -1 \equiv n - 1$
2. $c = 2$
3. $c =$ 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 [43], assim mantendo o valor de c restritamente pequeno de modo que possamos trocar uma multiplicação e uma redução por uma multiplicação por dígito simples e $O(1)$ subtrações.

Listing 5.9: Função multiplicação por fator pequeno.

```
/**
 * Multiplies a multiple precision integer for a small digit using best method
 * for each digit.
 *
 * @param[out] c   - the result.
 * @param[in] a    - the multiple precision integer to multiply.
 * @param[in] b    - the digit to multiply.
 * @param[in] n    - the modulus.
 * @return 1 if the result was reduced modulo n, 0 otherwise
 */
int bn_pol_mul_swt(bn_t c, bn_t a, bn_t b, bn_t n);
```

Reuso de uma exponenciação modular

É possível reduzir a quantidade de exponenciações modulares necessárias nos seguintes passos do algoritmo TFQS [43]:

1. z^n.
2. $z^{(n^2-1)/8}$.

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.

5.2.7 Redução Preguiçosa

Em uma multiplicação de dois inteiros com k *bits* cada, espera-se um resultado com pelo menos $2k-1$ *bits* e no máximo $2k$ *bits*. Portanto, na multiplicação de polinômios, além de ser necessária a redução dos coeficientes em $\mathbb{Z}_n$ ao final da operação, também deve-se garantir que sucessivas multiplicações não excedam a precisão máxima da biblioteca.

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:

1. Elimina-se a redução intermediária e define um fator de redução $r = n \cdot 2^{|n|}$.

2. 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.

5.3 Testes

A cada passo da implementação, alteração ou otimização no código, é importante efetuar testes que garantam a corretude dos componentes para que se obtenha uma coleta de dados consistente.

O artefato de testes foi desenvolvido em um ambiente fornecido pela própria biblioteca Relic. Foram construídas operações que verificam diversas assertivas, as principais estão descritas na Tabela 5.2.

Tabela 5.2: Teste de assertivas em componentes.

Util	Definir dígito com sinal está correto. Raiz quadrada está consistente. Quadrado perfeito está correto.
Polinômio	Comparação entre polinômios está correta. Polinômio zero e teste zero estão corretos. Cópia de polinômio está correta.
	Adição de polinômios é comutativa. Adição de polinômios é associativa. Adição de polinômios tem identidade. Adição de polinômios tem inverso.
	Multiplicação em $R(n, c)$ é comutativa. Multiplicação em $R(n, c)$ é associativa. Multiplicação em $R(n, c)$ é distributiva. Multiplicação em $R(n, c)$ tem identidade. Multiplicação em $R(n, c)$ tem propriedade zero. Quadrado em $R(n, c)$ está correto.
	Exponenciação modular está correta. Negação e exponenciação modular estão corretas. Expoente zero está correto. Redução de Montgomery na multiplicação está correta. Redução de Montgomery no quadrado está correta. Exponenciação modular com Montgomery está correta.
MR2 e TFQS	Teste para compositividade está correto. Teste para primalidade está correto.

O ambiente de testes do projeto tem a seguinte configuração:

Tabela 5.3: 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.

5.4 Resultados

5.4.1 Consumo de Tempo

Ao final das etapas de implementação e teste, chegou-se a um código inicial e funcional TFQS versão 1. Para maior compreensão do consumo de tempo do teste, foi dividido em

quatro partes principais:

1. Teste MR2.
2. Encontrar z aleatório tal que $(N(z), n) = -1$ (Jacobi).
3. Calcular z^n (Cmp 1).
4. Calcular $z^{\frac{n^2-1}{8}}$ (Cmp 2).

Com esta divisão e dado um primo provável à entrada do teste, foi possível analisar o consumo de tempo de cada parte individualmente, apresentado na Figura 5.2. A partir desta informação, foi criada uma estratégia de otimização com enfoque nas partes de maior volume.

