Notas em Matemática Aplicada e-ISSN 2236-5915

Volume 77, 2014

Editores

SBMAC **Sociedade Brasileira de Matemática Aplicada e Computacional**

2014

A Sociedade Brasileira de Matemática Aplicada e Computacional - SBMAC publica, desde as primeiras edições do evento, monografias dos cursos que são ministrados nos CNMAC.

Para a comemoração dos 25 anos da SBMAC, que ocorreu durante o XXVI CNMAC em 2003, foi criada a série **Notas em Matemática Aplicada** para publicar as monografias dos minicursos ministrados nos CNMAC, o que permaneceu até o XXXIII CNMAC em 2010.

A partir de 2011, a série passa a publicar, também, livros nas áreas de interesse da SBMAC. Os autores que submeterem textos à série Notas em Matemática Aplicada devem estar cientes de que poderão ser convidados a ministrarem minicursos nos eventos patrocinados pela SBMAC, em especial nos CNMAC, sobre assunto a que se refere o texto.

O livro deve ser preparado em **Latex (compatível com o Miktex versão 2.9), as figuras em eps** e deve ter entre **80 e 150 páginas**. O texto deve ser redigido de forma clara, acompanhado de uma excelente revisão bibliográfica e de **exercícios de verificação de aprendizagem** ao final de cada capítulo.

Veja todos os títulos publicados nesta série na página
http://www.sbmac.org.br/p_notas.php

SBMAC **Sociedade Brasileira de Matemática Aplicada e Computacional**

2014

UMA INTRODUÇÃO À CRIPTOGRAFIA DE CHAVE PÚBLICA ATRAVÉS DO MÉTODO EL GAMAL

Luis Menasché Schechter
luisms@dcc.ufrj.br

Departamento de Ciência da Computação
Instituto de Matemática
Universidade Federal do Rio de Janeiro

Sociedade Brasileira de Matemática Aplicada e Computacional

São Carlos - SP, Brasil
2014

Coordenação Editorial: Maria do Socorro Nogueira Rangel

Coordenação Editorial da Série: Fernando Rodrigo Rafaeli

Editora: SBMAC

Capa: Matheus Botossi Trindade

Patrocínio: SBMAC

Catalogação elaborada pela Biblioteca do IBILCE/UNESP
Bibliotecária: Maria Luiza Fernandes Jardim Froner

Schechter, L. M.
Uma Introdução à Criptografia de Chave Pública Através do Método El Gamal - São Carlos, SP : SBMAC, 2014, 124 p., 21.5 cm - (Notas em Matemática Aplicada; v. 77)

e-ISBN 978-85-8215-063-4

1. Criptografia de Chave Pública 2. Método El Gamal 3. Grupos Finitos 4. Problema do Logaritmo Discreto
I. Schechter, Luis Menasché. II. Título. III. Série

CDD - 51

Para Rosa e Luziane,
meus pilares,
meus oásis.

Agradecimentos

Gostaria de agradecer, em primeiro lugar, à Sociedade Brasileira de Matemática Aplicada e Computacional (SBMAC) pela oportunidade oferecida a mim de publicar este livro e de apresentar o seu conteúdo na forma de um mini-curso no CNMAC 2014.

Agradeço aos professores Juliana Vianna Valério, Luziane Ferreira de Mendonça e Marcello Goulart Teixeira, meus colegas no Departamento de Ciência da Computação (DCC) da UFRJ e meus amigos, pelo incentivo para que eu submetesse a proposta deste livro/mini-curso para a SBMAC.

Agradeço também a outro querido amigo, o professor Severino Collier Coutinho, que também é meu colega no DCC/UFRJ. Quando eu ainda era um aluno de graduação em Ciência da Computação, foi com ele que vi pela primeira vez os conceitos de criptografia de chave pública e com ele que aprendi os fundamentos do método El Gamal. Desta forma, o professor Collier criou a raiz do meu interesse presente por esta área de estudo e pesquisa.

A interação com todos os alunos que realizaram comigo a disciplina de graduação de "Números Inteiros e Criptografia" no DCC/UFRJ ao longo dos últimos anos sempre me proporcionou um aprendizado de novas ideias sobre como melhorar o curso para os semestres seguintes e sobre maneiras diferentes de abordar alguns tópicos centrais do conteúdo da disciplina. Também pude aprender bastante com todos os alunos que realizaram comigo atividades de pesquisa na área de criptografia, nos níveis de iniciação científica, mestrado e doutorado. Gostaria, então, de agradecer a todos eles. Espero ter conseguido utilizar este aprendizado da melhor forma na elaboração deste livro.

Finalmente, devo agradecer também ao CNPq pelo auxílio financeiro concedido a mim ao longo do período de elaboração deste trabalho.

Conteúdo

Prefácio **xi**

1 Introdução **1**
1.1 Objetivos da Criptografia 1
1.2 Breve Panorama Histórico 3
1.3 Mudança de Paradigma: Chave Pública 6
1.4 Roteiro 8
1.5 Exercícios 10

2 Aritmética Modular **13**
2.1 Divisibilidade 13
2.2 Relações de Equivalência 24
2.3 Inteiros Módulo n 26
2.4 Operações Modulares 29
2.5 Sistemas de Congruências 35
2.6 Exercícios 38

3 Números Primos **41**
3.1 Conceitos e Resultados Fundamentais 41
3.2 Crivo de Eratóstenes 46
3.3 Pequeno Teorema de Fermat 51
3.4 Teste de Miller-Rabin 56
3.5 Exercícios 59

4 Grupos **61**
4.1 Definições Básicas 61
4.2 Os Grupos Finitos $U(n)$ 64
4.3 Subgrupos 66
4.4 Grupos e Subgrupos Cíclicos 68
4.5 Exercícios 76

5 O Método El Gamal **77**
5.1 Esquema Geral 77
5.2 Criptografia El Gamal 82
5.3 Assinatura Digital El Gamal 88
5.4 Assinatura Digital DSA 94
5.5 Exercícios 98

6 Resolução do Problema do Logaritmo Discreto **101**
6.1 Algoritmo Ingênuo 101
6.2 Algoritmo "Baby-Step/Giant-Step" de Shanks 103
6.3 Algoritmo "Rho" de Pollard 105
6.4 Algoritmo de Pohlig-Hellman 109
6.5 Algoritmo do Cálculo de Índices 115
6.6 Exercícios 120

Prefácio

Estamos experimentando, de forma cada vez maior, o crescimento do tráfego de informações na Internet e o seu uso para operações financeiras, como o comércio eletrônico e o Internet banking, por exemplo. Com isso, a segurança destas trocas de informação através de redes de computadores torna-se cada vez mais um requisito fundamental do ponto de vista comercial, econômico e militar. Ao utilizarmos métodos de criptografia, estamos buscando "esconder" o conteúdo de uma mensagem, de forma que, caso alguém mal-intencionado "roube" uma mensagem criptografada antes dela chegar ao seu destinatário, esta pessoa não será capaz de ler o seu conteúdo original.

O objetivo principal deste livro é fornecer ao aluno um primeiro contato com os fundamentos matemáticos da *criptografia de chave pública* e da *assinatura digital*, que são utilizadas, por exemplo, na troca de mensagens na Internet (como números de cartão de crédito em sites de compras virtuais). Para isso, iremos estudar um dos mais simples métodos de criptografia de chave pública e assinatura digital, mas, ao mesmo tempo, um dos mais utilizados na prática: o *método El Gamal* (que possui este nome devido ao pesquisador que originalmente desenvolveu este método). Iremos estudar os fundamentos matemáticos deste método e entender o que o torna ao mesmo tempo eficiente e seguro.

Os fundamentos matemáticos do método El Gamal envolvem o estudo de propriedades dos números inteiros, o estudo de números primos e o estudo de grupos finitos, conceitos que são, em si, razoavelmente simples, mas que se mostram bastante úteis nesta aplicação prática na área de computação. É devido à simplicidade dos conceitos básicos necessários para este estudo que o método El Gamal se mostra como um tema adequado para um primeiro contato do estudante com os conceitos da área de criptografia.

A criptografia de chave pública é um tópico de estudo interessante na interseção entre Matemática e Computação, pois destaca uma possível aplicação prática de conceitos de matemática discreta assim como ilustra a presença de fundamentações matemáticas nas atividades computacionais. Desta forma, este livro pode interessar tanto a alunos de Matemática e Matemática Aplicada que buscam aplicações práticas para os conceitos e resultados matemáticos que estudam (neste caso, conceitos e resultados de matemática discreta) quanto para alunos de Computação que se interessam pelos fundamentos matemáticos envolvidos nos processos computacionais. O livro também pode ser adequado para alunos de Engenharias que se encaixem nestes perfis. Por se tratar de um livro de caráter introdutório, ele é adequado tanto para alunos de graduação quanto para alunos em início de pós-graduação que ainda estão buscando potenciais temas de pesquisa.

Buscamos abordar os principais conceitos matemáticos necessários para o entendimento do método El Gamal, tornando o texto fortemente auto-contido. Assim, acreditamos que os únicos pré-requisitos para um bom entendimento deste livro são

um conhecimento básico a respeito dos números inteiros e a respeito de números primos, habilidades matemáticas básicas para cálculos aritméticos, manipulações algébricas e formulações de provas matemáticas e familiaridade com a noção de algoritmo.

Rio de Janeiro, 28 de fevereiro de 2014.

Luis Menasché Schechter

Capítulo 1

Introdução

Nosso objetivo principal neste livro é estudar o *método El Gamal* de criptografia de chave pública e de assinatura digital.

Neste capítulo, iniciamos a caminhada rumo a este objetivo discutindo alguns conceitos básicos a respeito dos métodos de criptografia e assinatura digital e falando a respeito dos objetivos por trás da utilização destes métodos. Faremos também um breve apanhando histórico da evolução dos métodos de criptografia, falando de alguns métodos famosos historicamente, como a Cifra de César, usada pelo imperador romano, e a Máquina Enigma, usada pela marinha e aeronáutica alemãs durante a Segunda Guerra Mundial, tendo sido decifrada com a ajuda do matemático inglês Alan Turing.

Além disso, falaremos sobre a grande mudança de paradigma ocorrida na área da criptografia na década de 1970. Neste momento, foram desenvolvidos os primeiros métodos de criptografia conhecidos como *métodos de chave pública* e houve o surgimento dos métodos de *assinatura digital*.

Finalmente, encerramos este capítulo com um roteiro dos assuntos que serão abordados no decorrer dos próximos capítulos.

1.1 Objetivos da Criptografia

A criptografia é o estudo de métodos que possibilitam "esconder" o conteúdo de uma mensagem, de forma que, caso alguém mal-intencionado "roube" uma mensagem criptografada antes dela chegar ao seu destinatário, esta pessoa não será capaz de ler o seu conteúdo original.

A necessidade do uso de criptografia na troca de mensagens surge a partir da existência de diversas fontes de insegurança no trânsito da mensagem entre o remetente e o destinatário.

Neste sentido, um uso clássico e já muito antigo de criptografia é militar. Suponha que dois agrupamentos de um mesmo exército desejam trocar mensagens a respeito da estratégia e da tática de batalha. O comandante de um agrupamento irá escrever a mensagem e entregar a um soldado para que ele a carregue até a localização do segundo agrupamento, onde ela deve ser entregue ao outro comandante. Entretanto, uma vez que estes dois agrupamentos estão inseridos dentro de uma zona de conflito, a viagem do mensageiro é repleta de inseguranças. Uma destas inseguranças, talvez a mais grave do ponto de vista da tática de batalha, é o risco do mensageiro ser capturado por alguma tropa inimiga. Neste caso, os inimigos

teriam acesso à mensagem que contém diversas instruções táticas, ganhando então vantagem na batalha.

Em termos mais gerais, podemos dizer que, no exemplo acima, o mensageiro funciona como o "canal" de transmissão da mensagem entre o remetente (comandante do primeiro agrupamento) e o destinatário (comandante do segundo agrupamento). Temos então que este é um "canal" inseguro para a transmissão de mensagens, já que há a possibilidade de interceptações das mensagens que trafegam neste "canal".

Sempre que temos um cenário como este, em que um remetente deseja enviar uma mensagem para um destinatário onde

1. o canal de transmissão da mensagem é inseguro e

2. a mensagem contém algum tipo de conteúdo privado que só deve ser visto pelo remetente e pelo destinatário,

o uso de algum método de criptografia se faz necessário.

Na aplicação de um método de criptografia, desejamos esconder o conteúdo de uma mensagem m antes de enviá-la através de um canal inseguro. Vamos descrever então como isto é feito de forma genérica no modelo tradicional de criptografia. Dado um método de criptografia, o remetente e o destinatário selecionam conjuntamente uma *chave* k, que é um valor numérico dentro de um intervalo $n_1 \leq k \leq n_2$ estabelecido pelo método. O valor k selecionado deverá ser mantido secreto, sendo de conhecimento apenas do remetente e do destinatário.

O método estabelece também uma função de *encriptação* f e uma função de *decriptação* g. A função de encriptação nos permite, dada uma mensagem m e uma chave k, calcular a mensagem encriptada $\widehat{m} = f(m, k)$. A mensagem que será enviada no canal será então $\widehat{m}$. Desta forma, caso algum intruso intercepte a mensagem trafegando no canal, o que ele verá será $\widehat{m}$ e não m. É importante notar que a mensagem encriptada depende não somente da mensagem original, mas também da chave k que foi escolhida inicialmente. Com isso, torna-se difícil para um intruso que tenha conseguido interceptar a mensagem encriptada $\widehat{m}$ calcular, a partir dela, a mensagem original m sem ter o conhecimento da chave k que foi utilizada na encriptação.

Por outro lado, a função de decriptação nos permite, dada uma mensagem encriptada $\widehat{m}$ e a chave k utilizada na encriptação, calcular novamente a mensagem original $m = g(\widehat{m}, k)$. Obviamente, para que o método tenha utilidade prática, é necessário que a função de decriptação realmente consiga recuperar o conteúdo original de uma mensagem encriptada com a função f. Desta forma, devemos ter a igualdade $g(f(m, k), k) = m$. É importante notar que a função g só se torna útil em conjunto com o conhecimento da chave k utilizada na encriptação. Caso o valor de k tenha sido mantido secreto, sendo de conhecimento apenas do remetente e do destinatário, então apenas o destinatário poderá usar a função g para recuperar o conteúdo original de uma mensagem encriptada. Intrusos que consigam interceptar mensagens no canal inseguro terão acesso apenas a mensagens encriptadas, mas não conseguirão recuperar o conteúdo original delas, já que não tem conhecimento da chave k utilizada.

Conforme explicamos acima, neste modelo de criptografia, a chave de encriptação k deve ser mantida privada entre o remetente e o destinatário, pois qualquer intruso que intercepte uma mensagem encriptada transitando no canal será capaz de aplicar a função de decriptação e recuperar a mensagem original se possuir o conhecimento de k. Por esta razão, a chave k é denominada *chave privada* e este modelo de criptografia é conhecido como *criptografia de chave privada*.

Historicamente, além de aplicações militares como a que descrevemos no exemplo acima, os métodos de criptografia também possuem amplo uso para a proteção de segredos comerciais, empresariais e governamentais.

Atualmente, com o crescimento massivo do uso da Internet e a possibilidade da execução de operações financeiras através dela, como o comércio eletrônico e o Internet banking, por exemplo, o uso da criptografia deixou de ser algo restrito aos contextos mencionados acima e passou a ter uma relevância central no cotidiano de todos os usuários de computadores.

1.2 Breve Panorama Histórico

Um dos métodos conhecidos de criptografia mais antigos é o método denominado *Cifra de César*, que era utilizado pelo imperador romano César para a troca de mensagens com os seus generais em batalha.

Na Cifra de César, assumimos que a mensagem é composta apenas de letras e não fazemos diferenciação entre maiúsculas e minúsculas. Como consideramos o alfabeto contendo 26 letras, a chave de encriptação k será um número inteiro no intervalo $1 \leq k < 26$. A encriptação é realizada letra a letra ao longo da mensagem. A encriptação de uma letra é igual à letra que está k posições mais adiante no alfabeto em relação à letra original. Por exemplo, se $k = 3$, então a encriptação da letra "A" é igual à letra "D". Para que este método de encriptação possa funcionar com todas as letras, independentemente de suas posições no alfabeto, consideramos o alfabeto de forma circular, isto é, a letra seguinte ao "Z" é a letra "A" novamente.

Exemplo 1.1. *Vamos encriptar a mensagem "CESAR" com a Cifra de César, utilizando a chave* $k = 4$.

$$\begin{array}{ccccc} C & E & S & A & R \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ G & I & W & E & V \end{array}$$

Logo, a mensagem encriptada é "GIWEV".

O valor de k utilizado por César para encriptar as mensagens deveria ser acordado com o general que iria recebê-las antes que ele se deslocasse para o campo de batalha. Assim, quando o mensageiro entrega uma mensagem encriptada ao general, ele pode recuperar o seu conteúdo original, já que possui o valor de k que foi utilizado na encriptação. Para realizar a decriptação da mensagem e recuperar o conteúdo original, ele deve proceder letra a letra, calculando a letra que está k posições para trás no alfabeto em relação a cada letra da mensagem encriptada. Vemos que o procedimento de decriptação é extremamente simples para quem conhece o valor de k. Por outro lado, se o mensageiro de César for capturado durante o seu trajeto até o general, os inimigos terão acesso apenas à mensagem encriptada e, não conhecendo o valor de k, terão dificuldades em obter o seu conteúdo.

A Cifra de César possui uma fragilidade. Uma vez fixado o valor de k, todas as letras "A" de uma mensagem serão encriptadas de uma mesma forma. O mesmo acontecerá com todas as letras "B", todas as letras "C" e assim por diante. Desta forma, a frequência relativa das letras na mensagem original é preservada de certa forma na mensagem encriptada. Como exemplo disto, vamos considerar que a mensagem original foi escrita na Língua Portuguesa. A letra que aparece com maior frequência em palavras da Língua Portuguesa é a letra "A". Desta forma, se a mensagem original for longa o suficiente, é bastante provável que a letra que aparece com maior frequência na mensagem também seja o "A". Como todas as

letras "A" são encriptadas de uma mesma forma na mensagem encriptada, temos então que é bastante provável que a letra que aparece com maior frequência na mensagem encriptada seja a encriptação da letra "A". Seguimos então com este processo analisando a segunda letra mais frequente, a terceira letra mais frequente e assim por diante. Em diversas situações, esta análise estatística pode ser suficiente para descobrirmos o conteúdo original da mensagem mesmo sem o conhecimento do valor de k. Mesmo uma análise estatística de apenas algumas poucas letras pode já fornecer indícios suficientes para elaborarmos uma lista de prováveis valores de k. Esta análise estatística é conhecida como *método de contagem de frequência*.

Atualmente, com o advento dos computadores, o método de contagem de frequência pode ser completamente automatizado, de forma que a Cifra de César não oferece mais nenhuma segurança real no momento presente. É possível ainda utilizar os computadores em uma abordagem mais radical. Dado que existem apenas 25 possíveis valores para k, podemos programar um computador para testar exaustivamente todos estes valores, até encontrarmos uma decriptação que resulte em uma mensagem que faz sentido do ponto de vista linguístico.

Como última observação a respeito da Cifra de César, podemos notar que ela é um método de criptografia de *chave privada*, já que a mesma chave utilizada na encriptação é utilizada também na decriptação, o que significa que ela deve permanecer privada, sendo de conhecimento apenas do remetente e do destinatário.

Diversos métodos de criptografia desenvolvidos posteriormente buscaram maneiras de contornar a fragilidade exibida pela Cifra de César. Para isto, tais métodos buscavam maneiras de dificultar o máximo possível a aplicação de técnicas relacionadas ao método de contagem de frequência.

Um método que ficou famoso tanto por sua dificuldade em ser quebrado quanto pelo seu papel histórico é o método de criptografia implementado pela chamada *Máquina Enigma*. Esta máquina foi desenvolvida pela Alemanha nazista para ser utilizada na encriptação das mensagens transmitidas pelos comandos militares da marinha e da aeronáutica alemãs para seus navios, submarinos e aviões durante a Segunda Guerra Mundial, entre 1939 e 1945. Assim como na Cifra de César, a encriptação de uma mensagem é realizada letra a letra. Entretanto, o processo de encriptação de uma letra é consideravelmente mais complexo.

A Máquina Enigma é muito semelhante em aparência a uma máquina de escrever. Entretanto, além do conjunto de teclas, sendo uma para cada letra, a máquina também possui um painel de lâmpadas, sendo novamente uma para cada letra, e um conjunto de 3 ou 4 rotores, sendo que cada um deles pode ser colocado em 26 posições diferentes.

Quando uma tecla da máquina é pressionada, uma das lâmpadas do painel se acende. A letra correspondente à lâmpada que se acendeu representa a encriptação da letra correspondente à tecla que foi pressionada. Além do acendimento de uma das lâmpadas, quando se pressiona uma das teclas, o primeiro rotor gira uma posição. Quando o primeiro rotor completa uma volta, o segundo rotor gira uma posição. Por sua vez, quando o segundo rotor completa uma volta, o terceiro rotor gira uma posição. Um processo análogo ocorre entre o terceiro e o quarto rotores no caso de uma máquina com quatro rotores.

Antes do início da encriptação de uma mensagem, escolhe-se uma posição inicial para os rotores da máquina. Este posicionamento inicial dos rotores funciona como *chave* de encriptação, uma vez que ele irá determinar a encriptação de todas as letras da mensagem. A encriptação de uma letra depende da posição atual de todos os rotores da máquina no momento em que a tecla correspondente àquela letra é pressionada. Uma vez que o pressionamento de qualquer tecla gera uma

movimentação nos rotores, se a mesma tecla for pressionada duas vezes seguidas, a letra correspondente a esta tecla será encriptada pela máquina de duas formas diferentes. Assim, o mapeamento entre letras na mensagem original e letras na mensagem encriptada não é mais fixo como na Cifra de César, de modo que o método de contagem de frequências não pode mais ser diretamente aplicado para burlar a segurança da criptografia da Máquina Enigma. Considerando que existem 26^3 ou 26^4 possíveis posicionamentos dos rotores, nota-se que qualquer análise estatística com o objetivo de obter o conteúdo da mensagem original a partir da interceptação de uma mensagem encriptada e sem o conhecimento da chave de encriptação deverá ser consideravelmente mais refinada do que a análise estatística utilizada no caso da Cifra de César.

Pode-se notar que, apesar de ser mais sofisticada do que a Cifra de César, a criptografia da Máquina Enigma também é um método de criptografia de *chave privada*. Caso um intruso saiba a posição inicial dos rotores da máquina (e possua uma Máquina Enigma à disposição), ele poderá realizar a decriptação das mensagens.

Quando a Máquina Enigma começou a ser utilizada pelos alemães na Segunda Guerra Mundial, acreditava-se que a criptografia implementada por ela era impossível de ser quebrada. De fato, durante um bom tempo, os esforços para acessar o conteúdo das mensagens encriptadas com a Enigma se mostraram inúteis. Isto, por sua vez, se refletiu em uma grande vantagem da Alemanha sobre a Inglaterra e a França. Em particular, os submarinos alemães conseguiram impor grandes derrotas à marinha britânica.

Com o passar do tempo, entretanto, a Máquina Enigma mostrou possuir vulnerabilidades. Um time de cientistas, matemáticos e militares ingleses conseguiu desenvolver um método para obter o conteúdo original de mensagens encriptadas com a Enigma. Como esperado, este método era baseado em análises estatísticas extremamente sofisticadas. De fato, estas análises eram tão sofisticadas que foi necessária a construção de computadores dedicados somente a realizar todos os cálculos necessários para a obtenção do conteúdo das mensagens. Devido a isso, a quebra da criptografia da Máquina Enigma acabou por se tornar uma das primeiras aplicações de grande porte da computação.

Com a quebra da criptografia da Máquina Enigma, pouco a pouco a vantagem militar alemã foi sendo revertida. O momento da quebra desta criptografia coincide aproximadamente com o momento da mudança dos rumos da guerra, que passou a tender para o lado dos aliados. Desta forma, o trabalho de quebra da criptografia da Máquina Enigma teve relevância fundamental para o momento que estava sendo vivido na Europa.

O matemático inglês Alan Turing fazia parte do time que conseguiu a quebra da criptografia da Máquina Enigma e teve um papel fundamental tanto na elaboração dos cálculos necessários para a obtenção do conteúdo das mensagens como no projeto e construção dos computadores que passaram a realizar tais cálculos. Alan Turing é um dos pioneiros da Ciência da Computação, tendo desenvolvido, além deste trabalho fundamental na área de criptografia, trabalhos nas áreas de Teoria da Computação, Inteligência Artificial e Computação Científica. Para maiores detalhes sobre sua vida e suas diversas contribuições ao desenvolvimento da computação, o livro [9] pode ser consultado.

Para os leitores que se interessam pelo desenvolvimento histórico dos métodos de criptografia, recomendamos a consulta ao livro [11]. Este livro realiza uma análise histórica bastante detalhada e completa a respeito da evolução dos métodos de criptografia ao longo do tempo, desde a antiguidade até os dias de hoje.

1.3 Mudança de Paradigma: Chave Pública

Todos os métodos de criptografia desenvolvidos e utilizados desde a antiguidade até a década de 1970 pertenciam à categoria dos métodos de *chave privada*. Isto começou a mudar em 1976 com a publicação do artigo "*New Directions in Cryptography*" (Novas Direções na Criptografia) [4] por Whitfield Diffie e Martin Hellman.

Este artigo, que se inicia com a profética frase "*We stand today on the brink of a revolution in cryptography*" (Estamos hoje no limiar de uma revolução na criptografia), apresenta pela primeira vez os conceitos de *criptografia de chave pública* e *assinatura digital*.

Em um método de criptografia de chave privada, o remetente e o destinatário devem estabelecer uma chave de encriptação e decriptação antes de iniciarem a troca de mensagens em um canal inseguro. Isto significa, em particular, que um não pode comunicar ao outro o valor da chave através deste canal, tendo que realizar esta comunicação pessoalmente ou através de algum meio confiável. Tal requisito de pré-comunicação da chave através de um canal secundário (e seguro) passa a se mostrar fortemente inconveniente em um mundo onde o uso de redes de computadores passa a se difundir e as potenciais aplicações econômicas e comerciais da computação em rede começam a despertar interesse.

A mudança de paradigma dos métodos de chave pública, em relação aos métodos anteriores de chave privada, é que eles não dependem mais do segredo completo da chave utilizada na encriptação. Nos modelos de criptografia de chave pública, a chave usada para encriptar uma mensagem pode ser de conhecimento público e apenas a chave utilizada para decriptá-la deve ser mantida de forma privada pelo *destinatário* da mensagem. Assim, qualquer um será capaz de encriptar uma mensagem, mas somente o destinatário legítimo dela possuirá a chave que permite decriptá-la e lê-la.

O desenvolvimento de métodos de criptografia de chave pública também possibilitou o surgimento de outro conceito complementar, o de *assinatura digital*. Um método de criptografia tem o objetivo de oferecer uma garantia de *privacidade* com relação à mensagem, isto é, garantir que o seu conteúdo estará acessível apenas para o remetente e o destinatário legítimo da mensagem. Já um método de assinatura digital tem o objetivo de oferecer uma garantia de *autenticidade* com relação à mensagem, isto é, garantir que uma determinada mensagem foi realmente criada por seu suposto remetente. Desta forma, enquanto a criptografia provê segurança com relação ao acesso no lado do destinatário, a assinatura digital provê segurança com relação à origem no lado do remetente.

A partir desta ideia, nos modelos de assinatura digital, a chave de assinatura deverá ser mantida de forma privada pelo *remetente* da mensagem, enquanto a chave utilizada na verificação da autenticidade da assinatura pode ser de conhecimento público. Assim, ninguém além do remetente será capaz de assinar suas mensagens em seu lugar, mas qualquer um será capaz de verificar se uma assinatura em uma mensagem foi realmente produzida pelo suposto remetente.

Estes novos modelos de criptografia e assinatura digital impõem novos desafios teóricos, pois, para que eles sejam realmente seguros, não deve haver nenhum procedimento computacional simples para calcular o valor da chave privada (de decriptação ou de assinatura) a partir do valor conhecido da chave pública (de encriptação ou de verificação). Caso tal procedimento exista, então o método não estará oferecendo segurança nenhuma, porque qualquer um que possua a chave privada será capaz de decriptar e ler as mensagens, mesmo não sendo o seu destinatário legítimo, no caso de um método de criptografia, ou será capaz de falsificar assina-

turas em mensagens, no caso de um método de assinatura digital. Desta forma, a segurança dos métodos de criptografia de chave pública e de assinatura digital se baseia na dificuldade computacional de se resolver alguns problemas matemáticos. Em outras palavras, em um método de criptografia de chave pública ou de assinatura digital seguro, a tarefa de se calcular o valor da chave privada a partir do valor conhecido da chave pública é equivalente à tarefa de se resolver um problema matemático com altíssima complexidade computacional.

Atualmente, os métodos de criptografia de chave pública e de assinatura digital assumiram uma importância muito grande com a difusão e popularização dos computadores e da Internet. Os métodos de chave pública e de assinatura são usados massivamente em serviços de comércio eletrônico, Internet banking e em protocolos de autenticação em páginas web assim como para a troca de dados sensíveis através da Internet. A segurança destes métodos tornou-se então um requisito fundamental do ponto de vista comercial, econômico e militar.

Dois dos métodos de criptografia de chave pública e de assinatura digital mais utilizados atualmente são o *RSA* [25], desenvolvido por Ron Rivest, Adi Shamir e Leonard Adleman e o *El Gamal* [5], desenvolvido pelo pesquisador de mesmo nome. No caso do RSA, o cálculo do valor da chave privada a partir do valor conhecido da chave pública é equivalente ao problema de fatorar um número inteiro muito grande, o que é um problema muito difícil de ser resolvido, mesmo com a ajuda de um computador (não há algoritmos eficientes para a fatoração de inteiros no caso geral). Já no caso do El Gamal, o cálculo da chave privada a partir da chave pública é equivalente à resolução de uma instância do Problema do Logaritmo Discreto, um problema oriundo da teoria de grupos que estudaremos em detalhes mais adiante neste livro. Este também é um problema muito difícil de ser resolvido, mesmo com a ajuda de um computador (também não há algoritmos eficientes para ele no caso geral).

Mesmo não sendo conhecidos algoritmos eficientes para resolver estes problemas, existem alguns algoritmos que, apesar de não serem bons o suficiente para resolvê-los no caso geral, podem ser capazes de resolver algumas instâncias do Problema da Fatoração ou do Problema do Logaritmo Discreto até um certo limite de tamanho. Isto impõe um tamanho mínimo que as chaves utilizadas devem possuir para que o sistema seja seguro, isto é, para que não seja possível utilizar-se na prática um dos algoritmos conhecidos para realizar o cálculo da chave privada. Conforme o poder de processamento dos computadores vai aumentando ao longo do tempo, o tamanho mínimo das chaves para que o sistema seja seguro também aumenta.

O aumento do tamanho das chaves para a manutenção da segurança também acarreta em um efeito colateral: dadas duas cópias do mesmo sistema de criptografia ou assinatura digital, uma que utiliza uma chave pequena e outra que utiliza uma chave grande, o processamento de encriptação e decriptação ou de assinatura e verificação das mensagens por usuários legítimos do sistema será mais lento no caso em que a chave é maior. Desta forma, o aumento do tamanho das chaves para a manutenção da segurança acaba acarretando em uma perda de desempenho para os usuários legítimos que utilizam o sistema.

O estudo dos métodos de criptografia de chave pública e de assinatura digital possui um lado "construtivo" e um lado "destrutivo". No lado "construtivo", estudamos a fundamentação matemática do método de criptografia ou assinatura digital em consideração, a segurança deste método e a sua implementação de forma eficiente no computador. No lado "destrutivo", estudamos os diversos algoritmos existentes para a resolução do problema matemático subjacente ao método de criptografia de chave pública ou assinatura digital em questão. No caso específico do

método El Gamal, que será o método estudado ao longo deste livro, devemos estudar os algoritmos existentes para a resolução do Problema do Logaritmo Discreto em grupos finitos. Este estudo do lado "destrutivo" não interessa apenas a pessoas mal intencionadas. Ele é muito importante também para quem implementa o sistema e para os usuários legítimos dele. O conhecimento dos algoritmos disponíveis para a resolução do problema matemático associado a um método de criptografia de chave pública ou assinatura digital é importante do ponto de vista prático, pois, mesmo que este problema seja difícil de ser computacionalmente resolvido no caso geral, os algoritmos existentes são eficientes em alguns casos particulares. Tais casos devem então ser evitados durante a construção dos sistemas, para que sua segurança não seja comprometida.

Desta forma, tentando obter uma visão global dos vários aspectos envolvidos na implementação segura e eficiente de métodos de criptografia de chave pública e de assinatura digital, estudaremos o problema matemático subjacente ao método El Gamal, que é o Problema do Logaritmo Discreto em grupos finitos, como ele é utilizado na construção do método e também diversos algoritmos apresentados na literatura para a resolução de alguns casos deste problema matemático.

Encerramos esta seção apresentando alguns outros livros que tratam de métodos de criptografia de chave pública e de assinatura digital, assim como dos problemas matemáticos subjacentes a eles. Para um estudo detalhado do método RSA, descrito brevemente acima, o livro [3] é uma ótima referência em Língua Portuguesa. Outros livros que abordam estes tópicos com um bom nível de detalhe são [16], [13] e [10].

1.4 Roteiro

Neste livro, temos o objetivo de estudar o *método El Gamal* de criptografia de chave pública e de assinatura digital. De modo a delinear nosso caminho até este objetivo, apresentamos a seguir um roteiro dos assuntos que serão abordados no decorrer dos próximos capítulos.

Após o panorama introdutório apresentado ao longo deste capítulo, iniciamos, no **Capítulo 2**, a apresentação dos conceitos matemáticos subjacentes à construção do método El Gamal.

No **Capítulo 2**, apresentamos a chamada *aritmética modular*, que será utilizada em todos os cálculos realizados pelo método El Gamal. A noção central da aritmética modular é a noção de *relação de congruência módulo* n, onde n é um inteiro maior do que 1.

Iniciamos o capítulo discutindo os conceitos básicos de divisibilidade entre números inteiros. Um conceito muito importante que será apresentado é o *máximo divisor comum* entre dois inteiros positivos a e b. Iremos apresentar também um algoritmo que, além de calcular o máximo divisor comum, calcula simultaneamente outros dois inteiros que serão úteis para nossas aplicações. Este é o Algoritmo Euclidiano Estendido.

Em seguida, apresentamos o conceito de *relação de equivalência* para, logo após, discutirmos o caso particular de relação de equivalência que irá nos interessar pelo restante do livro: a relação de congruência módulo n. Para podermos definir esta relação, utilizamos os conceitos de divisibilidade entre inteiros. Descreveremos então os conjuntos $\mathbb{Z}_n$, onde $n > 1$, dos inteiros módulo n, construídos a partir das relações de congruência módulo n. Explicaremos também como realizar as operações aritméticas de soma, diferença, produto e potenciação com elementos destes conjuntos e como calcular inversos multiplicativos destes elementos. Em

particular, as operações de potenciação e de cálculo de inversos multiplicativos são operações fundamentais para os cálculos realizados durante a execução do método El Gamal. Mostraremos ainda que o Algoritmo Euclidiano Estendido e os valores que ele calcula são especialmente úteis no cálculo de inversos multiplicativos de elementos de um conjunto $\mathbb{Z}_n$.

Finalizando este capítulo, discutimos o chamado Teorema Chinês do Resto e o seu algoritmo associado, que são úteis para a resolução de sistemas de congruências modulares.

No **Capítulo 3**, continuamos a nossa apresentação de conceitos matemáticos básicos necessários para as nossas aplicações com uma pequena discussão a respeito dos números primos.

Primeiramente, apresentamos os conceitos fundamentais de *número primo* e *número composto*. Em seguida, descrevemos uma prova bastante simples de que existem infinitos números primos. Certamente, este é um resultado de que ninguém duvida. Mesmo assim, é importante termos uma prova formal dele, uma vez que números primos são necessários para a construção das chaves utilizadas pelo método El Gamal. Portanto, a infinidade dos números primos nos garante que sempre podemos construir novas chaves conforme seja necessário, isto é, o método El Gamal nunca se tornará obsoleto pelo esgotamento dos números primos utilizados.

Outro resultado fundamental que apresentamos a respeito dos números primos é o da unicidade da fatoração de qualquer inteiro positivo $n \geq 2$ em fatores primos.

Em seguida, apresentamos o crivo de Eratóstenes, um método bastante simples para obter uma lista de todos os números primos até um determinado limite superior.

Retornando à aritmética modular, apresentamos o Pequeno Teorema de Fermat, um resultado muito útil para cálculos com inteiros modulares em um conjunto $\mathbb{Z}_p$, onde p é primo. Descrevemos também como utilizar o Pequeno Teorema de Fermat em conjunto com o Algoritmo Chinês do Resto para simplificar bastante o cálculo de potências modulares em alguns casos.

Finalmente, encerrando nossa discussão sobre números primos, apresentaremos o teste de Miller-Rabin, que é uma forma muito eficiente de testar computacionalmente se um dado número é ou não primo sem a necessidade de obter a sua fatoração. Como precisamos de números primos para construir as chaves do método El Gamal, se não possuíssemos uma forma eficiente de testar a primalidade de números, então não teríamos como utilizar o El Gamal na prática.

No **Capítulo 4**, apresentamos nosso último tópico entre os conceitos matemáticos básicos necessários para as nossas aplicações, com a discussão da estrutura algébrica conhecida como *grupo*.

Primeiramente, iremos apresentar a definição formal do que é um grupo, sendo que estaremos mais interessados em grupos finitos. Em seguida, descreveremos o nosso foco principal, que são grupos construídos a partir das relações de *congruência módulo n*. Tais grupos são denotados por $U(n)$ e seus elementos são os elementos de $\mathbb{Z}_n$ que possuem inverso multiplicativo módulo n. Iremos também estudar de maneira particular os grupos $U(p)$, onde p é um primo, já que os cálculos do método El Gamal são realizados em grupos deste tipo.

Ao longo do capítulo, apesar de nosso foco principal ser os grupos $U(n)$ e $U(p)$, também apresentaremos algumas noções gerais oriundas da teoria de grupos. Entre elas, discutiremos as noções de subgrupos e de grupos e subgrupos cíclicos. Esta última noção também se mostrará muito importante para a construção do método El Gamal. Além disso, completaremos nossa discussão apresentando e provando alguns resultados fundamentais sobre grupos, como o Teorema de Lagrange e o

Teorema da Raiz Primitiva, entre outros. De maneira mais ou menos explícita, cada um destes resultados é importante na construção do método El Gamal.

Finalmente, apresentamos também neste capítulo a formulação do Problema do Logaritmo Discreto, um problema da teoria de grupos que está intimamente relacionado à segurança do método El Gamal.

No **Capítulo 5**, após todos os conceitos matemáticos necessários a respeito dos números inteiros, da aritmética modular e da teoria de grupos terem sido apresentados, estamos prontos para entender o funcionamento do método El Gamal.

Neste capítulo, apresentamos o método El Gamal para criptografia e para assinatura digital, assim como o método DSA para assinatura digital, que é uma pequena variação do El Gamal. Mostraremos como construir as chaves de encriptação e decriptação (ou chaves de assinatura e verificação, no caso da assinatura digital) e quais são os cálculos necessários para encriptar e decriptar uma mensagem (ou para assinar uma mensagem e verificar uma assinatura).

Discutiremos o funcionamento do método, mostrando que a mensagem original sempre pode ser recuperada por quem possui a chave correta de decriptação. Analogamente, no caso da assinatura digital, mostramos também que uma assinatura sempre pode ser produzida para qualquer mensagem por quem possui a chave correta de assinatura. Por outro lado, mostraremos que o método é bastante seguro, mostrando que a única maneira conhecida para um usuário não autorizado calcular o valor da chave de decriptação a partir apenas do conhecimento da chave de encriptação (ou o valor da chave de assinatura a partir da chave de verificação) envolve a resolução do Problema do Logaritmo Discreto em um grupo finito, que é um problema computacionalmente difícil. Desta forma, podemos concluir que o método El Gamal é um método efetivo e seguro de criptografia de chave pública e assinatura digital.

Encerrando o livro, apresentamos, no **Capítulo 6**, alguns algoritmos para a resolução do Problema do Logaritmo Discreto em grupos finitos cíclicos.

Este estudo é importante, pois, apesar da resolução do Problema do Logaritmo Discreto ser computacionalmente difícil no caso geral, existem casos particulares em que os algoritmos conhecidos são eficientes. Desta forma, é importante o conhecimento destes algoritmos e a análise de em quais casos eles são eficientes para que, ao implementarmos um método de criptografia ou assinatura digital baseado em grupos, como o El Gamal e o DSA, possamos contornar e evitar tais casos, já que estes são justamente os casos em que a segurança dos métodos estará comprometida.

Neste capítulo, estudaremos o chamado algoritmo ingênuo, também conhecido como algoritmo de busca exaustiva ou algoritmo de força bruta, o Algoritmo "Baby-Step/Giant-Step" de Shanks, o Algoritmo "Rho" de Pollard, o Algoritmo de Pohlig-Hellman e o Algoritmo do Cálculo de Índices. Estes algoritmos, em conjunto com algumas variantes, compõem a maioria dos métodos conhecidos para a resolução do Problema do Logaritmo Discreto, de forma que estaremos fazendo um estudo bastante completo deste tópico.

1.5 Exercícios

1. Qual é o objetivo principal da criptografia? O que torna a sua aplicação necessária?

2. Explique as diferenças entre um método de criptografia de chave privada e um método de criptografia de chave pública. Dê dois exemplos de métodos pertencentes a cada categoria.

3. Explique as diferenças entre os objetivos de um método de criptografia de chave pública e de um método de assinatura digital. Explique também as diferenças entre os seus funcionamentos.

4. Explique o funcionamento da Cifra de César. Ela é um método de chave privada ou de chave pública? Por quê?

5. Encripte o seu primeiro nome com a Cifra de César, utilizando $k = 6$.

6. Sabendo que a mensagem "OVQKWKV" foi encriptada com a Cifra de César utilizando $k = 10$, decripte a mensagem e recupere o seu conteúdo original.

7. Explique a fragilidade da Cifra de César.

Capítulo 2

Aritmética Modular

Após o panorama introdutório apresentado no capítulo anterior, iniciamos, neste capítulo, a apresentação dos conceitos matemáticos subjacentes à construção do método El Gamal.

Neste capítulo, apresentaremos a chamada *aritmética modular*, que será utilizada em todos os cálculos realizados pelo método El Gamal. A noção central da aritmética modular é a noção de *relação de congruência módulo* n, onde n é um inteiro maior do que 1.

Iniciamos o capítulo discutindo os conceitos básicos de divisibilidade entre números inteiros. Um conceito muito importante que será apresentado é o *máximo divisor comum* entre dois inteiros positivos a e b. Iremos apresentar também um algoritmo que, além de calcular o máximo divisor comum, calcula simultaneamente outros dois inteiros que serão úteis para nossas aplicações. Este é o Algoritmo Euclidiano Estendido.

Em seguida, apresentamos o conceito de *relação de equivalência* para, logo após, discutirmos o caso particular de relação de equivalência que irá nos interessar pelo restante do livro: a relação de congruência módulo n. Para podermos definir esta relação, utilizamos o conceito de divisibilidade entre inteiros. Descreveremos então os conjuntos $\mathbb{Z}_n$, onde $n > 1$, dos inteiros módulo n, construídos a partir das relações de congruência módulo n. Explicaremos também como realizar as operações aritméticas de soma, diferença, produto e potenciação com elementos destes conjuntos e como calcular inversos multiplicativos destes elementos. Em particular, as operações de potenciação e de cálculo de inversos multiplicativos são operações fundamentais para os cálculos realizados durante a execução do método El Gamal. Mostraremos ainda que o Algoritmo Euclidiano Estendido e os valores que ele calcula são especialmente úteis no cálculo de inversos multiplicativos de elementos de um conjunto $\mathbb{Z}_n$.

Finalizando este capítulo, discutimos o chamado Teorema Chinês do Resto e o seu algoritmo associado, que são úteis para a resolução de sistemas de congruências modulares.

2.1 Divisibilidade

Nesta seção, apresentamos os conceitos básicos de divisibilidade entre números inteiros. Para isto, iniciamos pelas definições de *quociente* e *resto* de uma divisão inteira.

Definição 2.1 (Quociente e Resto)**.** *Dados dois inteiros a, chamado de* dividendo*, e $b > 0$, chamado de* divisor*, definimos o* quociente q *e o* resto r *da divisão inteira de a por b como os inteiros que satisfazem as seguintes condições:*

$$a = bq + r \qquad e \qquad 0 \leq r < b.$$

Tanto na definição acima como no restante do texto, estaremos sempre considerando que os divisores são positivos, já que este caso é suficiente para todas as nossas necessidades. Em particular, ele é suficiente para a nossa definição, mais adiante neste capítulo, da noção de *relação de congruência módulo n*.

Em princípio, a definição acima pode parecer fraca. Todos nós aprendemos na escola como realizar o cálculo de uma divisão inteira (também chamada de divisão com resto) de forma a obter quociente e resto que satisfaçam a definição acima. Entretanto, não é tão óbvio a partir dela que existirá apenas um par de quociente e resto que satisfaça as condições acima. Para um mesmo par de dividendo e divisor, poderia haver outros pares de quociente e resto, diferentes daquele que calculamos, que também satisfaçam a definição acima?

Felizmente, a resposta é não. Para cada divisão inteira, existe realmente apenas um par de quociente e resto satisfazendo as condições acima. Isto significa que, não importa quantos métodos diferentes possam existir para realizar este cálculo, caso os métodos efetivamente produzam o resultado correto do cálculo, eles todos retornarão os mesmos valores de quociente e resto.

Vamos provar este resultado no teorema abaixo. Este teorema é um bom exemplo inicial de um resultado de *unicidade*.

Teorema 2.1 (Unicidade do Quociente e Resto)**.** *Dados dois inteiros a e $b > 0$, existe um único par de inteiros q e r que satisfaz as condições da definição acima.*

Demonstração: Suponha que, para um dado par de inteiros a e $b > 0$, existam dois pares de inteiros (q, r) e (q', r') que satisfaçam as condições acima. Nosso objetivo é mostrar que $q = q'$ e $r = r'$. Temos então

$$a = bq + r \qquad \text{e} \qquad a = bq' + r',$$

onde $0 \leq r, r' < b$. Vamos supor, sem perda de generalidade, que $r' \geq r$. Como tanto r quanto r' são menores do que b, a sua diferença também é menor do que b. Temos então $0 \leq r' - r < b$. Subtraindo as duas igualdades acima, obtemos

$$0 = b(q - q') + (r - r'),$$

o que significa que $r' - r = b(q - q')$. Como $0 \leq r' - r < b$, temos $0 \leq b(q - q') < b$. Como $b \neq 0$, isto é equivalente a $0 \leq q - q' < 1$. Uma vez que q e q' são inteiros e o único inteiro maior ou igual a 0 e menor do que 1 é 0, temos que $q - q' = 0$. Isto nos permite concluir que $q = q'$ e, consequentemente, que $r = r'$. ■

Apresentamos a seguir um algoritmo bem simples, embora extremamente ineficiente no caso geral, para calcular o quociente e o resto de uma divisão inteira quando o dividendo e o divisor são ambos inteiros positivos. Apesar deste algoritmo não ter muita utilização prática, a sua simplicidade torna-o um bom exemplo para ilustrarmos como iremos apresentar os algoritmos ao longo deste livro.

Para descrever completamente um algoritmo, precisamos especificar três coisas: os dados que o algoritmo recebe como *entrada*, os dados que ele deve produzir como *saída* e a sequência de *instruções* que ele deve executar para conseguir isto.

Algoritmo 2.1: Algoritmo Simples para Divisão Inteira

Entrada: Dois números inteiros $a, b > 0$.

Saída: Dois números inteiros q e r tais que $a = bq + r$ e $0 \leq r < b$.

Instruções:

1. $q \leftarrow 0$ # A variável q armazena o valor 0
2. $r \leftarrow a$ # A variável r armazena o valor de a
3. Enquanto $r \geq b$, faça:
   3.1. $q \leftarrow q + 1$ # Soma-se 1 ao valor armazenado em q
   3.2. $r \leftarrow r - b$ # Subtrai-se b do valor armazenado em r
4. Retorne q e r.

Antes de discutirmos o algoritmo em si, vamos observar as notações que estamos utilizando na sua descrição acima. Estas mesmas notações serão utilizadas nos outros algoritmos apresentados neste livro.

As instruções no formato

$$x \leftarrow t$$

significam que o valor t será armazenado dentro da variável x. Ao mencionarmos variáveis em um algoritmo, é importante também lembrar que o termo "variável" tem significados distintos em um algoritmo e em uma equação matemática. Em um algoritmo, uma variável é um rótulo para uma posição de memória onde iremos armazenar valores. Desta forma, o valor armazenado em uma variável pode ser alterado durante a execução de um algoritmo. Como exemplo, no algoritmo acima, a expressão

$$q \leftarrow q + 1$$

significa que iremos tomar o valor que está armazenado em q, somar 1 a este valor e então armazenar o resultado desta soma de volta na variável q. Com isso, o valor armazenado em q é efetivamente alterado pela execução desta instrução. As instruções neste formato são chamadas de instruções de *atribuição*, já que atribuem um novo valor à uma variável, isto é, armazenam um novo valor na variável. Em uma instrução de atribuição, o lado direito da atribuição (a expressão que aparece à direita do símbolo $\leftarrow$) é sempre totalmente calculado para só então o resultado deste cálculo ser armazenado na variável indicada no lado esquerdo da atribuição.

Os algoritmos, em geral, apresentam instruções de repetição e/ou instruções condicionais. Uma instrução de repetição é uma instrução que orienta o algoritmo a repetir um determinado bloco de instruções enquanto uma determinada condição se mantiver verdadeira. Já uma instrução condicional é uma instrução que orienta o algoritmo a executar um determinado bloco de instruções apenas se uma determinada condição for verdadeira. Temos também instruções condicionais mais elaboradas, que orientam o algoritmo a executar um determinado bloco de instruções se uma determinada condição for verdadeira ou um outro bloco de instruções se esta mesma condição for falsa. Tais instruções condicionais funcionam como uma "bifurcação" no algoritmo.