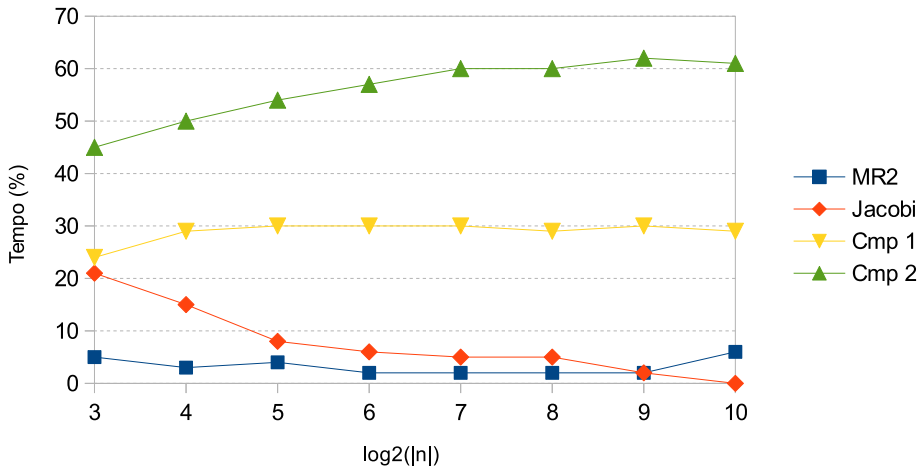

Figura 5.2: Consumo de tempo (%) vs. Tamanho de n (em potência de 2) do TFQS versão 1.

A etapa de otimização culminou no TFQS versão 5, que tem seu consumo de tempo apresentado na Figura 5.3.

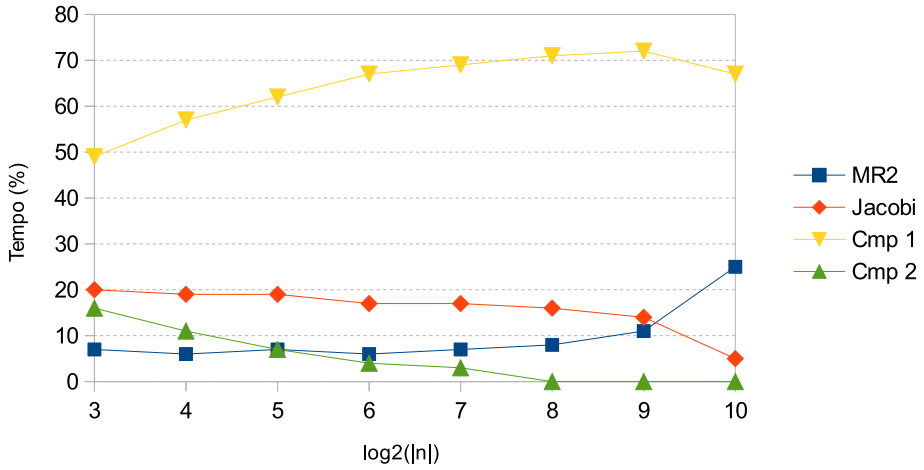

Figura 5.3: Consumo de tempo (%) vs. Tamanho de n (em potência de 2) do TFQS versão 5.

Pode-se notar que na Figura 5.3, última versão, a economia da segunda exponenciação modular (Cmp 2) reduziu o seu tempo para um valor desprezível. Também é visível que a única exponenciação modular restante continua custosa, seu cálculo é responsável por mais da metade do tempo de execução do teste. As outras etapas do teste aparentaram pouca ou nenhuma mudança.

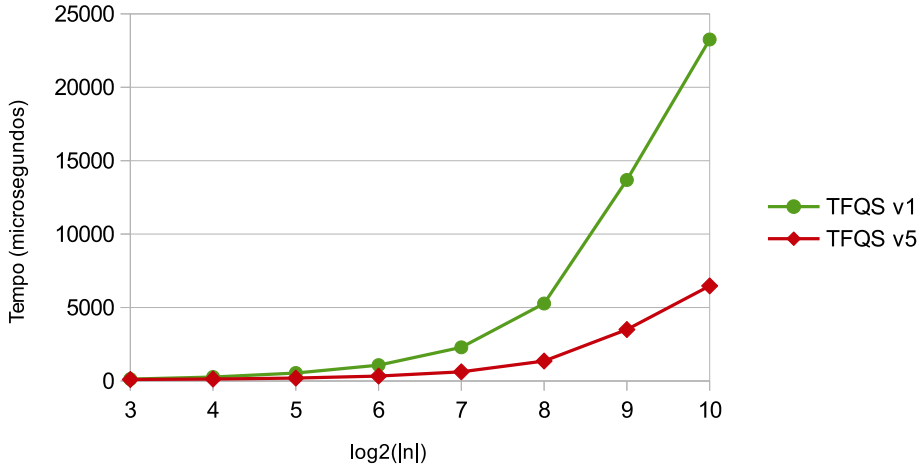

Figura 5.4: Gráfico do desempenho entre TFQS versão 1 e versão 5.

Do TFQS versão 1 para a versão 5, houve um *speedup* de 60%. Em média, a última versão se mostrou $2,5$ vezes mais rápida. Sem dúvida, Todas as técnicas de otimização foram importantes para este ganho, mas a que foi responsável pelo maior volume foi a

economia de uma exponenciação modular (Cmp 2), que é notável na comparação entre os Gráficos 5.2 e 5.3.

Também foi implementada uma versão chamada TFQS Subs, na qual a busca por um polinômio aleatório, linha 1 do algoritmo 12, só é necessária na primeira rodada do teste. A versão alternativa não mostrou significativa melhora no desempenho e também a partir de primos com 1000 *bits*, é executada apenas uma rodada do TFQS, o que tornou a versão TFQS Subs irrelevante quando é necessário alto nível de segurança para chaves criptográficas.

5.4.2 Teste de Primalidade

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}.

A Tabela 5.4 apresenta alguns dos novos valores mais relevantes para o número de rodadas ajustado para o Miller-Rabin. Estes valores foram gerados através da ferramenta mrtab [2].

Tabela 5.4: Rodadas do Miller-Rabin ajustada para erro de 2^{-100}.

t	1	2	3	4	5	6	7	8	9	10
k	3883	1761	1159	868	698	586	507	448	403	367

A coleta de dados dos testes de primalidade em si, foram divididas em dois cenários:

1. Cenário primo. Apenas primos prováveis já testados são entregues aos algoritmos.
2. Cenário composto. Apenas inteiros compostos são entregues aos testes. Este cenário tem o intuito de analisar o tempo de prova de compositividade para os testes.

A Figura 5.5 apresenta o desempenho dos testes com entrada de apenas primos prováveis. Para facilitar a visualização e comparação entre os gráficos, foi divido o tempo de cada teste pelo tamanho em *bits* do inteiro testado. Deste modo há maior separação entre as curvas.

Nota-se que o TFQS é mais veloz para inteiro de até 256 *bits* e a partir de 512 *bits*, o Miller-Rabin passa a ser mais eficiente.

[2]Gerador de tabelas de limite de iteração para Miller-Rabin. https://code.google.com/p/mrtab/