Na nossa notação para os algoritmos, os blocos de instruções que se encontram dentro de instruções de repetição ou de instruções condicionais aparecem com um recuo maior em relação à instrução de repetição ou instrução condicional inicial e com a numeração de suas instruções se iniciando sempre com o mesmo número da instrução inicial. Como exemplo, no algoritmo acima, temos a instrução de repetição

3. Enquanto $r \geq b$, faça:

   3.1. $q \leftarrow q + 1$

   3.2. $r \leftarrow r - b$

Todas as instruções que fazem parte do bloco que deve ser repetido estão recuadas em relação à instrução de repetição inicial e começam com o mesmo número da instrução inicial (3). Desta forma, com esta notação, é fácil determinar quais instruções devem ser repetidas ou quais instruções são condicionalmente executadas.

Finalmente, encerrando por hora a discussão sobre a notação que utilizamos para descrever os algoritmos, vamos explicar o que significa o símbolo #. Quando este símbolo aparece em alguma linha da sequência de instruções, tudo o que vem depois dele é um *comentário*, isto é, alguma explicação ou auxílio sobre aquela instrução. Um comentário serve apenas para as pessoas que estão lendo o algoritmo, mas não afeta de forma alguma a execução da instrução onde ele aparece.

Vamos agora discutir especificamente o Algoritmo da Divisão que apresentamos acima. Como ele possui uma instrução de repetição, é importante verificar que o algoritmo não fica repetindo eternamente o bloco de instruções dentro desta instrução de repetição. Também é importante verificar, como em qualquer algoritmo, que o resultado que ele produz ao final da execução de todas as instruções é correto, isto é, satisfaz a descrição dada pela *saída* do algoritmo. No caso particular do Algoritmo da Divisão, precisamos nos certificar que o resultado final é realmente o quociente e o resto da divisão inteira dos números fornecidos como entrada.

Para verificar estes dois pontos, vamos olhar a sequência de valores armazenados nas variáveis q e r ao longo da execução do algoritmo. O valor armazenado nestas variáveis é alterado a cada repetição da instrução de repetição que destacamos acima. A tabela abaixo exibe as sequências de valores.

Repetições	0	1	2	3	...	$u-1$	u
q	0	1	2	3	...	$u-1$	u
r	a	$a-b$	$a-2b$	$a-3b$	...	$a-(u-1)b$	$a-ub$

Vemos que a variável r assume a sequência estritamente decrescente de valores $a > a-b > a-2b > \ldots$. Enquanto o valor em r for maior ou igual a b, as repetições continuam. Mas todos estes valores na sequência são inteiros distintos e existe uma quantidade finita de inteiros menores do que a e maiores ou iguais a b. Logo, as repetições têm que parar eventualmente.

Os últimos valores armazenados nas variáveis q e r são u e $a - ub$, logo estes são os valores que o algoritmo retorna como quociente e resto, respectivamente. Se o último valor armazenado em r é $a - ub$, isto significa que não ocorreram mais repetições no algoritmo após este valor ser armazenado em r. Mas as repetições só param quando o valor em r se torna menor do que b. Assim, o resto retornado pelo algoritmo satisfaz a condição de ser menor do que b. Além disso, como o quociente retornado é u e o resto retornado é $a - ub$, eles também satisfazem a condição $a = bq + r$, já que $bu + (a - ub) = a$. A única condição que ainda precisamos verificar é que o resto retornado é maior ou igual a zero. O penúltimo valor armazenado em r é $a-(u-1)b$. Como ainda ocorre uma repetição após este valor ser armazenado em r, isto significa que $a-(u-1)b \geq b$. Subtraindo b de ambos os lados da inequação, obtemos $a - ub \geq 0$. Mas $a - ub$ é o resto retornado pelo algoritmo. Logo, ele é maior ou igual a zero. Podemos concluir então que o Algoritmo da Divisão descrito acima retorna o resultado correto.

O algoritmo que apresentamos só funciona quando o dividendo é um inteiro positivo, mas ele pode ser facilmente generalizado para um algoritmo que aceite como dividendo qualquer número inteiro. Deixamos esta generalização como exercício (Exercício 1).

Este algoritmo é muito ineficiente no caso geral, pois o número de repetições que ele realiza é sempre igual ao quociente da divisão. Um algoritmo de divisão inteira mais eficiente, mas que ainda é razoavelmente simples de ser programado, conhecido como Algoritmo de Divisão Longa, pode ser encontrado em [12].

Agora que já estamos bem familiarizados com a operação de divisão inteira e com as noções de quociente e resto, vamos apresentar uma definição que é derivada diretamente destas noções.

Definição 2.2. *Sejam a e $b > 0$ números inteiros. Dizemos que a é* múltiplo *de b, ou que a é* divisível *por b, ou que b é* divisor *de a, ou que b* divide *a, ou que b é* fator *de a, se $a = ba'$, para algum $a' \in \mathbb{Z}$, isto é, se a divisão de a por b produz resto* 0*.*

Muitas vezes, apressadamente, uma pessoa pode dizer que "x é divisível por y" quando queria dizer que "x divide y", ou vice-versa. É preciso sempre tomar muito cuidado com estas nomenclaturas.

Exemplo 2.1. *Os divisores do número* 12 *são os elementos do conjunto* $D = \{1, 2, 3, 4, 6, 12\}$.

Definição 2.3. *Um* divisor próprio *ou* fator próprio *de um número inteiro a é um fator de a diferente de* 1 *e do próprio a.*

Exemplo 2.2. *Os divisores próprios de* 12 *são os elementos do conjunto* $D' = \{2, 3, 4, 6\}$.

Vamos agora apresentar uma noção muito importante no estudo da teoria dos números inteiros: a noção de máximo divisor comum entre dois inteiros positivos.

Definição 2.4. *Sejam a e b dois números inteiros positivos. Definimos o* máximo divisor comum *entre a e b, denotado por $mdc(a, b)$, como o maior inteiro que é simultaneamente divisor de a e divisor de b.*

Em muitas situações, precisaremos ser capazes de calcular o máximo divisor comum entre dois inteiros positivos a e b. Uma das maneiras mais simples seria calcular o conjunto de todos os divisores de a e o conjunto de todos os divisores de b e então buscar o maior número que aparece em ambos os conjuntos. Entretanto, este método é extremamente ineficiente na prática quando a e b são números grandes.

Vamos apresentar um algoritmo para o cálculo do máximo divisor comum que é bem mais eficiente do que a estratégia acima e que ainda oferece um benefício extra: dados dois inteiros positivos a e b, além dele calcular $d = \text{mdc}(a, b)$, ele também calcula dois inteiros α e β tais que

$$\alpha a + \beta b = d.$$

Neste momento, algumas perguntas surgem imediatamente. Por que calcular dois inteiros que satisfaçam a igualdade acima? Qual a utilidade destes dois inteiros? Estas perguntas serão respondidas mais adiante neste livro. O que já pode ser dito neste momento é que tanto o cálculo de $\text{mdc}(a, b)$ quanto o cálculo dos inteiros α e β serão extremamente úteis para as nossas aplicações ao longo do livro.

O algoritmo que vamos apresentar é conhecido como Algoritmo Euclidiano Estendido. Ele possui este nome porque estende o Algoritmo Euclidiano para o cálculo do máximo divisor comum com algumas contas a mais para calcular os inteiros α e β.

Já o Algoritmo Euclidiano para o cálculo do máximo divisor comum tem este nome pois foi apresentado por Euclides, um matemático da Grécia Antiga que viveu em torno do ano 300 AC, no livro VII da sua coletânea matemática "Elementos" [6]. Este é um dos mais antigos algoritmos a permanecer ainda em uso prático.

O Algoritmo Euclidiano funciona através de divisões inteiras sucessivas. Se queremos calcular o máximo divisor comum entre os inteiros positivos a e b, começamos realizando a divisão inteira de a por b, obtendo um quociente q_1 e um resto r_1. Se este resto for nulo, então $\text{mdc}(a, b) = b$ e o algoritmo termina. Caso contrário, o divisor da última divisão será usando como dividendo de uma nova divisão e o resto da última divisão será usado como divisor desta nova divisão. Isto é, dividiremos b por r_1, obtendo um quociente q_2 e um resto r_2. Caso o resto r_2 não seja nulo, repetimos o processo descrito acima, dividindo r_1 por r_2 e assim por diante. O algoritmo termina quando obtivermos um resto zero. Teremos então que o máximo divisor comum entre a e b será o último resto diferente de zero nesta sequência de divisões.

O Algoritmo Euclidiano é resumido de forma esquemática nas duas primeiras colunas da Figura 2.1.

$$
\begin{array}{ccc}
a = bq_1 + r_1 & 0 < r_1 < b & \alpha_1 a + \beta_1 b = r_1 \\
b = r_1 q_2 + r_2 & 0 < r_2 < r_1 & \alpha_2 a + \beta_2 b = r_2 \\
r_1 = r_2 q_3 + r_3 & 0 < r_3 < r_2 & \alpha_3 a + \beta_3 b = r_3 \\
\vdots & \vdots & \vdots \\
r_{j-2} = r_{j-1} q_j + r_j & 0 < r_j < r_{j-1} & \alpha_j a + \beta_j b = r_j \\
r_{j-1} = r_j q_{j+1} + r_{j+1} & 0 < r_{j+1} < r_j & \alpha_{j+1} a + \beta_{j+1} b = r_{j+1} \\
r_j = r_{j+1} q_{j+2} + r_{j+2} & 0 < r_{j+2} < r_{j+1} & \alpha_{j+2} a + \beta_{j+2} b = r_{j+2} \\
\vdots & \vdots & \vdots \\
r_{n-3} = r_{n-2} q_{n-1} + r_{n-1} & 0 < r_{n-1} < r_{n-2} & \alpha_{n-1} a + \beta_{n-1} b = r_{n-1} \\
r_{n-2} = r_{n-1} q_n + r_n & r_n = 0 &
\end{array}
$$

Figura 2.1: Forma Esquemática do Algoritmo Euclidiano Estendido

De acordo com os cálculos na Figura 2.1, o resto r_n é igual a zero e os restos anteriores são todos diferentes de zero. Assim, o algoritmo termina após o cálculo de r_n, sem realizar mais divisões. Já que $r_n = 0$ e o Algoritmo Euclidiano nos diz que $\text{mdc}(a, b)$ é igual ao último resto diferente de zero nesta sequência de divisões, então temos que $\text{mdc}(a, b) = r_{n-1}$.

Analogamente ao que fizemos anteriormente para o Algoritmo da Divisão, precisamos mostrar que a sequência de divisões sucessivas do Algoritmo Euclidiano não pode continuar eternamente e também que o último resto diferente de zero nesta sequência de divisões é realmente o máximo divisor comum entre a e b.

Consultado a segunda coluna da Figura 2.1, notamos que os restos produzidos pela sequência de divisões formam uma sequência estritamente decrescente de números inteiros: $b > r_1 > r_2 > r_3 > \ldots$. Como estes números são todos inteiros e distintos e existe uma quantidade finita de inteiros menores do que b e maiores do

que zero, eventualmente terá que aparecer na sequência de divisões um resto igual a zero, fazendo o algoritmo terminar.

Vamos agora mostrar que o último resto diferente de zero na sequência de divisões do Algoritmo Euclidiano é realmente o máximo divisor comum entre a e b. Para isso, precisamos de um lema auxiliar.

Lema 2.1. *Sejam a, b, s e t quatro números inteiros positivos tais que $a = bs + t$. Então, $mdc(a, b) = mdc(b, t)$.*

Demonstração: Sejam $d_1 = \text{mdc}(a, b)$ e $d_2 = \text{mdc}(b, t)$.

Como d_1 é o máximo divisor comum entre a e b, d_1 divide a e d_1 divide b. Logo, $a = d_1a'$, para algum $a' \in \mathbb{Z}$ e $b = d_1b'$, para algum $b' \in \mathbb{Z}$. Substituindo na igualdade do enunciado, obtemos $d_1a' = d_1b's + t$, que pode ser escrito como $d_1(a' - b's) = t$. Esta igualdade implica que d_1 divide t. Então, d_1 é um divisor comum entre b e t. Como d_2 é o máximo divisor comum entre b e t, temos que $d_1 \leq d_2$.

Analogamente, como d_2 é o máximo divisor comum entre b e t, d_2 divide b e d_2 divide t. Logo, $b = d_2b''$, para algum $b'' \in \mathbb{Z}$ e $t = d_2t'$, para algum $t' \in \mathbb{Z}$. Substituindo na igualdade do enunciado, obtemos $a = d_2b''s + d_2t'$, que pode ser escrito como $a = d_2(b''s + t')$. Esta igualdade implica que d_2 divide a. Então, d_2 é um divisor comum entre a e b. Como d_1 é o máximo divisor comum entre a e b, temos que $d_2 \leq d_1$.

De $d_1 \leq d_2$ e $d_2 \leq d_1$, podemos concluir que $d_1 = d_2$. ■

Vamos agora olhar para a sequência de divisões na Figura 2.1 na ordem reversa, começando pela última. Como $r_n = 0$, a última divisão pode ser escrita como $r_{n-2} = r_{n-1}q_n$. Isto significa que r_{n-1} divide r_{n-2}. Como r_{n-1} também divide a si próprio, temos que r_{n-1} é um divisor comum entre r_{n-2} e r_{n-1}. Entretanto, r_{n-1} não pode possuir nenhum divisor maior do que si mesmo. Assim, não pode haver nenhum divisor comum entre r_{n-2} e r_{n-1} maior do que r_{n-1}, o que significa que $\text{mdc}(r_{n-2}, r_{n-1}) = r_{n-1}$.

Vamos observar agora a penúltima divisão: $r_{n-3} = r_{n-2}q_{n-1} + r_{n-1}$. Aplicando o lema acima a esta igualdade, temos que $\text{mdc}(r_{n-3}, r_{n-2}) = \text{mdc}(r_{n-2}, r_{n-1})$. Mas acabamos de mostrar que $\text{mdc}(r_{n-2}, r_{n-1}) = r_{n-1}$, logo $\text{mdc}(r_{n-3}, r_{n-2}) = r_{n-1}$. Se continuarmos aplicando o lema acima a todas as divisões (de baixo para cima), iremos concluir que, em todas elas, como nas duas que já analisamos, o máximo divisor comum entre o dividendo e o divisor é sempre igual a r_{n-1}. Em particular, da primeira divisão $a = bq_1 + r_1$, concluímos que $\text{mdc}(a, b) = r_{n-1}$. Logo, o último resto diferente de zero na sequência de divisões (r_{n-1}) é realmente o máximo divisor comum entre a e b. Podemos concluir então que o Algoritmo Euclidiano, conforme esquematizado nas duas primeiras colunas da Figura 2.1, retorna o resultado correto.

Para obtermos o Algoritmo Euclidiano Estendido, "estendemos" os cálculos realizados pelo Algoritmo Euclidiano com os cálculos descritos na terceira coluna da Figura 2.1. A ideia do Algoritmo Euclidiano Estendido para calcular os inteiros α e β tais que

$$\alpha a + \beta b = \text{mdc}(a, b)$$

é bastante simples e efetiva. Uma vez que mdc(a,b) é um resto produzido na sequência de divisões do Algoritmo Euclidiano, ao invés de tentar calcular diretamente os valores de α e β, o algoritmo calcula valores α_i e β_i tais que

$$\alpha_i a + \beta_i b = r_i,$$

para cada resto r_i, onde $1 \leq i \leq n-1$. A motivação por trás desta ideia é que o valor de α_i possa ser calculado a partir dos valores α_j, com $j < i$, já calculados anteriormente, com um procedimento análogo para o valor de β_i. Assim, os valores de α_i e β_i seriam calculados simultaneamente com os valores de q_i e r_i durante a sequência de divisões. Como $\text{mdc}(a, b) = r_{n-1}$, os valores de α e β que desejamos como resposta do algoritmo serão α_{n-1} e β_{n-1}.

Vamos agora mostrar como calcular os valores de α_{j+2} e β_{j+2} assumindo que já foram calculados anteriormente os valores de α_j, β_j, α_{j+1} e β_{j+1}, tais que

$$\alpha_j a + \beta_j b = r_j \quad \text{e} \quad \alpha_{j+1} a + \beta_{j+1} b = r_{j+1},$$

conforme as igualdades presentes na terceira coluna da Figura 2.1.

Da primeira coluna da Figura 2.1, temos a divisão

$$r_j = r_{j+1} q_{j+2} + r_{j+2},$$

que pode ser escrita como

$$r_{j+2} = r_j - q_{j+2} r_{j+1}. \tag{2.1.1}$$

Queremos calcular α_{j+2} e β_{j+2} satisfazendo a igualdade $\alpha_{j+2} a + \beta_{j+2} b = r_{j+2}$ (novamente, conforme ilustrado na terceira coluna da Figura 2.1).

Substituindo, na igualdade (2.1.1), r_j, r_{j+1} e r_{j+2} pelos valores dados pelas igualdades da terceira coluna da Figura 2.1, obtemos

$$\alpha_{j+2} a + \beta_{j+2} b = (\alpha_j a + \beta_j b) - q_{j+2}(\alpha_{j+1} a + \beta_{j+1} b),$$

que é equivalente a

$$\alpha_{j+2} a + \beta_{j+2} b = (\alpha_j - q_{j+2}\alpha_{j+1}) a + (\beta_j - q_{j+2}\beta_{j+1}) b.$$

Esta última igualdade nos diz que, se tomarmos

$$\begin{cases} \alpha_{j+2} &= \alpha_j - q_{j+2}\alpha_{j+1} \quad \text{e} \\ \beta_{j+2} &= \beta_j - q_{j+2}\beta_{j+1}, \end{cases} \tag{2.1.2}$$

estes valores irão satisfazer a igualdade $\alpha_{j+2} a + \beta_{j+2} b = r_{j+2}$. Assim, à medida que formos produzindo os quocientes e restos das divisões sucessivas, podemos simultaneamente calcular os valores de α_{j+2} e β_{j+2}, bastando para isso armazenar os dois valores anteriores α_j e α_{j+1} e os dois valores anteriores β_j e β_{j+1} de forma a utilizá-los nas fórmulas acima.

Nos resta ainda um problema final para podermos utilizar de fato o Algoritmo Euclidiano Estendido. Sabemos calcular os valores parciais de α e β se conhecermos os dois valores parciais anteriores de cada um. Mas como fazemos para calcular α_1 e β_1? Olhando novamente para o raciocínio acima, vemos que o que nos permitiu obter α_{j+2} e β_{j+2} a partir dos valores anteriores foi a divisão

$$r_j = r_{j+1} q_{j+2} + r_{j+2},$$

isto é, a divisão que produz o resto r_{j+2}. Portanto, para descobrirmos como calcular α_1 e β_1, devemos observar a divisão que produz o resto r_1:

$$a = b q_1 + r_1.$$

Se utilizarmos esta divisão no raciocínio que desenvolvemos acima, conseguiremos calcular α_1 e β_1 a partir de valores $\widetilde{\alpha}$, $\widetilde{\beta}$, $\widehat{\alpha}$ e $\widehat{\beta}$ tais que:

$$\begin{cases} \widetilde{\alpha} a + \widetilde{\beta} b &= a \quad \text{e} \\ \widehat{\alpha} a + \widehat{\beta} b &= b. \end{cases}$$

Mas, dadas estas equações, é imediato determinar que os valores $\widetilde{\alpha} = 1$, $\widetilde{\beta} = 0$, $\widehat{\alpha} = 0$ e $\widehat{\beta} = 1$ vão satisfazê-la. Assim, sempre iniciamos o Algoritmo Euclidiano Estendido com estes quatro valores "pré-calculados" para serem utilizados como valores iniciais na aplicação da fórmulas (2.1.2).

Abaixo, descrevemos o Algoritmo Euclidiano Estendido, levando em consideração tudo o que discutimos sobre ele. A formulação do Algoritmo Euclidiano Estendido neste formato em que o apresentamos é devida a Donald Knuth [12].

Algoritmo 2.2: Algoritmo Euclidiano Estendido

Entrada: Dois números inteiros positivos a e b.

Saída: Um número inteiro positivo d e dois números inteiros α e β tais que $d = \text{mdc}(a, b)$ e $\alpha a + \beta b = d$.

Instruções:

1. $x'' \leftarrow 1$, $y'' \leftarrow 0$, $x' \leftarrow 0$, $y' \leftarrow 1$
2. $q \leftarrow$ Quociente da divisão de a por b
3. $r \leftarrow$ Resto da divisão de a por b
4. Enquanto $r \neq 0$, faça:
   4.1. $x \leftarrow (x'' - q * x')$
   4.2. $y \leftarrow (y'' - q * y')$
   4.3. $x'' \leftarrow x'$, $y'' \leftarrow y'$, $x' \leftarrow x$, $y' \leftarrow y$
   4.4. $a \leftarrow b$, $b \leftarrow r$
   4.5. $q \leftarrow$ Quociente da divisão de a por b
   4.6. $r \leftarrow$ Resto da divisão de a por b
5. $d \leftarrow b$, $\alpha \leftarrow x'$, $\beta \leftarrow y'$
6. Retorne d, α e β.

Exemplo 2.3. *Vamos aplicar o Algoritmo Euclidiano Estendido a* 1768 *e* 62*. Começamos construindo uma tabela com quatro colunas: uma para os restos das divisões sucessivas (R), outra para os quocientes (Q) e as duas últimas para as diversas etapas dos cálculos dos valores de* α *e* β *respectivamente.*

R	Q	α	β

Colocamos então os valores 1768 *e* 62 *nas duas primeiras linhas da tabela, na coluna R. Preenchemos a primeira linha da coluna* α *com o valor* 1*, a segunda linha desta coluna com o valor* 0*, a primeira linha da coluna* β *com o valor* 0 *e a segunda linha desta coluna com o valor* 1*.*

R	Q	α	β
1768	$-$	1	0
62	$-$	0	1

Agora realizamos a divisão de 1768 *por* 62*, os dois elementos da coluna R, obtendo* $1768 = 62.28 + 32$*. Logo, o quociente é* 28 *e o resto é* 32*. Acrescentamos estes valores nas colunas Q e R, respectivamente.*

R	Q	α	β
1768	$-$	1	0
62	$-$	0	1
32	28		

Agora devemos calcular o valor parcial de α e β nesta linha. O valor parcial de α nesta linha será igual ao valor parcial de α duas linhas acima menos o produto do quociente desta linha pelo valor parcial de α na linha acima. Assim, o valor parcial de α nesta linha será $1 - 28.0 = 1$. O cálculo do valor parcial de β nesta linha é inteiramente análogo, sendo $0 - 28.1 = -28$.

R	Q	α	β
1768	–	1	0
62	–	0	1
32	28	1	−28

Realizamos agora a divisão de 62 por 32, os dois últimos elementos na coluna R, obtendo novamente o quociente e o resto e acrescentado estes valores na tabela como anteriormente. Em seguida, calculamos os novos valores parciais de α e β, também como anteriormente. Paramos este processo quando o valor 0 for acrescentado à coluna R da tabela. O último valor diferente de zero na coluna R é o $mdc(1768, 62)$. Os valores de α e β que aparecem na tabela na mesma linha do máximo divisor comum são os valores finais de α e β.

R	Q	α	β
1768	–	1	0
62	–	0	1
32	28	1	−28
30	1	−1	29
2	1	2	−57
0	15	–	–

Temos então que $mdc(1768, 62) = 2$, $\alpha = 2$ e $\beta = -57$. De fato, calculando $2.1768 - 57.62 = 2 = mdc(1768, 62)$.

Um erro muito comum é calcular os valores finais de α e β como se fossem os valores que aparecem na tabela na mesma linha do zero na coluna R. Estes não são os valores corretos. Os valores corretos aparecem uma linha acima, na mesma linha do máximo divisor comum.

Exemplo 2.4. *Vamos fazer mais um exemplo de Aplicação do Algoritmo Euclidiano Estendido. Vamos aplicar o algoritmo a 76 e 2404. Começamos fazendo o preenchimento inicial da tabela da mesma forma que no exemplo anterior.*

R	Q	α	β
76	–	1	0
2404	–	0	1

Repare que, desta vez, o número menor apareceu acima do número maior na coluna R, ao contrário do exemplo anterior. Vamos mostrar que a ordem em que colocamos os números da tabela não afeta a correção do resultado. Dividindo 76 por 2404, obtemos quociente 0 e resto 76.

R	Q	α	β
76	–	1	0
2404	–	0	1
76	0		

Vamos calcular os valores parciais de α e β nesta nova linha da tabela. O valor parcial de α nesta linha será igual ao valor parcial de α duas linhas acima menos

o produto do quociente desta linha pelo valor parcial de α *na linha acima. Assim, o valor parcial de* α *nesta linha será* $1 - 0.0 = 1$*. O cálculo do valor parcial de* β *nesta linha é inteiramente análogo, sendo* $0 - 0.1 = 0$*.*

R	Q	α	β
76	$-$	1	0
2404	$-$	0	1
76	0	1	0

Caso houvéssemos colocado os números 76 *e* 2404 *na coluna* R *na ordem inversa, o início da tabela ficaria da seguinte forma:*

R	Q	α	β
2404	$-$	1	0
76	$-$	0	1

Em ambas as tabelas, o próximo passo seria realizar a divisão de 2404 *por* 76*. Daí em diante, os cálculos nas duas tabelas seriam os mesmos. Vemos então que as únicas diferenças entre as tabelas é que a primeira tabela terá uma linha a mais (a primeira linha) e as colunas* α *e* β *são permutadas entre as duas. Assim, podemos concluir que o algoritmo produz o resultado correto independentemente da ordem com que colocamos os dois números na coluna* R *da tabela. Precisamos apenas ficar atentos com os resultados produzidos nas colunas* α *e* β*. O resultado final na coluna* α *sempre será o multiplicador do primeiro número colocado na coluna* R *e o resultado final produzido na coluna* β *sempre será o multiplicador do segundo número colocado nesta coluna. Se trocarmos a ordem dos números iniciais na coluna* R*, os resultados das colunas* α *e* β *também serão permutados.*

Vamos agora calcular o restante da nossa tabela original, com o valor 76 *no topo.*

R	Q	α	β
76	$-$	1	0
2404	$-$	0	1
76	0	1	0
48	31	-31	1
28	1	32	-1
20	1	-63	2
8	1	95	-3
$\underline{4}$	2	$\underline{-253}$	$\underline{8}$
0	2	$-$	$-$

Temos então que $mdc(76, 2404) = 4$*,* $\alpha = -253$ *e* $\beta = 8$*. De fato, calculando* $-253.76 + 8.2404 = 4 = mdc(76, 2404)$*.*

Precisamos fazer apenas mais uma última observação a respeito dos valores de α e β. Se $\text{mdc}(a, b) = d$, os valores de α e β calculados pelo Algoritmo Euclidiano Estendido não são os únicos que satisfazem a igualdade $\alpha a + \beta b = d$. De fato, se α e β satisfazem esta igualdade, então temos também

$$(\alpha - kb)a + (\beta + ka)b = d, \qquad \text{para todo } k \in \mathbb{Z},$$

o que significa que podemos calcular uma infinidade de pares de valores satisfazendo a igualdade.

Encerramos esta seção com um lema muito importante que será utilizado diversas vezes ao longo deste livro para auxiliar a prova de vários resultados.

Lema 2.2. *Sejam m e n dois números inteiros positivos tais que $mdc(m,n) = 1$ e seja a um número inteiro. Então, as seguintes afirmações são verdadeiras.*

1. *Se n divide am, então n divide a.*

2. *Se m divide a e n divide a, então mn divide a.*

Demonstração: (1) Como $\text{mdc}(m,n) = 1$, podemos utilizar o Algoritmo Euclidiano Estendido para obter uma igualdade na forma $\alpha m + \beta n = 1$. Multiplicando ambos os lados desta igualdade por a, obtemos $\alpha am + \beta an = a$. Como, por hipótese, n divide am, temos $am = nt$, para algum $t \in \mathbb{Z}$. Substituindo am na igualdade anterior, obtemos $\alpha nt + \beta an = a$, que é equivalente a $n(\alpha t + \beta a) = a$. Esta última igualdade implica que n divide a.

(2) Se m divide a, então $a = ma'$, para algum $a' \in \mathbb{Z}$. Como n divide a, então n divide ma'. Como $\text{mdc}(m,n) = 1$ e n divide ma', o item anterior nos diz que então n divide a'. Desta forma, $a' = na''$, para algum $a'' \in \mathbb{Z}$. Mas então $a = ma' = (mn)a''$, o que implica que mn divide a. ■

2.2 Relações de Equivalência

Nesta seção, apresentamos as chamadas *relações de equivalência*. Ao longo de todo o livro, estaremos constantemente lidando com um caso particular de relação de equivalência. Por isso, é importante compreender bem os conceitos básicos a respeito destas relações.

Definição 2.5 (Relação Binária)**.** *Dado um conjunto S, uma* relação binária *entre elementos de S é um conjunto $R \subseteq S \times S$. Para $a, b \in S$, dizemos que a está relacionado com b pela relação binária R se $(a,b) \in R$. Neste caso, também usamos a notação aRb.*

Definição 2.6 (Relação de Equivalência)**.** *Dado um conjunto S e uma relação binária $\sim \subseteq S \times S$, dizemos que a relação $\sim$ é uma* relação de equivalência *em S se ela satisfaz as seguintes propriedades:*

1. *Reflexividade: para todo $a \in S$, $a \sim a$;*

2. *Simetria: para todo $a, b \in S$, se $a \sim b$, então $b \sim a$;*

3. *Transitividade: para todo $a, b, c \in S$, se $a \sim b$ e $b \sim c$, então $a \sim c$.*

Exemplo 2.5. *Dadas duas frações $\frac{a}{b}$ e $\frac{a'}{b'}$, não precisamos ter $a = a'$ e/ou $b = b'$ para que as duas sejam iguais, isto é, para que representem a mesma razão. Frações como $\frac{2}{6}$ e $\frac{1}{3}$ representam a mesma razão, apesar de serem visualmente diferentes. Duas frações $\frac{a}{b}$ e $\frac{a'}{b'}$ são iguais se e somente se $ab' = a'b$. Vamos mostrar que a igualdade de frações é uma relação de equivalência:*

Reflexividade: *Como $ab = ab$, temos que $\frac{a}{b} = \frac{a}{b}$.*

Simetria: *Se $\frac{a}{b} = \frac{a'}{b'}$, então $ab' = a'b$. Mas a igualdade de inteiros é simétrica, logo temos $a'b = ab'$, o que significa que $\frac{a'}{b'} = \frac{a}{b}$.*

Transitividade: *Se* $\frac{a}{b} = \frac{a'}{b'}$ *e* $\frac{a'}{b'} = \frac{a''}{b''}$, *então* $ab' = a'b$ *e* $a'b'' = a''b'$. *Multiplicando os dois lados da primeira igualdade por* b'', *obtemos* $ab'b'' = a'bb''$, *que pode ser escrita como* $ab'b'' = a'b''b$. *Pela segunda igualdade, podemos substituir* $a'b''$ *por* $a''b'$, *obtendo* $ab'b'' = a''b'b$. *O denominador de uma fração sempre é diferente de zero, logo* $b' \neq 0$. *Podemos então cancelá-lo dos dois lados da igualdade, obtendo* $ab'' = a''b$, *o que nos permite concluir que* $\frac{a}{b} = \frac{a''}{b''}$.

Exemplo 2.6. *Se considerarmos uma caixa* C *com bolas coloridas e bolas* $a, b \in C$, *a relação* $a \sim b$ *se e somente se* a *e* b *tem a mesma cor é uma relação de equivalência.*

Informalmente, uma relação de equivalência é uma relação que considera dois objetos como "iguais" se eles possuem uma determinada característica fixada em comum, como a razão que representam, no primeiro exemplo, ou a sua cor, no segundo.

Vamos ver agora como uma relação de equivalência em um conjunto X define subconjuntos particulares de X.

Definição 2.7. *Seja* X *um conjunto,* $\sim$ *uma relação de equivalência em* X *e* $a \in X$. *A* classe de equivalência *de* a, *denotada por* $\overline{a}$, *é definida como*

$$\overline{a} = \{b \in X : a \sim b\}.$$

Exemplo 2.7. *De volta ao exemplo das bolas coloridas, as classes de equivalência podem ser pensadas como urnas rotuladas com o nome de uma cor, onde apenas as bolas com a cor igual à do rótulo podem ser colocadas.*

Uma observação importante é que $\overline{a}$ não é necessariamente a única maneira possível de descrever a classe de equivalência a que a pertence. Se existe outro elemento $u \neq a$ tal que $u \in \overline{a}$, podemos descrever a classe de equivalência tanto como $\overline{a}$ quanto como $\overline{u}$, isto é, as duas notações representam a mesma classe de equivalência. Em outras palavras, qualquer um dos elementos de uma classe de equivalência pode ser selecionado como "representante" da classe. Vamos mostrar este resultado com cuidado.

Propriedade 2.1. *Seja* X *um conjunto e* $\overline{a}$ *uma classe de equivalência em* X. *Se* $u \in X$ *e* $u \in \overline{a}$, *então* $\overline{u} = \overline{a}$.

Demonstração: Se $u \in \overline{a}$, então, pela definição de $\overline{a}$, temos que $a \sim u$. Para mostrar que $\overline{u} = \overline{a}$, precisamos mostrar que, se $c \in X$, então $c \in \overline{u}$ se e somente se $c \in \overline{a}$.

Suponha que $c \in \overline{u}$. Então, pela definição de $\overline{u}$, temos que $u \sim c$. Como $\sim$ é uma relação de equivalência, por transitividade, como $a \sim u$ e $u \sim c$, então $a \sim c$. Isto, pela definição de $\overline{a}$, significa que $c \in \overline{a}$.

Suponha agora que $c \in \overline{a}$. Então, temos que $a \sim c$. Como $\sim$ é uma relação de equivalência, por simetria, como temos $a \sim u$, temos também $u \sim a$. Agora, por transitividade, como $u \sim a$ e $a \sim c$, então $u \sim c$. Isto, pela definição de $\overline{u}$, significa que $c \in \overline{u}$. ■

É simples ver que X pode ser obtido como a união de todas as suas classes de equivalência. Entretanto, podemos obter um resultado mais forte do que este: as classes de equivalência formam uma *partição* do conjunto X, isto é, a união das classes de equivalência é *disjunta*.

Propriedade 2.2. *Suponha que* $u \in \overline{a}$ *e* $u \in \overline{b}$. *Então,* $\overline{a} = \overline{b}$.

Demonstração: Se $u \in \overline{a}$, então $a \sim u$. Se $u \in \overline{b}$, então $b \sim u$ e, por simetria, $u \sim b$. Por transitividade, como $a \sim u$ e $u \sim b$, então $a \sim b$. Isto significa que $b \in \overline{a}$. Pela Propriedade 2.1, se $b \in \overline{a}$, então $\overline{b} = \overline{a}$. ■

Uma vez que as classes de equivalência nos fornecem uma partição do conjunto X e que todos os elementos dentro de uma mesma classe são considerados, com o perdão da redundância, "equivalentes", podemos construir um novo conjunto tomando um representante de cada uma das classes de equivalência.

Definição 2.8. *Seja X um conjunto e $\sim$ uma relação de equivalência em X. Definimos o* conjunto quociente *de X por $\sim$, denotado por $X/_\sim$, da seguinte forma:*

$$X/_\sim = \{\overline{a} : a \in X\}.$$

Todas as classes de equivalência estão presentes no conjunto acima. Por outro lado, como elementos nunca aparecem repetidos dentro de um conjunto, cada classe de equivalência no conjunto acima será representada por um único elemento dela. Qual representante escolhemos para cada classe no conjunto acima não importa, desde que todas as classes estejam representadas.

Exemplo 2.8. *Retornando ao exemplo das frações tendo estudado estes resultados básicos sobre relações de equivalência, vemos que uma fração não é simplesmente um par de inteiros (a, b) onde $b \neq 0$, como às vezes ouvimos na escola. Como diversos pares diferentes representam a mesma fração, uma fração na realidade é um elemento do conjunto quociente $X/_\sim$, onde X é o conjunto de pares de inteiros (a, b) onde $b \neq 0$ e $\sim$ é a relação de igualdade de frações que estudamos no exemplo acima. Em outras palavras, uma fração é uma classe de equivalência. Qual dos elementos dessa classe nós escolhemos para representar a fração não importa. O resultado das contas com frações não depende desta escolha. Entretanto, a escolha mais comum é pela chamada fração reduzida, isto é, a fração em que o máximo divisor comum entre o numerador a e o denominador b é igual a 1.*

2.3 Inteiros Módulo n

Nesta seção, apresentamos a relação de *congruência módulo n*, onde $n \geq 2$ é um inteiro, e a utilizamos para construir o conjunto dos *inteiros módulo n*, denotado por $\mathbb{Z}_n$.

Vamos iniciar a apresentação com um exemplo simples que motiva este estudo.

Exemplo 2.9. *Se agora são 4 horas da madrugada e perguntarmos a alguém que horas serão daqui a 15 horas, a pessoa poderá responder "19 horas" ou "7 horas da noite". Por outro lado, se agora são 23 horas (11 horas da noite) e fizermos a mesma pergunta, a pessoa poderá responder "14 horas" ou "2 horas da tarde", mas será muito improvável que ela responda "38 horas", a soma direta de 23 com 15. Vemos então que a aritmética das horas do dia não é exatamente igual à aritmética básica da escola. Na aritmética das horas do dia, $23 + 15 = 14$ e $23 + 15 = 2$ são resultados considerados corretos, apesar de serem totalmente estranhos do ponto de vista da aritmética tradicional. Isto se deve ao fato de que as horas do dia começam a se repetir após um intervalo de 24 horas ou mesmo após um intervalo de 12 horas, se considerarmos a maneira como um relógio analógico tradicional exibe as horas. Assim, para modelar corretamente os cálculos com horas do dia, o modelo ideal não*

é o da reta infinita dos números inteiros, mas sim o de um relógio circular fechado com 12 ou 24 casas. A relação de congruência módulo n e a aritmética modular *derivada dela são uma generalização desta ideia: ao invés dos cálculos serem feitos sobre a reta infinita dos inteiros, eles são feitos sobre um "relógio circular fechado" com n casas.*

Após esta motivação inicial, estamos prontos para definir a relação de congruência módulo n.

Definição 2.9. *Seja $n \geq 2$ um inteiro. Dados dois inteiros a e b, dizemos que a é congruente a b módulo n, denotado por $a \equiv b \pmod{n}$, se $a - b$ é múltiplo de n. O número n é chamado de* módulo *da congruência.*

Repare que não temos uma única relação de congruência. Cada inteiro $n \geq 2$ dá origem a uma relação de congruência distinta.

Exemplo 2.10. *Temos que $36 \equiv 15 \pmod{7}$ já que $36 - 15 = 21$ e 21 é múltiplo de 7. Entretanto, $36 \not\equiv 15 \pmod{11}$, já que $36 - 15$ não é múltiplo de 11. Vemos então que ao alterar o módulo, alteramos os pares de números que são congruentes entre si. Concluímos então que não faz sentido falar simplesmente de "congruência", sem especificar qual é o módulo que estamos utilizando.*

Exemplo 2.11. *No caso das horas do dia, estamos trabalhando com a relação de congruência módulo 24 ou com a relação de congruência módulo 12. Utilizando a notação de congruências, temos:*

1. *$38 \equiv 14 \pmod{24}$, já que $38 - 14 = 24$, que é múltiplo de si próprio, e*

2. *$38 \equiv 2 \pmod{12}$, já que $38 - 2 = 36$, que é múltiplo de 12.*

As congruências acima justificam as respostas "14 horas" e "2 horas" apresentadas no exemplo inicial.

Vamos mostrar agora que a relação de congruência módulo n é uma relação de equivalência.

Teorema 2.2. *Seja $n \geq 2$ um inteiro. A relação de congruência módulo n é uma relação de equivalência.*

Demonstração: Vamos mostrar que a relação de congruência módulo n satisfaz as três propriedades de uma relação de equivalência: reflexividade, simetria e transitividade.

Reflexividade: Seja a um inteiro. Temos que 0 é múltiplo de n, o que significa que $a - a$ é múltiplo de n. Pela definição de congruência módulo n, temos então que $a \equiv a \pmod{n}$.

Simetria: Sejam a e b inteiros tais que $a \equiv b \pmod{n}$. Então, pela definição de congruência módulo n, temos que $a - b$ é múltiplo de n. Podemos escrever então $a - b = nk$, para algum $k \in \mathbb{Z}$. Multiplicando esta última igualdade por -1 em ambos os lados, obtemos $b - a = n(-k)$, o que significa que $b - a$ também é múltiplo de n. Logo, pela definição de congruência módulo n, $b \equiv a \pmod{n}$.

Transitividade: Sejam a, b e c inteiros tais que $a \equiv b \pmod{n}$ e $b \equiv c \pmod{n}$. Então, pela definição de congruência módulo n, temos que $a - b$ e $b - c$ são múltiplos de n. Temos então $a - b = nk$, para algum $k \in \mathbb{Z}$, e $b - c = nk'$, para algum $k' \in \mathbb{Z}$. Se somarmos estas igualdades, obtemos $a - c = n(k + k')$, o que significa que $a - c$ também é múltiplo de n. Logo, pela definição de congruência módulo n, $a \equiv c \pmod{n}$. ■

Já que a relação de congruência módulo n é uma relação de equivalência, podemos pensar nas classes de equivalência definidas por esta relação.

Definição 2.10. *Sejam $n \geq 2$ e a inteiros. A classe de equivalência de a pela relação de congruência módulo n, denotada por $\overline{a}$, é definida como*

$$\overline{a} = \{b \in \mathbb{Z} : a \equiv b \pmod{n}\}.$$

Conforme os resultados que estudamos na seção anterior, estas classes de equivalência formam uma partição do conjunto $\mathbb{Z}$ dos números inteiros. Isto significa que todo número inteiro pertence a uma e apenas uma destas classes de equivalência. Precisamos agora determinar quais são e quantas são estas classes.

Propriedade 2.3. *Sejam r e r' dois inteiros distintos. Se $0 < r - r' < n$, então $r \not\equiv r' \pmod{n}$.*

Demonstração: Para que $r \equiv r' \pmod{n}$, precisaríamos que $r - r'$ fosse múltiplo de n. Isto significaria que $r - r' = nk$, para algum $k \in \mathbb{Z}$. Como temos que $0 < r - r' < n$, teríamos então $0 < nk < n$. Dado que $n \neq 0$, isto é equivalente a $0 < k < 1$. Mas k é inteiro e não existe nenhum inteiro maior do que 0 e menor do que 1. Logo, $r - r'$ não pode ser múltiplo de n, nos permitindo concluir que $r \not\equiv r' \pmod{n}$. ■

Propriedade 2.4. *Seja $n \geq 2$ e a inteiros. Então $a \equiv r \pmod{n}$, onde r é o resto da divisão de a por n.*

Demonstração: Dividindo a por n, obtemos $a = nq + r$, onde $0 \leq r < n$. Podemos escrever esta igualdade como $a - r = nq$, o que significa que $a - r$ é múltiplo de n. Logo, pela definição de congruência módulo n, temos que $a \equiv r \pmod{n}$. ■

A partir da Propriedade 2.4, vemos que todo inteiro será congruente módulo n a um dos inteiros no conjunto $\{0, 1, 2, \ldots, n-1\}$, já que estes são os possíveis restos de uma divisão em que n é o divisor. Desta maneira, não existem outras classes de equivalência além das classes $\overline{0}, \overline{1}, \overline{2}, \ldots, \overline{n-1}$. Por outro lado, dados quaisquer dois valores distintos do conjunto acima, a Propriedade 2.3 nos diz que eles não serão congruentes entre si módulo n. Desta maneira, as classes de equivalência que listamos são todas distintas. Conhecemos então todas as classes de equivalência de $\mathbb{Z}$ pela relação de congruência módulo n e sabemos que elas totalizam n classes distintas. Podemos então reuní-las no conjunto quociente de $\mathbb{Z}$ pela relação de congruência módulo n.

Definição 2.11. *Dado o conjunto $\mathbb{Z}$ dos números inteiros e a relação de congruência módulo n, que nesta definição abreviaremos como $\equiv_n$, o conjunto quociente de $\mathbb{Z}$ por $\equiv_n$, denotado por $\mathbb{Z}/_{\equiv_n}$, é definido da seguinte forma:*

$$\mathbb{Z}/_{\equiv_n} = \{\overline{0}, \overline{1}, \overline{2}, \ldots, \overline{n-1}\}.$$

No caso específico da relação de congruência módulo n, a nomenclatura "conjunto quociente" e a notação $\mathbb{Z}/_{\equiv_n}$ acabam não sendo muito utilizadas. O conjunto acima é normalmente denotado por $\mathbb{Z}_n$ e chamado de *conjunto dos inteiros módulo* n. Temos então

$$\mathbb{Z}_n = \{\overline{0}, \overline{1}, \overline{2}, \ldots, \overline{n-1}\}.$$

A nomenclatura e a notação $\mathbb{Z}_n$ não deixam tão explícito, mas é importante sempre lembrar que os elementos de $\mathbb{Z}_n$ não são números inteiros, mas sim *classes de equivalência* dos números inteiros de acordo com a congruência módulo n. Escolhemos como representantes das classes os números inteiros no intervalo $0 \leq i < n$, mas todos os outros inteiros também estão incluídos em alguma das classes acima. Por exemplo, $\overline{n}$ não aparece explicitamente no conjunto acima, mas $\overline{n} = \overline{0}$, isto é, a classe de equivalência de n é a mesma classe de equivalência de 0.

Uma maneira intuitiva de pensar nestas classes de equivalência é considerar um relógio redondo com n casas, em que cada casa é rotulada, em sentido horário, pelos números $0, 1, \ldots, n-1$, sendo que a casa com 0 fica no topo do relógio. Pegamos então a reta infinita dos inteiros, sobrepomos o 0 da reta com o 0 do relógio e então "enrolamos" a reta dos inteiros em volta do relógio, no sentido horário a semi-reta dos inteiros positivos e no sentido anti-horário a semi-reta dos inteiros negativos. Após este processo, todos os inteiros que ficarem situados sobre a mesma casa do relógio estão na mesma classe de equivalência módulo n: todos que ficarem situados sobre a casa rotulada com 0 estão em $\overline{0}$, todos que ficarem situados sobre a casa rotulada com 1 estão em $\overline{1}$, e assim por diante até $\overline{n-1}$.

Todo número inteiro a é congruente módulo n a algum número no intervalo $0 \leq i < n$, o número que corresponde ao resto da divisão de a por n, como vimos na Propriedade 2.4. Este valor único no intervalo $0 \leq i < n$ que é congruente módulo n ao inteiro a é chamado de *forma reduzida de a módulo n* e denotado por $a \bmod n$.

Exemplo 2.12. *A forma reduzida de* 19 *módulo* 5 *é* 4*, já que* 4 *é o resto da divisão de* 19 *por* 5*. Na nossa notação, escrevemos* $19 \bmod 5 = 4$.

2.4 Operações Modulares

Nesta seção, mostramos como realizar as operações aritméticas de soma, subtração, produto e potenciação em $\mathbb{Z}_n$. Mostramos também como determinar se um elemento de $\mathbb{Z}_n$ possui ou não inverso multiplicativo e como calcular este inverso se ele existir.

Vamos iniciar este estudo da aritmética do conjunto $\mathbb{Z}_n$ pela operação da soma. Neste caso, é simples usarmos a analogia do relógio de n casas que vimos no final da última seção para entender o que a soma de dois elementos de $\mathbb{Z}_n$ deve representar.

Suponha que queremos calcular $\overline{a} + \overline{b}$, onde $\overline{a}, \overline{b} \in \mathbb{Z}_n$. Para fazer esta operação de soma com o nosso relógio de n casas, fazemos os seguintes passos. Primeiro, colocamos o ponteiro do relógio na casa correspondente a $\overline{a}$, isto é, na casa sobre a qual o inteiro a fica quando a reta dos inteiros é enrolada em volta do relógio. Em seguida, movemos o ponteiro b casas adiante, no sentido horário. A casa em que o ponteiro parar corresponde ao resultado da soma $\overline{a} + \overline{b}$.

Na nossa notação matemática, o procedimento acima pode ser descrito da seguinte forma:

$$\overline{a} + \overline{b} = \overline{a + b}.$$

Isto significa que o resultado da soma de duas classes de equivalência é igual à classe de equivalência da soma de seus representantes. Para que esta definição realmente

faça sentido, precisamos nos certificar de que o resultado da soma será sempre o mesmo, independentemente de quais representantes sejam escolhidos para as duas classes que estão sendo somadas. Formalizamos isto no teorema abaixo.

Teorema 2.3. *Se $\overline{a} = \overline{a'}$ e $\overline{b} = \overline{b'}$, então $\overline{a+b} = \overline{a'+b'}$. Em outras palavras, o resultado da soma é independente do representante que escolhemos para a primeira classe (a ou a') e a segunda classe (b ou b') sendo somadas.*

Demonstração: Como estamos lidando com classes de equivalência em $\mathbb{Z}_n$, se $\overline{a} = \overline{a'}$, então podemos concluir que $a \equiv a' \pmod{n}$. Analogamente, se $\overline{b} = \overline{b'}$, então $b \equiv b' \pmod{n}$. Destas duas congruências, concluímos que $a - a'$ e $b - b'$ são múltiplos de n. Logo, $a - a' = nk$, para algum $k \in \mathbb{Z}$, e $b - b' = nl$, para algum $l \in \mathbb{Z}$. Somando estas duas últimas igualdades, obtemos $(a - a') + (b - b') = n(k + l)$, que pode ser escrita como

$$(a + b) - (a' + b') = n(k + l).$$

Temos então que $(a + b) - (a' + b')$ é um múltiplo de n. Logo, $a + b \equiv a' + b' \pmod{n}$, o que significa que $\overline{a+b} = \overline{a'+b'}$ ■

Vemos então que a operação modular de soma é definida de maneira bem simples, utilizando para o seu cálculo a soma tradicional de inteiros.

Exemplo 2.13. *Vamos calcular a soma de $\overline{13}$ e $\overline{17}$ em $\mathbb{Z}_{25}$. Pela fórmula acima, $\overline{13} + \overline{17} = \overline{13 + 17} = \overline{30}$. Podemos substituir 30 por sua forma reduzida, que será o resto da divisão de 30 por 25, ou seja, 5. Então, $\overline{13} + \overline{17} = \overline{5}$ em $\mathbb{Z}_{25}$.*

No caso da subtração, o procedimento é inteiramente análogo. Novamente utilizando a nossa analogia do relógio de n casas, a operação de subtração é muito parecida com a da soma, com a única diferença sendo que, na última etapa, ao invés de mover o ponteiro b casas para frente, no sentido horário, movemos o ponteiro b casas para trás, no sentido anti-horário. Na notação matemática, o que temos é

$$\overline{a} - \overline{b} = \overline{a - b}.$$

Novamente, precisamos nos certificar de que o resultado da subtração será sempre o mesmo, independentemente de quais representantes sejam escolhidos para as duas classes que estão sendo subtraídas. Entretanto, esta verificação é inteiramente análoga ao que fizemos no teorema acima para o caso da soma. Deixamos então esta verificação como exercício (Exercício 5).

Exemplo 2.14. *Vamos calcular $\overline{9} - \overline{13}$ em $\mathbb{Z}_7$. Pela fórmula acima, $\overline{9} - \overline{13} = \overline{9 - 13} = \overline{-4}$. Calcular a forma reduzida de um número negativo não é tão imediato quanto de um número positivo. Entretanto, podemos sempre lembrar que, para qualquer inteiro a, temos $n + a \equiv a \pmod{n}$, como se pode verificar diretamente da definição de congruência módulo n. Assim, para um número negativo, podemos somar n a ele quantas vezes for necessário até obtermos um valor no intervalo $0 \leq i < n$. Este valor será a forma reduzida do número original. No caso do número -4, temos $-4 + 7 = 3$. Assim, $\overline{9} - \overline{13} = \overline{3}$ em $\mathbb{Z}_7$.*

Vamos agora analisar o caso da operação de produto. Na aritmética tradicional dos inteiros, o produto $a.b$ significa somar a com si mesmo b vezes. Para o produto de dois elementos de $\mathbb{Z}_n$, vamos utilizar esta mesma ideia, realizando esta soma repetida no nosso relógio de n casas. Começamos com o ponteiro na casa do 0 e então fazemos b vezes seguidas o movimento de avançar o ponteiro a casas para

frente, no sentido horário. A casa em que o ponteiro parar corresponde ao resultado do produto $\overline{a}.\overline{b}$.

Na notação matemática, este procedimento pode então ser descrito da seguinte forma:

$$\overline{a}.\overline{b} = \overline{a.b},$$

que é novamente muito semelhante às expressões que obtivemos para as operações anteriores. Mais uma vez, precisamos nos certificar de que o resultado do produto será sempre o mesmo, independentemente de quais representantes sejam escolhidos para as duas classes que estão sendo multiplicadas. É isto que fazemos no teorema abaixo.

Teorema 2.4. *Se $\overline{a} = \overline{a'}$ e $\overline{b} = \overline{b'}$, então $\overline{a.b} = \overline{a'.b'}$. Em outras palavras, o resultado do produto é independente do representante que escolhemos para a primeira classe (a ou a') e a segunda classe (b ou b') sendo multiplicadas.*

Demonstração: Se $\overline{a} = \overline{a'}$, então podemos concluir que $a \equiv a' \pmod{n}$. Analogamente, se $\overline{b} = \overline{b'}$, então $b \equiv b' \pmod{n}$. Destas duas congruências, concluímos que $a - a'$ e $b - b'$ são múltiplos de n. Logo, $a - a' = nk$, para algum $k \in \mathbb{Z}$, e $b - b' = nl$, para algum $l \in \mathbb{Z}$. Estas igualdades podem ser escritas como $a = a' + nk$ e $b = b' + nl$. Multiplicando estas duas últimas igualdades, obtemos

$$ab = (a' + nk)(b' + nl) = a'b' + n(a'l + b'k + nkl).$$

Temos então que $ab - a'b'$ é um múltiplo de n. Logo, $ab \equiv a'b' \pmod{n}$, o que significa que $\overline{a.b} = \overline{a'.b'}$. ∎

Exemplo 2.15. *Vamos calcular $\overline{4}.\overline{5}$ em $\mathbb{Z}_3$. Pela fórmula acima, $\overline{4}.\overline{5} = \overline{4.5} = \overline{20}$. A forma reduzida de 20 módulo 3 é 2, logo $\overline{4}.\overline{5} = \overline{2}$ em $\mathbb{Z}_3$.*

Podemos notar, a partir das definições acima, que as operações de soma e produto satisfazem diversas propriedades da soma e produto tradicionais de inteiros. A soma modular satisfaz as propriedades de associatividade, comutatividade, existência de elemento neutro (o elemento $\overline{0}$) e existência de inverso aditivo (o inverso aditivo de $\overline{a}$ é $\overline{-a} = \overline{n - a}$). Já o produto modular satisfaz as propriedades de associatividade, comutatividade e existência de elemento neutro (o elemento $\overline{1}$). Veremos mais adiante que só alguns elementos de $\mathbb{Z}_n$ possuem inverso multiplicativo. Além destas propriedades, a soma e o produto modulares também satisfazem a propriedade de distributividade, isto é, $\overline{a}.(\overline{b} + \overline{c}) = \overline{a}.\overline{b} + \overline{a}.\overline{c}$.

Entretanto, existe uma propriedade do produto tradicional de inteiros que o produto modular nem sempre satisfaz. Se a e b são inteiros e $a.b = 0$, então $a = 0$ ou $b = 0$. No produto modular, esta propriedade pode ser falsa. Como exemplo, em $\mathbb{Z}_6$, temos $\overline{2}.\overline{3} = \overline{2.3} = \overline{6} = \overline{0}$. Entretanto, tanto $\overline{2} \neq \overline{0}$ quanto $\overline{3} \neq \overline{0}$.

Trataremos agora da operação de potenciação modular. Assim como no caso da potenciação tradicional, a potenciação $\overline{a}^t$, onde $t \geq 0$, significa multiplicar $\overline{a}$ por si mesmo t vezes. Portanto, a maneira mais simples de realizar este cálculo seria realizar as operações de multiplicação em sequência, possivelmente calculando a forma reduzida de cada resultado parcial antes de prosseguir com a próxima multiplicação. Entretanto, este método é extremamente ineficiente quando o expoente k é muito grande. Vamos apresentar a seguir um algoritmo bem mais eficiente para a operação de potenciação modular.

Começamos escrevendo o expoente t da seguinte forma:

$$t = t_0 + t_1.2 + t_2.2^2 + \ldots + t_{k-1}.2^{k-1} + t_k.2^k,$$

onde $t_i \in \{0, 1\}$, para todo $0 \leq i < k$ e $t_k = 1$. Dito de outra forma, t_0, t_1, ..., t_k são os algarismos da representação de t na base 2, ordenados do menos significativo para o mais significativo.

Tendo esta representação de t, podemos escrever a potenciação como

$$\overline{a}^t = \overline{a}^{t_0 + t_1.2 + t_2.2^2 + \ldots + t_{k-1}.2^{k-1} + t_k.2^k} =$$

$$= (\overline{a})^{t_0}.(\overline{a}^2)^{t_1}.(\overline{a}^{2^2})^{t_2}.\ldots.(\overline{a}^{2^{k-1}})^{t_{k-1}}.(\overline{a}^{2^k})^{t_k}.$$

Para calcular a potência $\overline{a}^t$, vamos então calcular o produto das potências acima, calculando as potências da esquerda para direita. Como os expoentes t_i, $0 \leq i \leq k$ são iguais a 0 ou a 1, teremos $(\overline{a}^{2^i})^{t_i} = 1$, se $t_i = 0$, ou $(\overline{a}^{2^i})^{t_i} = \overline{a}^{2^i}$, se $t_i = 1$. Ou seja, para cada potência, ou o resultado é 1 ou é a própria base. Uma outra observação é que cada uma das bases desta sequência de potências acima é igual ao quadrado da base imediatamente à sua esquerda na sequência. Por fim, temos uma maneira simples de calcular a sequência de expoentes t_0, t_1, ..., t_k. Basta dividirmos repetidamente k por 2 e tomar os restos destas divisões. A sequência de restos será igual a sequência t_0, t_1, ..., t_k.

Vamos reunir todas estas ideias no nosso algoritmo. Criamos uma variável para armazenar o resultado do produto das potências acima. Ela começa com o valor 1 e a cada nova potência calculada, multiplicamos o valor da variável pelo resultado desta nova potência, calculando sempre a forma reduzida módulo n desta multiplicação. Ao final da sequência de potências, esta variável conterá a forma reduzida de $\overline{a}^t$ módulo n.

Para calcular cada uma das potências da sequência, criamos uma variável para armazenar a base das potências. O valor inicial desta variável é a, a base da primeira potência. A cada nova potência que formos calcular, elevamos ao quadrado o valor desta variável e calculamos sua forma reduzida módulo n. Finalmente, para calcular a sequência dos expoentes t_0, t_1, ..., t_k, armazenamos o expoente t em uma variável e, a cada potência que formos calcular, armazenamos o quociente da divisão do valor desta variável por 2 de volta nesta variável e utilizamos o resto desta divisão como o próximo elemento da sequência t_0, t_1, ..., t_k.

O algoritmo que implementa estas ideias é apresentado abaixo.

Algoritmo 2.3: Potenciação Modular

Entrada: Dois números inteiros não-negativos a e t e um número inteiro $n \geq 2$.

Saída: Forma reduzida de $a^t \bmod n$.

Instruções:

1. $R \leftarrow 1$, $A \leftarrow a$, $E \leftarrow t$
2. Enquanto $E \neq 0$, faça:
   - 2.1. Se E for ímpar, então:
     - 2.1.1. $R \leftarrow (R * A) \bmod n$
     - 2.1.2. $E \leftarrow (E - 1)/2$
   - 2.2. Senão, $E \leftarrow E/2$

2.3. $A \leftarrow (A * A) \bmod n$

3. Retorne R.

Exemplo 2.16. *Vamos calcular a forma reduzida de* $\overline{6}^{19457}$ *em* $\mathbb{Z}_{33}$*. Em outras palavras, vamos calcular o resto da divisão de* 6^{19457} *por* 33*. Começamos construindo uma tabela com uma coluna para cada variável do algoritmo e uma quarta coluna para avaliar quais comandos condicionais serão executados em cada etapa.*

R	A	E	E *ímpar?*
1	6	19457	*Sim*

Como em qualquer etapa do algoritmo, devemos substituir 19457 *pelo quociente da sua divisão por* 2 *na variável* E *e substituir o valor da variável* A *pela forma reduzida módulo* 33 *do quadrado deste valor. Entretanto, como* 19457 *é ímpar, devemos também multiplicar o valor atual de* R *pelo valor atual de* A *e calcular a forma reduzida deste produto. O quociente de* 19457 *por* 2 *é* 9728*.* $6^2 = 36$ *e* $36 \equiv 3 \pmod{33}$*. Finalmente,* 1 *multiplicado por* 6 *é* 6*, que já está reduzido módulo* 33*. A tabela fica então*

R	A	E	E *ímpar?*
1	6	19457	*Sim*
6	3	9728	*Não*

Agora, 9728 *não é ímpar. Então não vamos alterar o valor de* R*, apenas o de* A *e* E*. Construímos o restante da tabela de acordo com esta metodologia:*

R	A	E	E *ímpar?*
1	6	19457	*Sim*
6	3	9728	*Não*
6	9	4864	*Não*
6	15	2432	*Não*
6	27	1216	*Não*
6	3	608	*Não*
6	9	304	*Não*
6	15	152	*Não*
6	27	76	*Não*
6	3	38	*Não*
6	9	19	*Sim*

Neste momento, como 19 *é ímpar, além de atualizar os valores de* A *e* E*, também precisamos atualizar o valor de* R*. O valor de* R *será a forma reduzida do produto de* 6 *(valor atual de* R*) por* 9 *(valor atual de* A*).* $54 \equiv 21 \pmod{33}$*, logo o novo valor de* R *será* 21*. O restante da tabela será então:*

R	A	E	E *ímpar?*
21	15	9	*Sim*
18	27	4	*Não*
18	3	2	*Não*
18	9	1	*Sim*
<u>30</u>	15	0	*Não*

Assim, temos que $6^{19457} \equiv 30 \pmod{33}$*.*

Vamos agora encerrar esta seção discutindo como determinar se um dado elemento de $\mathbb{Z}_n$ possui ou não inverso multiplicativo e também como calcular este inverso nos casos em que ele existe. Em primeiro lugar, vamos definir formalmente o que é o inverso multiplicativo de um elemento de $\mathbb{Z}_n$.

Definição 2.12. *Seja $\overline{a} \in \mathbb{Z}_n$. Dizemos que $\overline{b}$ é o* inverso multiplicativo *de $\overline{a}$ em $\mathbb{Z}_n$ se $\overline{a}.\overline{b} = \overline{1}$ em $\mathbb{Z}_n$, ou, dito de outra forma, se $ab \equiv 1 \pmod{n}$.*

O teorema abaixo nos dá uma caracterização de quais elementos de $\mathbb{Z}_n$ possuem inverso multiplicativo. Ele também nos fornece uma outra caracterização, que nos será útil mais adiante, a respeito de quais elementos de $\mathbb{Z}_n$ possuem uma potência congruente a 1 módulo n. Tanto no caso do inverso multiplicativo quanto no caso das potências, a resposta é a mesma: os elementos $\overline{a} \in \mathbb{Z}_n$ tais que $\text{mdc}(a, n) = 1$.

Teorema 2.5 (Teorema da Inversão Modular)**.** *Sejam a e $n \geq 2$ números inteiros. As seguinte três afirmativas são equivalentes entre si:*

1. *$\overline{a}$ possui inverso multiplicativo em $\mathbb{Z}_n$.*
2. *$mdc(a, n) = 1$.*
3. *Existe um inteiro positivo k tal que $a^k \equiv 1 \pmod{n}$.*

Demonstração: $(1 \Rightarrow 2)$ Suponha que $\overline{a}$ possui inverso multiplicativo em $\mathbb{Z}_n$. Então, existe um $\overline{\alpha} \in \mathbb{Z}_n$ tal que $\alpha a \equiv 1 \pmod{n}$. Isto é equivalente a dizer que $\alpha a - 1$ é múltiplo de n, isto é, $\alpha a - 1 = nt$, para algum $t \in \mathbb{Z}$. Seja $d = \text{mdc}(a, n)$. Então, d divide a e d divide n, isto é $a = da'$, para algum $a' \in \mathbb{Z}$ e $n = dn'$, para algum $n' \in \mathbb{Z}$. Podemos reescrever a igualdade $\alpha a - 1 = nt$ como $d\alpha a' - 1 = dn't$, que é equivalente a $d(\alpha a' - n't) = 1$. Esta última igualdade implica que d divide 1, ou seja, $d = 1$.

$(2 \Rightarrow 1)$ Suponha que $\text{mdc}(a, n) = 1$. Utilizando o Algoritmo Euclidiano Estendido, obtemos a igualdade $\alpha a + \beta n = 1$, que pode ser escrita como $\alpha a - 1 = (-\beta)n$. Esta igualdade é equivalente a dizer que $\alpha a \equiv 1 \pmod{n}$, logo $\overline{\alpha}$ é o inverso multiplicativo de $\overline{a}$ em $\mathbb{Z}_n$.

$(1 \Rightarrow 3)$ Suponha que $\overline{a}$ possui inverso multiplicativo em $\mathbb{Z}_n$. Vamos considerar a sequência de potências $a, a^2, a^3, \ldots$, todas reduzidas módulo n. Suponha, por contradição, que nenhuma delas é congruente a 1 módulo n. Entretanto, como $\mathbb{Z}_n$ é um conjunto finito, essa sequência infinita de potências não pode conter para sempre valores distintos entre si. Eventualmente, para algum inteiro positivo l, o valor de $a^l \bmod n$ será igual ao de uma potência anterior da sequência, $a^m \bmod n$, com $m < l$. Temos então $a^l \equiv a^m \pmod{n}$. Seja α o inverso multiplicativo de a. Multiplicando em ambos os lados da congruência por α^m, temos $a^l\alpha^m \equiv a^m\alpha^m$, que pode ser escrita como $\alpha^{l-m} \equiv 1$, o que é uma contradição com a hipótese anterior de que nenhuma potência de a é congruente a 1.

$(3 \Rightarrow 1)$ Suponha que existe um inteiro positivo k tal que $a^k \equiv 1 \pmod{n}$. Temos então $aa^{k-1} \equiv 1 \pmod{n}$, o que significa que a^{k-1} é o inverso multiplicativo de a módulo n. ■

A prova do teorema acima nos fornece um método tanto para determinar a existência do inverso multiplicativo de um elemento de $\mathbb{Z}_n$ quando para calcular este inverso quando ele existe: podemos realizar ambas as tarefas ao mesmo tempo utilizando o Algoritmo Euclidiano Estendido.

Seja $\overline{a} \in \mathbb{Z}_n$. Aplicando o Algoritmo Euclidiano Estendido a a e n, obteremos o máximo divisor comum $d = \text{mdc}(a, n)$ e dois inteiros α e β tais que

$$\alpha a + \beta n = d.$$

Se $d \neq 1$, então $\overline{a}$ não possui inverso multiplicativo em $\mathbb{Z}_n$. Já se $d = 1$, o inverso multiplicativo de $\overline{a}$ existe e já foi calculado pelo Algoritmo Euclidiano Estendido,

conforme nos mostrou a prova do teorema acima: a classe de equivalência do inteiro α calculado pelo algoritmo ($\overline{\alpha}$) é o inverso multiplicativo de $\overline{a}$ em $\mathbb{Z}_n$. Não existe nenhuma garantia de que o valor retornado pelo algoritmo vá estar no intervalo $0 \leq i < n$. Entretanto, se desejarmos um valor neste intervalo, basta calcularmos a forma reduzida de α módulo n. Deixamos como exercício a descrição formal deste algoritmo de teste de existência e cálculo do inverso multiplicativo modular no formato dos outros algoritmos apresentados neste capítulo (Exercício 8).

Exemplo 2.17. *Vamos verificar se $\overline{9}$ possui inverso multiplicativo em $\mathbb{Z}_{24}$ e calcular quem ele é, caso exista. Para isso, vamos aplicar o Algoritmo Euclidiano Estendido a 9 e 24.*

R	Q	α	β
9	–	1	0
24	–	0	1
9	0	1	0
6	2	-2	1
$\underline{3}$	1	3	-1
0	2	–	–

Temos que $mdc(9, 24) = 3 \neq 1$, logo $\overline{9}$ não possui inverso em $\mathbb{Z}_{24}$.

Exemplo 2.18. *Vamos verificar se $\overline{9}$ possui inverso multiplicativo em $\mathbb{Z}_{32}$ e calcular quem ele é, caso exista. Para isso, vamos aplicar o Algoritmo Euclidiano Estendido a 9 e 32.*

R	Q	α	β
9	–	1	0
32	–	0	1
9	0	1	0
5	3	-3	1
4	1	4	-1
$\underline{1}$	1	$\underline{-7}$	2
0	4	–	–

Temos que $mdc(9, 32) = 1$, logo $\overline{9}$ possui inverso em $\mathbb{Z}_{32}$. Este inverso é $\overline{\alpha} = \overline{-7}$. A forma reduzida de -7 módulo 32 é 25, então também podemos escrever o inverso de $\overline{9}$ como $\overline{25}$. De fato, $9.25 \equiv 225 \equiv 1 \pmod{32}$.

2.5 Sistemas de Congruências

Finalizamos este capítulo apresentando um método que nos permite resolver o seguinte problema. Suponha que desejamos calcular a classe de equivalência de um inteiro x módulo u (ou, de modo equivalente, a forma reduzida de x módulo u), mas que realizar este cálculo diretamente seja muito trabalhoso. Se conhecermos dois fatores m e n de u tais que $u = mn$ e $\text{mdc}(m, n) = 1$, ao invés de realizar o cálculo diretamente, muitas vezes é mais eficiente calcular as classes de equivalência de x módulo m e módulo n e então "colar" estas duas soluções para obter a solução módulo u. Apresentamos nesta seção um resultado que nos permite realizar esta "colagem".

Este resultado é conhecido como Teorema Chinês do Resto, pois sua descrição mais antiga se encontra em um livro chinês de Matemática publicado entre os séculos III e V da era comum. O autor deste livro é conhecido apenas como Sunzi (ou Sun Tzu), que significa "Mestre Sun", e o livro é intitulado "O Clássico Matemático de Sunzi" [28].

Teorema 2.6 (Teorema Chinês do Resto)**.** *Considere dois números inteiros $m, n \geq 2$ tais que $mdc(m, n) = 1$ e dois números inteiros $0 \leq a < m$ e $0 \leq b < n$. Então, o sistema de congruências*

$$\begin{cases} x \equiv a \pmod{m} \\ x \equiv b \pmod{n} \end{cases}$$

tem exatamente uma solução módulo mn.

Demonstração: Primeiramente, vamos mostrar que a solução existe. Em seguida, mostraremos que ela é única.

Para mostrar que a solução existe, vamos mostrar como calculá-la. Escrevendo a primeira congruência como uma igualdade de inteiros, temos $x = a+mk$, para algum $k \in \mathbb{Z}$. Substituímos este valor de x na segunda congruência, obtendo $a + mk \equiv b \pmod{n}$. Esta congruência é equivalente a $mk \equiv b - a \pmod{n}$. Para isolarmos a indeterminada k, precisamos multiplicar os dois lados da congruência pelo inverso de m módulo n. Como temos a hipótese de que $\text{mdc}(m, n) = 1$, este inverso existe (Teorema 2.5). Seja α o inverso de m módulo n. Temos então $k \equiv \alpha(b - a) \pmod{n}$. Escrevendo esta congruência como uma igualdade de inteiros, obtemos $k = \alpha(b - a) + nl$, para algum $l \in \mathbb{Z}$. Substituindo este valor de k de volta na primeira igualdade, obtemos

$$x = a + mk = a + m(\alpha(b - a) + nl) = a(1 - m\alpha) + bm\alpha + mnl.$$

Podemos manipular esta expressão um pouco mais, lembrando que α e m estão relacionados. Temos que α é o inverso de m módulo n e pode ser obtido aplicando o Algoritmo Euclidiano Estendido a m e n, que nos fornece a igualdade $\alpha m + \beta n = 1$. Logo, $1 - m\alpha = \beta n$. Substituindo este valor de $1 - m\alpha$ na expressão acima, ficamos com $x = an\beta + bm\alpha + mnl$, que é equivalente à congruência

$$x \equiv an\beta + bm\alpha \pmod{mn}.$$

Desta forma, para determinar a solução do sistema de congruências, aplicamos o Algoritmo Euclidiano Estendido aos módulos m e n, obtendo os valores α e β e calculamos a solução módulo mn de acordo com a congruência acima.

Vamos agora mostrar que esta solução é única módulo mn. Suponha, por contradição, que $x \not\equiv y \pmod{mn}$ sejam ambos soluções do sistema de congruências. Então, temos

$$\begin{cases} x \equiv a \pmod{m} \\ x \equiv b \pmod{n}; \end{cases} \quad \begin{cases} y \equiv a \pmod{m} \\ y \equiv b \pmod{n}. \end{cases}$$

Subtraindo as duas congruências módulo m, obtemos $x - y \equiv 0 \pmod{m}$. Analogamente, temos $x - y \equiv 0 \pmod{n}$. Desta forma, tanto m quanto n dividem $x - y$. Como ambos dividem $x - y$ e $\text{mdc}(m, n) = 1$, então mn divide $x - y$ (Lema 2.2), o que significa que $x \equiv y \pmod{mn}$, contradizendo a hipótese anterior de que x e y eram soluções distintas módulo mn. Desta forma, a solução do sistema de congruências é realmente única. ■

Assim como vimos anteriormente no caso do Teorema 2.5, a prova do Teorema Chinês do Resto também nos fornece um algoritmo para calcular a classe de equivalência de x módulo mn quando $\text{mdc}(m, n) = 1$ e conhecemos as classes de equivalência de x módulo m e módulo n. O algoritmo envolve a aplicação do Algoritmo Euclidiano Estendido a m e n e o uso dos inteiros α e β retornados por ele para o cálculo da classe de equivalência de x módulo mn através da fórmula

$$x \equiv an\beta + bm\alpha \pmod{mn}.$$

Esta fórmula não garante o cálculo de um valor no intervalo $0 \leq i < mn$. Entretanto, se quisermos um valor neste intervalo, basta calcular a forma reduzida de $an\beta + bm\alpha$ módulo mn. Este algoritmo derivado da prova do Teorema Chinês do Resto é conhecido como *Algoritmo Chinês do Resto*. Deixamos como exercício a sua descrição formal no formato dos outros algoritmos apresentados neste capítulo (Exercício 10).

Não é difícil perceber que o Algoritmo Chinês do Resto pode ser generalizado para situações em que temos uma sequência de congruências

$$x \equiv a_i \pmod{n_i} \qquad 1 \leq i \leq k,$$

onde $k > 2$ e $\text{mdc}(n_i, n_j) = 1$, se $i \neq j$. Para isto, aplicamos a versão básica de duas congruências do Algoritmo Chinês do Resto às duas primeiras congruências da sequência. Ele dará como resposta uma congruência $x \equiv a' \pmod{n_1 n_2}$. Agora, como $\text{mdc}(n_1, n_3) = 1$ e $\text{mdc}(n_2, n_3) = 1$, então $\text{mdc}(n_1 n_2, n_3) = 1$. Podemos então aplicar novamente a versão básica do algoritmo ao par de congruências formado pela terceira congruência da sequência e pela congruência $x \equiv a' \pmod{n_1 n_2}$ que calculamos na aplicação anterior. Esta segunda aplicação nos dará como resposta uma congruência $x \equiv a'' \pmod{n_1 n_2 n_3}$. Continuamos com este processo até que todas as congruências da sequência tenham sido utilizadas e obtenhamos a nossa resposta final $x \equiv \widehat{a} \pmod{n_1 n_2 \ldots n_k}$. A formalização desta versão mais elaborada do Algoritmo Chinês do Resto também é deixada como exercício (Exercício 11).

Exemplo 2.19. *Vamos resolver o seguinte sistema de congruências utilizando o Algoritmo Chinês do Resto:*

$$\begin{cases} x \equiv 1 \pmod{3} \\ x \equiv 5 \pmod{11} \\ x \equiv 12 \pmod{32}. \end{cases}$$

Temos que $mdc(3, 11) = 1$, $mdc(3, 32) = 1$ *e* $mdc(11, 32) = 1$, *logo podemos aplicar o Algoritmo Chinês do Resto para encontrar a classe de equivalência de* x *módulo* $3.11.32 = 1056$. *Começamos resolvendo o sistema formado pelas duas primeiras congruências, aplicando o Algoritmo Euclidiano Estendido aos módulos* 3 *e* 11.

R	Q	α	β
3	–	1	0
11	–	0	1
3	0	1	0
2	3	-3	1
1	1	4	-1
0	2	–	–

Temos então a confirmação de que $mdc(3, 11) = 1$ *e temos* $\alpha = 4$ *e* $\beta = -1$. *Aplicamos estes valores na fórmula*

$$x \equiv an\beta + bm\alpha \pmod{mn},$$

onde a *é o valor do lado direito da primeira congruência (*1*),* b *é o valor do lado direito da segunda congruência (*5*),* m *é o módulo da primeira congruência (*3*) e* n *é o módulo da segunda congruência (*11*). Temos então*

$$x \equiv an\beta + bm\alpha \equiv 1.11.(-1) + 5.3.4 \equiv -11 + 60 \equiv 49 \equiv 16 \pmod{33}.$$

Resolvemos agora o sistema formado pela terceira congruência do sistema original e pela congruência que acabamos de obter:

$$\begin{cases} x \equiv 12 \pmod{32} \\ x \equiv 16 \pmod{33}. \end{cases}$$

Aplicamos o Algoritmo Euclidiano Estendido aos módulos 32 *e* 33*.*

R	Q	α	β
32	$-$	1	0
33	$-$	0	1
32	0	1	0
1	1	-1	1
0	32	$-$	$-$

Temos $\alpha = -1$ *e* $\beta = 1$*. Utilizamos novamente a fórmula dada pelo Teorema Chinês do Resto com os valores do sistema de congruências atual.*

$$x \equiv an\beta + bm\alpha \equiv 12.33.1 + 16.32.(-1) \equiv 396 - 512 \equiv -116 \equiv 940 \pmod{1056}.$$

Logo, a solução para o nosso sistema original é $x \equiv 940 \pmod{1056}$*.*

2.6 Exercícios

1. Generalize algoritmo da divisão apresentado neste capítulo para que o dividendo possa ser qualquer inteiro.
2. Aplique o Algoritmo Euclidiano Estendido aos seguintes pares de inteiros positivos:
   2.1. 2568 e 122
   2.2. 78 e 1046
   2.3. 1127 e 175
   2.4. 112 e 2046
3. Uma equação diofantina linear em duas variáveis é uma equação
   $$ax + by = c,$$
   em que a, b e c são inteiros e as variáveis x e y também só podem assumir valores inteiros. Descreva como utilizar o Algoritmo Euclidiano Estendido para testar se uma equação diofantina possui solução e para calculá-la, caso exista.
4. Calcule as seguintes formas reduzidas:
   4.1. 38 mod 14
   4.2. -12 mod 8
   4.3. 15 mod 4
   4.4. -43 mod 27
5. Verifique que, se $\overline{a} = \overline{a'}$ e $\overline{b} = \overline{b'}$, então $\overline{a - b} = \overline{a' - b'}$.

6. Utilize as operações de aritmética modular para provar os seguintes critérios de divisibilidade:

   6.1. Um número é divisível por 2 se e somente se o seu algarismo das unidades é divisível por 2.

   6.2. Um número é divisível por 3 se e somente se a soma de seus algarismos é divisível por 3.

   6.3. Um número é divisível por 5 se e somente se o seu algarismo das unidades é divisível por 5 (o algarismo das unidades é 0 ou 5).

   6.4. Um número é divisível por 9 se e somente se a soma de seus algarismos é divisível por 9.

   6.5. Um número é divisível por 11 se e somente se a soma alternada de seus algarismos é divisível por 11.

7. Determine a forma reduzida das seguintes potências:

   7.1. $7^{1234} \bmod 14$

   7.2. $3^{14593} \bmod 8$

   7.3. $12^{3157} \bmod 20$

   7.4. $2^{20017} \bmod 11$

8. Escreva uma descrição formal do algoritmo para teste de existência e cálculo do inverso multiplicativo modular no formato dos outros algoritmos apresentados neste capítulo.

9. Determine se os seguintes elementos possuem inversos multiplicativos e calcule-os, caso existam:

   9.1. $\overline{9}$ em $\mathbb{Z}_{20}$

   9.2. $\overline{12}$ em $\mathbb{Z}_{75}$

   9.3. $\overline{2}$ em $\mathbb{Z}_{101}$

   9.4. $\overline{42}$ em $\mathbb{Z}_{147}$

10. Escreva uma descrição formal do Algoritmo Chinês do Resto no formato dos outros algoritmos apresentados neste capítulo.

11. Escreva uma descrição formal da versão mais elaborada do Algoritmo Chinês do Resto apresentada no final deste capítulo. Esta versão deve poder operar com mais de duas congruências.

12. Resolva, utilizando o Algoritmo Chinês do Resto, o sistema

$$\begin{cases} x \equiv 1 \pmod 2 \\ x \equiv 2 \pmod 5 \\ x \equiv 5 \pmod{11}. \end{cases}$$

13. Determine o menor inteiro positivo que deixa resto 2 na divisão por 7, resto 4 na divisão por 15 e resto 10 na divisão por 19.

Capítulo 3

Números Primos

Neste capítulo, continuamos a nossa apresentação de conceitos matemáticos básicos necessários para as nossas aplicações com uma pequena discussão a respeito dos números primos.

Primeiramente, apresentamos os conceitos fundamentais de *número primo* e *número composto*. Em seguida, descrevemos uma prova bastante simples de que existem infinitos números primos. Certamente, este é um resultado de que ninguém duvida. Mesmo assim, é importante termos uma prova formal dele, uma vez que números primos são necessários para a construção das chaves utilizadas pelo método El Gamal. Portanto, a infinidade dos números primos nos garante que sempre podemos construir novas chaves conforme seja necessário, isto é, o método El Gamal nunca se tornará obsoleto pelo esgotamento dos números primos utilizados.

Outro resultado fundamental que apresentamos a respeito dos números primos é o da unicidade da fatoração de qualquer inteiro positivo $n \geq 2$ em fatores primos.

Em seguida, apresentamos o crivo de Eratóstenes, um método bastante simples para obter uma lista de todos os números primos até um determinado limite superior.

Retornando à aritmética modular, apresentamos o Pequeno Teorema de Fermat, um resultado muito útil para cálculos com inteiros modulares em um conjunto $\mathbb{Z}_p$, onde p é primo. Descrevemos também como utilizar o Pequeno Teorema de Fermat em conjunto com o Algoritmo Chinês do Resto para simplificar bastante o cálculo de potências modulares em alguns casos.

Finalmente, encerrando nossa discussão sobre números primos, apresentaremos o teste de Miller-Rabin, que é uma forma muito eficiente de testar computacionalmente se um dado número é ou não primo sem a necessidade de obter sua fatoração em primos. Como precisamos de números primos para construir as chaves do método El Gamal, se não possuíssemos uma forma eficiente de testar a primalidade de números, então não teríamos como utilizar o El Gamal na prática.

3.1 Conceitos e Resultados Fundamentais

Iniciamos esta seção com o conceito central deste capítulo: o conceito de *número primo*. Apresentamos também o seu conceito dual, o de *número composto*.

Nesta seção, apresentamos também alguns resultados fundamentais a respeito dos números primos, como a infinidade dos primos, a chamada Propriedade Fundamental dos Primos e a unicidade da fatoração em primos de um inteiro $n \geq 2$.

Definição 3.1 (Números Primos e Compostos)**.** *Seja n um número inteiro positivo maior do que* 1*. Dizemos que n é um número* primo *caso seus únicos divisores sejam* 1 *e o próprio n. Por outro lado, um número inteiro positivo maior do que* 1 *que não é primo é chamado de número* composto*.*

De acordo com a definição de *divisor próprio* (Definição 2.3), podemos definir um número primo como um inteiro $n \geq 2$ que *não possui* divisores próprios. Por outro lado, um número composto é um inteiro $n \geq 2$ que *possui* divisores próprios.

Repare que não classificamos o número 1 nem como primo nem como composto. Dentro do conjunto dos números inteiros positivos, o 1 é diferente de todos os outros em um aspecto importante: ele é o único que possui inverso multiplicativo neste mesmo conjunto. Este fato faz com que 1 seja denominado uma *unidade* do conjunto dos inteiros (sendo a única outra unidade o inteiro negativo -1).

Exemplo 3.1. *Os números* $2, 3, 5, 7$ *e* 11 *são os 5 menores números primos. O número* $2^{31} - 1 = 2147483647$ *também é primo, embora esta conclusão já não seja imediata como nos números anteriores. Por outro lado, os números* $4 = 2.2$*,* $10 = 2.5$*,* $21 = 3.7$ *e* $2^{23} - 1 = 8388607 = 47.178481$ *são exemplos de números compostos.*

Definição 3.2. *Um* divisor primo *ou* fator primo *de um inteiro $n \geq 2$ é um fator f de n tal que $f > 1$ e f é um número primo.*

Apresentamos abaixo duas propriedade básicas a respeito dos números compostos. Intuitivamente, a primeira propriedade esclarece a razão do nome "composto" para estes números: estes números podem ser obtidos pela composição de dois ou mais números primos através da operação de produto.

Propriedade 3.1. *Seja $n \geq 2$ um número composto. Então, n possui no mínimo dois fatores que são primos e próprios.*

Demonstração: Se n é composto, então, pela definição, n possui um conjunto não-vazio de fatores próprios. Seja f o *menor* destes fatores. Logo, $n = fk$, para algum $k \in \mathbb{Z}$. Suponha, por contradição, que f seja composto. Então, novamente pela definição, f possuirá um conjunto não-vazio de fatores próprios. Seja f' um destes fatores. Logo, $f = f'l$, para algum $l \in \mathbb{Z}$. Mas então, podemos escrever $n = f'(lk)$, o que significa que f' divide n. Entretanto, $1 < f' < f$, já que f' é fator próprio de f, o que é uma contradição com a nossa escolha de f como o menor fator de n maior do que 1. Assim, f deve ser primo.

Como $n = fk$ e $1 < f < n$, temos então que $1 < k < n$. Se k também for primo, temos então dois fatores primos e próprios de n (f e k) e a propriedade é verdadeira. Caso k seja composto, ele terá um conjunto não-vazio de fatores próprios. Podemos repetir o raciocínio acima tomando o *menor* destes fatores de k, denotado por k'. Temos então que $k = k'm$, para algum $m \in \mathbb{Z}$. Pelo raciocínio do parágrafo anterior, k' também será primo. De $n = fk$ e $k = k'm$, temos que $n = k'(fm)$. Assim, k' divide n. Além disso, $1 < k' < k < n$, já que k' é fator próprio de k. Isso significa que k' também é fator próprio de n. Temos então f e k' como dois fatores primos e próprios de n. ■

Propriedade 3.2. *Seja $n \geq 2$ um número composto. Então, o menor fator f de n é tal que $f \leq \lfloor \sqrt{n} \rfloor$, onde $\lfloor \sqrt{n} \rfloor$ denota a parte inteira da raiz quadrada positiva de n.*

Demonstração: Como f é fator de n, então $n = fk$, para algum $k \in \mathbb{Z}$. Temos então que k também é fator de n. Mas como f é o *menor* fator de n, então devemos ter $f \leq k$. Multiplicando os dois lados desta desigualdade por f, obtemos $f^2 \leq fk = n$. Portanto, temos que $f \leq \sqrt{n}$. Como f é um número inteiro, se $f \leq \sqrt{n}$, temos necessariamente que $f \leq \lfloor \sqrt{n} \rfloor$. ■

Uma vez que estudamos algumas propriedades dos números compostos, passemos agora ao estudo de algumas propriedades dos números primos.

Iniciamos este estudo com uma prova de que existem infinitos números primos entre os inteiros positivos. Ao longo do tempo, dezenas de provas da infinitude dos primos foram desenvolvidas por diversos matemáticos diferentes. O livro [24] realiza uma catalogação de várias destas provas.

A prova que optamos por exibir abaixo é uma das provas mais simples, bastante similar à prova original deste resultado elaborada por Euclides no livro IX da sua coletânea "Elementos" [6].

Como primeiro passo antes de podermos descrever esta prova, precisamos da definição do *primorial* de um inteiro positivo.

Definição 3.3 (Primorial)**.** *Definimos o* primorial *de um número inteiro positivo $n \geq 2$, denotado por $n^{\#}$, como o produto de todos os números primos menores ou iguais a n.*

Exemplo 3.2. *Temos $6^{\#} = 5^{\#} = 5.3.2 = 30$. Por outro lado, $10^{\#} = 7.5.3.2 = 210$.*

Utilizando a noção acima de primorial, podemos agora provar que existem infinitos números primos.

Teorema 3.1 (Infinidade dos Primos)**.** *Existem* infinitos *números primos.*

Demonstração: Suponha, por contradição, que exista uma quantidade finita de números primos. Logo, existe um número primo p que é *o maior* de todos os primos. O primorial de p, denotado por $p^{\#}$, será então o produto de *todos* os números primos. Vamos considerar o número $n = p^{\#} + 1$. Como $n > p$ e p é o maior número primo, então n é um número composto. Desta forma, pela Propriedade 3.1, existe um número primo $1 < q < n$ que é divisor de n, isto é, $n = qn'$, para algum $n' \in \mathbb{Z}$. Por outro lado, como q é primo, q também é divisor de $p^{\#}$, isto é, $p^{\#} = qk$, para algum $k \in \mathbb{Z}$. Podemos então escrever a igualdade $n = p^{\#} + 1$ como $qn' = qk + 1$, o que é equivalente a $1 = q(n' - k)$. Mas temos então, por esta última igualdade, que q é divisor de 1. Isto implica que $q = 1$, o que é uma contradição com a hipótese anterior de que $q > 1$. ■

Continuamos nosso estudo sobre os números primos com a chamada Propriedade Fundamental dos Primos. Esta é uma propriedade muito útil para a prova de diversos resultados ao longo deste livro.

Propriedade 3.3 (Propriedade Fundamental dos Primos)**.** *Seja p um número primo e a e b números inteiros. Se p divide ab, então p divide a ou p divide b.*

Demonstração: Suponha que p é primo e que p divide ab. Se p divide a, não é necessário provar mais nada. Suponha então que p não divide a. Neste caso, queremos mostrar que p divide b.

Como p é primo, os únicos divisores de p são 1 e p. Desta forma, ou $\text{mdc}(a, p) = 1$ ou $\text{mdc}(a, p) = p$. Mas, como estamos supondo que p não divide a, p não é um divisor comum entre p e a. Assim, $\text{mdc}(a, p) = 1$. De acordo com o Lema 2.2, se p

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

Finalmente, descrevemos abaixo como os números compostos e primos se relacionam, mostrando que todo número inteiro $n \geq 2$ possui uma *única* fatoração em primos.

Definição 3.4 (Fatoração em Primos)**.** *Dado um número inteiro $n \geq 2$, definimos a sua* fatoração em primos*, chamada também apenas de* fatoração*, da seguinte forma:*

$$n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k},$$

onde $1 < p_1 < p_2 < \ldots < p_k$ são primos e $e_i \geq 1$, para todo $1 \leq i \leq k$, chamados de multiplicidades*.*

Teorema 3.2 (Unicidade da Fatoração ou Teorema Fundamental da Aritmética)**.** *Seja $n \geq 2$ um número inteiro. Então, a sua fatoração em primos é única.*

Demonstração: Suponha, por contradição, que existam inteiros positivos que possuam pelo menos duas fatorações em primos distintas. Seja n o menor destes números. Escrevemos então duas fatorações distintas de n:

$$n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k} = q_1^{f_1} q_2^{f_2} \dots q_l^{f_l},$$

onde $1 < p_1 < p_2 < \ldots < p_k$ e $1 < q_1 < q_2 < \ldots < q_l$ são primos, $e_i \geq 1$, para todo $1 \leq i \leq k$, e $f_j \geq 1$, para todo $1 \leq j \leq l$.

Como temos $n = p_1(p_1^{e_1-1} p_2^{e_2} \dots p_k^{e_k})$, p_1 divide n. Por outro lado, como $n = q_1^{f_1} q_2^{f_2} \dots q_l^{f_l}$, p_1 divide este produto. Como p_1 é primo, se p_1 divide um produto, então ele divide ao menos um dos fatores, pela Propriedade Fundamental dos Primos (Propriedade 3.3). Assim, existe q_i, $1 \leq i \leq l$, tal que p_1 divide q_i. Mas p_1 e q_i são ambos primos. A única forma de um primo dividir outro é se ambos forem iguais. Logo, $q_i = p_1$. Substituindo q_i por p_1 na segunda fatoração, obtemos

$$n = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k} = q_1^{f_1} q_2^{f_2} \dots q_{i-1}^{f_{i-1}} p_1^{f_i} q_{i+1}^{f_{i+1}} \dots q_l^{f_l}.$$

Temos então que $n = p_1 n'$ e podemos obter duas fatorações distintas para n' eliminando um dos fatores p_1 de cada uma das fatorações acima. Temos então

$$n' = p_1^{e_1-1} p_2^{e_2} \dots p_k^{e_k} = q_1^{f_1} q_2^{f_2} \dots q_{i-1}^{f_{i-1}} p_1^{f_i-1} q_{i+1}^{f_{i+1}} \dots q_l^{f_l}.$$

Mas, como $p_1 > 1$, temos que $n' < n$. Então, as duas fatorações distintas para n' são uma contradição com a nossa escolha de n como o menor inteiro positivo que possui pelo menos duas fatorações em primos distintas. Desta forma, a fatoração em primos de qualquer inteiro $n \geq 2$ é única. ■

Exemplo 3.3. *Como exemplo, a fatoração em primos de* 18 *é* $18 = 2.3^2$*. Já a fatoração de* 100 *é* $100 = 2^2.5^2$*. Finalmente, a fatoração de* 13230 $= 2.3^3.5.7^2$*. Conforme mostramos acima, estas fatorações são* únicas*.*

Resta-nos apenas descrever algum método para calcular a fatoração em primos de um número $n \geq 2$.

Existem vários algoritmos conhecidos na literatura para o cálculo da fatoração de um inteiro positivo. Cada um deles é mais eficiente para fatorar números que atendam determinadas características, mas todos eles são bastante ineficientes no

caso geral. Devido a isso, o problema da fatoração é considerado um problema bastante complexo, mesmo com ajuda computacional.

A dificuldade computacional de se calcular a fatoração de um inteiro no caso geral é a base da segurança do método de criptografia de chave pública RSA, que possui este nome devido às iniciais dos sobrenomes de seus criadores (Ron Rivest, Adi Shamir e Leonard Adleman). Para realizar o cálculo da chave privada de uma implementação do RSA a partir somente do conhecimento de sua chave pública, seria necessário o cálculo eficiente da fatoração de um número inteiro, o que não conseguimos realizar, no caso geral, com os algoritmos de fatoração conhecidos.

Tanto os detalhes do problema da fatoração de inteiros quanto os algoritmos para resolvê-lo estão fora do escopo deste livro. Da mesma forma, também não trataremos aqui do método RSA. Mais adiante neste livro, descreveremos um outro método de criptografia de chave pública, o El Gamal. Para maiores detalhes sobre o problema da fatoração de inteiros, sobre alguns algoritmos de fatoração conhecidos e sobre o método RSA, o livro [3] é uma ótima referência em Língua Portuguesa. Outras boas referências, que apresentam alguns outros algoritmos de fatoração, são os livros [10], [13] e [16]. A descrição original do método RSA foi feita no artigo [25].

Um método simples, porém muito ineficiente, para determinar um fator primo de um número inteiro $n \geq 2$ nos é dado pelas Propriedades 3.1 e 3.2. A prova da Propriedade 3.1 nos mostra que se tentarmos encontrar um fator $2 \leq f \leq n$ de n testando os valores deste intervalo um por um em ordem crescente, o menor fator que encontrarmos será necessariamente um *fator primo*. Além disso, a Propriedade 3.2 nos garante que, caso não tenhamos encontrado nenhum fator $f \leq \lfloor \sqrt{n} \rfloor$ para n, então n não possui nenhum fator *próprio*. Utilizamos estas duas ideias no algoritmo abaixo, que calcula o *menor fator primo* de um inteiro $n \geq 2$.

Algoritmo 3.1: Algoritmo Simples para Fatoração

Entrada: Um número inteiro $n \geq 2$.

Saída: Um número inteiro f que é o menor fator primo de n.

Instruções:

1. $f \leftarrow 2$
2. Enquanto $f \leq \lfloor \sqrt{n} \rfloor$, faça: $\#\lfloor \sqrt{n} \rfloor =$ parte inteira de $\sqrt{n}$
   2.1. $r \leftarrow$ Resto da divisão de n por f.
   2.2. Se $r = 0$, então retorne f.
   2.3. Senão, $f \leftarrow f + 1$
3. Retorne n.

Este algoritmo calcula apenas o menor fator primo de um inteiro $n \geq 2$, mas podemos aplicá-lo sucessivas vezes de forma a obter a fatoração completa de n. Após aplicarmos o algoritmo pela primeira vez a n, obtemos o fator f_1. Calculamos então o quociente n' da divisão de n por f_1 ($n = f_1 n'$) e aplicamos o algoritmo a n', obtendo o fator f_2. Prosseguimos com este processo até termos obtido todos os fatores de n. Deixamos a formalização deste algoritmo para a fatoração completa como exercício (Exercício 1).

3.2 Crivo de Eratóstenes

Nesta seção, apresentamos um algoritmo simples e bastante útil para o cálculo da lista de todos os números primos até um determinado limite superior. Este algoritmo é o *Crivo de Eratóstenes*.

O algoritmo possui este nome por ter sido descrito originalmente por Eratóstenes, um matemático da Grécia Antiga que viveu entre os séculos III e II AC. Apesar de todos os escritos originais de Eratóstenes terem se perdido, o crivo é descrito e atribuído a Eratóstenes por Nicômaco, um matemático que viveu entre os séculos I e II DC, em seu livro "Introdução à Aritmética" [20].

O Crivo de Eratóstenes funciona como uma peneira ou um filtro, separando os números compostos dos números primos. Dado um limite superior n, começamos com uma lista de todos os inteiros maiores ou iguais a 2 e menores ou iguais a n. Após uma passagem do crivo, alguns números compostos na lista são identificados e "filtrados". Após uma quantidade suficiente de passagens do crivo, restarão na lista apenas os números primos. É importante notar que, em nenhum momento o crivo "filtra" um número primo. Desta forma, ao final do processo, obtemos uma lista que contém *todos* os primos menores ou iguais ao limite superior n.

Uma primeira observação que podemos fazer diz respeito aos números pares na lista. Não precisamos de nenhum método elaborado para determinar quais números pares são primos e quais são compostos. Sabemos que o único número par que é primo é o 2, sendo todos os outros pares compostos. Desta maneira, é desnecessário passar os números pares da lista pelo processo do crivo. Podemos então colocar na nossa lista apenas os números *ímpares* maiores ou iguais a 3 e menores ou iguais a n. O único cuidado necessário ao fazermos isto é lembrarmos que o primo 2 não estará incluído na lista obtida diretamente pelo crivo, precisando ser acrescentado a esta lista no final do processo.

Cada número na nossa lista terá associado a ele um *índice*, denotando a sua posição na lista. O primeiro número da lista terá índice 0. Como exemplo, construímos abaixo a lista de todos os ímpares maiores ou iguais a 3 e menores ou iguais a 13:

Índice	0	1	2	3	4	5
Número	3	5	7	9	11	13

A partir deste exemplo, podemos notar que, dado um índice i, o número que é armazenado na lista na posição indicada por este índice é $j = 2i + 3$. Reciprocamente, dado um número j armazenado na lista, ele é armazenado na posição indicada pelo índice $i = (j - 3)/2$. Repare que, como estamos armazenando apenas números ímpares na lista, j necessariamente é ímpar. Logo $j - 3$ é par, o que significa que $(j - 3)/2$ é um número inteiro.

Conforme foi dito acima, a ideia principal do crivo é ir eliminando aos poucos os números compostos que aparecem na lista até que restem apenas os números primos. Para isso, o procedimento de cada etapa é o seguinte:

1. Buscamos o primeiro elemento da lista que ainda não utilizamos em passos anteriores e que ainda não foi eliminado. No início da execução do crivo, este será o primeiro elemento da lista.

2. Seja j o número selecionado no passo anterior e i o seu índice. Temos a relação $j = 2i + 3$ entre eles. A partir da posição i da lista, iremos eliminar números a cada j posições. Isto é, iremos eliminar os números nas posições $i + j, i + 2j, i + 3j, \ldots$ até que encontremos o final da lista.

O que este processo de eliminação está fazendo é eliminar todos os múltiplos de j que estão na lista, já que, se são múltiplos de j, então são compostos.

Para realizar o processo de eliminação de um número, não devemos simplesmente excluí-lo da lista, pois isto mudaria o tamanho da lista e alteraria a relação entre os números e os índices da lista, tornando a igualdade $j = 2i + 3$ falsa. A melhor forma de realizar a eliminação de um número é substituí-lo por 0 na lista.

3. Repetimos este processo em uma nova etapa, até que não restem mais números que possam ser selecionados no primeiro passo acima.

Podemos reparar pela descrição acima que o maior mérito do Crivo de Eratóstenes é conseguir encontrar quais são os números que possuem um determinado fator sem realizar nenhuma conta de divisão. As contas de divisão são os cálculos aritméticos mais custosos do ponto de vista computacional.