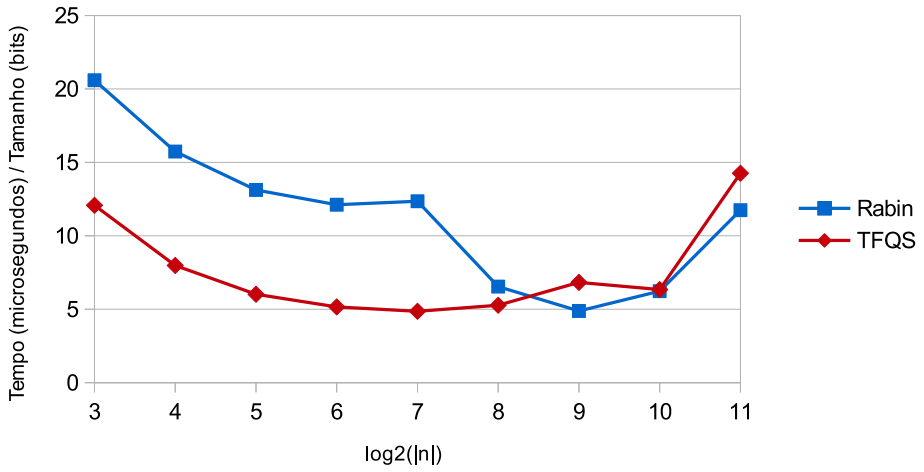

Figura 5.5: Gráfico Rabin vs. TFQS para entrada primo.

Não houve significativa diferença entre o desempenho dos testes para compositividade apresentado na Figura 5.6.

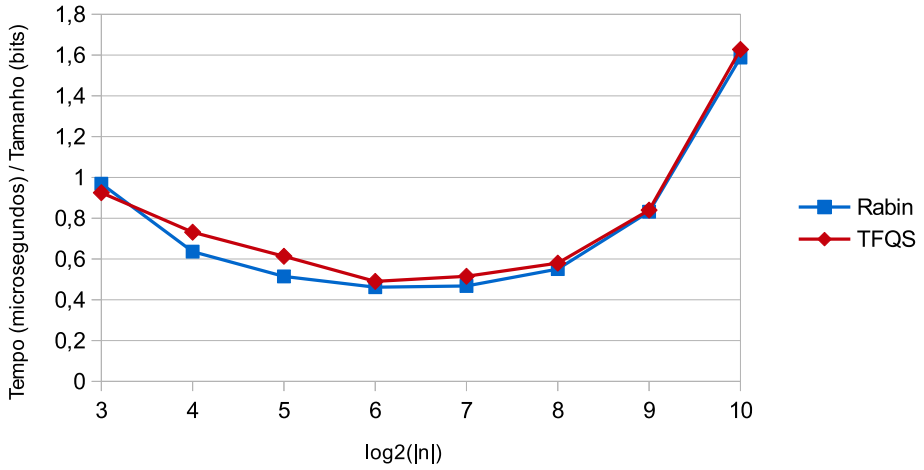

Figura 5.6: Gráfico Rabin vs. TFQS para entrada composto.

5.4.3 Gerador de Primos

A comparação de desempenho final entre o TFQS e o teste de Miller-Rabin foi efetuada sobre a utilidade real do teste de primalidade nas bibliotecas criptográficas, função de gerar primos, descrita no Algoritmo 15.

Algorithm 15 Gerador de primos

Input: Tamanho a em *bits*.
Output: Primo provável n com tamanho t *bits*.

```
1: while true do
2:     n ← número aleatóroi com t bits.
3:     if n não for divisível por primos triviais then
4:         if Teste_de_primalidade(n) = PRIMO then return n
5:     end if
6: end while
```

A linha 3 do algoritmo verifica se o inteiro n é divisível por primos pequenos, os 400 ou 500 primeiros primos. Esta técnica é uma otimização para provar a compositividade de números que tenham fatores pequenos.

Com o objetivo de obter uma análise completa e equilibrada, as comparações foram novamente divididas em 2 cenários:

1. Cenário justo. A semente do gerador pseudoaleatório foi fixada de modo que os testes recebam exatamente a mesma sequência de inteiros para verificação de primalidade.

2. Cenário aleatório. Não há fixação da semente do gerador, cada teste recebe um inteiro aleatório para verificação.

Deve-se ressaltar que os resultados apresentados são o da média de tempo de geração para 100 primos de cada tamanho.