Exemplo 3.4. *Vamos realizar os passos descritos acima na lista com os inteiros ímpares entre* 3 *e* 40*.*

Índice	0	1	2	3	4	5	6	7	8	9
Número	3	5	7	9	11	13	15	17	19	21
Índice	10	11	12	13	14	15	16	17	18	
Número	23	25	27	29	31	33	35	37	39	

Nossa primeira seleção será o número $j = 3$, com índice $i = 0$. Iremos então, a partir da posição 0*, eliminar os elementos da lista de* 3 *em* 3*. Isto significa que vamos eliminar os elementos nas posições* $3, 6, 9, 12, 15$ *e* 18*, substituindo-os por* 0*.*

Índice	0	1	2	3	4	5	6	7	8	9
Número	3	5	7	0	11	13	0	17	19	0
Índice	10	11	12	13	14	15	16	17	18	
Número	23	25	0	29	31	0	35	37	0	

Este processo de eliminação retirou da lista todos os múltiplos de $j = 3$.

Realizamos agora uma nova etapa, com uma nova seleção de um número. O primeiro número da lista que ainda não utilizamos e que não foi eliminado é $j = 5$, com índice $i = 1$. Iremos então, a partir da posição 1*, eliminar os elementos da lista de* 5 *em* 5*. Vamos eliminar os elementos nas posições* $6, 11$ *e* 16*.*

Índice	0	1	2	3	4	5	6	7	8	9
Número	3	5	7	0	11	13	0	17	19	0
Índice	10	11	12	13	14	15	16	17	18	
Número	23	0	0	29	31	0	0	37	0	

Analogamente ao que ocorreu na etapa anterior, este processo de eliminação retirou da lista todos os múltiplos de $j = 5$. Repare que o elemento na posição 6 *já havia sido eliminado antes. Vemos então que, ocasionalmente, o crivo realiza eliminações redundantes. Não é possível evitar todas estas eliminações redundantes, mas discutiremos mais adiante algumas maneiras de diminuir a ocorrência delas.*

O próximo número selecionado é $j = 7$, com índice $i = 2$. As eliminações ocorreriam nas posições 9 *e* 16*, mas elas são ambas redundantes.*

Em seguida, como o número 9 *já foi eliminado na lista, o próximo número selecionado é* $j = 11$*, com índice* $i = 4$*. A única eliminação ocorreria na posição* 15*, mas ela é redundante.*

Na próxima etapa, o número selecionado é $j = 13$*, com índice* $i = 5$*. A única eliminação ocorreria na posição* 18*, mas ela é redundante.*

Na sequência, como o número 15 *já foi eliminado na lista, o próximo número selecionado é* $j = 17$*, com índice* $i = 7$*. Nenhuma eliminação será feita na lista já que a primeira posição de eliminação seria* 24*, uma posição que não existe nesta lista. O mesmo fenômeno acontecerá com todos os outros números maiores do que* 17 *que estão na lista.*

Podemos concluir então que o nosso processo de eliminação está encerrado. Todos os números que não foram eliminados são primos, já que não possuem nenhum fator próprio. Assim, a lista de todos os primos menores ou iguais a 40 *é formada pelos números que não foram eliminados na lista acima com a adição do número* 2*, o único primo par:*

$$L = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37].$$

Vimos no exemplo que, ocasionalmente, o Crivo de Eratóstenes realiza algumas eliminações redundantes. Não é possível evitá-las completamente, mas vamos discutir agora duas melhorias para o Crivo que causam a diminuição de eliminações redundantes.

A primeira melhoria é sugerida diretamente pela Propriedade 3.2. Ela nos diz que se m é um número *composto* que está na lista do crivo, ele vai possuir um fator menor ou igual a $\lfloor\sqrt{m}\rfloor$. Mas todo número na lista é menor ou igual ao seu limite superior n. De $m \leq n$, podemos concluir que $\lfloor\sqrt{m}\rfloor \leq \lfloor\sqrt{n}\rfloor$. Assim, todos os números compostos da lista terão algum fator menor ou igual a $\lfloor\sqrt{n}\rfloor$. Logo, no momento em que já houvermos realizado todas as eliminações com números menores ou iguais a $\lfloor\sqrt{n}\rfloor$, os números que restarem sem terem sido eliminados são necessariamente primos. Assim, podemos terminar o processo do crivo neste momento.

Exemplo 3.5. *No exemplo anterior, o nosso limite superior era* $n = 40$*. Temos que* $\lfloor\sqrt{40}\rfloor = 6$*. Logo, após as eliminações realizadas com os números* 3 *e* 5*, não eram necessárias mais eliminações. Após estas duas eliminações, os números restantes já eram os primos que buscávamos. Isso está de acordo com o que vimos no exemplo, quando todas as eliminações que tentamos fazer com números maiores do que* 6 *se mostraram redundantes.*

Para a segunda melhoria, utilizamos um raciocínio semelhante ao que utilizamos na prova da Propriedade 3.2.

Suponha que estamos realizando as eliminações com um inteiro j. Todos os múltiplos de j na lista serão eliminados. Se um destes múltiplos de j possuir também algum outro fator *menor* do que j, então ele já terá sido eliminado em alguma etapa anterior do crivo e a eliminação atual com j será redundante.

Vamos tentar então determinar alguma propriedade dos inteiros positivos que possuem j como seu *menor* fator. Suponha que m é um inteiro positivo e que j é o seu menor fator. Temos então que $m = jm'$, para algum $m' \in \mathbb{Z}$. Mas como j é o menor fator de m, então $m' \geq j$. Multiplicando esta igualdade por j de ambos os lados obtemos $m = jm' \geq j^2$. Concluímos então que, se m é um múltiplo de j tal que $m < j^2$, então ele possui algum fator menor do que j, o que torna a sua eliminação com j no crivo redundante. Assim, quando estamos

eliminando com j no crivo, podemos iniciar nossa eliminação pelo número j^2. Todos os números anteriores que forem múltiplos de j já terão sido eliminados em alguma etapa anterior do crivo.

Vamos denotar por k a posição inicial do processo de eliminação levando em consideração esta segunda melhoria. Temos que k deve ser o índice de j^2 na lista. Como temos a relação de que o índice i de um inteiro j na lista satisfaz a igualdade $i = (j-3)/2$, o índice k de j^2 será $k = (j^2-3)/2$. Assim, ao invés de eliminarmos os números de j em j na lista fazendo a primeira eliminação na posição $i+j$, onde i é o índice de j, continuamos eliminando os números de j em j, mas realizamos a primeira eliminação na posição $k = (j^2-3)/2$.

Exemplo 3.6. *No exemplo acima, se utilizarmos esta segunda melhoria, não haverá mudanças nas eliminações que realizamos com o* 3*, mas teremos diferenças nas eliminações com o* 5*. Sem a melhoria, eliminamos os números nas posições* $6, 11$ *e* 16*. Com a melhoria, a primeira eliminação ocorreria na posição* $k = (5^2-3)/2 = 11$*, prosseguindo de* 5 *em* 5*. Assim, não seria feita a eliminação na posição* 6*, mas seriam mantidas as eliminações nas posições* 11 *e* 16*. Pelo que vemos no exemplo acima, a eliminação na posição* 6 *era justamente a eliminação redundante que ocorria com* $j = 5$*. Assim, esta segunda melhoria foi capaz de evitar esta eliminação redundante.*

Exemplo 3.7. *Vamos agora apresentar a aplicação do crivo com as duas melhorias acima para encontrar a lista de todos os primos menores ou iguais a* 100*.*

De acordo com a primeira melhoria, só precisamos realizar eliminações com números menores ou iguais a $\lfloor\sqrt{100}\rfloor = 10$*.*

Índice	0	1	2	3	4	5	6	7	8	9
Número	3	5	7	9	11	13	15	17	19	21
Índice	10	11	12	13	14	15	16	17	18	19
Número	23	25	27	29	31	33	35	37	39	41
Índice	20	21	22	23	24	25	26	27	28	29
Número	43	45	47	49	51	53	55	57	59	61
Índice	30	31	32	33	34	35	36	37	38	39
Número	63	65	67	69	71	73	75	77	79	81
Índice	40	41	42	43	44	45	46	47	48	
Número	83	85	87	89	91	93	95	97	99	

Nossa primeira seleção será o número $j = 3$*, com índice* $i = 0$*. Pela segunda melhoria, iremos eliminar os números de* 3 *em* 3 *começando a eliminação na posição* $k = (3^2-3)/2 = 3$*.*

Índice	0	1	2	3	4	5	6	7	8	9
Número	3	5	7	0	11	13	0	17	19	0
Índice	10	11	12	13	14	15	16	17	18	19
Número	23	25	0	29	31	0	35	37	0	41
Índice	20	21	22	23	24	25	26	27	28	29
Número	43	0	47	49	0	53	55	0	59	61
Índice	30	31	32	33	34	35	36	37	38	39
Número	0	65	67	0	71	73	0	77	79	0
Índice	40	41	42	43	44	45	46	47	48	
Número	83	85	0	89	91	0	95	97	0	

Em seguida, selecionamos o número $j = 5$*, com índice* $i = 1$*. Novamente utilizando a segunda melhoria, iremos eliminar os números de* 5 *em* 5 *começando a eliminação na posição* $k = (5^2 - 3)/2 = 11$*.*

Índice	0	1	2	3	4	5	6	7	8	9
Número	3	5	7	0	11	13	0	17	19	0
Índice	10	11	12	13	14	15	16	17	18	19
Número	23	0	0	29	31	0	0	37	0	41
Índice	20	21	22	23	24	25	26	27	28	29
Número	43	0	47	49	0	53	0	0	59	61
Índice	30	31	32	33	34	35	36	37	38	39
Número	0	0	67	0	71	73	0	77	79	0
Índice	40	41	42	43	44	45	46	47	48	
Número	83	0	0	89	91	0	0	97	0	

Na sequência, selecionamos o número $j = 7$*, com índice* $i = 2$*. Iremos eliminar os números de* 7 *em* 7 *começando a eliminação na posição* $k = (7^2 - 3)/2 = 23$*.*

Índice	0	1	2	3	4	5	6	7	8	9
Número	3	5	7	0	11	13	0	17	19	0
Índice	10	11	12	13	14	15	16	17	18	19
Número	23	0	0	29	31	0	0	37	0	41
Índice	20	21	22	23	24	25	26	27	28	29
Número	43	0	47	0	0	53	0	0	59	61
Índice	30	31	32	33	34	35	36	37	38	39
Número	0	0	67	0	71	73	0	0	79	0
Índice	40	41	42	43	44	45	46	47	48	
Número	83	0	0	89	0	0	0	97	0	

Em seguida, como o número 9 *já foi eliminado, selecionaríamos o número* $j = 11$*. Porém, a primeira melhoria nos diz que isto não é necessário e que já obtivemos a lista de primos que buscávamos. A lista de primos será formada por todos os números que não foram eliminados pelo crivo com a adição do número* 2*:*

$$L = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41,$$
$$43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97].$$

Apresentamos abaixo a descrição formal do algoritmo do Crivo de Eratóstenes com as duas melhorias discutidas acima. Usamos a notação de colchetes para representar listas.

É importante termos em mente de que o Crivo de Eratóstenes *não é* um teste de primalidade e não deve ser utilizado como tal. Quando fornecemos um inteiro n para o Crivo de Eratóstenes, ele irá calcular a lista de todos os primos menores ou iguais a n, que pode ser uma lista grande se n for um inteiro grande. Muitas vezes, em um teste de primalidade, desejamos testar números de mais de 100 algarismos. O Crivo não conseguirá construir a lista de todos os números primos com até 100 algarismos, tanto pelo tempo necessário quanto pela memória necessária para armazenar tal lista. Desta forma, o crivo não pode ser usado como um substituto para os testes de primalidade. A sua função é gerar listas de primos, sendo que ele é bastante eficiente nesta função para limites superiores até em torno de 100 milhões, onde a lista gerada terá em torno de 5 milhões e meio de primos.

Algoritmo 3.2: Crivo de Eratóstenes

Entrada: Um número inteiro $n \geq 3$.

Saída: Uma lista L contendo todos os números primos menores ou iguais a n.

Instruções:

1. $N \leftarrow [\ ]$ # lista vazia
2. $j \leftarrow 3$
3. Enquanto $j \leq n$, faça:
   3.1. Adicione j à lista N.
   3.2. $j \leftarrow j + 2$
4. $t \leftarrow \lfloor (n-1)/2 \rfloor$ # tamanho da lista
5. $i \leftarrow 0$
6. Enquanto $(2 * i + 3) \leq \lfloor \sqrt{n} \rfloor$, faça: # primeira melhoria
   6.1. $j \leftarrow (2 * i + 3)$
   6.2. $k \leftarrow (j * j - 3)/2$ # segunda melhoria
   6.3. Enquanto $k < t$, faça:
      6.3.1. $N[k] \leftarrow 0$
      6.3.2. $k \leftarrow k + j$
   6.4. $i \leftarrow i + 1$
   6.5. Enquanto $N[i] = 0$, faça $i \leftarrow i + 1$.
7. $L \leftarrow [2]$ # lista contendo apenas o 2
8. Adicione a L todos os elementos de N que são diferentes de 0.
9. Retorne L.

3.3 Pequeno Teorema de Fermat

Nesta seção, apresentamos um resultado muito importante que permite simplificar diversos cálculos modulares quando o módulo que estamos utilizando é um número primo. Este resultado é conhecido como Pequeno Teorema de Fermat, pois este teorema foi enunciado em 1640 pelo matemático francês Pierre de Fermat, que nasceu na primeira década do século XVII e faleceu em 1665.

Como era, de certa forma, costumeiro para Fermat, ele enunciou o teorema sem apresentar uma prova (fez o mesmo com o resultado conhecido como Último Teorema de Fermat). Provas do Pequeno Teorema de Fermat foram inicialmente apresentadas pelo matemático alemão Gottfried Leibniz (1646-1716) e pelo matemático suíço Leonhard Euler (1707-1783).

Teorema 3.3 (Pequeno Teorema de Fermat)**.** *Se p é primo e a é um número inteiro tal que $a \not\equiv 0 \pmod{p}$, então $a^{p-1} \equiv 1 \pmod{p}$.*

Demonstração: Inicialmente, repare que se $a \equiv 0 \pmod{p}$, então todas as suas potências, incluindo a^{p-1} serão também congruentes a zero módulo p. Vamos supor então que $a \not\equiv 0 \pmod{p}$, o que significa que p não divide a.

Vamos considerar a sequência de números abaixo, todos reduzidos módulo p:

$$a, \quad 2a, \quad 3a, \quad \ldots, \quad (p-1)a \qquad (\text{mod } p).$$

Existem $p-1$ números inteiros nesta sequência e todos estão no intervalo $[1, p-1]$, devido à redução módulo p. Vamos mostrar que todos os números desta sequência são diferentes entre si, o que significa que todos os números inteiros do intervalo $[1, p-1]$ aparecem nesta sequência.

Suponha que tenhamos $1 \leq i \leq j \leq p-1$ e $ia \equiv ja \pmod{p}$. Temos então que $(j-i)a \equiv 0 \pmod{p}$, o que significa que p divide $(j-i)a$. Como, por hipótese, p não divide a, temos que p divide $(j-i)$. Entretanto, $0 \leq j-i < p-1$, e o único inteiro neste intervalo que é divisível pelo primo p é 0. Logo, temos $j-i=0$, o que implica $i=j$. Desta forma, se $i \neq j$, $ia \not\equiv ja \pmod{p}$.

Como a sequência de números acima contém exatamente os inteiros no intervalo $[1, p-1]$, o produto de todos os números na sequência acima será igual ao produto de todos os inteiros neste intervalo. Desta forma,

$$a(2a)(3a)\ldots(p-1)a \equiv (p-1)! \pmod{p},$$

o que implica

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

Como $(p-1)! \not\equiv 0 \pmod{p}$, podemos multiplicar a igualdade acima pelo seu inverso em ambos os lados, obtendo $a^{p-1} \equiv 1 \pmod{p}$. ■

Exemplo 3.8. *O teorema nos diz que, sem precisarmos fazer nenhuma conta,* $12^{18} \equiv 1 \pmod{19}$, $3^{22} \equiv 1 \pmod{23}$ *e* $8^{40} \equiv 1 \pmod{41}$, *uma vez que* $12 \not\equiv 0 \pmod{19}$, $3 \not\equiv 0 \pmod{23}$ *e* $8 \not\equiv 0 \pmod{41}$ *e* $19, 23$ *e* 41 *são primos.*

É muito importante manter sempre em mente as duas hipóteses do Pequeno Teorema de Fermat. Podemos ter $a^{n-1} \not\equiv 1 \pmod{n}$ se n for composto. Além disso, se $a \equiv 0 \pmod{p}$, então qualquer potência a^k, $k \geq 0$, também irá satisfazer $a^k \equiv 0 \pmod{p}$, o que significa que $a^k \not\equiv 1 \pmod{p}$, já que $0 \not\equiv 1 \pmod{p}$.

Analisando com cuidado o Pequeno Teorema de Fermat, podemos ver que ele nos fornece uma maneira de testar se um dado inteiro $n > 2$ é *composto*.

Suponha que n seja primo e seja b um inteiro tal que $2 \leq b \leq n-1$. Como b é menor do que n e diferente de zero, certamente b não é múltiplo de n. Assim, temos que $b \not\equiv 0 \pmod{n}$. Logo, como estamos supondo que n é primo, então o Pequeno Teorema de Fermat nos diz que $b^{n-1} \equiv 1 \pmod{n}$.

Vamos então observar este raciocínio no sentido inverso. Suponha que tenhamos um valor de b no intervalo $2 \leq b \leq n-1$ tal que $b^{n-1} \not\equiv 1 \pmod{n}$. Se n fosse primo, este último resultado estaria contradizendo o resultado esperado a partir do Pequeno Teorema de Fermat. Logo, neste caso, n é necessariamente composto.

Obtivemos então o seguinte teste de *composicionalidade*, baseado no Pequeno Teorema de Fermat.

Corolário 3.1. *Seja* $n > 2$ *um inteiro. Se existe um inteiro* b *no intervalo* $2 \leq b \leq n-1$ *tal que* $b^{n-1} \not\equiv 1 \pmod{n}$, *então* n *é um número composto. Neste caso, dizemos que* b *é uma* testemunha *de que* n *é composto.*

Este teste é conhecido como Teste de Fermat. Repare que se o teste nos informa que um número é composto, podemos concluir com absoluta certeza que isto é verdade, mesmo que não saibamos nenhum de seus fatores próprios. Isto nos

mostra um dos resultados mais contra-intuitivos deste estudo dos números inteiros: é possível determinar que um número é composto mesmo sem saber fatorá-lo.

Como vimos anteriormente neste capítulo, o problema da fatoração é um problema muito difícil do ponto de vista computacional. Logo, é uma boa notícia que o problema de determinar se um número é composto e o problema de calcular a fatoração de um número não sejam equivalentes, apesar de parecerem ser à primeira vista.

Se conseguimos encontrar um inteiro b no intervalo $2 \leq b \leq n-1$ tal que $b^{n-1} \not\equiv 1 \pmod{n}$, então sabemos que n é composto. Entretanto, para um valor fixado de b neste intervalo, se $b^{n-1} \equiv 1 \pmod{n}$, então *não podemos* concluir que n é primo. Existem números n que são compostos mas satisfazem a congruência $b^{n-1} \equiv 1 \pmod{n}$. Estes números são conhecidos como *pseudoprimos de Fermat*, ou simplesmente *pseudoprimos*, para a base b, já que, apesar de serem compostos, eles têm o mesmo comportamento que um primo teria com relação a esta congruência com base b.

Exemplo 3.9. *O número* 341 *é composto, já que* $341 = 11.31$. *Entretanto,* $2^{340} \equiv 1 \pmod{341}$. *Logo,* 341 *é um pseudoprimo para a base* 2.

A existência de pseudoprimos nos mostra que, quando selecionamos uma base b e executamos o Teste de Fermat com esta base, calculando a forma reduzida de b^{n-1} módulo n, o teste poderá nos dar dois resultados: o número n é composto, se $b^{n-1} \not\equiv 1 \pmod{n}$, ou o resultado do teste é *inconclusivo*, se $b^{n-1} \equiv 1 \pmod{n}$. O teste é inconclusivo neste último caso porque tanto os números primos quanto os pseudoprimos para a base b (que são números compostos) satisfazem esta última congruência. Assim, o Teste de Fermat com uma base b é capaz de distinguir alguns números compostos dos números primos, mas não todos. Como consequência, ao aplicar o Teste de Fermat com uma base, poderemos em alguns casos concluir com absoluta certeza que um número é composto, mas nunca poderemos concluir com esta mesma certeza que um número é primo.

Se o Teste de Fermat com uma base fixada não é suficiente para determinar se um número é primo, uma outra ideia poderia ser aplicar a recíproca do Teste de Fermat. Se $n > 2$ é um inteiro e, para todo inteiro b no intervalo $2 \leq b \leq n-1$, temos que $b^{n-1} \equiv 1 \pmod{n}$, então n é um número primo?

A resposta é sim. Se $n > 2$ for um número composto, a congruência $b^{n-1} \equiv 1$ não pode ser verdadeira para todos os inteiros no intervalo $2 \leq b \leq n-1$. Se n é composto, n possui um fator próprio $2 \leq f \leq n-1$. Em particular, como f é fator de n, temos que $\text{mdc}(n, f) \neq 1$. Então, pelo Teorema da Inversão Modular (Teorema 2.5), podemos concluir que não existe nenhum inteiro positivo k tal que $f^k \equiv 1 \pmod{n}$. Assim, em particular, $f^{n-1} \not\equiv 1 \pmod{n}$.

Vimos então que, caso n seja composto, existirão sempre algumas bases no intervalo $2 \leq b \leq n-1$ tais que $b^{n-1} \not\equiv 1$. Desta forma, caso pudéssemos testar todas as bases no intervalo, poderíamos concluir com absoluta certeza se n é primo ou composto. Porém, quando n é um número muito grande, como efetivamente acontece na prática, este teste através da varredura exaustiva de todas as bases no intervalo é totalmente inviável.

Outra má notícia para a eficácia do Teste de Fermat em diferenciar totalmente números primos e compostos é a existência de uma família *infinita* de inteiros *compostos*, conhecidos como *Números de Carmichael*, para os quais as únicas bases b em que temos $b^{n-1} \not\equiv 1$ são justamente as bases que utilizamos no argumento acima, isto é, as bases em que $\text{mdc}(n, b) \neq 1$. Para todos os inteiros b no intervalo $2 \leq b \leq n-1$ em que $\text{mdc}(n, b) = 1$, teremos $b^{n-1} \equiv 1 \pmod{n}$.

Assim, para conseguirmos determinar conclusivamente que um Número de Carmichael é composto através do Teste de Fermat, precisaríamos testar sucessivamente as bases no intervalo $2 \leq b \leq n-1$ até termos a sorte de encontrar uma para a qual $\text{mdc}(n, b) \neq 1$, já que apenas neste caso a congruência $b^{n-1} \not\equiv 1 \pmod{n}$ se verificará. Entretanto, encontrar uma base satisfazendo esta propriedade é equivalente a efetivamente encontrar um fator de n, já que se $d = \text{mdc}(n, b) \neq 1$, então d é fator de n. Assim, o Teste de Fermat para Números de Carmichael não é mais eficiente do que um processo tradicional de fatoração.

Como os Números de Carmichael são pseudoprimos para a maioria das bases no intervalo $2 \leq b \leq n-1$, o Teste de Fermat terá enorme dificuldade em diferenciá-los dos números primos. Desta forma, a existência dos Números de Carmichael nos mostra que o Teste de Fermat é inerentemente limitado na sua capacidade de separar os números primos dos compostos, embora não deixe de ser útil em muitos casos, nos dando uma certificação de que determinado número é composto.

Para mais informações sobre os Números de Carmichael, o livro [3] pode ser consultado.

Na próxima seção, veremos um teste mais refinado que ameniza consideravelmente as limitações do Teste de Fermat, sendo muito utilizado na prática.

Uma outra utilidade muito importante do Pequeno Teorema de Fermat é auxiliar no cálculo de potências modulares quando o módulo é um número primo p. Suponha que queremos calcular a forma reduzida de a^k módulo p, onde $a \not\equiv 0 \pmod{p}$ e p é primo. Sabemos, pelo Pequeno Teorema de Fermat, que $a^{p-1} \equiv 1 \pmod{p}$. Assim, se $k \geq p-1$, podemos simplificar o cálculo de a^k com um raciocínio simples.

Começamos dividindo k por $p-1$, obtendo $k = (p-1)q + r$, com $0 \leq r < p-1$. Então, temos

$$a^k \equiv a^{(p-1)q+r} \equiv (a^{p-1})^q a^r \pmod{p}.$$

Como $a^{p-1} \equiv 1 \pmod{p}$, podemos substituir a^{p-1} por 1 na congruência acima:

$$(a^{p-1})^q a^r \equiv 1^q a^r \equiv a^r \pmod{p}.$$

Desta forma, concluímos que, quando o módulo p é primo, $a^k \equiv a^r \pmod{p}$, onde r é o resto da divisão de k por $p-1$. Podemos então realizar o cálculo de potenciação com o expoente menor r, ao invés de com o expoente maior k.

Exemplo 3.10. *Vamos calcular a forma reduzida de* $8^{293487559837}$ *módulo* 41. *Como* 41 *é primo e* $8 \not\equiv 0 \pmod{41}$*, podemos utilizar o Pequeno Teorema de Fermat para nos auxiliar, conforme explicado acima.*

Começamos dividindo 293487559837 *por* 40*, obtendo*

$$293487559837 = 40.7337188995 + 37.$$

Assim, $8^{293487559837} \equiv 8^{37} \pmod{41}$. *Podemos perceber a enorme diferença no tamanho dos dois expoentes. Aplicar o algoritmo de exponenciação modular ao segundo expoente é claramente vantajoso.*

R	A	E	E *ímpar?*
1	8	37	*Sim*
8	23	18	*Não*
8	37	9	*Sim*
9	16	4	*Não*
9	10	2	*Não*
9	18	1	*Sim*
39	37	0	*Não*

Logo, a forma reduzida de $8^{293487559837}$ *módulo* 41 *é* 39*, sendo que não precisamos realizar os cálculos com o expoente* 293487559837 *para obtermos o resultado.*

É sempre importante relembrar que o Pequeno Teorema de Fermat só pode ser aplicado quando o módulo é primo. Logo, a simplificação acima não pode ser feita da mesma forma se o módulo for composto.

O Pequeno Teorema de Fermat pode ser utilizado de outra maneira para simplificar o cálculo de potências em alguns casos em que o módulo é composto. Neste caso, ele é auxiliado pelo Teorema Chinês do Resto. Os casos em que esta segunda abordagem irá funcionar são os casos em que o módulo n é um número composto que possui fatoração na forma $n = p_1 p_2 \dots p_t$, com $1 < p_1 < p_2 < \dots < p_t$. Em outras palavras, temos que todos os primos distintos que ocorrem na fatoração de n possuem multiplicidade 1.

Neste caso, se queremos calcular a forma reduzida de a^k módulo n, começamos calculando as formas reduzidas de a^k módulo p_i, $1 \leq i \leq t$, onde p_i são os primos distintos que ocorrem na fatoração de n. Para estes cálculos, podemos utilizar o Pequeno Teorema de Fermat da forma que explicamos acima para obter simplificações, já que os módulos p_i são primos. Vamos chamar de a_i a forma reduzida de a^k módulo p_i, $1 \leq i \leq t$. Podemos então construir o sistema de congruências

$$\begin{cases} x & \equiv & a_1 & \pmod{p_1} \\ x & \equiv & a_2 & \pmod{p_2} \\ \vdots & \vdots & \vdots & \vdots \\ x & \equiv & a_t & \pmod{p_t} \end{cases}$$

e resolvê-lo através do Algoritmo Chinês do Resto. O Teorema Chinês do Resto (Teorema 2.6) nos garante que ele possui uma única solução módulo o produto $p_1 p_2 \dots p_t$, que é exatamente o nosso módulo original n. Assim, a resolução do sistema nos dará a forma reduzida de a^k módulo n.

Exemplo 3.11. *Vamos calcular a forma reduzida de* $6^{4274623}$ *módulo* 3003*. O módulo* 3003 *não é primo, logo não podemos aplicar diretamente o Pequeno Teorema de Fermat como fizemos no exemplo anterior.*

Temos que $3003 = 3.7.11.13$*. Como todos os primos aparecem com multiplicidade* 1 *na fatoração de* 3003*, podemos utilizar a estratégia acima para realizar o cálculo. Para isso, precisamos calcular as formas reduzidas de* $6^{4274623}$ *módulo* 3*, módulo* 7*, módulo* 11 *e módulo* 13*.*

No caso do módulo 3*, não podemos aplicar o Pequeno Teorema de Fermat, pois* $6 \equiv 0 \pmod 3$*. Entretanto, isto não é um problema, já que toda potência* 6^k *de* 6 *também será congruente a* 0 *módulo* 3*. Assim,* $6^{4274623} \equiv 0 \pmod 3$*.*

Para o módulo 7*, como* $6 \not\equiv 0 \pmod 7$*, podemos utilizar o Pequeno Teorema de Fermat. Assim, temos que* $6^6 \equiv 1 \pmod 7$*. Realizando a divisão de* 4274623 *por* 6*, obtemos resto* 1*. Assim,* $6^{4274623} \equiv 6 \pmod 7$*.*

Vamos agora para o módulo 11*. Podemos, novamente, utilizar o Pequeno Teorema de Fermat, já que* $6 \not\equiv 0 \pmod{11}$*. Assim, temos que* $6^{10} \equiv 1 \pmod{11}$*. Realizando a divisão de* 4274623 *por* 10*, obtemos resto* 3*. Assim,* $6^{4274623} \equiv 6^3 \equiv 7 \pmod{11}$*.*

Finalmente, vamos para o módulo 13*. Temos que* $6 \not\equiv 0 \pmod{13}$*, assim, pelo Pequeno Teorema de Fermat,* $6^{12} \equiv 1 \pmod{13}$*. Realizando a divisão de* 4274623 *por* 12*, obtemos resto* 7*. Assim,* $6^{4274623} \equiv 6^7 \equiv 7 \pmod{13}$*.*

Obtemos então o sistema

$$\begin{cases} x \equiv 0 \pmod 3 \\ x \equiv 6 \pmod 7 \\ x \equiv 7 \pmod{11} \\ x \equiv 7 \pmod{13}. \end{cases}$$

Podemos resolvê-lo com o Algoritmo Chinês do Resto. Deixamos os detalhes da resolução deste sistema como exercício (Exercício 5), sendo 2295 *a solução encontrada. Assim,* $6^{4274623} \equiv 2295 \pmod{3003}$.

3.4 Teste de Miller-Rabin

Finalizando o capítulo, nesta seção apresentamos um refinamento do Teste de Fermat, apresentado na seção anterior. O teste que vamos apresentar é conhecido como Teste de Miller-Rabin, pois foi originalmente proposto por Gary Miller no artigo [17] em 1976 e depois melhorado por Michael Rabin no artigo [23] em 1980. Ele é um dos testes mais utilizados atualmente para a distinção de números primos e compostos. uma vez que ele não é dependente de métodos de fatoração (assim como o Teste de Fermat), possui um baixo custo computacional e consegue distinguir primos e compostos com uma alta margem de segurança e com a utilização de muito menos bases do que o Teste de Fermat.

O Teste de Miller-Rabin se preocupa apenas com números $n > 2$ que são *ímpares*, uma vez que qualquer número par neste intervalo será necessariamente composto, sem a necessidade de nenhuma verificação computacional.

Se n é ímpar, então $n-1$ é par. Podemos então escrever $n-1$ na forma $n-1 = 2^k q$, onde $k \geq 1$ e q é ímpar. Para obter os valores de k e q, basta apenas que façamos divisões sucessivas de $n-1$ e dos quocientes que forem obtidos por 2, até obtermos um quociente ímpar. Este último quociente será q e o número de divisões que houvermos feito será k.

Suponha que n seja primo. Se b for um inteiro no intervalo $2 \leq b \leq n-1$, então o Pequeno Teorema de Fermat nos diz que $b^{n-1} \equiv 1 \pmod n$. Temos então

$$1 \equiv b^{n-1} \equiv b^{2^k q} \pmod p.$$

Pela definição de congruência, isto significa que n divide $b^{2^k q} - 1$. Mas como $k \geq 1$, então $b^{2^k q} - 1$ é uma diferença de dois quadrados, sendo $b^{2^k q}$ o quadrado de $b^{2^{k-1} q}$ e 1 o quadrado de si mesmo. Podemos então escrever $b^{2^k q} - 1$ como

$$b^{2^k q} - 1 = (b^{2^{k-1} q} + 1)(b^{2^{k-1} q} - 1).$$

Pela Propriedade Fundamental dos Primos (Propriedade 3.3), como n é primo e n divide $b^{2^k q} - 1$, n deve dividir um dos termos do produto acima. Isto significa que n divide $b^{2^{k-1} q} + 1$ ou n divide $b^{2^{k-1} q} - 1$. No primeiro caso, temos que

$$b^{2^{k-1} q} \equiv -1 \pmod n.$$

Já no segundo caso, temos que

$$b^{2^{k-1} q} \equiv 1 \pmod n.$$

No segundo caso, temos uma congruência semelhante à nossa congruência original $b^{2^k q} \equiv 1 \pmod{n}$, apenas com a diminuição em uma unidade do expoente de 2, que passou de k para $k-1$.

Vamos então denotar por j o menor expoente maior ou igual a zero tal que $b^{2^j q} \equiv 1 \pmod{n}$. Sabemos que $j \leq k$, pelo exposto acima.

Se $j = 0$, obtemos

$$b^q \equiv 1 \pmod{n}. \tag{3.4.1}$$

Por outro lado, se $j \geq 1$, podemos repetir o raciocínio do parágrafo anterior. Podemos escrever

$$b^{2^j q} - 1 = (b^{2^{j-1} q} + 1)(b^{2^{j-1} q} - 1).$$

Da mesma forma que anteriormente, como n é primo e divide $b^{2^j q} - 1$, então n divide $b^{2^{j-1} q} + 1$ ou n divide $b^{2^{j-1} q} - 1$. Mas agora o segundo caso não é mais possível. Se n dividisse $b^{2^{j-1} q} - 1$, teríamos $b^{2^{j-1} q} \equiv 1 \pmod{n}$ o que é uma contradição com a nossa escolha de j como o *menor* expoente tal que $b^{2^j q} \equiv 1 \pmod{n}$. Logo, temos necessariamente que n divide $b^{2^{j-1} q} + 1$, o que significa que

$$b^{2^{j-1} q} \equiv -1 \pmod{n}. \tag{3.4.2}$$

Vamos recapitular os resultados que obtivemos nas Equações (3.4.1) e (3.4.2). Constatamos que, se n é primo e $n - 1 = 2^k q$, onde $k \geq 1$ e q é ímpar, então uma das seguintes coisas necessariamente acontece:

1. $b^q \equiv 1 \pmod{n}$ (Equação (3.4.1)) ou
2. $b^{2^i q} \equiv -1 \pmod{n}$, para algum i no intervalo $0 \leq i \leq k-1$ (Equação (3.4.2), onde $j - 1$ foi substituído por i).

Vamos observar agora este raciocínio no sentido inverso. Seja $n > 2$ um inteiro ímpar e $n - 1 = 2^k q$, onde $k \geq 1$ e q é ímpar. Suponha que tenhamos um valor de b no intervalo $2 \leq b \leq n-1$ tal que $b^q \not\equiv 1 \pmod{n}$ e, para todo i no intervalo $0 \leq i \leq k-1$, $b^{2^i q} \not\equiv -1 \pmod{n}$. Se n fosse primo, estes últimos resultados estariam contradizendo os resultados esperados expostos no parágrafo anterior. Logo, neste caso, n é composto.

A partir desta conclusão, descrevemos o Teste de Miller-Rabin. O teste consiste no cálculo de uma sequência de potências:

$$b^q, \quad b^{2q}, \quad b^{2^2 q}, \quad b^{2^3 q}, \quad \ldots, \quad b^{2^{k-1} q} \qquad \pmod{n}.$$

Se a primeira potência não for congruente nem a 1 nem a -1 módulo n e nenhuma outra potência for congruente a -1 módulo n, então n é composto. Neste caso, dizemos que b é uma *testemunha* de que n é composto. É importante observar que não precisamos calcular separadamente cada uma destas k potências. Olhando com atenção para a lista acima, vemos que cada potência é o quadrado da potência anterior. Desta forma, precisamos apenas calcular a primeira potência da lista, sendo as outras obtidas elevando a potência imediatamente anterior ao quadrado e tomando sua forma reduzida módulo n.

Entretanto, analogamente ao que ocorria no Teste de Fermat, se para um valor fixado de b no intervalo $2 \leq b \leq n-1$ temos que $b^q \equiv 1 \pmod{n}$ ou temos que $b^{2^i q} \equiv -1 \pmod{n}$, para algum i no intervalo $0 \leq i \leq k-1$, *não podemos* concluir que n é primo. Existem números n que são compostos mas satisfazem alguma destas congruências. Estes números são conhecidos como *pseudoprimos fortes* para a base b, já que, apesar de serem compostos, eles têm o mesmo comportamento que um primo teria com relação às congruências do Teste de Miller-Rabin.

Exemplo 3.12. *O número* $n = 3277$ *é composto, já que* $3277 = 29.113$. *Entretanto, ao realizarmos o Teste de Miller-Rabin com a base* $b = 2$, *ele não é capaz de determinar que este número é composto.*

Temos $n - 1 = 3276 = 2^2.819$, *logo* $k = 2$ *e* $q = 819$. *Assim, começamos calculando a forma reduzida da potência* 2^{819} *módulo* 3277. *Obtemos* $2^{819} \equiv 128 \pmod{3277}$. *Como* $128 \neq 1 \pmod{3277}$ *e* $128 \neq -1 \pmod{3277}$, *prosseguimos o teste com a próxima potência, que será* $2^{2.819}$. *Temos que* $2^{2.819} \equiv (2^{819})^2 \equiv 128^2 \equiv 3276 \equiv -1 \pmod{3277}$. *Como esta segunda potência é congruente a* -1 *módulo* 3277, *o Teste de Miller-Rabin com a base* $b = 2$ *não é capaz de determinar que* 3277 *é composto. Logo,* 3277 *é um pseudoprimo forte para a base* 2.

A existência de pseudoprimos fortes nos mostra que, analogamente ao que acontecia no Teste de Fermat, quando selecionamos uma base b e executamos o Teste de Miller-Rabin com esta base, o teste poderá nos dar dois resultados: o número n é composto, se $b^q \not\equiv 1 \pmod{n}$ e se $b^{2^i q} \not\equiv -1$, para todo $0 \leq i \leq k-1$, ou o resultado do teste é *inconclusivo*, caso contrário ($b^q \equiv 1 \pmod{n}$ ou $b^{2^i q} \equiv -1$, para *algum* $0 \leq i \leq k-1$). O teste é inconclusivo no segundo caso porque tanto os números primos quanto os pseudoprimos fortes para a base b (que são números compostos) satisfazem alguma destas últimas congruências. Assim, o Teste de Miller-Rabin com uma base b é capaz de distinguir alguns números compostos dos números primos, mas não todos. Como consequência, ao aplicar o Teste de Miller-Rabin com uma base, poderemos em alguns casos concluir com absoluta certeza que um número é composto, mas nunca poderemos concluir com esta mesma certeza que um número é primo.

Parece então que temos os mesmos problemas que tínhamos com o Teste de Fermat para distinguir números primos de compostos em alguns casos. Entretanto, o Teste de Miller-Rabin tem algumas vantagens bastante significativas na prática. Em primeiro lugar, existem menos *pseudoprimos fortes* para uma base b do que *pseudoprimos de Fermat* para esta mesma base. Isto ocorre porque todo pseudoprimo forte também é necessariamente um pseudoprimo de Fermat, mas a recíproca não é verdadeira. Deixamos a prova deste resultado como exercício (Exercício 7). Podemos concluir então que o Teste de Miller-Rabin consegue distinguir mais números compostos do que o Teste de Fermat.

Em segundo lugar, não existe nenhum análogo dos Números de Carmichael para o Teste de Miller-Rabin, isto é, números que mesmo sendo compostos, retornam resultado inconclusivo no teste para a grande maioria das bases. De fato, Rabin mostra em seu artigo [23] que, se um inteiro ímpar $n \geq 5$ é composto, então o Teste de Miller-Rabin conseguirá chegar ao resultado conclusivo com mais de $3/4$ das bases no intervalo $2 \leq b \leq n-1$. Isso nos indica que, se o número n que quisermos testar for composto, temos uma probabilidade maior do que $3/4$ de conseguir obter uma resposta conclusiva no Teste de Miller-Rabin ao escolhermos uma base b ao acaso e uma probabilidade menor do que $1/4$ de que o teste retorne resultado inconclusivo.

Podemos também olhar este resultado de outra maneira. Considere inicialmente que, se n for primo, o Teste de Miller-Rabin irá retornar resultado inconclusivo para todas as bases. Por outro lado, ao testarmos um número composto n com uma dada base b, o teste retornará um resultado inconclusivo em menos de $1/4$ dos casos. Então, se testarmos um número n que não sabemos se é primo ou composto e o resultado do teste for inconclusivo, a probabilidade de n ser composto é menor do que $1/4$ e de n ser primo é maior do que $3/4$.

Assim, podemos utilizar o Teste de Miller-Rabin com várias bases para testar se um dado n é primo com uma margem de confiança tão grande quanto se queira.

Ao realizarmos o teste sucessivamente com t bases escolhidas ao acaso, se o teste retornar o resultado de que n é composto com qualquer das bases, podemos encerrar o teste e temos certeza absoluta de que n é realmente composto. Por outro lado, se o teste retornar resultado inconclusivo para *todas* as t bases, a probabilidade de n não ser primo é menor do que $1/4^t$. Desta maneira, na prática, o Teste de Miller-Rabin nos fornece uma maneira eficiente de distinguir entre primos e compostos.

Como exemplo, se testarmos um número n com 10 bases e o teste retornar resultado inconclusivo para todas elas, a chance de n não ser realmente um número primo é menor do que $1/4^{10}$, ou seja, menor do que uma em um milhão. Se aumentarmos o número de bases para 15, a chance passa a ser menor do que uma em um bilhão.

É importante salientar que estas probabilidades levam em consideração que as t bases são escolhidas ao acaso. Na prática, costuma-se escolher como bases os t menores primos. Assim, neste caso, as probabilidades acima passam a ser apenas uma aproximação. Tendo isto em mente, o procedimento que costuma ser utilizado na prática é aumentarmos o número de bases usadas no teste conforme o número de algarismos de n aumenta.

Encerrando esta seção, apresentamos abaixo a descrição formal do Teste de Miller-Rabin.

Algoritmo 3.3: Teste de Miller-Rabin

Entrada: Um número inteiro ímpar $n \geq 3$ e um número inteiro $2 \leq b \leq n-1$.

Saída: "Número composto" ou "Resultado inconclusivo".

Instruções:

1. $k \leftarrow 0$, $q \leftarrow n-1$
2. Enquanto q for par, faça:
   2.1. $k \leftarrow k+1$
   2.2. $q \leftarrow q/2$
3. $t \leftarrow b^q \bmod n$
4. Se $t = 1$ ou $t = n-1$, então retorne "Resultado inconclusivo".
5. $i \leftarrow 1$
6. Enquanto $i < k$, faça:
   6.1. $t \leftarrow t^2 \bmod n$
   6.2. Se $t = n-1$, então retorne "Resultado inconclusivo".
   6.3. $i \leftarrow i+1$
7. Retorne "Número composto".

3.5 Exercícios

1. Escreva uma descrição formal do algoritmo para a fatoração completa de um inteiro $n \geq 2$ no formato dos outros algoritmos apresentados neste capítulo. O algoritmo deverá obter a fatoração conforme a descrição na Definição 3.4. Para isso, o algoritmo deverá retornar duas listas: uma lista de fatores primos distintos de n e uma lista das respectivas *multiplicidades* destes fatores na fatoração de n.

2. Determine a fatoração dos seguintes números:

   2.1. 952875

   2.2. 171990

   2.3. 1386000

3. Utilize o Crivo de Eratóstenes com as duas melhorias propostas para construir a lista de todos os primos menores do que 400.

4. Calcule a forma reduzida das seguintes potências:

   4.1. $3^{124755} \bmod 41$

   4.2. $5^{423744} \bmod 29$

   4.3. $6^{326532} \bmod 53$

   4.4. $2^{256364} \bmod 59$

5. Desenvolva os detalhes da resolução do sistema de congruências do Exemplo 3.11 através do Algoritmo Chinês do Resto.

6. Calcule a forma reduzida das seguintes potências:

   6.1. $4^{165234} \bmod 2121$

   6.2. $3^{734253} \bmod 1490$

   6.3. $6^{524187} \bmod 741$

   6.4. $10^{922531} \bmod 3445$

7. Mostre que todo pseudoprimo forte para uma base b é também um pseudoprimo de Fermat para esta mesma base. Em seguida, mostre que nem todo pseudoprimo de Fermat para uma base b é um pseudoprimo forte para esta base, mostrando que o número 341, que é um pseudoprimo de Fermat para a base 2, conforme o exemplo 3.9, não é um pseudoprimo forte para esta base.

8. Aplique o Teste de Miller-Rabin com a base 2 aos números abaixo. Caso o resultado retornado seja inconclusivo, aplique o teste novamente com a base 3.

   8.1. 10027

   8.2. 13823

   8.3. 15841

   8.4. 1373653

Capítulo 4

Grupos

Neste capítulo, apresentamos nosso último tópico entre os conceitos matemáticos básicos necessários para as nossas aplicações, com a discussão da estrutura algébrica conhecida como *grupo*.

Primeiramente, iremos apresentar a definição formal do que é um grupo, sendo que estaremos mais interessados em grupos finitos. Em seguida, descreveremos o nosso foco principal, que são grupos construídos a partir das relações de *congruência módulo* n. Tais grupos são denotados por $U(n)$ e seus elementos são os elementos de $\mathbb{Z}_n$ que possuem inverso multiplicativo módulo n. Iremos também estudar de maneira particular os grupos $U(p)$, onde p é um primo, já que os cálculos do método El Gamal são realizados em grupos deste tipo.

Ao longo do capítulo, apesar de nosso foco principal ser os grupos $U(n)$ e $U(p)$, também apresentaremos algumas noções gerais oriundas da teoria de grupos. Entre elas, discutiremos as noções de subgrupos e de grupos e subgrupos cíclicos. Esta última noção também se mostrará muito importante para a construção do método El Gamal. Além disso, completaremos nossa discussão apresentando e provando alguns resultados fundamentais sobre grupos, como o Teorema de Lagrange e o Teorema da Raiz Primitiva, entre outros. De maneira mais ou menos explícita, cada um destes resultados é importante na construção do método El Gamal.

Finalmente, apresentamos também neste capítulo a formulação do Problema do Logaritmo Discreto, um problema da teoria de grupos que está intimamente relacionado à segurança do método El Gamal.

4.1 Definições Básicas

A *teoria de grupos*, isto é, o estudo sistemático das estruturas algébricas conhecidas como *grupos*, foi iniciada pelo matemático francês Évariste Galois (1811-1832), que foi o primeiro a utilizar a nomenclatura "grupo". Entretanto, muitos resultados da teoria de grupos já haviam sido apresentados por matemáticos anteriores a Galois, embora em contextos menos gerais e sem utilizar a notação e a nomenclatura da teoria de grupos. Em particular, diversos resultados relacionados já haviam sido desenvolvidos por matemáticos como o suíço Leonhard Euler (1707-1783), o franco-italiano Joseph-Louis Lagrange (1736-1813) e o alemão Carl Friedrich Gauss (1777-1855).

O trabalho de Gauss, em especial, foi de extrema importância para a sedimentação de notações e resultados fundamentais. Em seu livro "*Disquisitiones*

Arithmeticae" [7], escrito em Latim em 1798, quando ele tinha 21 anos, e publicado em 1801, Gauss cataloga de maneira bastante completa resultados a respeito dos números inteiros, de aritmética modular e também resultados que acabaram incorporados à teoria de grupos, como o Teorema da Raiz Primitiva. Neste trabalho de catalogação, Gauss apresenta resultados de matemáticos anteriores (como Euclides, Fermat, Euler e Lagrange, entre outros), além de acrescentar resultados e provas de sua própria autoria. Em particular, a notação de aritmética modular que é utilizada até hoje, e que nós também utilizamos neste livro, é herdada das "*Disquisitiones Arithmeticae*".

Iniciamos o nosso estudo da teoria de grupos pela definição mais básica, isto é, a definição de um *grupo*.

Definição 4.1. *Um* grupo *é um par $\mathcal{G} = (G, *)$, onde G é um conjunto e $*$ é uma operação $* : G \times G \to G$ (uma operação que toma dois operandos em G e retorna como resultado um elemento de G) que satisfaz as seguintes propriedades:*

1. *Associatividade: para todo $a, b, c \in G$, $(a * b) * c = a * (b * c)$;*
2. *Existência de elemento neutro: existe um elemento $e \in G$ tal que, para todo $a \in G$, $a * e = e * a = a$;*
3. *Existência de inversos: para todo $a \in G$, existe um elemento $a^{-1} \in G$ tal que $a * a^{-1} = a^{-1} * a = e$, onde e é o elemento neutro.*

Quando estivermos falando de um grupo em um contexto genérico, sempre utilizaremos a notação e para o elemento neutro do grupo.

Podemos reparar também que não exigimos em um grupo que a operação $*$ seja comutativa no caso geral.

Definição 4.2. *Um grupo $\mathcal{G} = (G, *)$ é chamado de* grupo comutativo *ou* grupo abeliano *se, além das propriedades acima, ele satisfaz a seguinte propriedade:*

4. *Comutatividade: para todo $a, b \in G$, $a * b = b * a$.*

A nomenclatura *grupo abeliano* é uma homenagem ao matemático norueguês Niels Henrik Abel (1802-1829), outro pioneiro da teoria de grupos ao lado de Galois. Tragicamente, tanto Abel quanto Galois morreram com menos de 30 anos de idade.

Vamos analisar agora alguns exemplos de pares de conjuntos e operações que *não são* grupos e outros que *são* grupos.

Exemplo 4.1. *O par $\mathcal{G} = (\mathbb{N}, +)$ não é um grupo. A soma de naturais é associativa e 0 é o elemento neutro da operação, porém nem todos os naturais possuem um inverso aditivo que também seja natural. Por exemplo, o inverso aditivo de 2 é -2, mas $-2 \notin \mathbb{N}$.*

Exemplo 4.2. *O par $\mathcal{G} = (\mathbb{Z}, +)$ é um grupo. Neste caso, o inverso aditivo de qualquer inteiro também é um número inteiro.*

Exemplo 4.3. *O par $\mathcal{G} = (\mathbb{Z}_{ímpares}, +)$, onde $\mathbb{Z}_{ímpares}$ representa o conjunto dos inteiros ímpares, não é um grupo, pois neste caso a soma nem sequer atende a nossa definição de* operação*. A operação de um grupo deve ser necessariamente uma função que toma dois elementos do conjunto e retorna um valor que* também pertença *ao conjunto. Mas no nosso exemplo atual, em que estamos trabalhando com o conjunto dos inteiros ímpares, isto não acontece, já que a soma de dois inteiros ímpares é um inteiro par.*

Exemplo 4.4. *O par $\mathcal{G} = (\mathbb{Z}, \cdot)$, onde $\cdot$ representa a operação de produto, não é um grupo, pois nem todos os inteiros possuem um inverso multiplicativo que também seja inteiro.*

Exemplo 4.5. *O par $\mathcal{G} = (\mathbb{Q}^*, \cdot)$, onde $\mathbb{Q}^*$ representa o conjunto dos racionais diferentes de 0 e $\cdot$ representa a operação de produto é um grupo. Neste caso, o inverso multiplicativo de qualquer racional diferente de zero também é um racional diferente de zero.*

Exemplo 4.6. *Para um determinado n fixado, o par $\mathcal{G} = (\mathbb{M}_n, \cdot)$, onde $\mathbb{M}_n$ representa o conjunto das matrizes quadradas n por n e $\cdot$ representa o produto matricial, não é um grupo. A operação de produto matricial é associativa, a matriz identidade n por n é o elemento neutro da operação, porém nem toda matriz quadrada n por n possui uma matriz inversa. Por outro lado, se tomarmos o par $\mathcal{G}' = (\mathbb{M}_n^*, \cdot)$, onde $\mathbb{M}_n^*$ representa o conjunto das matrizes quadradas n por n com determinante diferente de zero, teremos um grupo, já que toda matriz quadrada n por n com determinante não-nulo possui uma inversa e esta inversa também é uma matriz quadrada n por n com determinante não-nulo. Repare que este é um exemplo de grupo* não-abeliano *já que o produto de matrizes não é uma operação comutativa.*

Após estes exemplos, apresentamos outras definições básicas muito importantes: a definição da *ordem* de um grupo e a definição de um *grupo finito*

Definição 4.3. *A* ordem *de um grupo $\mathcal{G} = (G, *)$ é definida como a cardinalidade do conjunto G.*

Definição 4.4. *Um grupo $\mathcal{G} = (G, *)$ é um* grupo finito *se a sua ordem é finita, isto é, se G é um conjunto finito.*

Encerramos esta seção com uma observação importante. Seja $\mathcal{G} = (G, *)$ um grupo e sejam $a, b \in G$. Como $\mathcal{G}$ é um grupo, então, por definição, a e b possuem inversos em G com relação à operação $*$. Sejam a^{-1} e b^{-1} estes inversos, respectivamente. Por outro lado, também pela definição, $a * b$ também é um elemento de G e, desta forma, também deve ter um inverso em G. Mas quem seria então o inverso de $a*b$ em G? O impulso inicial seria acreditar que é $a^{-1}*b^{-1}$. Entretanto, veremos que isto só é verdade se o grupo for abeliano.

Vamos analisar então quem seria o inverso de $a*b$. Ele será um elemento $u \in G$ tal que $(a*b)*u = e$. Por associatividade, podemos escrever esta igualdade como $a*(b*u) = e$. Podemos operar os dois lados desta última igualdade com a^{-1}, da seguinte forma: $a^{-1}*a*(b*u) = a^{-1}*e = a^{-1}$. Repare que colocamos a^{-1} à esquerda nas operações dos dois lados da igualdade. Como não estamos supondo que o grupo é abeliano e sim tratando do caso geral, operar com a^{-1} à esquerda ou à direita pode gerar resultados diferentes. Assim, não podemos colocar a^{-1} à esquerda em um dos lados da igualdade e à direita no outro. Como a^{-1} é o inverso de a, temos que $a^{-1}*a = e$. Assim, a última igualdade pode ser escrita como $b*u = a^{-1}$. Finalmente, podemos operar os dois lados desta última igualdade com b^{-1}, da seguinte forma: $b^{-1}*b*u = b^{-1}*a^{-1}$. Como $b^{-1}*b = e$, obtemos $u = b^{-1}*a^{-1}$.

Vemos então que o inverso de $a*b$ em G é $b^{-1}*a^{-1}$. Se o grupo for abeliano, então $b^{-1}*a^{-1} = a^{-1}*b^{-1}$, que foi o nosso palpite inicial acima. Entretanto, no caso geral, teremos $b^{-1}*a^{-1} \neq a^{-1}*b^{-1}$ e o nosso palpite inicial se mostra incorreto.

4.2 Os Grupos Finitos $U(n)$

Para as nossas aplicações, estamos especialmente interessados em grupos finitos que são obtidos a partir das relações de *congruência módulo n* que estudamos anteriormente.

Se consideramos o conjunto $\mathbb{Z}_n$ com a operação de produto módulo n, não obtemos um grupo. A operação de produto modular é associativa, $\overline{1}$ é o seu elemento neutro, porém nem todos os elementos de $\mathbb{Z}_n$ possuem inverso multiplicativo. De acordo com o Teorema da Inversão Modular (Teorema 2.5), um elemento $\overline{a} \in \mathbb{Z}_n$ possui inverso multiplicativo em $\mathbb{Z}_n$ se e somente se $\text{mdc}(a, n) = 1$.

Neste caso, para obtermos um grupo a partir dos elementos de $\mathbb{Z}_n$, devemos nos restringir ao subconjunto dos seus elementos que possuem inverso multiplicativo. Definimos então o conjunto $U(n)$ da seguinte forma:

$$U(n) = \{\overline{a} \in \mathbb{Z}_n : \text{mdc}(a, n) = 1\}.$$

Este é o conjunto de todos elementos de $\mathbb{Z}_n$ que possuem inverso multiplicativo.

Vamos mostrar que o par $(U(n), \cdot)$, onde $\cdot$ representa o produto módulo n, é efetivamente um grupo. Para isso, precisamos mostrar que o produto de dois elementos de $U(n)$ resulta em um elemento de $U(n)$ e que $\overline{1} \in U(n)$. As outras propriedades, a associatividade do produto e a existência de inversos, são imediatas.

Para todo inteiro $n \geq 2$, temos que $\text{mdc}(1, n) = 1$, logo $\overline{1} \in U(n)$. Vamos agora mostrar que, se $a \in U(n)$ e $b \in U(n)$, então $ab \in U(n)$. Se $a \in U(n)$, então existe um elemento a^{-1} que é o inverso de a e também pertence a $U(n)$. Analogamente, existe um elemento b^{-1} que é o inverso de b em $U(n)$. Como vimos na seção anterior, $b^{-1}a^{-1}$ (ou $a^{-1}b^{-1}$, já que o produto modular é comutativo) será o inverso de ab. Assim, como ab possui inverso, então $ab \in U(n)$.

Podemos concluir então que o par $(U(n), \cdot)$ é um grupo. Com algum abuso de notação, iremos escrever apenas $U(n)$ para denotar o grupo $(U(n), \cdot)$.

Em geral, os elementos de um conjunto que possuem inverso multiplicativo são chamados de *unidades* deste conjunto. Assim, denotamos o nosso conjunto por $U(n)$ pois ele é o conjunto das unidades do conjunto $\mathbb{Z}_n$.

Vamos agora analisar como calcular a ordem de um grupo $U(n)$, isto é, o número de elementos no grupo $U(n)$ em função do valor de $n \geq 2$. A função que nos dá o número de elementos de $U(n)$ a partir do valor de n é denotada por $\phi(n)$ e conhecida como *função ϕ de Euler*. Esta função foi descrita inicialmente por Euler, embora a notação com a letra ϕ tenha sido sugerida por Gauss em suas "*Disquisitiones Arithmeticae*" [7].

Em primeiro lugar, podemos perceber que, independentemente do valor de n, o elemento $\overline{0}$ nunca pertencerá a $U(n)$. Como o produto de qualquer elemento de $\mathbb{Z}_n$ por $\overline{0}$ resulta em $\overline{0}$, então $\overline{0}$ não possui inverso multiplicativo (um elemento que multiplicado por $\overline{0}$ resultaria em $\overline{1}$). Desta forma, para todo $n \geq 2$, temos que $\phi(n) \leq n - 1$.

Vamos começar analisando o caso de $U(p)$, quando p é primo. Já sabemos que $\overline{0} \notin U(p)$. Por outro lado, para qualquer número a no intervalo $1 \leq a \leq p-1$, temos que $\text{mdc}(a, p) = 1$. Isto ocorre porque p é primo e, portanto, não possui divisores próprios. Assim,

$$U(p) = \mathbb{Z}_p - \{\overline{0}\},$$

o que significa que

$$\phi(p) = p - 1.$$

Em oposição ao caso acima, se n for composto, então n possui algum fator próprio $1 < f \leq n-1$. Assim, $\text{mdc}(f, n) > 1$, o que significa que, neste caso, além de $\overline{0} \notin U(n)$, temos também que $\overline{f} \notin U(n)$. Logo, se n é composto, temos que $\phi(n) < n-1$. A partir disto e do caso acima, podemos concluir a propriedade abaixo.

Propriedade 4.1. *$\phi(n) = n-1$ se e somente se n é primo.*

Vamos agora considerar o caso em que $n = p^k$, onde p é primo e $k \geq 1$. Queremos calcular quantos são os números a no intervalo $0 \leq a < p^k$ tais que $\text{mdc}(a, p^k) = 1$. Temos que $\text{mdc}(a, p^k) = 1$ se e somente se p não divide a. Temos que contar então quantos são os números no intervalo $0 \leq a < p^k$ que não são divisíveis por p. Neste caso, é mais simples fazer a contagem inversa: quantos são os números que são divisíveis por p. Se a é divisível por p, então $a = pa'$, para algum $a' \in \mathbb{Z}$. Por outro lado, como $0 \leq a < p^k$, temos que $0 \leq pa' < p^k$, o que significa que $0 \leq a' < p^{k-1}$, já que $p \neq 0$. Como existem p^{k-1} números no intervalo $0 \leq a' < p^{k-1}$, então existem p^{k-1} números no intervalo $0 \leq a < p^k$ que são divisíveis por p. Logo, existem $p^k - p^{k-1} = p^{k-1}(p-1)$ números no intervalo $0 \leq a < p^k$ que não são divisíveis por p. Estes são os números tais que $\text{mdc}(a, p^k) = 1$. Portanto,

$$\phi(p^k) = p^{k-1}(p-1).$$

Podemos reparar que o valor de $\phi(p)$ que calculamos anteriormente é um caso particular da fórmula acima quando $k = 1$.

Vamos agora utilizar o resultado acima para determinar o valor de $\phi(n)$ para qualquer valor de $n \geq 2$. Para isso, precisaremos ainda de alguns resultados auxiliares.

Lema 4.1. *Sejam $m, n \geq 2$ tais que $mdc(m, n) = 1$ e seja $\overline{x} \in \mathbb{Z}_{mn}$. Vamos denotar por a a forma reduzida de x módulo m e por b a forma reduzida de x módulo n. Então, $\overline{x} \in U(mn)$ se e somente se $\overline{a} \in U(m)$ e $\overline{b} \in U(n)$.*

Demonstração: ($\Rightarrow$) Se $\overline{x} \in U(mn)$, então existe $\overline{x^{-1}} \in U(mn)$ tal que $xx^{-1} \equiv 1 \pmod{mn}$. Logo, mn divide $xx^{-1} - 1$. Como m divide mn, então m divide $xx^{-1} - 1$, o que significa que $xx^{-1} \equiv 1 \pmod{m}$. Como $x \equiv a \pmod{m}$, então $ax^{-1} \equiv 1 \pmod{m}$, o que significa que $\overline{a}$ possui inverso multiplicativo módulo m, isto é, $\overline{a} \in U(m)$. Um raciocínio inteiramente análogo nos mostra que $\overline{b} \in U(n)$.

($\Leftarrow$) Se $\overline{a} \in U(m)$, então existe $\overline{a^{-1}} \in U(m)$ tal que $aa^{-1} \equiv 1 \pmod{m}$. Analogamente, se $\overline{b} \in U(n)$, então existe $\overline{b^{-1}} \in U(n)$ tal que $bb^{-1} \equiv 1 \pmod{n}$. Como $\text{mdc}(m, n) = 1$, podemos utilizar o Algoritmo Chinês do Resto para calcular a solução única módulo mn do sistema

$$\begin{cases} y \equiv a^{-1} & \pmod{m} \\ y \equiv b^{-1} & \pmod{n}. \end{cases}$$

Vamos mostrar agora que a solução única y deste sistema é o inverso multiplicativo de x módulo mn. Como $x \equiv a \pmod{m}$ e $y \equiv a^{-1} \pmod{m}$, então $xy \equiv aa^{-1} \equiv 1 \pmod{m}$. Assim, $xy - 1$ é divisível por m. Um raciocínio inteiramente análogo nos mostra que $xy - 1$ é divisível por n. Como $\text{mdc}(m, n) = 1$, se m e n dividem $xy - 1$, então mn divide $xy - 1$, de acordo com o Lema 2.2. Assim, $xy \equiv 1 \pmod{mn}$, o que significa que x possui inverso multiplicativo módulo mn. Logo, $x \in U(mn)$. ■

Teorema 4.1. *Se $m, n \geq 2$ e $mdc(m, n) = 1$, então $\phi(mn) = \phi(m)\phi(n)$.*

Demonstração: De acordo com o Lema 4.1, se $\text{mdc}(m,n) = 1$, cada elemento de $U(mn)$ possui uma forma reduzida módulo m que está em $U(m)$ e uma forma reduzida módulo n que está em $U(n)$. Assim, podemos contabilizar o número de elementos de $U(mn)$ contando o número de possíveis pares que podemos formar com o primeiro elemento do par vindo de $U(m)$ e o segundo elemento do par vindo de $U(n)$. O número de possíveis pares será então $\phi(m)\phi(n)$, o que nos permite concluir que $\phi(mn) = \phi(m)\phi(n)$. ■

Dado um inteiro $n \geq 2$, podemos então calcular o valor de $\phi(n)$ a partir da fatoração de n. Se $n = p_1^{e_1} p_2^{e_2} \dots p_t^{e_t}$, então

$$\phi(n) = \phi(p_1^{e_1} p_2^{e_2} \dots p_t^{e_t}) = \phi(p_1^{e_1})\phi(p_2^{e_2}) \dots \phi(p_t^{e_t}).$$

Na igualdade acima, aplicamos o Teorema 4.1, uma vez que o máximo divisor comum entre potências de primos distintos é sempre 1. Podemos agora aplicar a fórmula que deduzimos anteriormente para $\phi(p^k)$, obtendo

$$\phi(n) = p_1^{e_1-1} p_2^{e_2-2} \dots p_t^{e_t-1}(p_1 - 1)(p_2 - 1) \dots (p_t - 1).$$

Finalmente, é importante salientar que não é conhecida nenhuma maneira de calcular a função $\phi(n)$ no caso geral sem o conhecimento ou a obtenção da fatoração de n.

Exemplo 4.7. *Vamos calcular* $\phi(120)$*. A fatoração de* 120 *é* $120 = 2^3.3.5$*. Logo,*

$$\phi(120) = 2^{3-1}.3^{1-1}.5^{1-1}.(2-1).(3-1).(5-1) = 4.1.1.1.2.4 = 32.$$

Isto significa que existem 32 *elementos com inverso multiplicativo em* $\mathbb{Z}_{120}$*.*

4.3 Subgrupos

Nesta seção, apresentamos a noção de *subgrupo*, que se relaciona com a noção de *grupo* de forma análoga à relação entre subconjunto e conjunto. Entretanto, como um grupo possui uma estrutura interna dada por propriedades de sua operação, esta estrutura também precisa ser considerada na definição de subgrupo.

Definição 4.5. *Se* $\mathcal{G} = (G, *)$ *é um grupo, dizemos que* $\mathcal{H} = (H, *)$ *é um subgrupo de* $\mathcal{G}$ *se as seguintes propriedades são satisfeitas:*

1. $H \subseteq G$ *(*H *é um subconjunto de* G*);*
2. *Para todo* $h, j \in H$*, temos que* $h * j \in H$*;*
3. $e \in H$*, onde* e *é o elemento neutro da operação* $*$*;*
4. *Para todo* $h \in H$*, existe um elemento* $h^{-1} \in H$ *tal que* $h * h^{-1} = h^{-1} * h = e$*.*

A partir desta definição, podemos notar que um subgrupo não é obtido a partir de qualquer subconjunto do grupo original. Para que $\mathcal{H} = (H, *)$ seja um subgrupo de $\mathcal{G} = (G, *)$, ele deve satisfazer todas as propriedades de um grupo quando olhado isoladamente. Desta forma, para que $\mathcal{H} = (H, *)$ seja um subgrupo de $\mathcal{G} = (G, *)$, ele deve conter o elemento neutro da operação, todos os elementos de H devem possuir inversos que também pertençam a H e a operação $*$ deve estar bem definida quando restrita a H, isto é, quando operamos dois elementos de H, o resultado da operação também precisa pertencer a H.

Exemplo 4.8. *Dado o grupo* $U(18) = \{\overline{1}, \overline{5}, \overline{7}, \overline{11}, \overline{13}, \overline{17}\}$. *O conjunto* $H = \{\overline{1}, \overline{7}, \overline{13}\}$ *com a operação de produto módulo* 18 *forma um subgrupo de* $U(18)$*, uma vez que* H *é subconjunto de* $U(18)$*,* $\overline{1} \in H$*, o produto de quaisquer dois elementos de* H *também resulta em um elemento de* H *e o inverso de todo elemento de* H *também pertence a* H *(o inverso de* $\overline{1}$ *é ele próprio e* $\overline{7}$ *e* $\overline{13}$ *são inversos um do outro).*

Exemplo 4.9. *Se considerarmos o conjunto* $H' = \{\overline{5}, \overline{11}\}$ *não precisamos fazer nenhum cálculo para concluir que ele não forma um subgrupo de* $U(18)$*, uma vez que o elemento neutro de* $U(18)$ *(*$\overline{1}$*) não pertence a* H'*. Todo subgrupo de um grupo deve necessariamente conter o elemento neutro do grupo.*

Exemplo 4.10. *Se considerarmos o conjunto* $H'' = \{\overline{1}, \overline{7}, \overline{11}\}$*, agora o elemento neutro de* $U(18)$ *faz parte do conjunto, mas ele também não forma um grupo. Se operarmos os elementos* $\overline{7}$ *e* $\overline{11}$ *de* H''*, obtemos* $7.11 \equiv 77 \equiv 5 \pmod{18}$*, mas* $\overline{5} \notin H''$*. Assim, a operação entre dois elementos de* H'' *resultou em um elemento fora de* H''*, o que não pode ocorrer em um subgrupo.*

Pela definição de subgrupo, vemos que qualquer grupo $\mathcal{G} = (G, *)$ possuirá ao menos dois subgrupos: ele próprio e $(\{e\}, *)$, o subgrupo que contém como elemento apenas o elemento neutro da operação $*$. Para excluir estes casos triviais que ocorrem em qualquer grupo, definimos a noção de *subgrupo próprio*.

Definição 4.6. *Dado um grupo* $\mathcal{G} = (G, *)$ *e um subgrupo* $\mathcal{H} = (H, *)$*, dizemos que* $\mathcal{H}$ *é um* subgrupo próprio *de* $\mathcal{G}$ *se* $H \neq G$ *e* $H \neq \{e\}$*, onde* e *é o elemento neutro da operação* $*$*.*

Vamos agora apresentar um teorema central que rege a inter-relação entre um grupo e seus subgrupos. Este teorema foi provado inicialmente por Lagrange para um caso particular, em um momento anterior à criação da teoria de grupos por Galois. Posteriormente, ele foi generalizado para *grupos finitos*, mas o nome de Lagrange foi mantido como autor do teorema.

Este teorema nos mostra que a relação entre grupos e subgrupos é extremamente mais rígida do que a relação entre subconjuntos e conjuntos.

Teorema 4.2 (Teorema de Lagrange)**.** *Seja* $\mathcal{G} = (G, *)$ *um* grupo finito *e* $\mathcal{H} = (H, *)$ *um subgrupo de* $\mathcal{G}$*. Então, a ordem de* $\mathcal{H}$ *divide a ordem de* $\mathcal{G}$*.*

Demonstração: A ordem de $\mathcal{H}$ e a ordem de $\mathcal{G}$ são, respectivamente, a quantidade de elementos em H, denotada por $|H|$, e a quantidade de elementos em G, denotada por $|G|$.

Se $|H| = |G|$, o teorema é claramente verdadeiro. Vamos supor então que $|H| < |G|$. Assim, existe ao menos um elemento $g_1 \in G - H$.

Seja $H = \{h_1, h_2, \ldots, h_t\}$. Vamos construir o conjunto g_1H da seguinte forma:

$$g_1H = \{g_1 * h_1, g_1 * h_2, \ldots, g_1 * h_t\}.$$

É importante notar que $|g_1H| = |H|$, pois, para $i \neq j$, temos $g_1 * h_i \neq g_1 * h_j$. De fato, se tivéssemos $g_1 * h_i = g_1 * h_j$, teríamos $g_1^{-1} * g_1 * h_i = g_1^{-1} * g_1 * h_j$, obtendo $h_i = h_j$, o que seria uma contradição com a hipótese de que $i \neq j$.

Se $G = H \cup g_1H$, encerramos a construção neste ponto. Caso contrário, existe ao menos um elemento $g_2 \in G - (H \cup g_1H)$. Construímos então o conjunto g_2H de maneira análoga à construção de g_1H. Da mesma forma que no caso de g_1H, temos $|g_2H| = |H|$. Se $G = H \cup g_1H \cup g_2H$, encerramos a construção neste ponto. Caso

contrário, continuamos a construção com um elemento $g_3 \in G - (H \cup g_1H \cup g_2H)$ e assim por diante, até finalmente obtermos

$$G = g_0H \cup g_1H \cup g_2H \cup \ldots \cup g_sH,$$

onde $g_0 = e$, de forma que $g_0H = H$. Repare que, como G é finito, esta igualdade é eventualmente obtida.

Temos então

$$|G| = |g_0H \cup g_1H \cup g_2H \cup \ldots \cup g_sH| \leq$$
$$\leq |g_0H| + |g_1H| + |g_2H| + \ldots + |g_sH| = (s+1)|H|.$$

Só nos resta agora mostrar que todos os conjuntos na união $g_0H \cup g_1H \cup g_2H \cup \ldots \cup g_sH$ são disjuntos, de forma que temos efetivamente a igualdade

$$|G| = (s+1)|H|,$$

o que significa que a ordem de $\mathcal{H}$ divide a ordem de $\mathcal{G}$, provando o teorema.

Suponha, por contradição, que os conjuntos não são disjuntos. Assim, existem $s \geq i > j \geq 0$ e $1 \leq k, l \leq t$ tais que $g_i * h_k = g_j * h_l$. Mas então temos $g_i = g_j * h_l * h_k^{-1}$. Como $h_k, h_l \in H$, temos que $h_l * h_k^{-1} \in H$. Então, pela igualdade $g_i = g_j * h_l * h_k^{-1}$, $g_i \in g_jH$. Mas isto é uma contradição com a hipótese de que $i > j$ e com a restrição de g_i, que deve ser escolhido no conjunto $G - (H \cup g_1H \cup \ldots \cup g_{i-1}H)$. Desta forma os conjuntos são realmente disjuntos e o teorema está provado. ■

O Teorema nos diz que a ordem de um subgrupo será um divisor da ordem do grupo. Se a ordem do grupo é n, 1 e n são divisores de n. Entretanto, como o elemento neutro deve obrigatoriamente pertencer ao subgrupo, o único subgrupo possível de ordem 1 é o subgrupo que contém apenas o elemento neutro. Por outro lado, o único subgrupo de ordem n possível é o próprio grupo. Desta forma, a ordem de um *subgrupo próprio* de um grupo será um *divisor próprio* da ordem do grupo.

Exemplo 4.11. *Vamos considerar o grupo* $U(20) = \{\overline{1}, \overline{3}, \overline{7}, \overline{9}, \overline{11}, \overline{13}, \overline{17}, \overline{19}\}$. *Vemos que este grupo tem ordem* 8 *(o que pode ser confirmado pelo cálculo de* $\phi(20) = 8$*. Se tomarmos o subconjunto* $H = \{\overline{1}, \overline{7}, \overline{13}, \overline{17}, \overline{19}\}$*, não precisamos fazer nenhum cálculo com elementos de* H *para determinar se* H *vai ou não formar um subgrupo de* $U(20)$*. Pelo Teorema de Lagrange, a ordem de qualquer subgrupo de* $U(20)$ *será um divisor da ordem de* $U(20)$*. Como a ordem de* $U(20)$ *é 8, seus subgrupos terão ordem* $1, 2, 4$ *ou* 8*, sendo que os subgrupos próprios terão ordem* 2 *ou* 4*. Como* H *possui* 5 *elementos, ele não pode formar um subgrupo de* $U(20)$*.*

O Teorema de Lagrange nos permite ainda fazer uma conclusão interessante sobre grupos de ordem prima (grupos cuja quantidade de elementos é um número primo). Um número primo não possui divisores próprios, logo, a partir da observação que fizemos acima, um grupo de ordem prima não possui nenhum subgrupo próprio.

4.4 Grupos e Subgrupos Cíclicos

Nesta seção, vamos estudar outro conceito básico fundamental da teoria de grupos: o conceito de grupos e subgrupos *cíclicos*.

Seja $\mathcal{G} = (G, *)$ um *grupo finito* e $a \in G$ um de seus elementos. Vamos utilizar a seguinte notação:

$$a^k = a * a * \ldots * a \quad (k \text{ vezes}).$$

Chamamos a^k de *k-ésima potência* de a em $\mathcal{G}$. Definimos também que $a^0 = e$, onde e é o elemento neutro de $\mathcal{G}$.

Vamos calcular o conjunto H de todas as potências de a, com $k \geq 0$:

$$H = \{e, a, a^2, a^3, \ldots\}.$$

Em um primeiro olhar, o conjunto H parece infinito, já que os expoentes das potências podem crescer de forma ilimitada. Entretanto, como $a \in G$ e a operação de dois elementos de G sempre produz um resultado que também pertence a G, temos que $a^k \in G$, para todo $k \geq 0$. Assim, H é um subconjunto de G. Mas como $\mathcal{G}$ é um grupo finito, G é um conjunto finito. Desta forma, H também é necessariamente um conjunto finito.

Concluímos então que a listagem que fizemos acima dos elementos de H deve conter (muitas) redundâncias, uma vez que H só contém uma quantidade finita de elementos. Isto significa que existem elementos a^l e a^m, com $l > m$ tais que $a^l = a^m$ em $\mathcal{G}$. É simples percebermos que se a^{-1} é o inverso de a em $\mathcal{G}$, então $(a^{-1})^m$ é o inverso de a^m. Podemos escrever a igualdade $a^l = a^m$ como $a^{l-m} * a^m = a^m$. Operando então esta igualdade em ambos os lados com $(a^{-1})^m$ à direita, obtemos $a^{l-m} * a^m * (a^{-1})^m = a^m * (a^{-1})^m$, o que nos permite concluir que $a^{l-m} = e$ em $\mathcal{G}$.

Assim, para que H seja um conjunto finito, algumas potências de a deverão ser iguais ao elemento neutro.

Definição 4.7. *Seja $\mathcal{G} = (G, *)$ um grupo finito e $a \in G$ um de seus elementos. Definimos a* ordem *de a em $\mathcal{G}$ como o menor* inteiro positivo *k tal que $a^k = e$ em $\mathcal{G}$.*

Repare que $a^0 = e$, mas na definição de ordem estamos considerando apenas expoentes *positivos*.

Já havíamos definido na seção inicial deste capítulo um outro conceito intitulado *ordem*: a *ordem de um grupo*. Parece então uma péssima ideia nomear um segundo conceito, relativo agora a um elemento de um grupo, com o mesmo termo. Entretanto, conforme veremos mais adiante, a noção de ordem de um elemento está intimamente relacionada à noção de ordem de um grupo.

Seja k a ordem de a. Temos então que $a^k = e$. Operando com a dos dois lados da igualdade, obtemos $a^{k+1} = a$, $a^{k+2} = a^2$, e assim por diante. Desta forma, se k é a ordem de a, temos que $a^l = a^r$, onde r é o resto da divisão de l por k. Podemos concluir então que o conjunto H pode ser escrito como

$$H = \{e, a, a^2, \ldots, a^{k-1}\}.$$

Vamos mostrar que todos estes elementos de H são distintos. Suponha que $a^l = a^m$, com $0 \leq m \leq l \leq k-1$. Então, operando com $(a^{-1})^m$ dos dois lados da igualdade, obtemos $a^{l-m} = e$. Como m e l são menores ou iguais a $k-1$, temos que $0 \leq l - m \leq k - 1$. Entretanto, k é o menor inteiro positivo tal que $a^k = e$. Logo, se $l - m < k$ e $a^{l-m} = e$, a única possibilidade é $l - m = 0$, o que significa que $l = m$. Assim, todos os elementos na descrição de H acima são distintos. Podemos concluir então que, se a ordem de a é k, o conjunto H das potências de a terá k elementos.

Vemos que $e \in H$. Além disso, para uma potência a^l, com $1 \leq l \leq k-1$, temos que $a^l * a^{k-l} = a^k = e$, logo a^{k-l} é o inverso de a^l. Desta maneira, todo elemento de H possui inverso também em H. Podemos concluir então que $(H, *)$ é um *grupo finito* com ordem k. Além disso, como H é um subconjunto de G, $(H, *)$ é um subgrupo de $\mathcal{G}$.

Dizemos que $(H, *)$ é o *grupo cíclico* gerado por a ou, alternativamente, o *subgrupo cíclico* de $\mathcal{G}$ gerado por a. Dizemos que a é um *gerador* ou uma *raiz primitiva* do grupo $(H, *)$.

Vemos agora a relação entre a noção de *ordem* de um elemento a de um grupo (menor inteiro positivo k tal que $a^k = e$) e a noção de *ordem* de um grupo (número de elementos do grupo). Um elemento a de ordem k gera um grupo cíclico de ordem k (um grupo com k elementos).

A partir dos conceitos de *grupo cíclico* e de *gerador* ou *raiz primitiva*, podemos definir um problema central para as aplicações de grupos em criptografia: o *Problema do Logaritmo Discreto*.

Definição 4.8 (Problema do Logaritmo Discreto)**.** *Definimos o Problema do Logaritmo Discreto, abreviado como PLD, da seguinte forma: dados um grupo finito cíclico $\mathcal{G} = (G, *)$ de ordem n, um gerador g de $\mathcal{G}$ e um elemento $h \in G$, queremos determinar o valor de x no intervalo $0 \leq x < n$ tal que $g^x = h$ em $\mathcal{G}$.*

Repare que, se g é um gerador de $\mathcal{G}$ e este grupo possui ordem n, então todos os elementos deste grupo podem ser escritos como uma potência de g com expoente maior ou igual a zero e menor do que n. O objetivo do Problema do Logaritmo Discreto é justamente determinar quem é este expoente para um dado elemento h de $\mathcal{G}$.

Conforme veremos no próximo capítulo, a segurança do método El Gamal está baseada justamente na dificuldade computacional da resolução do Problema do Logaritmo Discreto no caso geral. Vários algoritmos para a resolução deste problema serão estudados no último capítulo deste livro. Eles se mostram eficientes para a obtenção da resposta em alguns casos particulares, mas nenhum deles é suficientemente eficiente no caso geral.

Vamos prosseguir analisando mais propriedades dos grupos cíclicos. Se $\mathcal{G} = (G, *)$ é um grupo finito e $a \in G$, então o grupo cíclico gerado por a é um subgrupo de $\mathcal{G}$. Logo, o Teorema de Lagrange nos diz que a ordem deste subgrupo deve dividir a ordem de $\mathcal{G}$. Entretanto, a ordem deste subgrupo é justamente a ordem de a. A partir deste raciocínio, obtemos o seguinte corolário do Teorema de Lagrange.

Corolário 4.1. *Seja $\mathcal{G}$ um grupo finito de ordem n e seja $a \in \mathcal{G}$. Então a ordem de a divide n.*

Demonstração: A ordem do subgrupo de $\mathcal{G}$ gerado por a é igual à ordem de a. Pelo Teorema de Lagrange (Teorema 4.2), a ordem deste subgrupo divide n. Logo, a ordem de a divide n. ■

Assim, sabemos que se $\mathcal{G}$ possui ordem n e l não divide n, l certamente não será o menor inteiro positivo tal que $a^l = e$.

Conforme vimos anteriormente, um grupo de ordem prima não possui subgrupos próprios. Assim, os únicos subgrupos de um grupo de ordem prima são o próprio grupo e o grupo que contém apenas o elemento neutro. Seja então $\mathcal{G} = (G, *)$ um grupo de ordem prima e seja $a \in G$ tal que $a \neq e$. Vamos considerar o subgrupo de $\mathcal{G}$ gerado por a. Como $a^1 = a \neq e$, o subgrupo cíclico gerado por a não tem ordem 1. Por outro lado, se o subgrupo cíclico gerado por a não é o grupo que contém apenas o elemento neutro, então a única opção restante entre os subgrupos de $\mathcal{G}$ é que ele seja o próprio grupo $\mathcal{G}$. Desta forma, $\mathcal{G}$ é um grupo cíclico gerado por a. Podemos concluir então que, se $\mathcal{G} = (G, *)$ é um grupo de ordem prima, então ele é um grupo cíclico e todo elemento $a \in G$ tal que $a \neq e$ é um gerador de $\mathcal{G}$.

É importante observarmos que a recíproca do raciocínio acima é falsa. Todo grupo de ordem prima é cíclico, mas nem todo grupo cíclico possui ordem prima. Em particular, como veremos mais adiante no Teorema da Raiz Primitiva, se p é primo, então o grupo $U(p)$ é cíclico. Mas a ordem de $U(p)$ é $\phi(p) = p - 1$, que é um número composto para todo primo $p > 3$.

Exemplo 4.12. *Vamos calcular todos os subgrupos cíclicos de* $U(20) = \{\overline{1}, \overline{3}, \overline{7}, \overline{9}, \overline{11}, \overline{13}, \overline{17}, \overline{19}\}$.

1. *Como* $\overline{1}^1 = \overline{1}$, *o subgrupo gerado por* $\overline{1}$ *é* $H_1 = \{\overline{1}\}$.

2. *Temos* $\overline{3}^1 = \overline{3}$, $\overline{3}^2 = \overline{9}$, $\overline{3}^3 = \overline{7}$ *e* $\overline{3}^4 = \overline{1}$. *Logo, a ordem de* $\overline{3}$ *é* 4 *e o subgrupo gerado por* $\overline{3}$ *é* $H_2 = \{\overline{1}, \overline{3}, \overline{7}, \overline{9}\}$.

3. *Temos* $\overline{7}^1 = \overline{7}$, $\overline{7}^2 = \overline{9}$, $\overline{7}^3 = \overline{3}$ *e* $\overline{7}^4 = \overline{1}$. *Logo, a ordem de* $\overline{7}$ *é* 4 *e o subgrupo gerado por* $\overline{7}$ *é* $H_2 = \{\overline{1}, \overline{3}, \overline{7}, \overline{9}\}$. *Vemos então que* $\overline{3}$ *e* $\overline{7}$ *são geradores do mesmo subgrupo.*

4. *Temos* $\overline{9}^1 = \overline{9}$ *e* $\overline{9}^2 = \overline{1}$. *Logo, a ordem de* $\overline{9}$ *é* 2 *e o subgrupo gerado por* $\overline{9}$ *é* $H_3 = \{\overline{1}, \overline{9}\}$.

5. *Temos* $\overline{11}^1 = \overline{11}$ *e* $\overline{11}^2 = \overline{1}$. *Logo, a ordem de* $\overline{11}$ *é* 2 *e o subgrupo gerado por* $\overline{11}$ *é* $H_4 = \{\overline{1}, \overline{11}\}$.

6. *Temos* $\overline{13}^1 = \overline{13}$, $\overline{13}^2 = \overline{9}$, $\overline{13}^3 = \overline{17}$ *e* $\overline{13}^4 = \overline{1}$. *Logo, a ordem de* $\overline{13}$ *é* 4 *e o subgrupo gerado por* $\overline{13}$ *é* $H_5 = \{\overline{1}, \overline{9}, \overline{13}, \overline{17}\}$.

7. *Temos* $\overline{17}^1 = \overline{17}$, $\overline{17}^2 = \overline{9}$, $\overline{17}^3 = \overline{13}$ *e* $\overline{17}^4 = \overline{1}$. *Logo, a ordem de* $\overline{17}$ *é* 4 *e o subgrupo gerado por* $\overline{17}$ *é* $H_5 = \{\overline{1}, \overline{9}, \overline{13}, \overline{17}\}$. *Vemos então que* $\overline{13}$ *e* $\overline{17}$ *são geradores do mesmo subgrupo.*

8. *Temos* $\overline{19}^1 = \overline{19}$ *e* $\overline{19}^2 = \overline{1}$. *Logo, a ordem de* $\overline{19}$ *é* 2 *e o subgrupo gerado por* $\overline{19}$ *é* $H_6 = \{\overline{1}, \overline{19}\}$.

Como nenhum elemento de $U(20)$ *possui a mesma ordem de* $U(20)$, *que é* $\phi(20) = 8$, *então* $U(20)$ não é *um grupo cíclico.*

No exemplo acima, calculamos metodicamente todos os subgrupos cíclicos de um dado grupo. É importante, no entanto, observarmos que, no caso geral, este método não irá produzir uma lista exaustiva de todos os subgrupos de um grupo. Isto ocorre porque nem todo subgrupo de um grupo precisa ser cíclico. Isto é, além dos subgrupos cíclicos do grupo, que podemos calcular como no exemplo acima, o grupo pode possuir também alguns subgrupos não-cíclicos, isto é, subgrupos que atendem a todas as propriedades na definição de subgrupo, mas cujos elementos não podem ser todos escritos como potências de um elemento gerador.

Exemplo 4.13. *Vamos ilustrar a existência de subgrupos não-cíclicos a partir de um exemplo com o grupo* $U(16) = \{\overline{1}, \overline{3}, \overline{5}, \overline{7}, \overline{9}, \overline{11}, \overline{13}, \overline{15}\}$. *O conjunto* $H = \{\overline{1}, \overline{7}, \overline{9}, \overline{15}\}$ *com a operação de produto módulo* 16 *forma um subgrupo de* $U(16)$. *De fato,* H *é um subconjunto de* $U(16)$, *o elemento neutro* $\overline{1} \in H$, *o produto de qualquer par de elementos de* H *produz um elemento de* H *e todo elemento de* H *possui um inverso que também pertence a* H. *Porém,* H não é *um grupo cíclico. Temos que a ordem de* $\overline{1}$ *é* 1 *e as ordens de* $\overline{7}, \overline{9}$ *e* $\overline{15}$ *são todas iguais a* 2, *já que*

$\overline{7}^2 = \overline{9}^2 = \overline{15}^2 = \overline{1}$. *Como a ordem de H é 4 e nenhum de seus elementos tem ordem 4, nenhum deles é um gerador de H.*

Repare que a ordem de H é um número composto. Isto não é uma coincidência, uma vez que vimos anteriormente que todo grupo de ordem prima deve necessariamente ser cíclico.

Apresentamos agora um lema central a respeito de grupos finitos. Este lema será muito importante como um componente na prova de diversos resultados sobre grupos.

Lema 4.2 (Lema Chave)**.** *Seja $\mathcal{G} = (G, *)$ um grupo finito e $a \in G$. Temos que $a^t = e$ se e somente se t é divisível pela ordem de a em $\mathcal{G}$.*

Demonstração: ($\Leftarrow$) Seja k a ordem de a em $\mathcal{G}$ e suponha que t seja divisível por k. Logo, $t = kt'$, para algum $t' \in \mathbb{Z}$. Temos então

$$a^t = a^{kt'} = (a^k)^{t'} = e^{t'} = e.$$

($\Rightarrow$) Suponha que $a^t = e$. Vamos dividir t por k, obtendo $t = kq + r$, com $0 \leq r < k$. Temos então

$$e = a^t = a^{kq+r} = (a^k)^q * a^r = e^q * a^r = a^r.$$

Como k é a ordem de a, ele é o menor inteiro positivo tal que $a^k = e$. Por outro lado, pela igualdade acima, temos que $a^r = e$, com $r < k$. Desta forma, o único valor possível para r é $r = 0$, o que significa que t é divisível por k. ■

Corolário 4.2. *Seja $\mathcal{G}$ um grupo finito de ordem n e seja $a \in \mathcal{G}$. Então $a^n = e$ em $\mathcal{G}$.*

Demonstração: Pelo Corolário 4.1, a ordem de a divide n. Isto, em conjunto com o Lema Chave (Lema 4.2), nos permite concluir que $a^n = e$ em $\mathcal{G}$. ■

Vamos agora apresentar alguns resultados a respeito das raízes primitivas de grupos finitos cíclicos. Começamos com resultados que nos mostram que um grupo cíclico possui, em geral, mais do que uma raiz primitiva.

Teorema 4.3. *Seja $\mathcal{G}$ um grupo finito cíclico de ordem n e g uma raiz primitiva de $\mathcal{G}$. Então, g^m também é uma raiz primitiva de $\mathcal{G}$ se e somente se $mdc(m, n) = 1$.*

Demonstração: ($\Leftarrow$) Suponha que $\text{mdc}(m, n) = 1$ e seja k a ordem de g^m. Então, $(g^m)^k = g^{mk} = e$. Isto implica que a ordem de g, que é n (já que g é uma raiz primitiva de um grupo cíclico de ordem n), divide mk. Como n divide o produto mk e $\text{mdc}(m, n) = 1$, então n divide k (Lema 2.2). Por outro lado, pelo Corolário 4.1, a ordem de g^m, que é k, divide a ordem do grupo, que é n. Como n divide k e k divide n, temos que $k = n$, o que significa que g^m também é uma raiz primitiva de $\mathcal{G}$.

($\Rightarrow$) Suponha que $\text{mdc}(m, n) = d > 1$. Podemos escrever $m = dm'$, para algum $m' \in \mathbb{Z}$, e $n = dn'$, para algum $n' \in \mathbb{Z}$. Temos então

$$(g^m)^{n'} = (g^{dm'})^{n'} = (g^{m'})^{dn'} = (g^{m'})^n = e,$$

pelo Corolário 4.2, o que significa que a ordem de g^m divide n', pelo Lema Chave (Lema 4.2). Mas como $d > 1$, então $n' < n$. Logo, a ordem de g^m é menor do que n, o que significa que g^m não é uma raiz primitiva de $\mathcal{G}$. ■

Corolário 4.3. *Seja $\mathcal{G}$ um grupo finito de ordem n. Se $\mathcal{G}$ possui ao menos uma raiz primitiva, então $\mathcal{G}$ possui exatamente $\phi(n)$ raízes primitivas. Em outras palavras, se $\mathcal{G}$ é um grupo finito cíclico de ordem n, $\mathcal{G}$ possui exatamente $\phi(n)$ raízes primitivas.*

Demonstração: Suponha que g é uma raiz primitiva de $\mathcal{G}$. Então, o conjunto de todas as raízes primitivas de $\mathcal{G}$ pode ser calculado como $\{g^i : 1 \leq i < n \text{ e } \text{mdc}(i, n) = 1\}$, de acordo com o Teorema 4.3. O número de elementos deste conjunto é igual à quantidade de números no intervalo $1 \leq i < n$ que satisfazem $\text{mdc}(i, n) = 1$. Mas esta quantidade é dada justamente por $\phi(n)$. ■

Nosso próximo objetivo é mostrar um resultado central a respeito dos grupos $U(n)$ que construímos a partir das relações de congruência módulo n. Este resultado é conhecido como Teorema da Raiz Primitiva e diz que, se p é primo, então o grupo $U(p)$ é cíclico.

Para mostrar este resultado, precisamos de dois lemas auxiliares.

Lema 4.3. *Seja $\mathcal{G} = (G, *)$ um grupo abeliano finito. Sejam $a, b \in G$ tais que a ordem de a é m e a ordem de b é n, onde $mdc(m, n) = 1$. Então, a ordem de ab é mn.*

Demonstração: Primeiramente, temos que $(ab)^{mn} = a^{mn}b^{mn}$, já que o grupo é abeliano. Prosseguindo os cálculos, $a^{mn}b^{mn} = (a^m)^n(b^n)^m = e$. Logo, pelo Lema Chave (Lema 4.2), a ordem de ab divide mn.

Por outro lado, suponha que $k \leq mn$ e $(ab)^k = e$. Temos então $((ab)^k)^n = e^n = e$. Mas $((ab)^k)^n = a^{kn}(b^n)^k = a^{kn}$. Logo, $a^{kn} = e$. Pelo Lema Chave (Lema 4.2), a ordem de a, que é m, divide kn. Como $\text{mdc}(m, n) = 1$, se m divide kn, m deve dividir k (Lema 2.2). Um argumento análogo, desta vez elevando os dois lados da igualdade $(ab)^k = e$ a m, mostra que n deve dividir k. Como m e n dividem k e $\text{mdc}(m, n) = 1$, então mn também divide k, pelo Lema 2.2, o que nos permite concluir que $k = mn$. ■

O segundo lema, apresentado abaixo, foi provado inicialmente por Lagrange. Optamos por apresentar uma prova por indução para este lema.

Lema 4.4. *Sejam p um primo e $f(x) = a_k x^k + a_{k-1}x^{k-1} + \ldots + a_1 x + a_0$ um polinômio tais que os coeficientes a_i, $1 \leq i \leq k$, são inteiros, a variável x também assume valores inteiros e $a_k \not\equiv 0 \pmod{p}$ (isto é, $f(x)$ tem grau k quando considerado módulo p). Então, a congruência $f(x) \equiv 0 \pmod{p}$ possui, no máximo k soluções distintas módulo p.*

Demonstração: A prova é feita por indução em k. Para $k = 0$, a congruência $f(x) \equiv 0 \pmod{p}$ assume a forma $a_0 \equiv 0 \pmod{p}$. Entretanto, com $k = 0$, a hipótese do enunciado impõe que $a_0 \not\equiv 0 \pmod{p}$. Desta forma, fica claro que a congruência $a_0 \equiv 0 \pmod{p}$ terá 0 soluções, satisfazendo o lema para $k = 0$.

Vamos assumir agora que o lema é verdadeiro para todos os polinômios com grau menor do que k que satisfazem as hipóteses do enunciado. Queremos mostrar então que o lema também é verdadeiro para os polinômios $f(x)$ com grau igual a k que satisfaçam estas hipóteses. Se a congruência $f(x) \equiv 0 \pmod{p}$, onde $f(x)$ tem grau k, possuir menos de k soluções distintas módulo p, o lema é satisfeito para $f(x)$. Suponha então que a congruência $f(x) \equiv 0 \pmod{p}$ possui k soluções distintas módulo p, denotadas por $s_1, s_2, \ldots, s_k$. Vamos mostrar então que a congruência não possui nenhuma outra solução módulo p distinta destas k soluções listadas anteriormente.

Seja

$$g(x) = f(x) - a_k(x - s_1)(x - s_2)\dots(x - s_k).$$

Note que o grau de $g(x)$ é menor do que k já que o termo $a_k x^k$ de $f(x)$ é cancelado pelo termo $a_k x^k$ que aparece em $a_k(x - s_1)(x - s_2)\dots(x - s_k)$. Pela hipótese de indução, se $g(x)$ satisfizer as hipóteses do enunciado, a congruência $g(x) \equiv 0 \pmod{p}$ terá menos de k soluções distintas módulo p. Entretanto, temos que $g(s_i) \equiv 0 \pmod{p}$ para todos os valores s_i, $1 \leq i \leq k$. Logo, a hipótese do enunciado de que o termo líder do polinômio não é congruente a zero módulo p não pode estar sendo satisfeita por $g(x)$. Isso significa que $g(x)$ é o polinômio identicamente nulo módulo p. Assim, a partir da equação acima, obtemos

$$f(x) \equiv a_k(x - s_1)(x - s_2)\dots(x - s_k) \pmod{p}.$$

Desta forma, $f(x) \equiv 0 \pmod{p}$ se e somente se p divide o produto $a_k(x - s_1)(x - s_2)\dots(x - s_k)$. Como p é primo, se p divide um produto, ele divide um dos termos deste produto (Propriedade 3.3). Por hipótese, p não divide a_k, já que $a_k \not\equiv 0 \pmod{p}$. Desta forma, p divide $x - s_i$, para algum $1 \leq i \leq k$, o que significa que $x \equiv s_i \pmod{p}$. Portanto, $f(x) \equiv 0$ se e somente se $x \equiv s_i \pmod{p}$, para algum $1 \leq i \leq k$. Assim, todas as soluções da congruência $f(x) \equiv 0 \pmod{p}$ são congruentes a uma das k soluções listadas anteriormente. ■

Com estes dois lemas acima, podemos agora provar o Teorema da Raiz Primitiva. Este teorema foi apresentado e provado por Gauss no artigo 55 de suas "*Disquisitiones Arithmeticae*" [7]. A prova que apresentamos abaixo é a mesma apresentada no texto de Gauss.

Teorema 4.4 (Teorema da Raiz Primitiva)**.** *Se p é primo, então $U(p)$ é um grupo cíclico.*

Demonstração: Como p é primo, $U(p) = \mathbb{Z}_p - \{\overline{0}\}$, de forma que a ordem de $U(p)$ é $p-1$. Vamos fatorar $p-1$, obtendo $p-1 = q_1^{e_1} q_2^{e_2} \dots q_k^{e_k}$, onde $1 < q_1 < q_2 < \dots < q_k$ são primos distintos e $e_i \geq 1$ para todo $1 \leq i \leq k$.

Para cada potência $q_i^{e_i}$, $1 \leq i \leq k$, nesta fatoração, é possível encontrar um elemento de $U(p)$ que tenha ordem $q_i^{e_i}$. Para isso, começamos buscando um elemento $\overline{a_i} \in U(p)$ tal que $a_i^{(p-1)/q_i} \not\equiv 1 \pmod{p}$. Este elemento precisa existir, já que os elementos $\overline{u} \in U(p)$ tais que $u^{(p-1)/q_i} \equiv 1 \pmod{p}$ são soluções da congruência $x^{(p-1)/q_i} - 1 \equiv 0 \pmod{p}$, que possui no máximo $(p-1)/q_i < p-1$ soluções distintas módulo p, de acordo com o Lema 4.4.