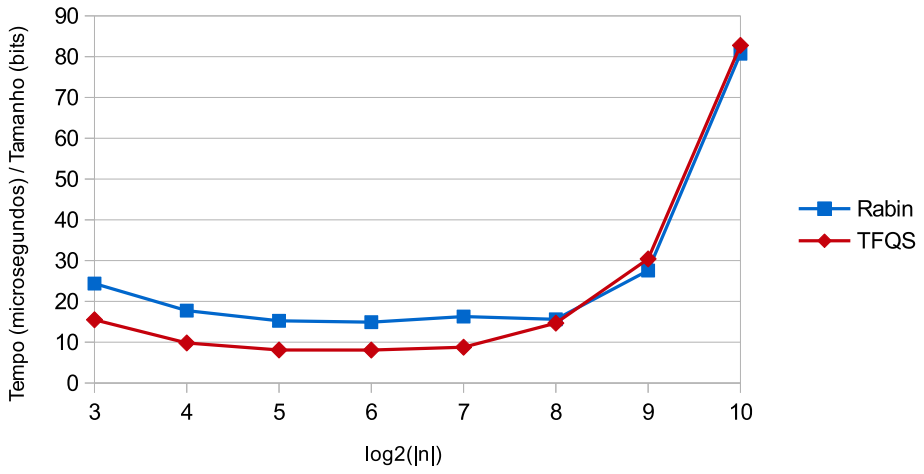

Figura 5.7: Gerador de primo com semente fixa. Rabin vs. TFQS.

Novamente, o TFQS supera o Miller-Rabin para a geração de primos de até 256 *bits* com ambos analisando a mesma sequência de inteiros.

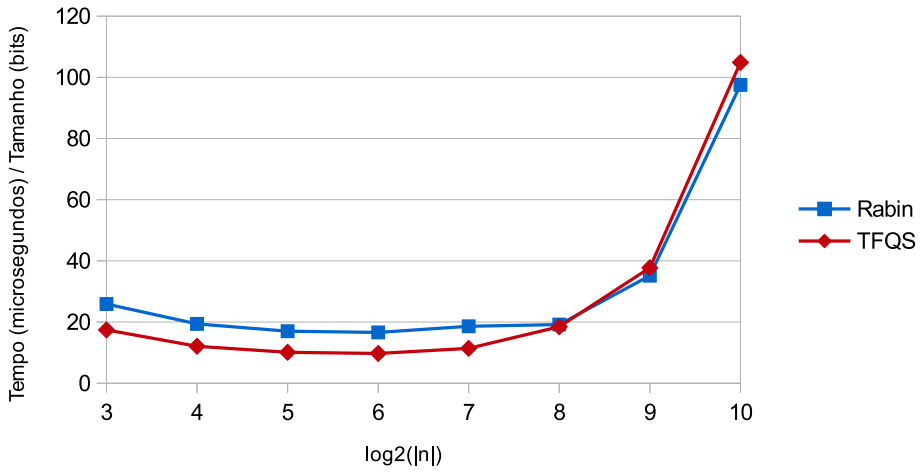

Figura 5.8: Gerador de primo com semente aleatória. Rabin vs. TFQS.

A Figura 5.8 reflete o mesmo resultado anterior. Embora com entradas aleatórias, devido a Lei dos Grandes Números, as médias das repetições convergem para o cenário justo.

Em uma visão panorâmica, o TFQS pode ser delimitado pelo custo de uma exponenciação modular em anel de polinômio por rodada, como evidenciado na Imagem 5.3. De maneira semelhante, o teste de Miller-Rabin pode ser delimitado por uma exponenciação modular simples por rodada.

Os dados de desempenho das duas funções é apresentado na Tabela 5.5. Assim, nota-se que a exponenciação modular em $R(n, c)$ é em média 3 vezes mais lenta que a exponenciação modular simples. Este fato, juntamente com o número de rodadas para cada teste, apoiam os resultados obtidos na análise dos gráficos de desempenho.

Tabela 5.5: Comparação entre as operações de exponenciação modular simples e em anel polinomial.

Bits	Tempo (nanosegundos)		
	`bn_mxp`	`bn_pol_mxp`	`bn_pol_mxp` / `bn_mxp`
32	15848	51871	3,27
64	29142	93074	3,19
128	60060	188831	3,14
256	138390	415677	3,00
512	414568	1206350	2,90
1024	1617537	4169730	2,57

Capítulo 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.