Uma vez encontrado o valor a_i, calculamos $h_i \equiv a_i^{(p-1)/q_i^{e_i}} \pmod{p}$. Temos que

$$h_i^{q_i^{e_i}} \equiv a_i^{p-1} \equiv 1 \pmod{p},$$

de acordo com o Corolário 4.2, logo a ordem de $\overline{h_i}$ divide $q_i^{e_i}$ pelo Lema Chave (Lema 4.2). Suponha então que a ordem de $\overline{h_i}$ seja q_i^t, onde $t < e_i$. Temos então

$$1 \equiv h_i^{q_i^t} \equiv (a_i^{(p-1)/q_i^{e_i}})^{q_i^t} \equiv a_i^{(p-1)/q_i^{e_i - t}} \pmod{p},$$

o que significa que a ordem de $\overline{a_i}$ divide $(p-1)/q_i^{e_i - t}$ pelo Lema Chave (Lema 4.2). Mas como $e_i - t > 1$, $(p-1)/q_i^{e_i - t}$ divide $(p-1)/q_i$. Logo, a ordem de $\overline{a_i}$ divide $(p-1)/q_i$, o que implica que $a_i^{(p-1)/q_i} \equiv 1 \pmod{p}$, também pelo Lema Chave. Mas isto é uma contradição com a escolha de a_i. Desta forma, a ordem de $\overline{h_i}$ é igual a $q_i^{e_i}$.

Realizado este cálculo para cada potência $q_i^{e_i}$ da fatoração, obtemos elementos $\overline{h_1}, \overline{h_2}, \ldots, \overline{h_k} \in U(p)$ tais que suas respectivas ordens são $q_1^{e_1}, q_2^{e_2}, \ldots, q_k^{e_k}$. Repare que, como estas ordens são potências de primos distintos, se m é a ordem de $\overline{h_i}$ e n é a ordem de $\overline{h_j}$, com $i \neq j$, então $\text{mdc}(m, n) = 1$. Temos então que o elemento $\overline{g}$, onde $g \equiv \prod_{1 \leq i \leq k} h_i \pmod{p}$ tem ordem $\prod_{1 \leq i \leq k} q_i^{e_i} = p - 1$, de acordo com o Lema 4.3. Logo, $\overline{g}$ é uma raiz primitiva de $U(p)$, o que significa que $U(p)$ é um grupo cíclico. ■

Corolário 4.4. *Se p é primo, o grupo $U(p)$ possui exatamente $\phi(\phi(p)) = \phi(p-1)$ raízes primitivas.*

Demonstração: Se p é primo, então o Teorema da Raiz Primitiva (Teorema 4.4) nos diz que $U(p)$ é cíclico. Como a ordem de $U(p)$ é $\phi(p) = p - 1$, o Corolário 4.3 nos diz que $U(p)$ possui exatamente $\phi(\phi(p)) = \phi(p-1)$ raízes primitivas. ■

Um aspecto interessante da prova de Gauss para o Teorema da Raiz Primitiva é que ela é uma *prova construtiva*, isto é, ela não somente mostra que $U(p)$ possui uma raiz primitiva, mas também nos mostra explicitamente uma forma de calcular tal raiz primitiva. Este método de cálculo pode ser facilmente transformado em um algoritmo para a obtenção de uma raiz primitiva de $U(p)$. Dado que este algoritmo é extraído diretamente da prova do teorema dada por Gauss, chamamos este algoritmo de Algoritmo de Gauss.

No algoritmo, fatoramos o número $p - 1$ na forma $p - 1 = q_1^{e_1} q_2^{e_2} \ldots q_k^{e_k}$. Então, para cada primo q_i, buscamos um valor a_i tal que $a^{(p-1)/q_i} \not\equiv 1 \pmod{p}$. Calculamos então $h_i \equiv a_i^{(p-1)/q_i^{e_i}} \pmod{p}$ e multiplicamos todos os valores de h_i, $1 \leq i \leq k$, para obter uma raiz primitiva. Apresentamos a descrição formal deste algoritmo abaixo.

Algoritmo 4.1: Algoritmo de Gauss

Entrada: Um número primo $p \geq 3$.

Saída: Uma raiz primitiva do grupo $U(p)$.

Instruções:

1. Fatore $p - 1$, obtendo $p - 1 = q_1^{e_1} q_2^{e_2} \ldots q_k^{e_k}$.
2. $i \leftarrow 1$, $g \leftarrow 1$
3. Enquanto $i \leq k$, faça:
   3.1. $a \leftarrow 2$
   3.2. Enquanto $a^{(p-1)/q_i} \equiv 1 \pmod{p}$, faça $a \leftarrow a + 1$.
   3.3. $h \leftarrow (a^{(p-1)/q_i^{e_i}}) \bmod p$
   3.4. $g \leftarrow (g * h) \bmod p$
   3.5. $i \leftarrow i + 1$
4. Retorne g.

O Algoritmo de Gauss retorna uma das raízes primitivas de $U(p)$, mas não há garantias de que ele irá retornar a *menor* raiz primitiva. De qualquer forma, uma vez que uma das raízes primitivas é conhecida, todas as outras podem ser calculadas utilizando-se o Teorema 4.3.

Exemplo 4.14. *Vamos utilizar o Algoritmo de Gauss para calcular uma raiz primitiva de* $U(41)$*. Temos então* $p = 41$*. Fatorando* $p - 1 = 40$*, obtemos* $40 = 2^3.5$*, logo* $q_1 = 2$*,* $q_2 = 5$*,* $e_1 = 3$ *e* $e_2 = 1$*.*

Começamos com $q_1 = 2$*.* $2^{(p-1)/q_1} \equiv 2^{20} \equiv 1 \pmod{41}$*, mas* $3^{20} \equiv 40 \not\equiv 1 \pmod{41}$*. Calculamos então* $h_1 \equiv 3^{(p-1)/q_1^{e_1}} \equiv 3^5 \equiv 38 \pmod{41}$*.*

Prosseguimos agora com $q_2 = 5$*. Fazendo* $2^{(p-1)/q_2} \equiv 2^8 \equiv 10 \not\equiv 1 \pmod{41}$*. Calculamos então* $h_2 \equiv 2^{(p-1)/q_2^{e_2}} \equiv 2^8 \equiv 10 \pmod{41}$*.*

Uma raiz primitiva de $U(41)$ *será então a forma reduzida de* h_1h_2 *módulo* 41*. Temos* $h_1h_2 \equiv 38.10 \equiv 380 \equiv 11 \pmod{41}$*. Logo,* 11 *é uma raiz primitiva de* $U(41)$*.*

Repare que existe ainda outra maneira diferente de escrever este algoritmo. Da maneira que implementamos o algoritmo, para cada fator primo q_i, buscamos um elemento a_i que fosse adequado. A outra maneira seria realizar a busca da forma inversa: para cada valor a no intervalo $2 \leq a \leq p - 1$, buscar quais fatores primos q_i são "atendidos" por a, isto é, quais fatores primos q_i satisfazem $a^{(p-1)/q_i} \not\equiv 1 \pmod{p}$, e calcular todos os respectivos valores de h_i utilizando este elemento a. Em seguida, enquanto restarem primos q_i que não foram "atendidos", incrementamos o valor de a e repetimos o processo. Deixamos a descrição formal desta segunda versão do Algoritmo de Gauss como exercício (Exercício 5).

4.5 Exercícios

1. Seja $\mathcal{G}$ um grupo. Mostre que, se o quadrado de qualquer elemento de $\mathcal{G}$ é igual ao elemento neutro, então o grupo é abeliano.

2. Dados os seguintes valores de n, calcule $\phi(n)$:

   2.1. 16807
   2.2. 9282
   2.3. 30375
   2.4. 694575

3. Determine todos os subgrupos cíclicos dos grupos abaixo, com seus respectivos geradores:

   3.1. $U(8)$
   3.2. $U(18)$
   3.3. $U(21)$
   3.4. $U(25)$

4. Utilizando o Algoritmo de Gauss, determine uma raiz primitiva para cada um dos grupos abaixo. Em seguida, encontre todas as raízes primitivas de cada grupo utilizando o Teorema 4.3:

   4.1. $U(7)$
   4.2. $U(13)$
   4.3. $U(19)$
   4.4. $U(53)$

5. Escreva a descrição formal da segunda versão do Algoritmo de Gauss que foi apresentada no final da Seção 4.4.

Capítulo 5

O Método El Gamal

Neste ponto, após todos os conceitos matemáticos necessários a respeito dos números inteiros, da aritmética modular e da teoria de grupos terem sido apresentados, estamos prontos para entender o funcionamento do método El Gamal.

Neste capítulo, apresentamos o método El Gamal para criptografia e para assinatura digital, assim como o método DSA para assinatura digital, que é uma pequena variação do El Gamal. Mostraremos como construir as chaves de encriptação e decriptação (ou chaves de assinatura e verificação, no caso da assinatura digital) e quais são os cálculos necessários para encriptar e decriptar uma mensagem (ou para assinar uma mensagem e verificar uma assinatura).

Discutiremos o funcionamento do método, mostrando que a mensagem original sempre pode ser recuperada por quem possui a chave correta de decriptação. Analogamente, no caso da assinatura digital, mostramos também que uma assinatura sempre pode ser produzida para qualquer mensagem por quem possui a chave correta de assinatura. Por outro lado, mostraremos que o método é bastante seguro, mostrando que a única maneira conhecida para um usuário não autorizado calcular o valor da chave de decriptação a partir apenas do conhecimento da chave de encriptação (ou o valor da chave de assinatura a partir da chave de verificação) envolve a resolução do Problema do Logaritmo Discreto em um grupo finito, que é um problema computacionalmente difícil. Desta forma, podemos concluir que o método El Gamal é um método efetivo e seguro de criptografia de chave pública e assinatura digital.

5.1 Esquema Geral

Antes de discutirmos as especificidades do método El Gamal, vamos apresentar nesta seção um esquema genérico do funcionamento das diversas etapas dos métodos de criptografia de chave pública e de assinatura digital.

Vamos utilizar, como é comum na literatura da área, os nomes de Alice e Bernardo para as duas pessoas ou sistemas computacionais que querem se comunicar através da troca de mensagens. O grande problema para a comunicação entre Alice e Bernardo é que o canal por onde trafegam as mensagens é um canal inseguro. Isto significa que pessoas ou entidades mal-intencionadas podem ocasionalmente observar, interceptar ou mesmo forjar mensagens que trafegam neste canal.

A insegurança do canal produz uma série de questões que precisam se resolvidas. De certa forma, podemos dizer que existem dois problemas principais de segurança

com relação à uma mensagem trafegando em um canal inseguro: um problema de segurança com relação ao destinatário da mensagem, que está relacionado à questão de *privacidade*, e um problema de segurança com relação ao remetente da mensagem, que está relacionado à questão de *autenticidade*.

Quando Alice envia uma mensagem para Bernardo, os dois gostariam que ninguém mais além de Bernardo fosse capaz de ler o seu conteúdo. Isto significa que o conteúdo da mensagem deveria ser *privado* entre os dois. Entretanto, se Alice enviar a mensagem destinada a Bernardo diretamente no canal, sem nenhum cuidado preliminar, este objetivo de privacidade e segurança não pode ser garantido.

Os métodos de criptografia buscam justamente lidar com esta situação em que uma mensagem deve ser enviada de forma segura por um canal inseguro. O objetivo da criptografia é esconder o conteúdo da mensagem enquanto ela trafega no canal, de forma que qualquer pessoa ou entidade que consiga observar a mensagem enquanto ela trafega no canal não consiga determinar o seu conteúdo original. Ou seja, alguém que observe a mensagem no canal verá uma mensagem obscura e sem sentido. Apenas Bernardo, com a sua chave de decriptação, poderá transformar de volta esta mensagem sem sentido na mensagem original escrita por Alice.

No caso de um método de criptografia de chave pública, a chave de encriptação de Bernardo será pública, permitindo a qualquer um, incluindo Alice, encriptar mensagens para serem enviadas a Bernardo. Por outro lado, a chave de decriptação de Bernardo é privada, sendo de posse exclusiva dele. Assim, apenas Bernardo é capaz de decriptar as mensagens que são endereçadas a ele. Para que um sistema nestes moldes seja seguro, Bernardo tem que garantir que a sua chave de decriptação permaneça efetivamente privada. Além disso, uma vez que a chave de encriptação é pública e, portanto, está acessível a todos, o cálculo da chave privada a partir da chave pública deve ser extremamente complexo do ponto de vista computacional.

Podemos resumir então os principais passos de um método de criptografia de chave pública:

1. Geração de chaves:
    1.1. Um método de criptografia de chave pública exige que Bernardo realize inicialmente uma escolha de valores para alguns parâmetros (estes valores são públicos após sua escolha).
    1.2. Bernardo cria um par de chaves de acordo com os valores dos parâmetros selecionados anteriormente: uma chave pública de encriptação e uma chave privada de decriptação.
    1.3. Bernardo divulga sua chave pública para todos.
    1.4. A partir deste momento, qualquer um (incluindo Alice) pode utilizar a chave pública de Bernardo para encriptar mensagens destinadas a ele antes de enviá-las.
    1.5. Por outro lado, apenas Bernardo pode decriptar com sua chave privada tais mensagens e acessar seu conteúdo original.
2. Encriptação:
    2.1. Se Alice quer enviar uma mensagem criptografada para Bernardo, o primeiro passo para ela é obter a chave pública de encriptação de Bernardo.
    2.2. Em seguida, Alice executa o algoritmo de encriptação oferecido pelo método que está sendo utilizando fornecendo como entradas a mensagem que ela quer enviar e a chave pública de Bernardo. O algoritmo produzirá uma mensagem encriptada.

2.3. Alice envia então a mensagem encriptada para Bernardo.

3. Decriptação:

3.1. Ao receber a mensagem encriptada de Alice, Bernardo executa o algoritmo de decriptação oferecido pelo método que está sendo utilizado fornecendo como entradas a mensagem recebida e a sua chave privada. O algoritmo produzirá uma mensagem decriptada.

3.2. A mensagem decriptada é igual à mensagem original elaborada por Alice. Assim, Bernardo consegue recuperar a mensagem original destinada a ele.

Desta forma, um método de criptografia de chave pública deve nos oferecer três algoritmos: um algoritmo de geração de chaves, um algoritmo de encriptação e um algoritmo de decriptação.

Por outro lado, quando Alice envia uma mensagem para Bernardo, Bernardo gostaria de ter alguma garantia de que a mensagem realmente foi criada por Alice, e não por outra pessoa.

Um exemplo clássico da necessidade deste tipo de garantia de *autenticidade* ocorre se Bernardo for o gerente responsável pela conta bancária de Alice. Se ele receber uma mensagem de alguém dizendo ser Alice e pedindo para esvaziar a sua conta bancária, ele precisa ter alguma forma de se certificar se a mensagem realmente foi criada por Alice ou se foi criada por outra pessoa tentando prejudicar Alice em uma fraude bancária.

Esta segurança com relação ao remetente da mensagem pode ser fornecida pelas assinaturas digitais. Os métodos de assinatura digital buscam justamente lidar com esta situação em que todas as mensagens chegam através de um canal inseguro e há a necessidade de se verificar qual foi a origem de cada mensagem. O objetivo da assinatura digital é impedir que uma pessoa ou entidade se aproveite do canal inseguro para enviar mensagens fingindo que é outra pessoa.

Em um método de assinatura digital, a chave de assinatura de Alice será privada, de maneira que apenas ela seja capaz de gerar assinaturas para as mensagens que ela criar. Por outro lado, a chave de verificação da assinatura será pública, permitindo a qualquer um, incluindo Bernardo, verificar a assinatura das mensagens recebidas de forma a determinar se a pessoa que criou originalmente a mensagem realmente é quem diz ser. Da mesma forma que na criptografia de chave pública, para que um sistema nestes moldes seja seguro, Alice tem que garantir que a sua chave de assinatura permaneça efetivamente privada. Além disso, uma vez que a chave de verificação é pública e, portanto, está acessível a todos, o cálculo da chave privada a partir da chave pública deve ser extremamente complexo do ponto de vista computacional.

Podemos resumir então os principais passos de um método de assinatura digital:

1. Geração de chaves:

1.1. Um método de assinatura digital exige que Alice realize inicialmente uma escolha de valores para alguns parâmetros (estes valores são públicos após sua escolha).

1.2. Alice cria um par de chaves de acordo com os valores dos parâmetros selecionados anteriormente: uma chave privada de assinatura e uma chave pública de verificação.

1.3. Alice divulga sua chave pública para todos.

1.4. A partir deste momento, apenas Alice pode assinar digitalmente, utilizando sua chave privada, uma mensagem que foi criada por ela.

1.5. Por outro lado, qualquer um (incluindo Bernardo) pode utilizar a chave pública de Alice para verificar assinaturas em mensagens que, ao menos supostamente, foram criadas por ela.

2. Assinatura:

2.1. Para que Alice possa enviar uma mensagem assinada a Bernardo, ela executa o algoritmo de assinatura oferecido pelo método que está sendo utilizado fornecendo como entradas a mensagem que ela quer enviar e a sua chave privada. O algoritmo produzirá uma assinatura para a mensagem.

2.2. Em seguida, Alice envia para Bernardo o par formado pela mensagem e pela assinatura produzida pelo algoritmo.

3. Verificação:

3.1. Se Bernardo quer verificar as assinaturas em mensagens que, ao menos supostamente, foram criadas por Alice, o primeiro passo para ele é obter a chave pública de verificação de Alice.

3.2. Ao receber a mensagem assinada de Alice (o par formado por mensagem e assinatura), Bernardo executa o algoritmo de verificação oferecido pelo método que está sendo utilizado fornecendo como entradas a mensagem e a assinatura que foram recebidas e a chave pública de Alice. O algoritmo produzirá uma resposta do tipo Sim/Não, informando se a assinatura da mensagem é válida, isto é, foi realmente feita por Alice, ou não.

3.3. Caso a assinatura seja verificada como legítima, Bernardo pode ter segurança a respeito da origem da mensagem. Caso contrário, Bernardo descarta a mensagem, pois ela provavelmente foi criada por alguém tentando se passar por Alice.

Desta forma, analogamente ao caso de um método de criptografia de chave pública, um método de assinatura digital deve nos oferecer três algoritmos: um algoritmo de geração de chaves, um algoritmo de assinatura e um algoritmo de verificação.

É possível também a utilização combinada de criptografia e assinatura digital, de maneira a fornecer segurança a respeito dos dois lados da comunicação. Neste caso, Alice deve primeiro assinar a mensagem com a sua chave privada de assinatura. Em seguida, ela encripta o par formado pela mensagem e pela assinatura com a chave pública de encriptação de Bernardo. Finalmente, ela envia para Bernardo esta última mensagem encriptada. Desta forma, ela esconde tanto a mensagem que ela irá enviar como também a informação de qual é a assinatura que corresponde àquela mensagem. Quando Bernardo recebe uma mensagem encriptada, ele, primeiramente, decripta a mensagem aplicando a sua chave privada de decriptação. Com isso, ele obtém um par formado por uma mensagem e uma assinatura. Ele então realiza a verificação desta assinatura usando a chave pública de verificação de Alice. Caso a assinatura não seja legítima, ele descarta a mensagem. Caso contrário, ele pode então acessar o conteúdo da mensagem original enviada por Alice.

É importante salientar que os valores dos parâmetros dos métodos de criptografia de chave pública e de assinatura digital são valores numéricos. Analogamente, as

chaves, tanto as públicas quanto as privadas, são números ou conjuntos de números. Por fim, os algoritmos de encriptação, decriptação, assinatura e verificação são algoritmos que realizam cálculos numéricos.

Assim, para que sejamos capazes de processar as mensagens de nosso interesse com estes algoritmos, é essencial que estas mensagens também estejam representadas de forma numérica. No caso de um número de cartão de crédito que queiramos criptografar, por exemplo, já temos a mensagem no formato desejado. Mas no caso de mensagens textuais, precisamos realizar algum tipo de pré-processamento para transformar a sequência de caracteres em uma sequência de números. Este procedimento é conhecido como pré-codificação. Na pré-codificação, cada caractere da mensagem original é transformado em uma sequência de algarismos.

O remetente da mensagem aplica o procedimento de pré-codificação de forma a obter uma mensagem em formato numérico onde possam ser aplicados os algoritmos de criptografia e/ou assinatura digital. Do outro lado do canal de comunicação, o destinatário, após realizar a decriptação da mensagem e/ou a verificação da assinatura, deve desfazer o procedimento de pré-codificação, de forma a obter novamente o conteúdo original da mensagem.

Para que este processo de reversão da pré-codificação possa ser feito de modo único, é fundamental que a pré-codificação nunca transforme duas mensagens diferentes no mesmo valor numérico. Assim, caracteres distintos nunca podem ser pré-codificados na mesma sequência de algarismos e todos os caracteres devem ser pré-codificados em sequências de algarismos de mesmo tamanho.

Como exemplo da importância deste último requisito, vamos considerar um pequeno exemplo. Suponha que o caractere “A” seja pré-codificado como 1, já que é a primeira letra do alfabeto, “B” seja pré-codificado como 2 e assim, por diante. Então, “L” será pré-codificado como 12. Entretanto, se o destinatário receber a mensagem 12, ele não saberá se a mensagem original era “AB” ou “L”, tornando o processo de reversão da pré-codificação ambíguo. Isto ocorreu porque, neste exemplo, alguns caracteres foram pré-codificados em sequências de 1 algarismo enquanto outros foram pré-codificados em sequências de 2 algarismos.

Uma maneira simples de realizar o processo de pré-codificação é utilizar as mesmas tabelas de correspondência utilizadas pelos sistemas operacionais e programas computacionais para armazenar caracteres textuais em formato numérico na memória interna do computador. Estas tabelas de correspondência atendem ao requisito de representar todos os caracteres como sequências de algarismos de mesmo tamanho. Como exemplo, podemos citar a tabela Unicode [27], a mais utilizada por programas computacionais contemporâneos.

Como um segundo exemplo de tabela de pré-codificação, para um caso mais simples em que a mensagem contém apenas letras maiúsculas, apresentamos a tabela abaixo, em que todos os caracteres são pré-codificados em sequências distintas de dois algarismos.

A	B	C	D	E	F	G	H	I	J	K	L	M
10	11	12	13	14	15	16	17	18	19	20	21	22
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
23	24	25	26	27	28	29	30	31	32	33	34	35

5.2 Criptografia El Gamal

Nesta seção, apresentamos o método El Gamal de criptografia de chave pública. Este método foi desenvolvido pelo cientista da computação egípcio Taher El Gamal em 1985 [5].

O método El Gamal de criptografia é um método baseado em grupos abelianos finitos cíclicos. Originalmente, ele foi desenvolvido a partir da utilização dos grupos finitos $U(p)$, onde p é primo, já que o Teorema da Raiz Primitiva (Teorema 4.4) nos garante que tais grupos são necessariamente cíclicos. No entanto, o método pode ser generalizado para utilizar quaisquer outros grupos abelianos finitos cíclicos.

Uma variante do método El Gamal que está sendo largamente desenvolvida e utilizada atualmente é conhecida como *Criptografia com Curvas Elípticas*. Ela opera da mesma forma que o método El Gamal original apenas substituindo os grupos $U(p)$ por grupos de pontos em uma curva elíptica. Não trataremos desta variante neste livro, mas [2] é uma boa referência para seu estudo.

Conforme discutimos na seção anterior, um método de criptografia de chave pública deve prover três algoritmos: um algoritmo de geração de chaves, um algoritmo de encriptação e um algoritmo de decriptação. Vamos então analisar o funcionamento destes três algoritmos no caso do método El Gamal.

Para a geração das chaves, sendo uma pública de encriptação e uma privada de decriptação, precisamos inicialmente de um primo p, uma vez que todos os cálculos do método serão realizados em um grupo finito $U(p)$, com p primo. Para selecionar este primo, começamos determinando um intervalo dos inteiros onde desejamos procurá-lo. Como veremos mais adiante, para que o método seja efetivamente seguro, uma condição necessária (mas que não é suficiente sozinha) é que o primo p selecionado seja grande. As recomendações atuais de segurança indicam que p deve ter pelo menos 2048 bits [19], o que significa que p será um número com aproximadamente 600 algarismos.

Para encontrar um número primo no intervalo definido, podemos testar sequencialmente ou aleatoriamente diversos números ímpares dentro do intervalo. Para cada número, testamos se ele é divisível por algum fator primo pequeno (por exemplo, fatores primos menores que 100000, que podem ser obtidos com a utilização do Crivo de Eratóstenes). Se o número não for divisível por nenhum destes primos, aplicamos a ele o Teste de Miller-Rabin com uma quantidade razoável de bases. Caso o resultado seja inconclusivo em todas as bases, obtivemos o número primo p que precisamos com altíssima probabilidade[1].

Uma vez selecionado o primo p, aplicamos o Algoritmo de Gauss para calcular uma raiz primitiva g de $U(p)$. Os valores de p e g são então os *parâmetros públicos* da nossa implementação do El Gamal. É interessante salientar que o parâmetro g não precisa ser necessariamente a raiz primitiva calculada pelo Algoritmo de Gauss. Qualquer raiz primitiva de $U(p)$ que seja escolhida como parâmetro g é válida. Entretanto, o Algoritmo de Gauss nos fornece uma maneira genérica de obter o parâmetro g dado o valor do parâmetro p.

Resta agora gerar as chaves propriamente ditas. Para a chave privada de decriptação, selecionamos um inteiro d no intervalo $1 < d < p - 1$. Em seguida, calculamos a chave pública de encriptação c como a forma reduzida de g^d módulo p.

[1] Discutimos a probabilidade de um número ser composto e retornar resultados inconclusivos no Teste de Miller-Rabin com várias bases no Capítulo 3 e mostramos que ela pode ser arbitrariamente pequena conforme aumentamos o número de bases.

Os valores de g, p e c serão todos públicos. Qualquer pessoa ou entidade que deseje enviar mensagens criptografadas para a pessoa ou entidade que gerou as chaves precisa apenas obter estes dados públicos para aplicá-los no algoritmo de encriptação. Por outro lado, o valor de d deve permanecer privado, de posse exclusiva da pessoa ou entidade que gerou esta chave. Desta forma, apenas ele poderá decriptar as mensagens que lhe são endereçadas. Caso alguém mais ganhe acesso a esta chave privada, poderá também decriptar as mensagens, violando a segurança pretendida com o uso do sistema criptográfico.

A etapa da geração das chaves costuma ser mais lenta do que as etapas de encriptação e decriptação, uma vez que encontrar um primo grande não é uma tarefa trivial. Isto pode ser notado pelo simples fato de que existem mais números compostos do que primos. Para determinarmos o número médio de tentativas que precisamos fazer para encontrar um número primo em um dado intervalo, precisaríamos estudar a distribuição e a densidade dos números primos entre os inteiros em um intervalo fixado. Este estudo está além do escopo deste livro, mas existem funções que medem estes parâmetros e nos permitem calcular a probabilidade associada de selecionarmos um primo. Para tais funções, [24] pode ser consultado.

De qualquer forma, o fato da geração de chaves ser a etapa mais lenta não é um problema, uma vez que ela é a etapa menos frequente entre as três. Após criarmos um par de chaves, podemos enviar muitas mensagens criptografadas utilizando este par, de forma que o tempo gasto com a geração das chaves é amortizado pelo longo tempo de duração destas chaves.

Descrevemos abaixo o algoritmo de geração de chaves, de acordo com os procedimentos que apresentamos.

Algoritmo 5.1: Algoritmo de Geração de Chaves do Método de Criptografia El Gamal

Entrada: Dois inteiros n_1 e n_2.

Saída: Os parâmetros públicos p e g, onde $n_1 \leq p \leq n_2$ é primo e g é uma raiz primitiva de $U(p)$, a chave pública c de encriptação e a chave privada d de decriptação.

Instruções:

1. Selecione um primo p tal que $n_1 \leq p \leq n_2$.
2. Calcule, utilizando o Algoritmo de Gauss, uma raiz primitiva g de $U(p)$.
3. Selecione d no intervalo $1 < d < p - 1$.
4. $c \leftarrow g^d \bmod p$ $\quad \#c$ é a forma reduzida de g^d módulo p
5. Retorne p, g, c e d.

Algumas questões importantes podem ser observadas neste processo de geração das chaves. Primeiramente, nós descartamos as possibilidades $d = 0$, $d = 1$ e $d = p - 1$, pois elas resultam em chaves privadas fracas. Isto se deve ao fato de que, nestes casos, a chave pública mostra explicitamente quem é a chave privada. Se $d = 0$ ou $d = p - 1$, então $c = g^d \bmod p = 1$ e se $d = 1$, então $c = g^d \bmod p = g$. Por outro lado, como g é uma raiz primitiva de $U(p)$ e este grupo tem ordem $p - 1$, as potências de g com expoentes no intervalo $1 < d < p - 1$ resultam em elementos distintos de $U(p)$ e, em particular, também distintos de g^0 e g^1. Desta forma, uma pessoa que observe uma chave pública $c = 1$ saberá automaticamente que a chave

privada é $d = 0$ ou $d = p - 1$, sendo que não há diferença real entre os dois valores, como pode ser visto mais adiante no algoritmo de decriptação. Por outro lado, se a chave pública é $c = g$, também se sabe automaticamente que a chave privada é $d = 1$. Concluímos então que, para estes valores, a chave não permaneceria efetivamente privada.

Outra observação importante diz respeito à relação entre a chave pública e a chave privada. A chave pública é do conhecimento de todos. Assim, é crucial para a segurança do sistema que não seja possível do ponto de vista computacional a realização do cálculo da chave privada a partir do conhecimento da chave pública. A chave pública é construída como $c = g^d \bmod p$. Calcular a chave privada então significa calcular o valor de d no intervalo $1 < d < p-1$ tal que $g^d \equiv c \pmod{p}$, onde todos os outros valores envolvidos, g, c e p, são públicos. Isto significa que cacular a chave privada a partir dos dados públicos de uma implementação do El Gamal envolve resolver uma instância do Problema do Logaritmo Discreto (Definição 4.8), o que é computacionalmente complexo no caso geral. Assim, caso os parâmetros públicos sejam escolhidos de acordo com as recomendações, torna-se inviável na prática o cálculo da chave privada a partir dos valores que são de conhecimento público, mesmo com auxílio computacional.

Para que a resolução do Problema do Logaritmo Discreto que permite calcular a chave privada seja realmente inviável, precisamos manter sempre em mente que existem algoritmos para a resolução do Problema do Logaritmo Discreto (veremos alguns no próximo capítulo) e que estes algoritmos são eficientes em alguns casos. Desta forma, precisamos escolher os parâmetros de forma a evitar instâncias do Problema do Logaritmo Discreto que possam ser resolvidas de forma eficiente por algum dos algoritmos conhecidos. Concluímos a partir disto que o estudo dos algoritmos para a resolução do Problema do Logaritmo Discreto não é de interesse apenas de pessoas maliciosas que buscam quebrar a segurança do sistema. Ele é crucial também para quem implementa o sistema, pois é ele que permite estabelecer as diretrizes para a escolha dos parâmetros de forma a obter uma implementação segura.

No próximo capítulo, quando estudarmos alguns algoritmos para a resolução do Problema do Logaritmo Discreto, iremos determinar algumas restrições sobre os parâmetros do El Gamal que são impostas por cada um deles.

Prosseguimos agora com o algoritmo de encriptação do método El Gamal. A primeira observação que devemos fazer sobre ele diz respeito a um limite existente para as mensagens que podemos encriptar. Vamos considerar que a mensagem m já está em formato numérico, isto é, que m é um inteiro. Para aplicarmos o algoritmo de encriptação a m, precisamos que m esteja no intervalo $0 < m < p$, onde p é o parâmetro do El Gamal. Caso a mensagem seja maior ou igual a p, o algoritmo de decriptação não será capaz de recuperar a mensagem original.

Para lidar então com mensagens maiores ou iguais a p, devemos quebrá-las em blocos, onde cada bloco b deve estar no intervalo $0 < b < p$. Então, encriptamos e enviamos cada bloco separadamente para o destinatário. Ele, por sua vez, decriptará cada bloco separadamente e então reunirá os blocos já decriptados para reconstruir a mensagem original.

A quebra da mensagem em blocos pode ser feita de maneira razoavelmente arbitrária, desde que seja respeitado o requisito de cada bloco estar no intervalo $0 < b < p$ e desde que nenhum bloco comece com um 0. A razão para esta segunda restrição vem do fato de que um zero à esquerda em um número não tem valor numérico. Assim, quando um bloco que começa com 0 é encriptado e, posteriormente, decriptado, ele é recuperado sem o algarismo 0 à esquerda. Então,

quando ele for reunido aos outros blocos para recompor a mensagem original, uma mensagem diferente será obtida.

Exemplo 5.1. *Suponha que* $p = 127$ *e a mensagem que queremos encriptar é* $m = 110125$*. Como* $m \geq p$*, temos que quebrar esta mensagem em blocos. Podemos quebrá-la como* $1-101-25$*, como* $1-10-125$*, como* $110-125$ *e como* $110-12-5$*, dentre outras formas. Mas não podemos quebrá-la como* $11-01-25$ *ou* $11-012-5$*, dentre outras formas, porque nestes casos um dos blocos está começando com um zero à esquerda.*

Para o algoritmo de encriptação, vamos então assumir que temos os parâmetros p e g e a chave pública de encriptação c e que a mensagem m já é uma mensagem que está no intervalo $0 < m < p$.

Para encriptar esta mensagem, começamos selecionando *aleatoriamente* um valor k no intervalo $1 < k < p - 1$. Este valor é conhecido como *chave efêmera*, uma vez que ele é descartado logo após a finalização da encriptação, sendo selecionado um novo valor de k para cada novo processo de encriptação. O uso de uma chave efêmera coloca o método de criptografia El Gamal na família dos chamados métodos de *encriptação probabilística*. Em métodos de encriptação probabilística, uma mesma mensagem pode ser encriptada de diversas formas diferentes, devido à possibilidade de uso de chaves efêmeras diferentes em cada processo de encriptação.

É *absolutamente essencial* selecionar um novo valor aleatório de k a cada nova mensagem que quisermos encriptar. A repetição do mesmo valor de k na encriptação de duas mensagens distintas pode abrir uma séria brecha de segurança, conforme notado pelo próprio El Gamal em seu artigo original [5].

Estudaremos com mais detalhes os problemas decorrentes da re-utilização de valores de k em mensagens distintas ao estudarmos o método de assinatura digital El Gamal, onde eles são ainda mais sérios e, portanto, mais simples de serem percebidos e estudados.

Após a escolha do valor de k, calculamos dois valores s e t, onde

$$s = g^k \bmod p$$

e

$$t = mc^k \bmod p.$$

A mensagem encriptada será então o par (s, t), onde $0 < s, t < p$.

Descrevemos abaixo o algoritmo de encriptação, de acordo com os procedimentos que apresentamos.

Algoritmo 5.2: Algoritmo de Encriptação do Método de Criptografia El Gamal

Entrada: Um número primo p, uma raiz primitiva g de $U(p)$, uma chave pública c e uma mensagem $0 < m < p$.

Saída: Uma mensagem encriptada $\widehat{m} = (s, t)$, onde $0 < s, t < p$.

Instruções:

1. Selecione *aleatoriamente* um valor k no intervalo $1 < k < p - 1$.
2. $s \leftarrow g^k \bmod p$ $\quad \#s$ é a forma reduzida de g^k módulo p
3. $t \leftarrow (m * c^k) \bmod p$
4. $\widehat{m} \leftarrow (s, t)$

5. Retorne $\widehat{m}$.

É importante reparar que a chave privada d não é utilizada em nenhum momento do processo de encriptação. Este processo foi realizado apenas com o uso dos valores públicos, de maneira que qualquer um pode encriptar mensagens e enviá-las para o destinatário, que se mantém como o único portador do dado necessário para decriptá-las.

Outra observação importante a respeito deste algoritmo de encriptação é que a mensagem encriptada possui aproximadamente o dobro do tamanho da mensagem original. Assim, há um custo envolvido nesta encriptação já que uma quantidade maior de dados irá trafegar pelo canal de comunicação.

Exemplo 5.2. *Vamos realizar um exemplo de encriptação com o método El Gamal. Suponha que os parâmetros escolhidos são* $p = 151$ *e* $g = 77$ *(a raiz primitiva de* $U(151)$ *obtida com o Algoritmo de Gauss). Vamos considerar também que a chave pública que está sendo utilizada é* $c = 3$*. Se desejamos encriptar a mensagem* $m = 140$*, começamos selecionando um valor aleatório de* k*. Vamos supor que o valor selecionado é* $k = 7$*. Calculamos então* $s \equiv g^k \equiv 77^7 \equiv 115 \pmod{151}$*. Finalmente, calculamos* $t \equiv mc^k \equiv 140.3^7 \equiv 103 \pmod{151}$*. Logo a mensagem encriptada é o par* $(115, 103)$*.*

Vamos agora para a última etapa do método de criptografia El Gamal, apresentando o algoritmo de decriptação. O requisito essencial deste algoritmo é que ele "reverta" o procedimento feito pelo algoritmo de encriptação. Suponha então que utilizamos os parâmetros p e g e a chave pública c para encriptar uma mensagem m com o algoritmo de encriptação do método El Gamal, obtendo a mensagem encriptada $\widehat{m}$. Então, se d é a chave privada associada à c, ao aplicar o algoritmo de decriptação do método El Gamal a $\widehat{m}$ utilizando a chave privada d e os mesmos parâmetros p e g utilizados na encriptação, devemos obter novamente a mensagem m.

Dada uma mensagem encriptada $\widehat{m} = (s, t)$, começamos o processo de decriptação calculando o inverso de s módulo p, isto é, o valor s^{-1} tal que $ss^{-1} \equiv 1 \pmod{p}$. Para este cálculo, utilizamos o Algoritmo Euclidiano Estendido. Em seguida, calculamos o valor

$$s' = (s^{-1})^d \bmod p.$$

Para concluir, a mensagem decriptada m' é obtida como

$$m' = s't \bmod p.$$

Descrevemos abaixo o algoritmo de decriptação, de acordo com os procedimentos que apresentamos.

Algoritmo 5.3: Algoritmo de Decriptação do Método de Criptografia El Gamal

Entrada: Um número primo p, uma chave privada d e uma mensagem encriptada $\widehat{m} = (s, t)$, onde $0 < s, t < p$.

Saída: Uma mensagem $0 < m' < p$.

Instruções:

1. Calcule, utilizando o Algoritmo Euclidiano Estendido, o valor s^{-1} tal que $ss^{-1} \equiv 1 \pmod{p}$. $\#s^{-1}$ é o inverso de s módulo p

2. $s' \leftarrow (s^{-1})^d \bmod p$ $\quad \#s'$ é a forma reduzida de $(s^{-1})^d$ módulo p
3. $m' \leftarrow (s' * t) \bmod p$
4. Retorne m'.

Resta-nos apenas mostrar que este processo de decriptação realmente recupera a mensagem original. Em outras palavras, queremos mostrar que $m' = m$, onde m é a mensagem original. Começamos observando que s^{-1} é o inverso de s módulo p, então $(s^{-1})^d$ é o inverso de s^d módulo p, já que $(s^{-1})^d s^d \equiv (s^{-1}s)^d \equiv 1^d \equiv 1 \pmod{p}$. Mas s é calculado na encriptação como

$$s = g^k \bmod p,$$

de forma que $s^d \equiv (g^k)^d \equiv (g^d)^k \equiv c^k \pmod{p}$. Esta última congruência segue do fato de que, na geração das chaves, a chave pública c é calculada como a forma reduzida do parâmetro g elevado à chave privada d. Concluímos então que $s' \equiv (s^{-1})^d$ é o inverso de c^k módulo p.

Estamos calculando a mensagem decriptada m' como $m' = s't \bmod p$. Uma vez que t é calculado na encriptação como

$$t = mc^k \bmod p,$$

temos que $m' \equiv s't \equiv s'mc^k \pmod{p}$. Como o grupo $U(p)$ é abeliano, podemos comutar a ordem deste último produto, de forma que $m' \equiv s'c^k m \pmod{p}$. Como mostramos acima que s' é o inverso de c^k módulo p, temos que $s'c^k \equiv 1 \pmod{p}$. Assim, obtemos finalmente que $m' \equiv m \pmod{p}$. Entretanto, temos que $0 < m, m' < p$. Assim, se ambas as mensagens m e m' são congruentes módulo p, então elas são necessariamente iguais, isto é, $m = m'$. Desta forma, o algoritmo de decriptação efetivamente recupera a mensagem original.

Exemplo 5.3. *Vamos supor que não conhecemos a mensagem original que foi encriptada no exemplo anterior. Conhecendo apenas a mensagem encriptada* (115, 103) *e a chave de decriptação* $d = 21$*, vamos recuperar a mensagem original aplicando o algoritmo de decriptação.*

Primeiramente, podemos observar que a chave privada $d = 21$ *é efetivamente a chave privada correspondente à chave pública* $c = 3$*, já que* $g^d \equiv 77^2 1 \equiv 3 \equiv c \pmod{151}$*.*

Para decriptar a mensagem $(s, t) = (115, 103)$*, começamos calculando o inverso de* s *módulo* $p = 151$ *através do Algoritmo Euclidiano Estendido.*

R	Q	α	β
115	–	1	0
151	–	0	1
115	0	1	0
36	1	-1	1
7	3	4	-3
1	5	-21	16
0	7	–	–

Assim, -21 *é o inverso de* 115 *módulo* 151*. Colocando* -21 *na forma reduzida, temos* $s^{-1} = 130$*.*

Calculamos agora s'*, que é a forma reduzida de* $(s^{-1})^d$ *módulo* 151*. Temos* $(s^{-1})^d \equiv 130^2 1 \equiv 60 \pmod{151}$*, de modo que* $s' = 60$*.*

Finalmente, calculamos m'*, a mensagem recuperada, como a forma reduzida de* $s't$ *módulo* 151*. Temos* $s't \equiv 60.103 \equiv 140 \pmod{151}$*, de modo que* $m' = 140$*, que é justamente a mensagem original que encriptamos no exemplo anterior.*

5.3 Assinatura Digital El Gamal

O método El Gamal para assinatura digital foi desenvolvido por Taher El Gamal e apresentado originalmente no mesmo artigo do método de criptografia El Gamal [5]. Este método de assinatura digital também é baseado no uso de grupos abelianos finitos cíclicos, de forma análoga ao método de criptografia El Gamal. Em particular, este método de assinatura digital também realiza os cálculos em grupos $U(p)$, com p primo.

O algoritmo de geração de chaves da assinatura El Gamal é praticamente idêntico ao algoritmo de geração de chaves da criptografia El Gamal. A distinção entre os dois casos que precisa ser salientada é a inversão dos papéis do remetente e do destinatário das mensagens no processo de geração das chaves. Na criptografia, a geração das chaves deve ser feita por parte do destinatário das mensagens, que manterá sob seu conhecimento privado a chave de decriptação. Já no caso da assinatura digital, tanto a geração das chaves quanto a posse da chave privada ficam a cargo do remetente, uma vez que ele utilizará a chave privada para assinar suas mensagens, isto é, certificar sua origem. A chave pública de verificação poderá ser utilizada por qualquer destinatário para se certificar de que uma dada mensagem foi efetivamente criada pelo remetente.

Apresentamos abaixo o algoritmo de geração das chaves, de acordo com os mesmos procedimentos utilizados na criptografia El Gamal. Em particular, também no caso da assinatura digital, o cálculo da chave privada a partir dos dados públicos envolve resolver uma instância do Problema do Logaritmo Discreto.

Algoritmo 5.4: Algoritmo de Geração de Chaves do Método de Assinatura Digital El Gamal

Entrada: Dois inteiros n_1 e n_2.

Saída: Os parâmetros públicos p e g, onde $n_1 \leq p \leq n_2$ é primo e g é uma raiz primitiva de $U(p)$, a chave privada a de assinatura e a chave pública v de verificação.

Instruções:

1. Selecione um primo p tal que $n_1 \leq p \leq n_2$.
2. Calcule, utilizando o Algoritmo de Gauss, uma raiz primitiva g de $U(p)$.
3. Selecione a no intervalo $1 < a < p - 1$.
4. $v \leftarrow g^a \bmod p$ $\quad \#v$ é a forma reduzida de g^a módulo p
5. Retorne p, g, a e v.

Prosseguimos agora com o algoritmo de assinatura do método El Gamal. A mensagem a ser assinada pelo algoritmo deve estar em um formato específico. Ela deve ser uma mensagem escrita no formato de uma sequência de bits, isto é, uma sequência de 0's e 1's. A mensagem pode ter um comprimento arbitrário (embora finito, obviamente), desde que esteja neste formato. O conjunto de todas as mensagens neste formato é denotado por $\{0,1\}^*$.

Como todos os cálculos dos algoritmos de assinatura e verificação são relacionados ao grupo $U(p)$, precisamos de uma maneira de mapear as mensagens que queremos assinar a elementos de $U(p)$ ou $\mathbb{Z}_p$. Uma maneira de fazer isto é utilizar uma função $h : \{0,1\}^* \to \mathbb{Z}_p$ que, para cada mensagem no formato de uma

sequência de bits, atribui um elemento do conjunto $\mathbb{Z}_p$. Uma função neste formato é chamada de *função hash*. Como o conjunto $\{0,1\}^*$ é infinito e o conjunto $\mathbb{Z}_p$ é finito, uma função *hash* claramente não é injetiva, o que significa que existirão mensagens $m \neq m'$ tais que $h(m) = h(m')$.

Para que o método de assinatura seja efetivamente seguro contra a possibilidade de terceiros forjarem assinaturas que não sejam detectadas como inválidas pelo destinatário, a função *hash* utilizada deve ser o que chamamos de *função hash criptograficamente segura*.

Definição 5.1. *Uma* função hash criptograficamente segura *é uma função* hash h *que satisfaz as seguintes propriedades:*

1. *Dificuldade de inversão: dado um valor $t \in \mathbb{Z}_p$, é muito difícil encontrar uma mensagem $m \in \{0,1\}^*$ tal que $h(m) = t$. Mesmo no caso em que já se conheça inicialmente uma mensagem $m \in \{0,1\}^*$ tal que $h(m) = t$, ainda é muito difícil encontrar uma segunda mensagem $m' \in \{0,1\}^*$ tal que $m' \neq m$ e $h(m') = t$.*

2. *Resistência a colisões: é muito difícil encontrar duas mensagens $m, m' \in \{0,1\}^*$ tais que $h(m) = h(m')$.*

Não nos aprofundamos no estudo de funções *hash* criptograficamente seguras neste livro. Para mais detalhes sobre elas e diversos exemplos de tais funções, o livro [16] pode ser consultado.

Podemos concluir então que uma função *hash* criptograficamente segura h deverá fazer parte dos parâmetros do método de assinatura do sistema em conjunto com os parâmetros p e g. Esta função será usada tanto no algoritmo de assinatura quanto no algoritmo de verificação. Obviamente, para que uma assinatura possa ser corretamente verificada, a mesma função *hash* que foi utilizada no algoritmo de assinatura deverá ser utilizada no algoritmo de verificação.

Vamos assumir então que temos os parâmetros p e g, a chave privada de assinatura a e uma função *hash* criptograficamente segura $h : \{0,1\}^* \to \mathbb{Z}_p$. Para assinar uma mensagem $m \in \{0,1\}^*$, começamos selecionando *aleatoriamente* um valor k no intervalo $1 < k < p-1$ tal que $\text{mdc}(k, p-1) = 1$. Este valor de k é nossa *chave efêmera* para a assinatura. Dado que $\text{mdc}(k, p-1) = 1$, o valor de k selecionado possui inverso módulo $p-1$.

Assim como no caso da encriptação, é *absolutamente essencial* selecionar um novo valor aleatório de k a cada nova mensagem que queremos assinar. A repetição do mesmo valor de k na assinatura de duas mensagens distintas pode abrir uma séria brecha de segurança, como mostraremos mais adiante. Esta brecha já havia sido percebida pelo próprio El Gamal em seu artigo original [5].

Após a escolha do valor de k, calculamos dois valores r e s, onde

$$r = g^k \bmod p$$

e

$$s = (k^{-1}(h(m) - ar)) \bmod p - 1,$$

onde k^{-1} é o inverso de k módulo $p-1$, isto é, $kk^{-1} \equiv 1 \pmod{p-1}$. É fundamental observar que o cálculo de r é feito módulo p, mas o cálculo de s é feito módulo $p-1$. A assinatura da mensagem será então o par (r, s).

Descrevemos a seguir o algoritmo de assinatura, de acordo com os procedimentos que apresentamos.

Algoritmo 5.5: Algoritmo de Assinatura do Método de Assinatura Digital El Gamal

Entrada: Um número primo p, uma raiz primitiva g de $U(p)$, uma chave privada a, uma mensagem $m \in \{0,1\}^*$ e uma função *hash* criptograficamente segura $h : \{0,1\}^* \to \mathbb{Z}_p$.

Saída: Uma assinatura $A_m = (r, s)$ para a mensagem m.

Instruções:

1. Selecione *aleatoriamente* um valor k no intervalo $1 < k < p-1$ tal que $\text{mdc}(k, p-1) = 1$.
2. $r \leftarrow g^k \bmod p$ $\quad \#r$ é a forma reduzida de g^k módulo p
3. Calcule, utilizando o Algoritmo Euclidiano Estendido, o valor k^{-1} tal que $kk^{-1} \equiv 1 \pmod{p-1}$. $\quad \#k^{-1}$ é o inverso de k módulo $p-1$
4. $s \leftarrow (k^{-1} * (h(m) - ar)) \bmod p-1$
5. $A_m \leftarrow (r, s)$
6. Retorne A_m.

Podemos reparar que o valor da mensagem m não é usado diretamente no cálculo da assinatura, mas sim o valor de $h(m)$. Esta é a razão para a necessidade de h ser criptograficamente segura. Caso h não atenda a este requisito, pode ser possível então reutilizar uma assinatura produzida legitimamente para uma mensagem m para assinar fraudulentamente uma outra mensagem m' onde $h(m') = h(m)$.

Exemplo 5.4. *Vamos realizar um exemplo de assinatura com o método El Gamal. Suponha que os parâmetros escolhidos são* $p = 191$ *e* $g = 57$ *(a raiz primitiva de* $U(191)$ *obtida com o Algoritmo de Gauss). Vamos considerar também que a chave privada que está sendo utilizada é* $a = 5$*. Finalmente, vamos supor que a função* h *que estamos utilizando retorna a forma reduzida da quantidade de bits* 1 *em uma mensagem módulo* 191*. Esta função não é criptograficamente segura, mas sua simplicidade é apropriada para nosso exemplo. Se desejamos assinar a mensagem* $m = 1110111$*, começamos selecionando um valor aleatório de* k *tal que* $mdc(k, p-1) = 1$*. Vamos supor que o valor selecionado é* $k = 9$*. Calculamos então* $r \equiv g^k \equiv 57^9 \equiv 148 \pmod{191}$*. Em seguida, utilizando o Algoritmo Euclidiano Estendido, calculamos o inverso de* k *módulo* $p - 1 = 190$*, obtendo* $k^{-1} = 169$*. Calculamos então* $h(m) = 6 \bmod 191 = 6$ *e, finalmente, calculamos* $s \equiv k^{-1}(h(m) - ar) \equiv 169.(6 - 5.148) \equiv 169.(-734) \equiv 169.26 \equiv 4394 \equiv 24 \pmod{190}$*. Logo, a assinatura da mensagem* $m = 1110111$ *é o par* $(148, 24)$*.*

Vamos agora para a última etapa do método de assinatura digital El Gamal, apresentando o algoritmo de verificação de assinaturas. Dada uma mensagem $m \in \{0,1\}^*$ e uma assinatura $A_m = (r, s)$, precisamos dos parâmetros p e g, da função h e da chave pública v da pessoa ou entidade que supostamente assinou a mensagem. O algoritmo então deverá determinar se a assinatura é válida, tendo sido realmente produzida pelo suposto remetente da mensagem.

Para verificar a assinatura, começamos verificando se r está no intervalo $1 \leq r \leq p-1$. Todas as assinaturas produzidas pelo algoritmo de assinatura acima atendem a este requisito. Por outro lado, existe uma possibilidade de falsificação de assinaturas em que as assinaturas produzidas possuem r fora deste intervalo.

Desta forma, esta primeira verificação do algoritmo impede este tipo de falsificação. Para mais detalhes sobre a necessidade desta verificação para evitar uma potencial brecha de segurança, [16] pode ser consultado.

Em seguida, calculamos dois valores u_1 e u_2, onde

$$u_1 = (v^r r^s) \bmod p$$

e

$$u_2 = g^{h(m)} \bmod p.$$

Temos então que a assinatura é válida se e somente se $u_1 = u_2$.

Descrevemos abaixo o algoritmo de verificação de assinaturas, de acordo com os procedimentos que apresentamos.

Algoritmo 5.6: Algoritmo de Verificação do Método de Assinatura Digital El Gamal

Entrada: Um número primo p, uma raiz primitiva g de $U(p)$, uma chave pública v, uma mensagem $m \in \{0,1\}^*$, uma assinatura $A_m = (r, s)$ para m e uma função *hash* criptograficamente segura $h : \{0,1\}^* \to \mathbb{Z}_p$.

Saída: "Assinatura válida" ou "Assinatura inválida".

Instruções:

1. Verifique se r está no intervalo $1 \leq r \leq p-1$. Se não estiver, retorne "Assinatura inválida".
2. $u_1 \leftarrow (v^r * r^s) \bmod p$
3. $u_2 \leftarrow g^{h(m)} \bmod p$
4. Se $u_1 = u_2$, retorne "Assinatura válida".
5. Senão, retorne "Assinatura inválida".

Podemos reparar que a chave privada a não é utilizada em nenhum momento do processo de verificação da assinatura. Este processo é realizado apenas com os valores públicos, de maneira que qualquer um pode realizar a verificação de assinaturas em mensagens, sendo que a chave utilizada para assinar estas mensagens permanece de posse exclusiva do remetente legítimo.

Resta-nos agora mostrar que este processo de verificação realmente detecta as assinaturas válidas. De maneira a simplificar os cálculos, podemos começar reparando que, se g é uma raiz primitiva de $U(p)$ e se $x \equiv y \pmod{p-1}$, então $g^x \equiv g^y \pmod{p}$. De fato, se $x \equiv y \pmod{p-1}$, então $x = y + (p-1)l$, para algum $l \in \mathbb{Z}$. Então, $g^x \equiv g^{y+(p-1)l} \equiv g^y (g^{p-1})^l \equiv g^y \pmod{p}$, uma vez que $g^{p-1} \equiv 1 \pmod{p}$.

O valor u_1 é calculado como $u_1 = (v^r r^s) \bmod p$. Vamos calcular v^r e r^s separadamente. Temos que $v = g^a \bmod p$, de forma que

$$v^r \equiv (g^a)^r \equiv g^{ar} \pmod{p}.$$

Para o cálculo de r^s, se r e s foram calculados apropriadamente pelo algoritmo de assinatura, então $r = g^k \bmod p$ e $s = (k^{-1}(h(m) - ar)) \bmod p - 1$. Então,

$$r^s \equiv (g^k)^s \equiv g^{ks} \pmod{p}.$$

Como $s \equiv k^{-1}(h(m) - ar) \pmod{p-1}$, então $ks \equiv kk^{-1}(h(m) - ar) \pmod{p-1}$ e

$$g^{ks} \equiv g^{kk^{-1}(h(m)-ar)} \equiv (g^{kk^{-1}})^{h(m)-ar} \pmod{p}.$$

Por outro lado, como $kk^{-1} \equiv 1 \pmod{p-1}$, então $g^{kk^{-1}} \equiv g \pmod{p}$. Logo,

$$r^s \equiv g^{ks} \equiv (g^{kk^{-1}})^{h(m)-ar} \equiv g^{h(m)-ar} \pmod{p}.$$

Desta forma, temos que

$$u_1 \equiv v^r r^s \equiv g^{ar} g^{h(m)-ar} \equiv g^{ar+h(m)-ar} \equiv g^{h(m)} \equiv u_2 \pmod{p}.$$

Dado que, no algoritmo de verificação, u_1 e u_2 são calculados como formas reduzidas módulo p, então $0 \leq u_1, u_2 < p$. Assim, se eles são congruentes entre si, eles são necessariamente iguais. Desta forma, o teste realizado pelo algoritmo de verificação é correto em determinar a validade de uma assinatura.

Exemplo 5.5. *Vamos aplicar o algoritmo de verificação para testar a validade da assinatura que produzimos no exemplo anterior. Temos a mensagem* $m = 1110111$, $A_m = (r, s) = (148, 24)$, $h(m) = 6$, $p = 191$, $g = 57$ *e* $v = g^a \bmod p = 57^5 \bmod 191 = 37$. *Calculamos então*

$$u_1 \equiv v^r r^s \equiv 37^{148}.148^{24} \equiv 36.170 \equiv 8 \pmod{191}$$

e

$$u_2 \equiv g^{h(m)} \equiv 57^6 \equiv 8 \pmod{191}.$$

Como $u_1 = u_2$, *a assinatura é válida.*

Encerramos esta seção mostrando a brecha de segurança que se abre quando o mesmo valor de k é utilizado na assinatura de duas mensagens distintas m_1 e m_2, onde $m_1 \neq m_2$.

Seja $A_{m_1} = (r_1, s_1)$ a assinatura produzida para m_1 e $A_{m_2} = (r_2, s_2)$ a assinatura produzida para m_2. Pelo modo como uma assinatura é calculada no algoritmo de assinatura, temos que

$$\begin{cases} r_1 &= g^k \bmod p \\ r_2 &= g^k \bmod p, \end{cases}$$

já que o mesmo valor de k foi usado nos dois processos de assinatura. Desta forma, $r_1 = r_2$. Uma vez que temos esta igualdade, denotaremos então os dois por r daqui em diante. Vemos então que se duas assinaturas possuem o primeiro componente igual, isto é um indicador de que o mesmo valor de k foi utilizado para produzir ambas.

Por outro lado, temos

$$\begin{cases} s_1 &= k^{-1}(h(m_1) - ar) \bmod p - 1 \\ s_2 &= k^{-1}(h(m_2) - ar) \bmod p - 1, \end{cases}$$

sendo que estas igualdades podem ser escritas como

$$\begin{cases} ks_1 &\equiv h(m_1) - ar \pmod{p-1} \\ ks_2 &\equiv h(m_2) - ar \pmod{p-1}. \end{cases}$$

Uma vez que os dois módulos são iguais, podemos subtrair as duas igualdades, obtendo

$$k(s_1 - s_2) \equiv (h(m_1) - h(m_2)) \pmod{p-1}.$$

Se $\text{mdc}(s_1 - s_2, p-1) = 1$, o que possui uma chance realista de ocorrer, podemos calcular $(s_1 - s_2)^{-1}$, o inverso de $(s_1 - s_2)$ módulo $p-1$ e obter

$$k \equiv (s_1 - s_2)^{-1}(h(m_1) - h(m_2) \pmod{p-1}.$$

Com o valor de k encontrado, podemos calcular

$$ar \equiv h(m_1) - ks_1 \pmod{p-1}.$$

Se $\text{mdc}(r, p-1) = 1$, o que também possui uma chance realista de ocorrer, podemos calcular r^{-1}, o inverso de r módulo $p-1$ e obter

$$a \equiv r^{-1}(h(m_1) - ks_1) \pmod{p-1}.$$

Como a chave privada é um valor no intervalo $1 < a < p-1$, esta última congruência nos permite calcular a chave privada como $a = (r^{-1}(h(m_1) - ks_1)) \bmod p-1$. Com o conhecimento da chave privada, qualquer pessoa pode se fazer passar pelo remetente original, assinando mensagens que serão validadas pelo algoritmo de verificação.

Entretanto, mesmo que $\text{mdc}(r, p-1) \neq 1$ e não possamos calcular a chave privada diretamente, ainda temos informação suficiente para forjar assinaturas que serão aceitas pelo algoritmo de verificação. Como obtivemos o valor de k e conhecemos r, temos agora os dois valores que satisfazem $r = g^k \bmod p$. Podemos também calcular k^{-1}, o inverso de k módulo $p-1$. Finalmente, também já calculamos acima o valor de ar módulo $p-1$.

Se queremos forjar uma assinatura válida para uma mensagem $\widetilde{m}$, precisamos obter um valor $\widetilde{s}$ de forma que o par $(r, \widetilde{s})$ seja uma assinatura válida para $\widetilde{m}$ de acordo com a chave privada de assinatura a (mesmo sem saber precisamente o valor desta chave). Pela fórmula do algoritmo de geração de assinatura, temos que

$$\widetilde{s} = k^{-1}(h(\widetilde{m}) - ar) \bmod p-1.$$

Como conhecemos k^{-1}, h e o valor do produto ar módulo $p-1$, podemos calcular $\widetilde{s}$, forjando uma assinatura que será considerada como válida pelo algoritmo de verificação.

Podemos concluir então que é realmente *essencial* escolher um novo valor aleatório para k a cada nova assinatura.

Exemplo 5.6. *Suponha que temos o parâmetro* $p = 191$*, uma mensagem* m_1 *com* $h(m_1) = 8$ *e assinatura* $A_{m_1} = (r_1, s_1) = (105, 143)$ *e uma mensagem* m_2 *com* $h(m_2) = 15$ *e assinatura* $A_{m_2} = (r_2, s_2) = (105, 40)$*. Como* $r_1 = r_2$*, sabemos que o mesmo valor de k foi utilizado em ambas as assinaturas. Logo, podemos utilizar o ataque descrito acima para forjar novas assinaturas que serão consideradas como válidas pelo algoritmo de verificação.*

De acordo com o ataque acima, temos $k(s_1 - s_2) \equiv h(m_1) - h(m_2) \pmod{p-1}$*. Logo,* $k(143 - 40) \equiv 8 - 15 \pmod{190}$*, o que é equivalente a* $103k \equiv -7 \pmod{190}$*. Calculando o inverso de* 103 *módulo* 190*, obtemos* $103^{-1} \equiv 107 \pmod{190}$*. Assim* $k \equiv -7.107 \equiv 11 \pmod{190}$*. Tendo o valor de k e sabendo, pela fórmula da assinatura, que* $ks_1 \equiv h(m_1) - ar \pmod{p-1}$*, temos que* $ar \equiv h(m_1) - ks_1 \equiv 8 - 11.143 \equiv 145 \pmod{190}$*.*

Como $mdc(r, p-1) \neq 1$*, não podemos calcular explicitamente o valor da chave privada a a partir do valor de ar. Entretanto, este valor já é suficiente para forjarmos novas assinaturas. Suponha que queiramos forjar uma assinatura para uma*

mensagem m_3 com $h(m_3) = 13$. Iremos utilizar o mesmo valor de r presente nas assinaturas de m_1 e m_2, isto é, $r = 105$. Vamos agora calcular o valor de s pela fórmula da assinatura:

$$s \equiv k^{-1}(h(m_3) - ar) \pmod{p-1}.$$

Como conhecemos k, $h(m_3)$, ar e $p-1$, este cálculo pode ser feito diretamente.

$$s \equiv 11^{-1}(13 - 145) \equiv 121(13 - 145) \equiv 178 \pmod{190}.$$

Logo, a assinatura para m_3 é o par $(105, 178)$.

5.4 Assinatura Digital DSA

O método de assinatura digital DSA, abreviatura de "*Digital Signature Algorithm*" é uma variação do método de assinatura El Gamal. Ele foi desenvolvido pelo "*National Institute of Standards and Technology*" (NIST), órgão governamental dos Estados Unidos, tendo sido publicado no documento FIPS 186 deste instituto [18] e patenteado pelo governo norte-americano [15].

O algoritmo de geração de chaves do método DSA é muito semelhante ao do método El Gamal. A única diferença importante é que, ao invés de um único primo p, o DSA trabalha com dois parâmetros p e q, onde p e q são primos e $p = tq + 1$, para algum inteiro $t \geq 2$.

A ideia por trás disto é que a maioria dos cálculos seja feito em um subgrupo de ordem q de $U(p)$. Desta forma, busca-se aumentar a velocidade dos processos de assinatura e verificação para os usuários legítimos do sistema mantendo-se a segurança contra tentativas de obtenção da chave privada de assinatura por parte de terceiros. Isto ocorre porque os cálculos de assinatura e verificação são feitos com elementos de um subgrupo de ordem "menor" (ordem q) enquanto a instância do Problema do Logaritmo Discreto que deve ser resolvida para o cálculo da chave privada por alguém tentando quebrar o sistema é descrita no grupo de ordem "maior" ($U(p)$).

Como a ordem de $U(p)$ é $p-1$ e q divide $p-1$, então, pelo Teorema de Lagrange (Teorema 4.2), pode existir um subgrupo de ordem q em $U(p)$. Para mostrar que um subgrupo de ordem q efetivamente existe, basta encontrarmos um elemento de ordem q em $U(p)$. As potências deste elemento irão então gerar um subgrupo cíclico de ordem q de $U(p)$.

Para encontrarmos este elemento de ordem q, começamos calculando uma raiz primitiva g' de $U(p)$ com o Algoritmo de Gauss. Em seguida, calculamos o elemento $g = (g')^{(p-1)/q} \bmod p$. Temos que $g^q \equiv (g')^{p-1} \equiv 1 \pmod{p}$. Logo, pelo Lema Chave (Lema 4.2), a ordem de g em $U(p)$ divide q. Mas como q é primo, seus únicos divisores são 1 e p. A ordem de g não pode ser 1, pois isto implicaria que $1 \equiv g \equiv (g')^{(p-1)/q} \pmod{p}$. Entretanto, $(p-1)/q < (p-1)$ e $p-1$ é o menor inteiro positivo i tal que $(g')^i \equiv 1 \pmod{p}$ (já que g' é raiz primitiva de $U(p)$). Desta forma, a ordem de g é q e g é um gerador de um subgrupo cíclico de ordem q de $U(p)$.

O algoritmo de geração de chaves começa selecionando os valores do parâmetros p e q, selecionando o primo q em um intervalo $n_1 < q < n_2$ dado como entrada e selecionando $p = tq+1$ para algum inteiro $t \geq 2$, de forma que p também seja primo. Em seguida, o algoritmo encontra um elemento g de ordem q em $U(p)$, calculando

uma raiz primitiva g' de $U(p)$ com o Algoritmo de Gauss e, na sequência, obtendo $g = (g')^{(p-1)/q} \bmod p$.

O algoritmo então gera o par de chaves, selecionando como chave privada de assinatura um inteiro a no intervalo $1 < a < q$ e calculando a chave pública v de verificação correspondente como $v = g^a \bmod p$.

Descrevemos abaixo o algoritmo de geração das chaves, de acordo com os procedimentos que apresentamos.

Algoritmo 5.7: Algoritmo de Geração de Chaves do Método de Assinatura Digital DSA

Entrada: Dois inteiros n_1 e n_2.

Saída: Os parâmetros públicos p, q e g, onde p e q são primos, $n_1 \leq q \leq n_2$, q divide $p-1$ e g é um elemento de ordem q de $U(p)$, a chave privada a de assinatura e a chave pública v de verificação.

Instruções:

1. Selecione um primo q tal que $n_1 \leq p \leq n_2$.
2. Calcule $p = tq + 1$, para algum inteiro $t \geq 2$, de forma que p seja primo.
3. Calcule, utilizando o Algoritmo de Gauss, uma raiz primitiva g' de $U(p)$.
4. $g \leftarrow (g')^{(p-1)/q} \bmod p$
5. Selecione a no intervalo $1 < a < q$.
6. $v \leftarrow g^a \bmod p$ $\quad \#v$ é a forma reduzida de g^a módulo p
7. Retorne p, q, g, a e v.

Prosseguimos agora com o algoritmo de assinatura do método DSA. A mensagem m a ser assinada pelo algoritmo deve estar no mesmo formato das mensagens assinadas pelo método El Gamal, isto é, $m \in \{0,1\}^*$. De maneira análoga, o algoritmo do método DSA também faz uso de uma função *hash* criptograficamente segura $h : \{0,1\}^* \to \mathbb{Z}_q$. É importante salientar que a imagem $h(m)$ da mensagem é um elemento de $\mathbb{Z}_q$, onde q é o primo "menor" dentro do par p e q.

Vamos assumir então que temos os parâmetros p, q e g, a chave privada de assinatura a e uma função *hash* criptograficamente segura h. Para assinar uma mensagem $m \in \{0,1\}^*$, começamos selecionando *aleatoriamente* um valor k no intervalo $1 < k < q$. Este valor de k é a nossa *chave efêmera*. Novamente, também no caso do DSA é *absolutamente essencial* selecionar um novo valor aleatório de k a cada nova mensagem que queremos assinar. A repetição do mesmo valor de k na assinatura de duas mensagens distintas abre a mesma brecha de segurança que estudamos no caso da assinatura El Gamal.

Após a escolha do valor de k, calculamos dois valores r e s, onde

$$r = (g^k \bmod p) \bmod q$$

e

$$s = (k^{-1}(h(m) + ar)) \bmod q,$$

onde k^{-1} é o inverso de k módulo q. Como q é primo e $1 < k < q$, para qualquer valor selecionado de k teremos $\text{mdc}(k, q) = 1$, isto é, qualquer valor de k terá inverso módulo q. A assinatura da mensagem será então o par (r, s).

É interessante notar a dupla redução modular feita no cálculo de r. Primeiramente, calculamos a forma reduzida de g^k módulo o primo "maior" (p) e em seguida calculamos a forma reduzida deste resultado módulo o primo "menor" (q). Também é importante perceber uma pequena diferença entre as fórmulas para o cálculo de s no algoritmo do método El Gamal e no algoritmo do método DSA. No algoritmo do El Gamal, temos a expressão $h(m) - ar$, enquanto no algoritmo do DSA temos $h(m) + ar$.

Descrevemos abaixo o algoritmo de assinatura, de acordo com os procedimentos que apresentamos.

Algoritmo 5.8: Algoritmo de Assinatura do Método de Assinatura Digital DSA

Entrada: Números primos p e q tais que q divide $p-1$, uma elemento g de ordem q de $U(p)$, uma chave privada a, uma mensagem $m \in \{0,1\}^*$ e uma função *hash* criptograficamente segura $h : \{0,1\}^* \to \mathbb{Z}_q$.

Saída: Uma assinatura $A_m = (r, s)$ para a mensagem m.

Instruções:

1. Selecione *aleatoriamente* um valor k no intervalo $1 < k < q$.
2. $r \leftarrow (g^k \bmod p) \bmod q$
3. Se $r = 0$, reinicie o algoritmo, selecionando um novo valor k.
4. Calcule, utilizando o Algoritmo Euclidiano Estendido, o valor k^{-1} tal que $kk^{-1} \equiv 1 \pmod q$. $\#k^{-1}$ é o inverso de k módulo q
5. $s \leftarrow (k^{-1} * (h(m) + ar)) \bmod q$
6. Se $s = 0$, reinicie o algoritmo, selecionando um novo valor k.
7. $A_m \leftarrow (r, s)$
8. Retorne A_m.

Exemplo 5.7. *Vamos realizar um exemplo de assinatura com o método DSA. Suponha que os primos escolhidos são* $q = 131$ *e* $p = 2q + 1 = 263$. *O Algoritmo de Gauss nos dá* $g' = 259$ *como uma raiz primitiva de* $U(263)$. *Calculamos então* $g = (g')^{(p-1)/q} \bmod p = 259^2 \bmod 263 = 16$. *Vamos assinar novamente a mensagem* $m = 1110111$, *com a mesma função* h *que utilizamos nos exemplos anteriores. Logo, temos que* $h(m) = 6$. *Vamos considerar que a chave privada que está sendo utilizada é* $a = 7$ *e que o valor de* k *selecionado é* $k = 10$. *Calculamos então* $r = (g^k \bmod p) \bmod q = (16^{10} \bmod 263) \bmod 131 = 25 \bmod 131 = 25$. *Em seguida, utilizando o Algoritmo Euclidiano Estendido, calculamos o inverso de* k *módulo* q, *obtendo* $k^{-1} = 118$. *Finalmente, calculamos* $s \equiv k^{-1}(h(m)+ar) \equiv 118(6+7.25) \equiv 5 \pmod{131}$. *Logo, a assinatura da mensagem* $m = 1110111$ *é o par* $(25, 5)$.

Vamos agora para a última etapa do método DSA, apresentando o algoritmo de verificação de assinaturas. Dada uma mensagem $m = \{0,1\}^*$ e uma assinatura $A_m = (r, s)$, começamos verificando se r e s estão no intervalo $0 < r, s < q$. Toda assinatura produzida de acordo com o algoritmo irá satisfazer estes critérios e, analogamente ao caso da assinatura El Gamal, com este teste evitamos alguns métodos de falsificação de assinaturas. Em seguida, utilizando o Algoritmo Euclidiano Estendido, calculamos s^{-1}, o inverso de s módulo q. Como q é primo e $0 < s < q$, temos que $\text{mdc}(s, q) = 1$, o que significa que este inverso existe.

No próximo passo, calculamos três valores u_1, u_2 e u_3, onde

$$u_1 = (s^{-1}h(m)) \bmod q,$$

$$u_2 = (rs^{-1}) \bmod q$$

e

$$u_3 = ((g^{u_1}v^{u_2}) \bmod p) \bmod q.$$

Temos então que a assinatura é válida se e somente se $u_3 = r$.

Descrevemos abaixo o algoritmo de verificação de assinaturas, de acordo com os procedimentos que apresentamos.

Algoritmo 5.9: Algoritmo de Verificação do Método de Assinatura Digital DSA

Entrada: Números primos p e q, tais que q divide $p-1$, uma elemento g de ordem q de $U(p)$, uma chave pública v, uma mensagem $m \in \{0,1\}^*$, uma assinatura $A_m = (r,s)$ para m e uma função *hash* criptograficamente segura $h : \{0,1\}^* \to \mathbb{Z}_q$.

Saída: "Assinatura válida" ou "Assinatura inválida".

Instruções:

1. Verifique se r está no intervalo $0 < r < q$. Se não estiver, retorne "Assinatura inválida".
2. Verifique se s está no intervalo $0 < s < q$. Se não estiver, retorne "Assinatura inválida".
3. Calcule, utilizando o Algoritmo Euclidiano Estendido, o valor s^{-1} tal que $ss^{-1} \equiv 1 \pmod q$. $\#s^{-1}$ é o inverso de s módulo q
4. $u_1 \leftarrow (s^{-1} * h(m)) \bmod q$
5. $u_2 \leftarrow (r * s^{-1}) \bmod q$
6. $u_3 \leftarrow ((g^{u_1} * v^{u_2}) \bmod p) \bmod q$
7. Se $u_3 = r$, retorne "Assinatura válida".
8. Senão, retorne "Assinatura inválida".

Novamente, podemos reparar que a chave privada a não é utilizada em nenhum momento do processo de verificação da assinatura. Resta-nos mostrar que este processo realmente detecta as assinaturas válidas.

Inicialmente, temos que se g é um elemento de ordem q de $U(p)$ e $x \equiv y \pmod q$, então $g^x \equiv g^y \pmod p$. De fato, se $x \equiv y \pmod q$, então $x = y + ql$, para algum $l \in \mathbb{Z}$. Então $g^x \equiv g^{y+ql} \equiv g^y(g^q)^l \equiv g^y \pmod p$, uma vez que $g^q \equiv 1 \pmod p$.

Vamos calcular a forma reduzida de $g^{u_1}v^{u_2}$ módulo p. Como $u_1 \equiv s^{-1}h(m) \pmod q$ e $u_2 \equiv rs^{-1} \pmod q$, então

$$g^{u_1}v^{u_2} \equiv g^{s^{-1}h(m)}v^{rs^{-1}} \pmod p.$$

Por outro lado, como $v \equiv g^a \pmod p$, temos

$$g^{u_1}v^{u_2} \equiv g^{s^{-1}h(m)}g^{ars^{-1}} \equiv (g^{s^{-1}})^{(h(m)+ar)} \pmod p.$$

Temos que $s \equiv k^{-1}(h(m) + ar) \pmod{q}$, o que é equivalente a $ks \equiv h(m) + ar \pmod{q}$. Assim,

$$g^{u_1} v^{u_2} \equiv (g^{s^{-1}})^{(h(m)+ar)} \equiv (g^{s^{-1}})^{ks} \equiv (g^{ss^{-1}})^k \pmod{p}.$$

Finalmente, como $ss^{-1} \equiv 1 \pmod{q}$, obtemos

$$g^{u_1} v^{u_2} \equiv (g^{ss^{-1}})^k \equiv g^k \pmod{p}.$$

Concluímos então que a forma reduzida de $g^{u_1} v^{u_2}$ módulo p é igual a forma reduzida de g^k módulo p, isto é, $(g^{u_1} v^{u_2}) \bmod p = g^k \bmod p$. Assim, se realizarmos uma redução módulo q em ambos os lados desta última igualdade, obtemos $((g^{u_1} v^{u_2}) \bmod p) \bmod q = (g^k \bmod p) \bmod q$. Mas agora o lado esquerdo é igual a u_3 e o lado direito é igual a r, o que significa que temos $u_3 = r$. Desta forma, o teste realizado pelo algoritmo de verificação é correto em determinar a validade de uma assinatura.

Exemplo 5.8. *Vamos aplicar o algoritmo de verificação para testar a validade da assinatura que produzimos no exemplo anterior. Temos a mensagem* $m = 1110111$, $A_m = (r, s) = (25, 5)$, $h(m) = 6$, $p = 263$, $q = 131$, $g = 16$ *e* $v = g^a \bmod p = 16^7 \bmod 263 = 35$. *Começamos calculando* s^{-1}, *o inverso de* s *módulo* q, *obtendo* $s^{-1} = 105$. *Calculamos então*

$$u_1 = s^{-1} h(m) \bmod q = 105.6 \bmod 131 = 106,$$

$$u_2 = rs^{-1} \bmod q = 25.105 \bmod 131 = 5$$

e

$$u_3 = ((g^{u_1} v^{u_2}) \bmod p) \bmod q = ((16^{106}.35^5) \bmod 263) \bmod 131 =$$
$$= 25 \bmod 131 = 25.$$

Como $u_3 = r$, *então a assinatura é válida.*

No caso da assinatura DSA, também podemos mostrar que a repetição de um mesmo valor de k na assinatura de duas mensagens distintas abre uma brecha na segurança do método da mesma forma que estudamos no caso do método El Gamal. Entretanto, como o desenvolvimento do ataque é muito semelhante ao que fizemos no caso do El Gamal, deixamos o caso do método DSA como exercício (Exercício 7).

5.5 Exercícios

1. Dadas as mensagens, os parâmetros, as chaves públicas e as chaves efêmeras abaixo, encripte as mensagens com o método El Gamal.

   1.1. $p = 107$, $g = 103$, $c = 16$, $m = 38$, $k = 2$

   1.2. $p = 163$, $g = 159$, $c = 99$, $m = 89$, $k = 3$

   1.3. $p = 211$, $g = 155$, $c = 57$, $m = 102$, $k = 4$

   1.4. $p = 241$, $g = 14$, $c = 100$, $m = 191$, $k = 5$

2. Dadas as mensagens encriptadas, o parâmetro p e as chaves privadas a seguir, decripte as mensagens levando em conta que elas foram encriptadas com o El Gamal.

2.1. $p = 107$, $d = 5$, $\widehat{m} = (43, 39)$

2.2. $p = 163$, $d = 9$, $\widehat{m} = (117, 67)$

2.3. $p = 211$, $d = 12$, $\widehat{m} = (192, 129)$

2.4. $p = 241$, $d = 19$, $\widehat{m} = (32, 134)$

3. Dados os valores da função h para as mensagens, os parâmetros, as chaves privadas e as chaves efêmeras abaixo, produza assinaturas com o método El Gamal.

3.1. $p = 107$, $g = 103$, $a = 5$, $h(m) = 21$, $k = 3$

3.2. $p = 163$, $g = 159$, $a = 10$, $h(m) = 1$, $k = 5$

3.3. $p = 211$, $g = 155$, $a = 17$, $h(m) = 69$, $k = 11$

3.4. $p = 241$, $g = 14$, $a = 22$, $h(m) = 97$, $k = 7$

4. Dados os valores da função h para as mensagens, os parâmetros, as chaves públicas e as assinaturas abaixo, verifique se estas assinaturas são válidas levando em conta que elas foram produzidas com o El Gamal.

4.1. $p = 107$, $g = 103$, $v = 43$, $h(m) = 31$, $A_m = (46, 21)$

4.2. $p = 163$, $g = 159$, $v = 117$, $h(m) = 5$, $A_m = (12, 125)$

4.3. $p = 211$, $g = 155$, $v = 182$, $h(m) = 14$, $A_m = (57, 10)$

4.4. $p = 241$, $g = 14$, $v = 32$, $h(m) = 50$, $A_m = (105, 30)$

5. Dados os valores da função h para as mensagens, os parâmetros, as chaves privadas e as chaves efêmeras abaixo, produza assinaturas com o método DSA.

5.1. $p = 167$, $q = 83$, $g = 16$, $a = 5$, $h(m) = 21$, $k = 3$

5.2. $p = 179$, $q = 89$, $g = 173$, $a = 10$, $h(m) = 1$, $k = 5$

5.3. $p = 227$, $q = 113$, $g = 221$, $a = 17$, $h(m) = 69$, $k = 11$

5.4. $p = 347$, $q = 173$, $g = 341$, $a = 22$, $h(m) = 97$, $k = 7$

6. Dados os valores da função h para as mensagens, os parâmetros, as chaves públicas e as assinaturas abaixo, verifique se estas assinaturas são válidas levando em conta que elas foram produzidas com o DSA.

6.1. $p = 167$, $q = 83$, $g = 16$, $v = 89$, $h(m) = 31$, $A_m = (5, 69)$

6.2. $p = 179$, $q = 89$, $g = 173$, $v = 142$, $h(m) = 5$, $A_m = (11, 63)$

6.3. $p = 227$, $q = 113$, $g = 221$, $v = 161$, $h(m) = 14$, $A_m = (20, 39)$

6.4. $p = 347$, $q = 173$, $g = 341$, $v = 205$, $h(m) = 50$, $A_m = (38, 24)$

7. Descreva o ataque que podemos realizar ao método DSA quando um mesmo valor da chave efêmera k é utilizado na assinatura de duas mensagens distintas, mostrando que este ataque permite forjar assinaturas para novas mensagens.

Capítulo 6

Resolução do Problema do Logaritmo Discreto

Encerrando o livro, apresentamos neste capítulo alguns algoritmos para a resolução do Problema do Logaritmo Discreto em grupos finitos cíclicos.

Este estudo é importante, pois, apesar da resolução do Problema do Logaritmo Discreto ser computacionalmente difícil no caso geral, existem casos particulares em que os algoritmos conhecidos são eficientes. Desta forma, é importante o conhecimento destes algoritmos e a análise de em quais casos eles são eficientes para que, ao implementarmos um método de criptografia ou assinatura digital baseado em grupos, como o El Gamal e o DSA, possamos contornar e evitar tais casos, já que estes são justamente os casos em que a segurança dos métodos estará comprometida.

Neste capítulo, estudaremos o chamado algoritmo ingênuo, também conhecido como algoritmo de busca exaustiva ou algoritmo de força bruta, o Algoritmo "Baby-Step/Giant-Step" de Shanks, o Algoritmo "Rho" de Pollard, o Algoritmo de Pohlig-Hellman e o Algoritmo do Cálculo de Índices. Estes algoritmos, em conjunto com algumas variantes, compõem a maioria dos métodos conhecidos para a resolução do Problema do Logaritmo Discreto, de forma que estaremos fazendo um estudo bastante completo deste tópico.

6.1 Algoritmo Ingênuo

Nesta seção, apresentamos o chamado algoritmo ingênuo para a resolução do Problema do Logaritmo Discreto. Este algoritmo também é conhecido com algoritmo de busca exaustiva ou algoritmo de força bruta.

Antes, porém, vamos relembrar a definição do Problema do Logaritmo Discreto, tanto no caso geral de um grupo finito cíclico $\mathcal{G} = (G, *)$ qualquer quanto no caso particular dos grupos $U(p)$, com p primo, que são os grupos que utilizamos nos métodos El Gamal e DSA.

Definição 6.1 (Problema do Logaritmo Discreto Genérico)**.** *Definimos o Problema do Logaritmo Discreto, abreviado como PLD, da seguinte forma: dados um grupo finito cíclico $\mathcal{G} = (G, *)$ de ordem n, um gerador g de $\mathcal{G}$ e um elemento $h \in G$, queremos determinar o valor de x no intervalo $0 \leq x < n$ tal que $g^x = h$ em $\mathcal{G}$.*

Definição 6.2 (Problema do Logaritmo Discreto em $U(p)$)**.** *No caso particular do grupo $U(p)$, com p primo, o Problema do Logaritmo Discreto pode ser descrito da*

seguinte forma: dados um gerador g de $U(p)$ e um elemento $h \in U(p)$, queremos determinar o valor de x no intervalo $0 \leq x < p-1$ tal que $g^x \equiv h \pmod{p}$.

É importante também relembrarmos como o Problema do Logaritmo Discreto aparece no contexto da segurança dos métodos El Gamal e DSA. No método de criptografia El Gamal, temos um parâmetro público p, que é um primo, outro parâmetro público $g \in U(p)$, que é um gerador de $U(p)$, e uma chave pública c de encriptação. A chave privada d de decriptação pode então ser calculada como a solução da seguinte instância do Problema do Logaritmo Discreto em $U(p)$: $g^x \equiv c \pmod{p}$. Cálculos análogos também podem ser utilizados para o cálculo das chaves privadas nos métodos de assinatura El Gamal e DSA. Assim, em resumo, a chave privada é, em todos estes casos, a solução de uma instância do Problema do Logaritmo Discreto construída a partir da chave pública.

Uma vez que sabemos que o valor de x que é a solução de uma instância do Problema do Logaritmo Discreto está no intervalo $0 \leq x < p-1$ para o grupo $U(p)$ ou no intervalo $0 \leq x < n$ no caso geral de um grupo de ordem n, o que o algoritmo ingênuo faz é testar todos os valores deste intervalo um por um, começando pelo valor 0 e incrementando enquanto for necessário.

Apresentamos abaixo o algoritmo ingênuo para o a resolução do PLD em grupos $U(p)$. A generalização do algoritmo para a resolução do PLD em grupos finitos cíclicos quaisquer, desde que a ordem n do grupo seja conhecida, é bastante simples. Desta forma, ela é deixada como exercício (Exercício 1).

Algoritmo 6.1: Algoritmo Ingênuo

Entrada: Um número primo p, um gerador g de $U(p)$ e um inteiro $1 \leq h < p$.

Saída: Um número inteiro $0 \leq x < p-1$ tal que $g^x \equiv h \pmod{p}$.

Instruções:

1. $i \leftarrow 0$
2. Enquanto $g^i \not\equiv h \pmod{p}$, faça:
   2.1. $i \leftarrow i + 1$
3. Retorne i.

Este algoritmo pode precisar executar, no pior caso, aproximadamente p passos para encontrar a solução. Se p for um primo muito grande, este algoritmo poderá não ser capaz, na prática, de encontrar a solução desejada, pois ele poderá precisar de uma quantidade exorbitante de tempo para concluir sua execução.

Entretanto, é importante notar que apenas a escolha de um primo p muito grande não previne completamente o uso prático deste algoritmo para a quebra dos métodos El Gamal e DSA. Caso a chave privada utilizada seja muito pequena, o algoritmo acima conseguirá encontrá-la em uma quantidade de tempo admissível. Assim, para prevenir o uso do algoritmo acima como ferramenta de quebra dos métodos que estudamos, devemos tomar duas precauções ao implementar tais métodos:

1. Selecionar um primo p muito grande. A recomendação atual é que p possua pelo menos 2048 bits, isto é, aproximadamente 600 algarismos de comprimento [19].

2. Selecionar uma chave privada que não seja muito pequena. Uma sugestão é selecionar chaves privadas que estejam situadas na região central do intervalo $1 < x < p-1$.

6.2 Algoritmo "Baby-Step/Giant-Step" de Shanks

Nesta seção, apresentamos o Algoritmo "*Baby-Step/Giant-Step*", abreviado como BSGS. Este algoritmo foi proposto originalmente por Daniel Shanks no artigo [26].

O Algoritmo BSGS pode ser utilizado para a resolução do PLD em qualquer grupo finito cíclico, não apenas nos grupos $U(p)$. Por esta razão, dizemos que ele é um *algoritmo genérico* para a resolução do PLD. O algoritmo ingênuo da seção anterior também pertence a esta categoria. Entretanto, existe também um segundo grupo de algoritmos que são *específicos* para a resolução do PLD em grupos $U(p)$.

Vamos explicar o funcionamento do Algoritmo BSGS no caso do grupo $U(p)$, pois este é o grupo em que trabalhamos nos métodos de criptografia e assinatura digital que apresentamos. Entretanto, a generalização do algoritmo para grupos finitos cíclicos quaisquer é simples e direta.

Queremos encontrar o valor de x tal que $g^x \equiv h \pmod{p}$, onde conhecemos g, h e p. A única coisa que sabemos inicialmente sobre x é que ele pertence ao intervalo $0 \leq x < p-1$. Para encontrar x, vamos começar calculando $m = \lceil \sqrt{p-1} \rceil = \lfloor \sqrt{p-1} \rfloor + 1$, isto é, a parte inteira de $\sqrt{p-1}$ mais 1. Dividindo x por m, obtemos

$$x = im + j,$$

onde i é o quociente e j o resto da divisão. Assim, temos que $0 \leq j < m$. Como o dividendo x é não-negativo e o divisor m é positivo, temos também $i \geq 0$. Vamos mostrar agora que, devido a esta escolha particular do valor de m, temos $i < m$.

Repare inicialmente que $m > \sqrt{p-1}$, logo $m^2 > p-1$. Suponha, por contradição que $i \geq m$. Então, $x = im + j \geq m^2 > p-1$, o que é uma contradição com o fato de que x está no intervalo $0 \leq x < p-1$. Assim, temos que $0 \leq i, j < m$.

Vamos agora escrever $g^x \equiv h \pmod{p}$ como $g^{im+j} \equiv h \pmod{p}$, que por sua vez é equivalente a

$$g^j \equiv h(g^{-1})^{im} \pmod{p}. \tag{6.2.1}$$

A partir desta congruência, surge a ideia principal do Algoritmo BSGS: ao invés de calcular o valor de x diretamente, vamos calcular primeiramente os valores de i e j e então determinar o valor de x através da igualdade $x = im + j$.

Sabemos que o valor de j está no intervalo $0 \leq j < m$. Assim, construímos uma lista com os valores de $g^j \bmod p$ para cada valor de j neste intervalo:

$$B = [g^j \bmod p : 0 \leq j < m].$$

Esta lista é chamada de lista dos *baby-steps*.

Em seguida, calculamos g^{-1}, o inverso de g módulo p, e $t = (g^{-1})^m \bmod p$. Começamos então o cálculo dos chamados *giant-steps*. Para cada valor de i no intervalo $0 \leq i < m$, calculamos $ht^i \bmod p$ e verificamos se este valor pertence à lista B. Caso não pertença, incrementamos o valor de i e calculamos o próximo *giant-step*. Mas caso este valor esteja na posição j da lista B, então encontramos valores de i e j tais que $g^j \equiv ht^i \pmod{p}$, que é a mesma congruência da Equação (6.2.1). Assim, estes são os valores de i e j que buscamos e podemos calcular o valor de x como $x = im + j$.

Apresentamos a seguir o Algoritmo BSGS para a resolução do PLD em grupos $U(p)$. A generalização do algoritmo para grupos finitos cíclicos quaisquer é deixada como exercício (Exercício 1). O ponto menos imediato desta generalização é o cálculo do valor de m para um grupo qualquer. Ao invés de $\lceil \sqrt{p-1} \rceil$, deve ser utilizado $\lceil \sqrt{n} \rceil$, onde n é a ordem do grupo ou um limite superior para esta ordem, caso ela não seja conhecida com precisão.

Em particular, se g' é um gerador de um subgrupo de ordem n de $U(p)$ e h é um elemento deste subgrupo, podemos utilizar o BSGS para calcular o valor de x tal que $(g')^x \equiv h \pmod{p}$ utilizando o valor de $m = \lceil \sqrt{n} \rceil$. Esta ideia será importante na construção do Algoritmo de Pohlig-Hellman, que veremos mais adiante neste capítulo.

Algoritmo 6.2: Algoritmo "Baby-Step/Giant-Step" de Shanks

Entrada: Um número primo p, um gerador g de $U(p)$ e um inteiro $1 \leq h < p$.

Saída: Um número inteiro $0 \leq x < p-1$ tal que $g^x \equiv h \pmod{p}$.

Instruções:

1. $m \leftarrow \lceil \sqrt{p-1} \rceil \quad \#\lfloor \sqrt{p-1} \rfloor + 1$
2. $j \leftarrow 0$
3. $B \leftarrow [\ ] \quad \#$ lista vazia
4. Enquanto $j < m$, faça:
   4.1. $s \leftarrow g^j \bmod p$
   4.2. Adicione s na posição j da lista B.
   4.3. $j \leftarrow j + 1$
5. Calcule, utilizando o Algoritmo Euclidiano Estendido, o inverso de g módulo p. Vamos denotá-lo por g^{-1}.
6. $t \leftarrow (g^{-1})^m \bmod p$
7. $i \leftarrow 0$
8. Enquanto $i < m$, faça:
   8.1. $s \leftarrow (h * t^i) \bmod p$
   8.2. Se $s \in B$, então:
      8.2.1. Seja j a posição de s em B.
      8.2.2. $x \leftarrow (i * m + j)$
      8.2.3. Retorne x.
   8.3. $i \leftarrow i + 1$

Este algoritmo pode precisar, no pior caso, construir duas listas de tamanho aproximadamente $\sqrt{p}$ cada uma. Desta forma, no pior caso, o algoritmo irá executar aproximadamente $2\sqrt{p}$ passos. Embora esta quantidade de passos ainda seja muito elevada, caso p seja muito grande, ela é uma melhora substancial em relação aos p passos que o algoritmo ingênuo pode precisar executar.

Na prática, se p for muito grande, nem o algoritmo ingênuo nem o BSGS serão capazes de encontrar a solução desejada. Entretanto, como a quantidade de passos executada pelo BSGS é substancialmente menor do que a executada pelo algoritmo ingênuo, a seleção de um primo p muito grande torna-se ainda mais crucial para a segurança dos métodos de criptografia e assinatura digital que estudamos.

Exemplo 6.1. *Seja $p = 241$, $g = 14$ (a raiz primitiva de $U(241)$ obtida com o Algoritmo de Gauss) e $h = 65$. Vamos utilizar o Algoritmo BSGS para determinar o valor de x tal que $g^x \equiv h \pmod{p}$.*

Começamos calculando $m = \lceil \sqrt{p-1} \rceil = \lceil \sqrt{240} \rceil = 16$. *Calculamos então a lista* B *dos* baby-steps $g^j \bmod p = 14^j \bmod 241$, *para* $0 \leq j < 16$.

j	0	1	2	3	4	5	6	7
$14^j \bmod 241$	1	14	196	93	97	153	214	104
j	8	9	10	11	12	13	14	15
$14^j \bmod 241$	10	140	32	207	6	84	212	76

Em seguida, utilizando o Algoritmo Euclidiano Estendido, calculamos g^{-1}, *o inverso de* g *módulo* p, *obtendo* $g^{-1} = 155$. *Calculamos então* $t = (g^{-1})^m \bmod p = 155^16 \bmod 241 = 94$. *Finalmente, calculamos um por um os* giant-steps $ht^i \bmod p = 65.94^i$, *para* $0 \leq i < 16$, *até encontrarmos um valor que aparece na lista acima.*

i	0	1	2	3
65.94^i	65	85	37	104
Está em B*?*	*Não*	*Não*	*Não*	*Sim*

Assim, temos $i = 3$. *Por outro lado, o valor* 104 *aparece em* B *na posição* $j = 7$. *Logo,* $x = im + j = 3.16 + 7 = 55$. *De fato,* $14^{55} \equiv 65 \pmod{241}$.

6.3 Algoritmo "Rho" de Pollard

Nesta seção, apresentamos o Algoritmo "Rho", proposto originalmente por John Pollard no artigo [22]. O algoritmo foi batizado de "Rho" porque ele realiza um "percurso" entre os elementos do grupo que consiste de um segmento inicial seguido de um ciclo do qual o percurso não sai. Assim, o percurso se assemelha em formato à letra grega rho (ρ).

Assim como os dois algoritmos apresentados anteriormente, o Algoritmo "Rho" também é um *algoritmo genérico* para a resolução do PLD.

Da mesma maneira que fizemos com os algoritmos anteriores, vamos explicar o funcionamento do Algoritmo "Rho" no caso do grupo $U(p)$. Ao longo da explicação, discutiremos alguns dos detalhes menos imediatos envolvidos na sua generalização para trabalhar com grupos finitos cíclicos quaisquer.

No Algoritmo BSGS, construímos duas listas de elementos de $U(p)$ procurando por um elemento comum entre elas. Este elemento comum nos fornece então informações para obter a resposta que procuramos. O maior problema desta abordagem da maneira como é feita no Algoritmo BSGS é que uma das listas precisa ficar completamente armazenada na memória até o fim do algoritmo. Assim, o Algoritmo BSGS apresenta um consumo de memória elevado.

O Algoritmo "Rho" parte da mesma ideia de construir duas listas buscando um elemento comum entre elas, mas utiliza outra estratégia, que permite que apenas um elemento de cada lista seja armazenado na memória a cada passo. Assim, ao invés de uma lista completa armazenada em memória, o Algoritmo "Rho" armazena apenas dois elementos de cada vez, sendo um de cada lista.

Chamaremos os elementos de uma das listas de y_i e os elementos da outra lista de z_i, com $i \geq 0$. O primeiro elemento de ambas as listas é 1, isto é, $y_0 = z_0 = 1$. Utilizamos então uma função f para calcular o próximo elemento das listas. Se y_i é um elemento da primeira lista, calculamos o próximo elemento desta lista como $y_{i+1} = f(y_i)$. Por outro lado, se z_i é um elemento da segunda lista, calculamos o próximo elemento desta lista como $z_{i+1} = f(f(z_i))$.

Assim, se considerarmos a aplicação de f como um passo dado em um percurso pelos elementos de $U(p)$, a cada passo dado no percurso da primeira lista, dois passos são dados no percurso da segunda lista. Como os dois percursos começaram no mesmo ponto, o número 1, temos então que, para todo i, $z_i = y_{2i}$.

A função f sugerida por Pollard para utilização no Algoritmo "Rho" é a seguinte:

$$f(w) = \begin{cases} gw \bmod p & \text{se} \quad 0 \leq w < p/3, \\ w^2 \bmod p & \text{se} \quad p/3 \leq w < 2p/3, \\ wh \bmod p & \text{se} \quad 2p/3 \leq w < p. \end{cases}$$

Para generalizarmos esta função para a aplicação do Algoritmo "Rho" em grupos finitos cíclicos $\mathcal{G} = (G, *)$ quaisquer, realizamos uma partição do conjunto G em 3 conjuntos disjuntos G_1, G_2 e G_3 e aplicamos a primeira regra da função acima para os elementos de G_1, a segunda regra para os elementos de G_2 e a terceira regra para os elementos de G_3.

Em qualquer etapa do algoritmo, apenas o último elemento de cada lista que foi calculado até o momento é armazenado. Os elementos anteriores são descartados. Pelo formato da função f acima, podemos concluir que um elemento da primeira lista tem sempre o formato $y_i = g^{\gamma_1} h^{\delta_1}$. O mesmo ocorre com os elementos da segunda lista: $z_i = g^{\gamma_2} h^{\delta_2}$.

Estes valores de γ_1, δ_1, γ_2 e δ_2 podem ser calculados em paralelo à aplicação da função f. Inicialmente, temos $\gamma_1 = \delta_1 = \gamma_2 = \delta_2 = 0$, já que o primeiro elemento de ambas as listas é 1. A partir daí, a cada vez que aplicamos a primeira regra de f a um elemento da primeira lista, incrementamos γ_1 em uma unidade. A cada vez que aplicamos a segunda regra de f a um elemento da primeira lista, multiplicamos por dois tanto γ_1 quanto δ_1. Finalmente, a cada vez que aplicamos a terceira regra de f a um elemento da primeira lista, incrementamos δ_1 em uma unidade. Procedimentos análogos são válidos para a aplicação das regras de f aos elementos da segunda lista e a atualização dos valores de γ_2 e δ_2, lembrando que o próximo elemento da segunda lista é obtido após *duas aplicações sucessivas* de f. Podemos então definir duas funções auxiliares Γ, para a atualização de γ_1 e γ_2, e Δ, para a atualização de δ_1 e δ_2:

$$\Gamma(w, \gamma) = \begin{cases} \gamma + 1 \\ 2\gamma \\ \gamma \end{cases} \quad \text{e} \quad \Delta(w, \delta) = \begin{cases} \delta & \text{se } 0 \leq w < p/3, \\ 2\delta & \text{se } p/3 \leq w < 2p/3, \\ \delta + 1 & \text{se } 2p/3 \leq w < p. \end{cases}$$

Calculamos novos elementos das duas listas até chegarmos a um valor de j tal que $y_j = z_j$. Como temos que $z_j = y_{2j}$, temos $y_j = y_{2j}$, isto é, uma indicação de que um elemento da primeira lista irá se repetir. Como o próximo elemento de uma lista é completamente determinado pela aplicação da função f, se o elemento y_j se repete em y_{2j}, então a primeira lista passará a ter repetições dos elementos entre y_j e y_{2j-1} para sempre. Temos então o ciclo no percurso da primeira lista aludido pela letra grega rho (ρ), sendo que este ciclo possui comprimento j.