Esta conclusão é sustentada pela concordância obtida entre os dados teóricos do artigo de Seysen [43], a tese sobre a relação da exponenciação modular e também pela ampla análise de desempenho entre os testes que mostram, como fator comum, a vantagem do TFQS sobre o Miller-Rabin para inteiros de até 256 *bits*. A partir daí, as rodadas do TFQS são irredutíveis e o teste de Miller-Rabin com iterações mais simples apresenta maior eficiência.

O escopo principal do trabalho foi atingido em parte, houve a implementação de um teste mais eficiente, mas 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 [6] e Miller-Rabin combinado ao teste de Lucas-Selfridge [36] que é baseado no teste Baillie-PWS e aperfeiçoado por Selfridge.

Por fim, todo código produzido nesta obra, está disponível no repositório online[1] com um arquivo *bash* que gerencia a configuração, compilação e execução de testes, *benchmarks*, produção de gráfico de desempenho, entre outros.

[1] https://code.google.com/p/bruno-tg2/

Referências

[1] S. Aaronson. The Prime Facts: From Euclid to AKS. http://www.scottaaronson.com/writings/prime.pdf, 2003. 32, 37

[2] L. M. Adleman and M. A. Huang. *Primality Testing and Abelian Varieties Over Finite Fields*. Lecture Notes in Mathematics. Springer, 1992. 37

[3] M. Agrawal, N. Kayal, and N. Saxena. PRIMES is in P. *Annals of Mathematics*, 160(2):781–793, 2004. 37, 39

[4] D. F. Aranha. Cronologia Técnica da Criptografia. https://docs.google.com/file/d/0BwyUaEvgJMOnSFBvQmFUN1Y3OEk/edit, 2013. 1, 6

[5] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient Library for Cryptography. http://code.google.com/p/relic-toolkit/. 42

[6] R. Baillie and S. Wagstaff. Lucas pseudoprimes. *Mathematics of Computation*, 35(152):1391–1417, 1980. 60

[7] F. L. Bauer. *Decrypted secrets. Methodsand Maxims of Cryptology*. Springer-Verlag, Berlin, 1999. 1, 2, 11

[8] D. M. Burton. *Elementary Number Theory*. Allyn and Bacon Inc., Boston, 1980. 17, 22

[9] The OpenSSL Core and Development Team. Openssl: Documents, genrsa(1), julho 2013. http://www.openssl.org/docs/apps/genrsa.html. 25

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, New York, 2001. 16, 23, 25, 27

[11] M. Dietzfelbinger. *Primality Testing in Polynomial Time*. Lecture Notes in Computer Science. Springer, 2004. 36

[12] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976. 2, 23, 30

[13] Euclid, T. L. Heath (Translator), and D. Densmore (Editor). *Euclid's Elements*, volume 1. Green Lion Press, 2002. 18

[14] S. Even and Y. Yacobi. Cryptography and NP-completeness. In *Proceedings of the 7th International Colloquium on Automata, Languages and Programming*, pages 195–207. Springer-Verlag, 1980. 11

[15] G. Everest and T. Ward. *An Introduction to Number Theory*. Springer, 2005. 17, 33

[16] S. R. Finch. *Mathematical Constants*. Cambridge University Press, Cambridge, 2003. 31

[17] O. Goldreich. *Foundations of Cryptography: Basic Techniques*. Cambridge University Press, 2001. vi, 11

[18] J. Grantham. A probable prime test with high confidence. *J. Number Theory*, 72(1):32–47, 1998. 40

[19] J. L. Hein. *Discrete Structures, Logic, and Computability*. Jones Bartlett Publishers, Massachusetts, 2009. vi, 16

[20] OEIS Foundation Inc. The on-line encyclopedia of integer sequences, janeiro 2014. http://oeis.org/A051021. 31

[21] ISO/IEC 18032. Information technology - Security techniques - Prime number generation, 2005. 33, 47

[22] D. Kahn. *The Codebreakers: The Story of Secret Writing*. Macmillan Publishing Co., New York, 1967. 1

[23] A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. *Physics-Doklady*, 7:595–596, 1963. 47

[24] A. Kerckhoffs. La cryptographie militaire. *Journal des Sciences Militaires*, 9(1):5–38, 1883. 4

[25] D. E. Knuth. *Bitwise tricks techniques. The Art of Computer Programming*, volume 4. Addison–Wesley Professional, 2009. 26

[26] H. W. Lenstra. Primality testing with gaussian periods. In *Proceedings of the 7th International Colloquium on Automata, Languages and Programming*. Springer-Verlag, 2002. 39

[27] R. R. Maier. Teoria dos números. Universidade de Brasília - IE, 2005. 19, 20

[28] T. Matsumoto and H. Imai. On the key predistribution system: A practical solution to the key distribution problem. In *Advances in Cryptology – CRYPTO*, volume 293 of *Lecture Notes in Computer Science*, pages 185–193. Springer, 1987. 8

[29] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. vi, 2, 3, 7, 8, 9, 10, 12, 28, 35, 48

[30] G. L. Miller. Riemann's hypothesis and tests for primality. *Journal of Computer and Systems Sciences*, 13(3):300–317, 1976. 36

[31] W. H. Mills. A prime-representing function. *Bulletin of the American Mathematical Society*, 53(6):604, 1947. 31

[32] D. Naccache. Padding Attacks on RSA. *Information Security Technical Report*, 4(4):28–33, 1999. 29

[33] W. Narkiewicz. *The Development of Prime Number Theory : From Euclid to Hardy and Littlewood*. Springer-Verlag, Berlin, 2000. 31

[34] S. J. Piestrak. Efcient hamming weight comparators of binary vectors. *Electron. Lett*, 43(11):611–612, 2007. 26

[35] R. G. E. Pinch. The Carmichael Numbers up to 10^{21} , 2007. 35

[36] C. Pomerance, J. L. Selfridge, and S. Wagstaff. The pseudoprimes to $25 \cdot 10^9$. *Mathematics of Computation*, 35(151):1003–1026, 1980. 60

[37] M. O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128–138, 1980. 36

[38] P. A. D. Rezende. Como Controlar o Vigilantismo Global? http://www.cic.unb.br/docentes/pedro/trabs/operamundi.html, Janeiro 2014. 4

[39] P. A. D. Rezende. O que aprendemos com Edward Snowden? http://www.cic.unb.br/docentes/pedro/trabs/aprendercsnowden.html, Janeiro 2014. 4

[40] P. Ribenboim. *The Little Book of Bigger Primes*. Spring-Verlag, New York, 2004. 18, 35

[41] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications ACM*, 21(2):120–126, 1978. 23, 25

[42] B. Schneier. *Applied Cryptography*. John Wiley & Sons, New York, 1996. 2, 7, 11, 18

[43] M. Seysen. A simplified quadratic frobenius primality test, 2005. vii, 40, 41, 47, 51, 60

[44] C. E. Shannon. Communication theory of secrecy systems. *Bell System Technical Journal*, 28(4):656–715, 1949. 2

[45] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE Computer Society Press, 1994. 32

[46] S. Singh. *The Code Book: How to Make It, Break It, Hack It, Crack It*. Delacorte Press, 2003. 1, 4, 11

[47] R. Solovay and V. Strassen. A fast monte-carlo test for primality. *SIAM Journal on Computing*, 6(1):84–85, 1978. 35

[48] D. R. Stinson. *Cryptography: Theory and Practice*. Chapman and Hall/CRC, Florida, 2006. 6, 10, 21

[49] H. C. A. van Tilborg (Editor). *Encyclopedia of Cryptography and Security*. Springer, New York, 2005. 2

[50] xkcd. Webcomic of romance, sarcasm, math, and language, Julho 2013. http://xkcd.com/247/. vi, 24