A partir da igualdade $y_j = z_j$, podemos concluir que $g^{\gamma_1} h^{\delta_1} \equiv g^{\gamma_2} h^{\delta_2} \pmod{p}$, para os valores de γ_1, δ_1, γ_2 e δ_2 calculados no decorrer das aplicações de f. Como $h \equiv g^x \pmod{p}$, onde x é o valor que estamos tentando determinar, a congruência $g^{\gamma_1} h^{\delta_1} \equiv g^{\gamma_2} h^{\delta_2} \pmod{p}$ pode ser escrita como

$$g^{\gamma_1 + x\delta_1} \equiv g^{\gamma_2 + x\delta_2} \pmod{p}.$$

Esta congruência pode ainda ser escrita como $g^{(\gamma_1 - \gamma_2) + x(\delta_1 - \delta_2)} \equiv 1 \pmod{p}$. Então, o Lema Chave (Lema 4.2) nos permite concluir que a ordem de g, que é

$p-1$, divide $(\gamma_1 - \gamma_2) + x(\delta_1 - \delta_2)$. Isto significa que

$$(\gamma_1 - \gamma_2) \equiv (\delta_2 - \delta_1)x \pmod{p-1}.$$

Seja $u = (\gamma_1 - \gamma_2) \bmod p-1$ e $v = (\delta_2 - \delta_1)$. Se $v = 0$, temos que $\delta_1 = \delta_2$, o que implica também em $\gamma_1 = \gamma_2$. Desta forma, temos os mesmos expoentes aparecendo de ambos os lados da congruência $g^{\gamma_1}h^{\delta_1} \equiv g^{\gamma_2}h^{\delta_2} \pmod{p}$, o que não nos fornece informação suficiente para determinar o valor de x. Sendo assim, o algoritmo falha em calcular x no caso em que $v = 0$.

Felizmente, a probabilidade de obtermos $v = 0$ é bem pequena. De qualquer forma, caso este caso ocorra e o algoritmo falhe, podemos repetir a aplicação do algoritmo com um novo elemento inicial diferente de 1 nas duas listas. Para isso, sorteamos aleatoriamente dois valores a e b tais que $0 \leq a, b < p-1$ e fazemos $y_0 = z_0 = (g^a h^b) \bmod p$.

Considerando então que $v \neq 0$, se $\text{mdc}(v, p-1) = 1$, podemos calcular v^{-1}, o inverso de v módulo $p-1$ e obter x como $x = uv^{-1} \bmod p-1$. Entretanto, isto nem sempre acontece. No caso em que $\text{mdc}(v, p-1) = d > 1$, alguns cálculos extras são necessários para obter o valor de x.

Aplicando o Algoritmo Euclidiano Estendido a v e $p-1$, obtemos $\alpha v + \beta(p-1) = d$, que é equivalente a $\alpha v \equiv d \pmod{p-1}$. Por outro lado, temos a congruência $u \equiv vx \pmod{p-1}$. Multiplicando-a dos dois lados por α, obtemos $\alpha u \equiv \alpha vx \pmod{p-1}$. Como $\alpha v \equiv d \pmod{p-1}$, podemos escrever a congruência anterior como $\alpha u \equiv dx \pmod{p-1}$. Então, de forma equivalente, temos $\alpha u = dx + (p-1)l$ para algum $l \in \mathbb{Z}$. Como $d = \text{mdc}(v, p-1)$, temos $p - 1 = dm$, para algum $m \in \mathbb{Z}$. Substituindo o valor de $p-1$ na igualdade acima, obtemos $\alpha u = dx + dml = d(x + ml)$. Esta igualdade nos permite concluir que d divide αu. Assim, temos $\alpha u = dt$, para algum $t \in \mathbb{Z}$, o que nos dá $dt = d(x + ml)$. Como $d \neq 0$, esta última igualdade é equivalente a $t = x + ml$, que pode ser escrita na forma de congruência como $x \equiv t \pmod{m}$.

Desta forma, para obter o valor de x, calculamos $m = (p-1)/d$ e $t = ((\alpha u)/d) \bmod m$. Entretanto, se $d > 1$, existe mais de um valor de x no intervalo $0 \leq x < p-1$ tal que $x \equiv t \pmod{m}$. De fato, existem d possíveis valores de x satisfazendo esta congruência. Eles são dados pela lista

$$L = [t + m * i : 0 \leq i < d].$$

O que efetivamente obtemos então é uma lista de valores que são candidatos à solução x. Para encontrar efetivamente o valor de x, devemos testar cada valor da lista para determinar qual deles satisfaz a congruência $g^x \equiv h \pmod{p}$. O valor de d usualmente é pequeno, de forma que há poucos elementos na lista para serem testados.

Apresentamos abaixo o Algoritmo "Rho" para a resolução do PLD em grupos $U(p)$. A generalização do algoritmo para grupos finitos cíclicos quaisquer não é complexa, levando em conta que já discutimos acima o detalhe menos imediato desta generalização, que é a adaptação da função f. Desta forma, deixamos a descrição do algoritmo generalizado como exercício (Exercício 1).

Algoritmo 6.3: Algoritmo "Rho" de Pollard

Entrada: Um número primo p, um gerador g de $U(p)$ e um inteiro $1 \leq h < p$.

Saída: Um número inteiro $0 \leq x < p-1$ tal que $g^x \equiv h \pmod{p}$ ou "Insucesso".

Instruções:

1. $y \leftarrow 1$, $z \leftarrow 1$
2. $\gamma_1 \leftarrow 0$, $\delta_1 \leftarrow 0$, $\gamma_2 \leftarrow 0$, $\delta_2 \leftarrow 0$
3. $\gamma_1 \leftarrow \Gamma(y, \gamma_1)$, $\delta_1 \leftarrow \Delta(y, \delta_1)$
4. $y \leftarrow f(y)$
5. $\gamma_2 \leftarrow \Gamma(f(z), \Gamma(z, \gamma_2))$, $\delta_2 \leftarrow \Delta(f(z), \Delta(z, \delta_2))$
6. $z \leftarrow f(f(z))$
7. Enquanto $y \neq z$, faça:
   - 7.1. $\gamma_1 \leftarrow \Gamma(y, \gamma_1)$, $\delta_1 \leftarrow \Delta(y, \delta_1)$
   - 7.2. $y \leftarrow f(y)$
   - 7.3. $\gamma_2 \leftarrow \Gamma(f(z), \Gamma(z, \gamma_2))$, $\delta_2 \leftarrow \Delta(f(z), \Delta(z, \delta_2))$
   - 7.4. $z \leftarrow f(f(z))$
8. $u \leftarrow (\gamma_1 - \gamma_2) \bmod p - 1$
9. $v \leftarrow (\delta_2 - \delta_1) \bmod p - 1$
10. Se $v = 0$, então retorne "Insucesso".
11. Aplique o Algoritmo Euclidiano Estendido a v e $p - 1$, obtendo $\alpha v + \beta(p-1) = d$, onde $d = \text{mdc}(v, p-1)$.
12. $m \leftarrow (p-1)/d$
13. $t \leftarrow ((\alpha * u)/d) \bmod m$
14. $i \leftarrow 0$
15. Enquanto $i < d$, faça:
    - 15.1. $s \leftarrow t + m * i$
    - 15.2. Se $g^s \equiv h \pmod p$, então retorne s.
    - 15.3. $i \leftarrow i + 1$

A análise sobre a estimativa do número de passos que este algoritmo precisa executar até encontrar a solução do PLD é bem mais sofisticada do que a dos algoritmos anteriores, uma vez que ela depende do estudo da probabilidade de ocorrer uma coincidência entre os valores mais recentes das duas listas após um determinado número de passos. Não apresentamos esta análise aqui, mas ela pode ser consultada no livro [10]. Esta análise conclui que o algoritmo executa aproximadamente $\sqrt{p}$ passos. Desta forma, o seu tempo de execução é próximo do tempo do Algoritmo BSGS, mas o seu consumo de memória é radicalmente menor.

Exemplo 6.2. *Seja $p = 257$, $g = 3$ (a raiz primitiva de $U(257)$ obtida com o Algoritmo de Gauss) e $h = 18$. Vamos utilizar o Algoritmo "Rho" para determinar o valor de x tal que $g^x \equiv h \pmod p$.*

Começamos calculando as listas y_i e z_i. Temos $y_0 = z_0 = 1$. Calculamos $y_1 = f(y_0) = f(1) = g = 3$ e $z_1 = f(f(z_0)) = f(3) = 3.3 = 9$. Em seguida, calculamos $y_2 = f(x_1) = f(3) = 9$ e $z_2 = f(f(z_1)) = f(f(9)) = f(3.9) = f(27) = f(3.27) = 81$. Calculamos então $y_3 = f(y_2) = f(9) = 27$ e $z_3 = f(f(y_2)) = f(f(81)) = f(3.81) = f(243) = 243h \bmod p = 243.18 \bmod 257 = 5$. Continuamos calculando os valores

das duas listas, assim como os valores dos expoentes $\gamma_1, \delta_1, \gamma_2$ e δ_2, até encontrarmos uma coincidência entre os valores:

i	y_i	z_i	γ_1	δ_1	γ_2	δ_2
0	1	1	0	0	0	0
1	3	9	1	0	2	0
2	9	81	2	0	4	0
3	27	5	3	0	5	1
4	81	45	4	0	7	1
5	243	235	5	0	16	2
6	5	46	5	1	32	6
7	15	26	6	1	66	12
8	45	234	7	1	68	12
9	135	234	8	1	136	26
10	235	234	16	2	16	54
11	118	234	16	3	32	110
12	46	234	32	6	64	222
13	138	234	33	6	128	190
14	26	234	66	12	0	126
15	78	234	67	12	0	254
16	234	234	68	12	0	254

Calculamos então $u = \gamma_1 - \gamma_2 = 68 - 0 = 68$ e $v = \delta_2 = \delta_1 = 254 - 12 = 242$. Aplicando o Algoritmo Euclidiano Estendido a $v = 242$ e $p - 1 = 256$, obtemos $\alpha = -55$ e $d = mdc(v, p-1) = 2$. Em seguida, calculamos $m = (p-1)/d = 256/2 = 128$ e $t = ((\alpha u)/d) \bmod m = (-55.68)/2 \bmod 128 = 50$. Temos então que $x \equiv t \pmod m$, isto é, $x \equiv 50 \pmod{128}$. Como $0 \leq x < p - 1 = 256$, existem dois possíveis valores candidatos a serem a solução x: $x_0 = t = 50$ e $x_1 = t + m = 50 + 128 = 178$. Precisamos verificar qual deles satisfaz a congruência $g^x \equiv h \pmod p$.

Começamos testando $x_0 = 50$. Temos $g^{x_0} = 3^{50} \equiv 18 \equiv h \pmod{257}$. Logo, a solução para o PLD é $x = 50$.

6.4 Algoritmo de Pohlig-Hellman

Nesta seção, apresentamos o Algoritmo de Pohlig-Hellman, proposto originalmente por Stephen Pohlig e Martin Hellman no artigo [21].

Da mesma forma que os algoritmos apresentados nas seções anteriores, o Algoritmo de Pohlig-Hellman é um *algoritmo genérico* para a resolução do PLD em grupos finitos cíclicos. Vamos explicá-lo no contexto dos grupos $U(p)$, mas a sua generalização não oferece dificuldades.

A ideia principal do Algoritmo de Pohlig-Hellman é tentar decompor o PLD no grupo $U(p)$ de ordem $p - 1$ em problemas mais simples em subgrupos de ordens menores de $U(p)$. Inicialmente, obtemos a fatoração de $p - 1$:

$$p - 1 = q_1^{e_1} q_2^{e_2} \dots q_k^{e_k}.$$

Nosso objetivo então é, para cada $1 \leq i \leq k$, encontrar um elemento g_i que seja gerador de subgrupos de $U(p)$ de ordem $q_i^{e_i}$. Em outras palavras, a ordem de g_i deve ser $q_i^{e_i}$. Encontrar tal g_i não é difícil. Podemos tomar $g_i = (g^{(p-1)/q_i^{e_i}}) \bmod p$. Temos

$$g_i^{q_i^{e_i}} \equiv g^{p-1} \equiv 1 \pmod p.$$

Assim, pelo Lema Chave (Lema 4.2), a ordem de g_i divide $q_i^{e_i}$. Entretanto, se $t < q_i^{e_i}$, então $t' = ((p-1)/q_i^{e_i})t < p-1$. Desta forma, $g^{t'} \not\equiv 1 \pmod{p}$, o que significa que $g_i^t \not\equiv 1 \pmod{p}$. Portanto, a ordem de g_i é igual a $q_i^{e_i}$.

Como temos $g^x \equiv h \pmod{p}$, podemos elevar os dois lados a $(p-1)/q_i^{e_i}$, obtendo $g_i^x \equiv h_i \pmod{p}$, onde $h_i = (h^{(p-1)/q_i^{e_i}}) \bmod p$. Esta nova congruência nos dá um PLD em um subgrupo de $U(p)$ de ordem $q_i^{e_i}$, já que esta é a ordem de g_i. Podemos reparar a partir da congruência que x também satisfaz este novo PLD. Se buscarmos uma solução para este segundo PLD, iremos calcular um valor x_i no intervalo $0 \leq x_i < q_i^{e_i}$ tal que $g_i^{x_i} \equiv h_i \pmod{p}$. Como x também satisfaz esta congruência, temos então que $x \equiv x_i \pmod{q_i^{e_i}}$. Desta forma, se conseguirmos determinar os valores de x_i para todo $1 \leq i \leq k$, podemos encontrar o valor de x resolvendo, com o Algoritmo Chinês do Resto, o sistema

$$\begin{cases} x \equiv x_1 \pmod{q_1^{e_1}} \\ x \equiv x_2 \pmod{q_2^{e_2}} \\ \vdots \quad \vdots \quad \vdots \\ x \equiv x_k \pmod{q_k^{e_k}}. \end{cases}$$

Como os módulos das congruências do sistema são potências de primos distintos, o Teorema Chinês do Resto (Teorema 2.6) nos garante que a solução do sistema é única módulo $q_1^{e_1} q_2^{e_2} \ldots q_k^{e_k} = p-1$. Ou seja, a solução do sistema é o valor $0 \leq x < p-1$ que estamos buscando como solução do PLD original $g^x \equiv h \pmod{p}$.

Nesta primeira etapa, decompomos a resolução do PLD em $U(p)$ na resolução de k PLD's, sendo cada um em um subgrupo de ordem $q_i^{e_i}$, para $1 \leq i \leq k$. Na segunda etapa, vamos agora decompor um PLD em um subgrupo de ordem $q_i^{e_i}$ em e_i PLD's em um subgrupo de ordem q_i.

Para isso, começamos com o PLD $g_i^{x_i} \equiv h_i \pmod{p}$ em um subgrupo de ordem $q_i^{e_i}$. Temos que $0 \leq x_i < q_i^{e_i}$. Decompomos então x_i como

$$x_i = x_{i0} + x_{i1} q_i + x_{i2} q_i^2 + \ldots + x_{i(e_i-1)} q_i^{e_i-1}.$$

Vamos agora elevar os dois lados da congruência $g_i^{x_i} \equiv h_i \pmod{p}$ a $q_i^{e_i-1}$, obtendo $(g_i^{q_i^{e_i-1}})^{x_i} \equiv h_i^{q_i^{e_i-1}} \pmod{p}$. Substituímos então x_i pela decomposição acima, obtendo

$$(g_i^{q_i^{e_i-1}})^{x_{i0} + x_{i1} q_i + \ldots + x_{i(e_i-1)} q_i^{e_i-1}} \equiv h_i^{q_i^{e_i-1}} \pmod{p}.$$

Esta congruência pode ser escrita como

$$(g_i^{q_i^{e_i-1}})^{x_{i0}} (g_i^{q_i^{e_i}})^{x_{i1} + \ldots + x_{i(e_i-1)} q_i^{e_i-2}} \equiv h_i^{q_i^{e_i-1}} \pmod{p}.$$

Entretanto, temos que $g_i^{q_i^{e_i}} \equiv 1 \pmod{p}$, de forma que a congruência acima pode ser simplificada para

$$(g_i^{q_i^{e_i-1}})^{x_{i0}} \equiv h_i^{q_i^{e_i-1}} \pmod{p}.$$

Se $g_i' = (g_i^{q_i^{e_i-1}}) \bmod p$ e $h_{i0} = (h_i^{q_i^{e_i-1}}) \bmod p$, obtemos um novo PLD: $(g_i')^{x_{i0}} \equiv h_{i0} \pmod{p}$. Temos que $(g_i')^{q_i} \equiv g_i^{q_i^{e_i}} \equiv 1 \pmod{p}$. Logo, pelo Lema Chave (Lema 4.2), a ordem de g_i' divide q_i. Como q_i é primo, a ordem de g_i' é 1 ou q_i. Entretanto, $g_i' \not\equiv 1 \pmod{p}$, já que $g_i^{q_i^{e_i-1}} \not\equiv 1 \pmod{p}$. Assim, a ordem de g_i' é q_i e temos o PLD $(g_i')^{x_{i0}} \equiv h_{i0} \pmod{p}$ em um subgrupo de ordem q_i. Resolvemos então este

PLD com o Algoritmo BSGS, levando em conta que a ordem de g_i' é q_i. Com isso, obtemos o valor de x_{i0}.

Vamos utilizar um processo análogo para calcular o valor de x_{i1}. Elevamos os dois lados do PLD $g_i^{x_i} \equiv h_i \pmod{p}$ a $q_i^{e_i-2}$ agora. Obtemos

$$(g_i^{q_i^{e_i-2}})^{x_{i0}}(g_i^{q_i^{e_i-1}})^{x_{i1}}(g_i^{q_i^{e_i}})^{x_{i2}+\ldots+x_{i(e_i-1)}q_i^{e_i-3}} \equiv h_i^{q_i^{e_i-2}} \pmod{p}.$$

Novamente, como $g_i^{q_i^{e_i}} \equiv 1 \pmod{p}$, a congruência acima pode ser simplificada para

$$(g_i^{q_i^{e_i-2}})^{x_{i0}}(g_i^{q_i^{e_i-1}})^{x_{i1}} \equiv h_i^{q_i^{e_i-2}} \pmod{p}.$$

Finalmente, escrevemos esta congruência como

$$(g_i^{q_i^{e_i-1}})^{x_{i1}} \equiv h_i^{q_i^{e_i-2}}((g_i^{-1})^{q_i^{e_i-2}})^{x_{i0}} \pmod{p},$$

onde o valor de x_{i0} já é conhecido. Se $h_{i1} = h_i^{q_i^{e_i-2}}((g_i^{-1})^{q_i^{e_i-2}})^{x_{i0}} \bmod p$, obtemos mais um PLD com a mesma base g_i', isto é, mais um PLD em um subgrupo de ordem q_i: $(g_i')^{x_{i1}} \equiv h_{i1} \pmod{p}$. Novamente, podemos utilizar o Algoritmo BSGS para determinar o valor de x_{i1}.

Continuando este procedimento, para calcular o valor de x_{ij}, $0 \leq j < e_i$, resolvemos o PLD $(g_i')^{x_{ij}} \equiv h_{ij} \pmod{p}$, onde

$$h_{ij} = h_i^{q_i^{e_i-1-j}}((g_i^{-1})^{q_i^{e_i-1-j}})^{x_{i0}+x_{i1}q_i+\ldots+x_{i(j-1)}q_i^{j-1}} \bmod p.$$

Após termos calculado os valores de x_{ij} para todo $0 \leq j < e_i$, calculamos diretamente o valor de x_i pela fórmula

$$x_i = x_{i0} + x_{i1}q_i + x_{i2}q_i^2 + \ldots + x_{i(e_i-1)}q_i^{e_i-1}.$$

É desta maneira que calculamos os valores de x_i para todo $1 \leq i \leq k$.

Em resumo, para calcular o valor de x do PLD original, primeiro calculamos os valores de x_i, $1 \leq i \leq k$, que são soluções de PLD's em subgrupos de ordem $q_i^{e_i}$, respectivamente. Por sua vez, para resolver cada um estes PLD's, decompomos x_i de acordo com a fórmula acima e calculamos cada x_{ij}, $0 \leq j < e_i$ através da resolução de PLD's em subgrupos de ordem q_i. Para a resolução destes PLD's, utilizamos o Algoritmo BSGS. Uma vez tendo obtido todos os valores de x_{ij}, calculamos x_i de acordo com a fórmula acima. Finalmente, uma vez obtidos todos os valores de x_i, calculamos o valor de x desejado a partir da aplicação do Algoritmo Chinês do Resto.

Apresentamos abaixo o Algoritmo de Pohlig-Hellman para a resolução do PLD em grupos $U(p)$, levando em consideração os procedimentos e cálculos que apresentamos acima. Assim como nas seções anteriores, deixamos a versão generalizada do algoritmo como exercício (Exercício 1).

Algoritmo 6.4: Algoritmo de Pohlig-Hellman

Entrada: Um número primo p, um gerador g de $U(p)$ e um inteiro $1 \leq h < p$.

Saída: Um número inteiro $0 \leq x < p-1$ tal que $g^x \equiv h \pmod{p}$.

Instruções:

1. Fatore $p-1$, obtendo $p-1 = q_1^{e_1} q_2^{e_2} \dots q_k^{e_k}$.
2. $i \leftarrow 0$
3. Enquanto $i < k$, faça:
   3.1. $g_i \leftarrow g^{(p-1)/q_i^{e_i}} \bmod p$
   3.2. Utilizando o Algoritmo Euclidiano Estendido, calcule g_i^{-1}, o inverso de g_i módulo p.
   3.3. $h_i \leftarrow h^{(p-1)/q_i^{e_i}} \bmod p$
   3.4. $x_i \leftarrow 0$
   3.5. $g'_i \leftarrow g_i^{q_i^{e_i-1}} \bmod p$
   3.6. $j \leftarrow 0$
   3.7. Enquanto $j < e_i$, faça:
      3.7.1. $h_{ij} \leftarrow (h_i^{q_i^{e_i-1-j}} (g_i^{-1})^{q_i^{(e_i-1-j)x_i}}) \bmod p$
      3.7.2. Utilizando o Algoritmo BSGS, calcule o valor x_{ij} que satisfaz a congruência $(g'_i)^{x_{ij}} \equiv h_{ij} \pmod p$, levando em conta que g'_i possui ordem q_i.
      3.7.3. $x_i \leftarrow x_i + x_{ij} * q_i^j$
      3.7.4. $j \leftarrow j + 1$
   3.8. $i \leftarrow i + 1$
4. Utilizando o Algoritmo Chinês do Resto, encontre o valor de x que é solução do sistema

$$\begin{cases} x \equiv x_1 \pmod{q_1^{e_1}} \\ x \equiv x_2 \pmod{q_2^{e_2}} \\ \vdots \quad \vdots \quad \vdots \\ x \equiv x_k \pmod{q_k^{e_k}}. \end{cases}$$

5. Retorne x.

Como vimos anteriormente, para resolver um PLD em $U(p)$, o Algoritmo BSGS pode precisar de aproximadamente $\sqrt{p}$ passos. Por outro lado, se $p-1 = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$, então o Algoritmo de Pohlig-Hellman decompõe o problema em k PLD's em subgrupos de ordem $q_i^{e_i}$, com $1 \leq i \leq k$. Após esta decomposição, o Algoritmo de Pohlig-Hellman ainda realiza uma segunda decomposição de um PLD em um subgrupo de ordem $q_i^{e_i}$ em e_i PLD's em um subgrupo de ordem q_i. São estes últimos PLD's que são finalmente resolvidos utilizando-se o Algoritmo BSGS. Desta forma, como o Algoritmo BSGS precisa de aproximadamente $\sqrt{q_i}$ passos para resolver um PLD em um subgrupo de ordem q_i, o Algoritmo de Pohlig-Hellman irá utilizar aproximadamente $e_1\sqrt{q_1} + e_2\sqrt{q_2} + \ldots + e_k\sqrt{q_k}$ passos para resolver o PLD original. Este número de passos é consideravelmente menor do que $\sqrt{p}$, se $p-1$ puder ser completamente decomposto em fatores relativamente pequenos quando comparados com p.

Vemos então que o fato de p ser um primo muito grande não é suficiente para garantir a segurança dos métodos de criptografia e assinatura digital que estudamos anteriormente. Se p for muito grande, mas $p-1$ possuir apenas fatores relativamente pequenos, o uso do Algoritmo de Pohlig-Hellman poderá oferecer uma possibilidade real de obtenção da chave privada. Desta forma, para prevenir o uso deste algoritmo como ferramenta de quebra destes métodos, devemos sempre verificar que

$p-1$ possua ao menos um fator primo grande, preferencialmente de uma ordem de grandeza bem próxima a do próprio p. Desta maneira, resolver o PLD através do Algoritmo de Pohlig-Hellman se tornará praticamente equivalente a resolvê-lo diretamente através do Algoritmo BSGS.

Podemos reparar que esta preocupação está explicitamente presente na construção das chaves do método DSA. Neste método, selecionamos dois primos p e q de forma que q divida $p-1$. Assim, se ambos os primos escolhidos forem muito grandes, o Algoritmo de Pohlig-Hellman não servirá como ferramenta efetiva para a obtenção da chave privada de uma implementação do DSA.

Exemplo 6.3. *Seja $p = 433$, $g = 126$ (a raiz primitiva de $U(433)$ obtida com o Algoritmo de Gauss) e $h = 76$. Vamos utilizar o Algoritmo de Pohlig-Hellman para determinar o valor de x tal que $g^x \equiv h \pmod{p}$.*

Começamos fatorado $p-1 = 432$, obtendo $432 = 2^4.3^3$. Primeiramente, trabalhamos com a potência 2^4. Calculamos $g_1 = g^{(p-1)/q_1^{e_1}} \bmod p = 126^{432/2^4} \bmod 433 = 126^{27} \bmod 433 = 183$ e $h_1 = h^{(p-1)/q_1^{e_1}} \bmod p = 76^{27} \bmod 433 = 168$. Utilizando o Algoritmo Euclidiano Estendido, calculamos também g_1^{-1}, o inverso de g_1 módulo p, obtendo $g_1^{-1} = 168$. Finalmente, calculamos $g_1' = g_1^{q_1^{e_1-1}} \bmod p = 183^{2^3} \bmod 433 = 432$.

Vamos calcular o valor de x_1 tal que $g_1^{x_1} \equiv h_1 \pmod{p}$. Sabemos que $0 \leq x_1 < p_1^{e_1} = 2^4 = 16$. Escrevemos então $x_1 = x_{10} + x_{11}2 + x_{12}2^2 + x_{13}2^3$. Começamos fazendo $x_1 = 0$ e vamos incrementalmente atualizando o seu valor ao calcularmos os valores de x_{10}, x_{11}, x_{12} e x_{13}:

1. *Para determinar x_{10}, calculamos*

$$h_{10} = h_1^{q_1^{e_1-1-j}} (g_1^{-1})^{q_1^{(e_1-1-j)x_1}} \bmod p = 168^{2^3}.168^{2^3.0} \bmod 433 = 432.$$

O valor de x_{10} será a solução do PLD $(g_1')^{x_{10}} \equiv h_{10} \pmod{p}$, isto é, $432^{x_{10}} \equiv 432 \pmod{433}$. Fica claro então que $x_{10} = 1$. Atualizamos o valor de x_1 para $x_1 = x_{10} = 1$.

2. *Para determinar x_{11}, calculamos*

$$h_{11} = h_1^{q_1^{e_1-1-j}} (g_1^{-1})^{q_1^{(e_1-1-j)x_1}} \bmod p = 168^{2^2}.168^{2^2.1} \bmod 433 = 432.$$

O valor de x_{11} será a solução do PLD $(g_1')^{x_{11}} \equiv h_{11} \pmod{p}$, isto é, $432^{x_{11}} \equiv 432 \pmod{433}$. Fica claro então que $x_{11} = 1$. Atualizamos o valor de x_1 para $x_1 = x_{10} + x_{11}2 = 3$.

3. *Para determinar x_{12}, calculamos*

$$h_{12} = h_1^{q_1^{e_1-1-j}} (g_1^{-1})^{q_1^{(e_1-1-j)x_1}} \bmod p = 168^{2^1}.168^{2^1.3} \bmod 433 = 432.$$

O valor de x_{12} será a solução do PLD $(g_1')^{x_{12}} \equiv h_{12} \pmod{p}$, isto é, $432^{x_{12}} \equiv 432 \pmod{433}$. Fica claro então que $x_{12} = 1$. Atualizamos o valor de x_1 para $x_1 = x_{10} + x_{11}2 + x_{12}2^2 = 7$.

4. *Para determinar x_{13}, calculamos*

$$h_{13} = h_1^{q_1^{e_1-1-j}} (g_1^{-1})^{q_1^{(e_1-1-j)x_1}} \bmod p = 168^{2^0}.168^{2^0.7} \bmod 433 = 432.$$

O valor de x_{13} será a solução do PLD $(g_1')^{x_{13}} \equiv h_{12} \pmod{p}$, isto é, $432^{x_{13}} \equiv 432 \pmod{433}$. Fica claro então que $x_{13} = 1$. Calculamos o valor de x_1 como $x_1 = x_{10} + x_{11}2 + x_{12}2^2 + x_{13}2^3 = 15$.

Vamos agora trabalhar com a potência 3^3. *Calculamos* $g_2 = g^{(p-1)/q_2^{e_2}} \bmod p = 126^{432/3^3} \bmod 433 = 126^{16} \bmod 433 = 17$ *e* $h_2 = h^{(p-1)/q_2^{e_2}} \bmod p = 76^{16} \bmod 433 = 3$. *Utilizando o Algoritmo Euclidiano Estendido, calculamos também* g_2^{-1}, *o inverso de* g_2 *módulo* p, *obtendo* $g_2^{-1} = 51$. *Finalmente, calculamos* $g_2' = g_2^{q_2^{e_2-1}} \bmod p = 17^{3^2} \bmod 433 = 198$.

Vamos calcular o valor de x_2 *tal que* $g_2^{x_2} \equiv h_2 \pmod p$. *Sabemos que* $0 \leq x_2 < p_2^{e_2} = 3^3 = 27$. *Escrevemos então* $x_2 = x_{20} + x_{21}3 + x_{22}3^2$. *Começamos fazendo* $x_2 = 0$ *e vamos incrementalmente atualizando o seu valor ao calcularmos os valores de* x_{20}, x_{21} *e* x_{22}.

1. *Para determinar* x_{20}, *calculamos*

$$h_{20} = h_2^{q_2^{e_2-1-j}} (g_2^{-1})^{q_2^{(e_2-1-j)x_2}} \bmod p = 3^{3^2}.51^{3^2.0} \bmod 433 = 198.$$

O valor de x_{20} *será a solução do PLD* $(g_2')^{x_{20}} \equiv h_{20} \pmod p$, *isto é,* $198^{x_{20}} \equiv 198 \pmod{433}$. *Fica claro então que* $x_{20} = 1$. *Atualizamos o valor de* x_2 *para* $x_2 = x_{20} = 1$.

2. *Para determinar* x_{21}, *calculamos*

$$h_{21} = h_2^{q_2^{e_2-1-j}} (g_2^{-1})^{q_2^{(e_2-1-j)x_2}} \bmod p = 3^{3^1}.51^{3^1.1} \bmod 433 = 234.$$

O valor de x_{21} *será a solução do PLD* $(g_2')^{x_{21}} \equiv h_{21} \pmod p$, *isto é,* $198^{x_{21}} \equiv 234 \pmod{433}$.

Utilizamos então o Algoritmo BSGS para determinar o valor de x_{21}, *sabendo que a ordem de* $g_2' = 198$ *é* $p_2 = 3$. *Começamos calculando* $m = \lceil \sqrt{3} \rceil = 2$ *e construímos a lista* B *de* baby-steps.

l	0	1
$198^l \bmod 433$	1	198

Agora, utilizando o Algoritmo Euclidiano Estendido, calculamos $(g_2')^{-1}$, *o inverso de* g_2' *módulo* p, *obtendo* $(g_2')^{-1} = 234$. *Calculamos então*

$$t = ((g_2')^{-1})^m \bmod p = 234^2 \bmod 433 = 198.$$

Finalmente, calculamos um a um os giant-steps $h_{21}t^i \bmod p$, *com* $0 \leq k < m = 2$.

k	0	1
$234.198^i \bmod 433$	234	1
Está em B?	*Não*	*Sim*

Temos então $x_{21} = km + l = 2$. *Atualizamos o valor de* x_2 *para* $x_2 = x_{20} + x_{21}3 = 7$.

3. *Para determinar* x_{22}, *calculamos*

$$h_{22} = h_2^{q_2^{e_2-1-j}} (g_2^{-1})^{q_2^{(e_2-1-j)x_2}} \bmod p = 3^{3^0}.51^{3^0.7} \bmod 433 = 234.$$

O valor de x_{22} *será a solução do PLD* $(g_2')^{x_{22}} \equiv h_{22} \pmod p$, *isto é,* $198^{x_{22}} \equiv 234 \pmod{433}$. *Como resolvemos este PLD no item anterior, sabemos que* $x_{22} = 2$. *Calculamos o valor de* x_2 *como* $x_2 = x_{20} + x_{21}3 + x_{22}3^2 = 25$.

Finalmente, podemos obter a solução x como a solução do sistema

$$\begin{cases} x \equiv x_1 \equiv 15 \pmod{2^4} \\ x \equiv x_2 \equiv 25 \pmod{3^3}. \end{cases}$$

Aplicando o Algoritmo Euclidiano Estendido a $2^4 = 16$ e $3^3 = 27$, obtemos $\alpha = -5$ e $\beta = 3$. A solução dada pelo Algoritmo Chinês do Resto será então

$$x = (15.\beta.27 + 25.\alpha.16) \bmod 432 = 79.$$

De fato, $126^{79} \equiv 76 \pmod{433}$.

6.5 Algoritmo do Cálculo de Índices

Nesta seção, apresentamos o Algoritmo do Cálculo de Índices. A ideia principal do algoritmo, o uso de um sistema linear para resolver um PLD, pode ser encontrada no livro [14], mas a elaboração do algoritmo propriamente dito foi apresentada por Leonard Adleman (o "A" do RSA) no artigo [1].

Ao contrário dos algoritmos apresentados nas seções anteriores, o Algoritmo do Cálculo de Índices é um *algoritmo específico* para a resolução do PLD em grupos $U(p)$.

Este algoritmo recebe uma entrada extra além dos valores de g, h e p para os quais se quer determinar x tal que $g^x \equiv h \pmod{p}$. O algoritmo recebe uma base de primos $P = [q_1, q_2, \ldots, q_k]$, geralmente constituída pelos k menores primos. Tal base pode ser construída utilizando-se o Crivo de Eratóstenes.

Inicialmente, o algoritmo seleciona valores aleatórios de t no intervalo $0 \leq t < p-1$, buscando por valores da forma $g^t \bmod p$ que possam ser fatorados completamente utilizando-se apenas os primos da base P. Números que atendem a este requisito são chamados de *números P-suaves*.

Em outras palavras, o algoritmo busca valores de t tais que $g^t \equiv q_1^{e_1} q_2^{e_2} \ldots q_k^{e_k} \pmod{p}$, onde $e_j \geq 0$ para todo $1 \leq j \leq k$. Como g é um gerador de $U(p)$, existem valores $x_1, x_2, \ldots, x_k$ tais que $g^{x_j} \equiv q_j \pmod{p}$, para todo $1 \leq j \leq k$. A ideia do algoritmo é calcular tais valores e então utilizá-los para obter o valor x procurado.

Podemos escrever a congruência $g^t \equiv q_1^{e_1} q_2^{e_2} \ldots q_k^{e_k} \pmod{p}$ como

$$g^{e_1 x_1 + e_2 x_2 + \ldots + e_k x_k} \equiv g^t \pmod{p}.$$

Como temos duas potências da mesma base congruentes entre si, então os seus expoentes devem ser congruentes módulo a ordem desta base, isto é, módulo $p-1$. Desta forma,

$$e_1 x_1 + e_2 x_2 + \ldots + e_k x_k \equiv t \pmod{p-1}.$$

Para calcular os valores de $x_1, \ldots, x_k$, podemos construir um sistema de k congruências módulo $p-1$ como a congruência acima. Para isso, precisamos obter pelo menos k valores $t_1, t_2, \ldots, t_k$ tais que $g^{t_i} \bmod p$ seja um número P-suave, para todo $1 \leq i \leq k$. A partir destes valores, construímos o seguinte sistema linear com congruências no mesmo formato da congruência acima:

$$\begin{bmatrix} e_{11} & e_{12} & \ldots & e_{1k} \\ e_{21} & e_{22} & \ldots & e_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ e_{k1} & e_{k2} & \ldots & e_{kk} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{bmatrix} = \begin{bmatrix} t_1 \\ t_2 \\ \vdots \\ t_k \end{bmatrix} \pmod{p-1}.$$

Para resolver este sistema linear, podemos utilizar o método da Eliminação Gaussiana [8] na matriz ampliada

$$\left[\begin{array}{cccc|c} e_{11} & e_{12} & \dots & e_{1k} & t_1 \\ e_{21} & e_{22} & \dots & e_{2k} & t_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ e_{k1} & e_{k2} & \dots & e_{kk} & t_k \end{array}\right] \pmod{p-1},$$

tomando cuidado apenas de adaptar o método para o nosso contexto em que o sistema é módulo $p-1$. Em particular, todas as operações de soma e multiplicação que são feitas durante o processo de Eliminação Gaussiana deverão ser sempre reduzidas módulo $p-1$.

Na Eliminação Gaussiana tradicional, feita geralmente para sistemas com coeficientes reais, precisamos escolher em cada coluna um pivô que seja inversível. No conjunto dos reais, qualquer número diferente de zero é inversível, de modo que falharemos em encontrar um pivô para uma coluna somente se o valor da diagonal e os abaixo da mesma forem todos nulos. No caso dos inteiros módulo $p-1$, a escolha de pivôs não é tão simples. Se $p > 3$, então $p-1$ é um número composto, de forma que nem todos os inteiros no intervalo $1 \leq i < p-1$ serão inversíveis. Assim, o pivô de uma coluna deve ser um elemento v tal que $\text{mdc}(v, p-1) = 1$, pois estes são os elementos inversíveis de $\mathbb{Z}_{p-1}$. Desta forma, existe uma chance maior no caso da Eliminação Gaussiana módulo $p-1$ de não encontrarmos nenhum pivô viável em uma coluna da matriz. Se isto acontecer, voltamos à etapa anterior e buscamos mais valores t tais que $g^t \bmod p$ seja P-suave, adicionando novas linhas à matriz acima.

Ao final do processo da Eliminação Gaussiana módulo $p-1$ com a possível adição de novas linhas à matriz, se tomarmos apenas as k primeiras linhas da matriz ampliada, obtemos

$$\left[\begin{array}{cccc|c} 1 & 0 & \dots & 0 & t'_1 \\ 0 & 1 & \dots & 0 & t'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & t'_k \end{array}\right] \pmod{p-1},$$

onde $t'_1, t'_2, \dots, t'_k$ é a solução do sistema. Isto é, $x_1 = t'_1$, $x_2 = t'_2$, $\dots$, $x_k = t'_k$. As demais linhas da matriz ampliada obtida ao final do processo, se existirem, serão nulas.

Na última etapa do algoritmo, calculamos g^{-1}, o inverso de g módulo p. Em seguida, buscamos um novo valor de t, agora sob uma nova condição. Desejamos que $h(g^{-1})^t \bmod p$ seja P-suave. Ao contrário da primeira etapa, em que precisamos de vários valores de t, nesta última precisamos de apenas um valor. Temos então

$$h(g^{-1})^t \equiv q_1^{f_1} q_2^{f_2} \dots q_k^{f_k} \pmod{p},$$

onde $f_j \geq 0$ para todo $1 \leq j \leq k$. Como $h \equiv g^x \pmod{p}$, a congruência acima pode ser escrita como

$$g^x \equiv g^{t+f_1x_1+f_2x_2+\dots+f_kx_k} \pmod{p},$$

o que nos permite concluir que

$$x \equiv t + f_1x_1 + f_2x_2 + \dots + f_kx_k \pmod{p-1}.$$

Como conhecemos f_j e x_j, para todo $1 \leq i \leq k$, obtemos a resposta desejada como $x = (t + f_1x_1 + f_2x_2 + \ldots + f_kx_k) \bmod p - 1$.

Apresentamos abaixo o Algoritmo do Cálculo de Índices para a resolução do PLD em grupos $U(p)$, levando em consideração os procedimentos e cálculos que apresentamos acima.

Algoritmo 6.5: Algoritmo do Cálculo de Índices

Entrada: Um número primo p, um gerador g de $U(p)$, um inteiro $1 \leq h < p$ e uma lista $P = [q_1, q_2, \ldots, q_k]$ contendo k primos.

Saída: Um número inteiro $0 \leq x < p - 1$ tal que $g^x \equiv h \pmod{p}$.

Instruções:

1. Sejam A uma matriz de k colunas e b um vetor coluna inicialmente vazios.
2. Primeira Etapa:
   - 2.1. Sorteie aleatoriamente valores de t no intervalo $0 \leq t < p-1$ até que $g^t \bmod p$ possa ser fatorado completamente utilizando-se apenas os primos da lista P.
   - 2.2. Seja $g^t \bmod p = q_1^{e_1}q_2^{e_2}\ldots q_k^{e_k}$, onde $e_j \geq 0$, para todo $1 \leq j \leq k$.
   - 2.3. Adicione t ao vetor b e os expoentes $e_1, e_2, \ldots, e_k$ à linha correspondente da matriz A.
   - 2.4. O vetor b e a matriz A precisam conter no mínimo k linhas para que se possa prosseguir para a segunda etapa.
3. Segunda Etapa:
   - 3.1. Vamos considerar a matriz ampliada $A' = [A|b]$.
   - 3.2. Aplicamos o processo de Eliminação Gaussiana módulo $p - 1$ a A' para obter uma matriz na forma

$$\left[\begin{array}{c|c} I & b' \\ \hline 0 & 0 \end{array}\right],$$

     onde I é a matriz identidade $k \times k$, 0 representa uma matriz ou vetor nulo e as linhas nulas podem ou não estar presentes.
   - 3.3. Caso, em algum passo da Eliminação Gaussiana, não seja possível encontrar um pivô viável para a continuação do processo (os pivôs devem ser inversíveis módulo $p-1$), deve-se retornar à primeira etapa para a adição de mais linhas à matriz A e ao vetor b.
   - 3.4. Caso a Eliminação Gaussiana seja bem sucedida, os elementos b'_j do vetor b', com $1 \leq j \leq k$ são, respectivamente, as soluções dos PLD's $g^x \equiv q_j \pmod{p}$. Isto é, $g^{b'_j} \equiv q_j \pmod{p}$.
4. Terceira Etapa:
   - 4.1. Utilizando o Algoritmo Euclidiano Estendido, calcule g^{-1}, o inverso de g módulo p.
   - 4.2. Sorteie aleatoriamente valores de t no intervalo $0 \leq t < p - 1$ até que $h(g^{-1})^t \bmod p$ possa ser fatorado completamente utilizando-se apenas os primos da lista P. Ao contrário do passo semelhante realizado na primeira etapa, agora é necessário encontrar apenas um valor de t.

4.3. Seja $h(g^{-1})^t \bmod p = q_1^{f_1} q_2^{f_2} \dots q_k^{f_k}$, onde $f_j \geq 0$, para todo $1 \leq j \leq k$.

4.4. $x \leftarrow (t + f_1 b'_1 + f_2 b'_2 + \ldots + f_k b'_k) \bmod p - 1$

4.5. Retorne x.

Um detalhe importante de se notar no algoritmo acima é que, para efeito prático, ele não resolve apenas um PLD $g^x \equiv h \pmod{p}$. De certa forma, ele resolve *todos* os PLD's com esta mesma base g e este mesmo módulo p. Reparando atentamente nos passos do algoritmo, vemos que o valor de h só é utilizado na última etapa, onde, além deste valor, é utilizada também a solução do sistema que é obtida na etapa anterior. Assim, se armazenarmos esta solução (o vetor b'), seremos capazes de resolver o PLD $g^x \equiv h' \pmod{p}$ para qualquer $h' \in U(p)$ aplicando apenas o cálculo feito na última etapa do algoritmo. Enquanto nos algoritmos anteriores, ao mudar o valor de h, precisávamos executar novamente o algoritmo em sua totalidade para resolver o novo PLD, no Algoritmo do Cálculo de Índices isto não acontece. Para um dado $U(p)$ e um dado gerador g, as duas primeiras etapas precisam ser executadas apenas uma vez. A partir de então, podemos resolver o PLD para qualquer valor de h bastando apenas executar a última etapa do algoritmo.

A análise sobre a estimativa do número de passos que este algoritmo precisa executar até encontrar a solução do PLD é consideravelmente sofisticada e não será detalhada aqui, podendo ser consultada no livro [10]. Esta análise depende de um estudo da quantidade de números P-suaves dentro de um intervalo em função do tamanho da base P.

Em um nível mais básico de análise, o que podemos concluir sem muita dificuldade é que conforme a base de primos P aumenta, mais valores de t atenderão aos critérios tanto da primeira etapa quanto da última. Desta forma, estas etapas ficam mais rápidas conforme P aumenta. Por outro lado, o aumento de P também ocasiona o aumento da dimensão da matriz que é processada na etapa da Eliminação Gaussiana. Assim, esta etapa fica mais lenta com o aumento de P. Desta forma, o tamanho de P deve ser equilibrado de forma bastante delicada de maneira a obter o melhor desempenho do algoritmo.

De todos os algoritmos que apresentamos, este é o mais eficiente para resolver o PLD em grupos $U(p)$. Entretanto, ele é específico para tais grupos, não podendo ser utilizado em grupos de outros formatos. Este é um grande incentivo para o estudo de variantes dos métodos El Gamal e DSA que utilizem outros grupos diferentes do grupo $U(p)$. Em tais sistemas, o Algoritmo do Cálculo de Índices não poderá ser utilizado como ferramenta para obtenção da chave privada, restando apenas o uso dos algoritmos menos eficientes. Desta forma, a segurança do sistema aumenta pelo simples fato de que o algoritmo mais eficiente que poderia ser usado para a sua quebra não está mais disponível. Esta é a motivação de métodos como a *Criptografia com Curvas Elípticas*, em que o grupo utilizado não é um grupo $U(p)$, mas sim um grupo de pontos em uma curva elíptica [2].

Exemplo 6.4. *Seja $p = 271$, $g = 6$ (a raiz primitiva de $U(271)$ obtida com o Algoritmo de Gauss) e $h = 217$. Vamos utilizar o Algoritmo do Cálculo de Índices com base de primos $P = [2, 3, 5]$ para determinar o valor de x tal que $g^x \equiv h \pmod{p}$. Começamos sorteando aleatoriamente valores de t buscando valores em que $g^t \bmod p$ possa ser fatorado completamente com os primos 2, 3 e 5. Como estamos utilizando uma base de primos com 3 elementos, precisamos de pelo menos 3 valores distintos de t. Os seguintes valores podem ser utilizados:*

1. *$t_1 = 235$: $g^{t_1} \bmod p = 6^{235} \bmod 271 = 54 = 2.3^3$;*

2. $t_2 = 18$*:* $g^{t_2} \bmod p = 6^{18} \bmod 271 = 90 = 2.3^2.5$*;*

3. $t_3 = 231$*:* $g^{t_3} \bmod p = 6^{231} \bmod 271 = 192 = 2^6.3$.

Estes valores de t *nos permitem construir o sistema*

$$\begin{bmatrix} 1 & 3 & 0 \\ 1 & 2 & 1 \\ 6 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 235 \\ 18 \\ 231 \end{bmatrix} \pmod{270}.$$

As soluções deste sistema são tais que $6^{x_1} \equiv 2 \pmod{271}$, $6^{x_2} \equiv 3 \pmod{271}$ *e* $6^{x_3} \equiv 5 \pmod{271}$.

Aplicamos então a Eliminação Gaussiana módulo 270 *na matriz ampliada*

$$\left[\begin{array}{ccc|c} 1 & 3 & 0 & 235 \\ 1 & 2 & 1 & 18 \\ 6 & 1 & 0 & 231 \end{array}\right].$$

para encontrar o valor de x_1, x_2 *e* x_3. *Para a primeira coluna, podemos utilizar o elemento* 1 *na primeira linha como pivô. Para zerar os outros elementos desta coluna, faremos a segunda linha menos a primeira e a terceira linha menos* 6 *vezes a segunda, sendo que todos os cálculos devem ser feitos módulo* 270. *Obtemos então a matriz*

$$\left[\begin{array}{ccc|c} 1 & 3 & 0 & 235 \\ 0 & 269 & 1 & 53 \\ 0 & 253 & 0 & 271 \end{array}\right].$$

Agora, para a segunda coluna, podemos utilizar o elemento 269 *na segunda linha como pivô, já que ele possui inverso módulo* 270. *O inverso de* 269 *é o próprio* 269. *Multiplicamos então a segunda linha por* 269, *obtendo a matriz*

$$\left[\begin{array}{ccc|c} 1 & 3 & 0 & 235 \\ 0 & 1 & 269 & 217 \\ 0 & 253 & 0 & 271 \end{array}\right].$$

Para zerar os outros elementos desta coluna, fazemos a primeira linha menos três vezes a segunda e a terceira linha menos 253 *vezes a segunda. Obtemos a matriz*

$$\left[\begin{array}{ccc|c} 1 & 0 & 3 & 124 \\ 0 & 1 & 269 & 217 \\ 0 & 0 & 253 & 80 \end{array}\right].$$

Finalmente, para a terceira coluna, podemos utilizar o elemento 253 *na terceira linha como pivô, já que ele possui inverso módulo* 270. *O inverso de* 253 *é* 127. *Multiplicamos então a terceira linha por* 127, *obtendo*

$$\left[\begin{array}{ccc|c} 1 & 0 & 3 & 124 \\ 0 & 1 & 269 & 217 \\ 0 & 0 & 1 & 170 \end{array}\right].$$

Para zerar os outros elementos desta coluna, fazemos a primeira linha menos três vezes a terceira e a segunda linha menos 269 *vezes a terceira, obtendo*

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 154 \\ 0 & 1 & 0 & 117 \\ 0 & 0 & 1 & 170 \end{array}\right].$$

Assim, $x_1 = 154$, $x_2 = 117$ *e* $x_3 = 170$. *De fato,* $6^{154} \equiv 2 \pmod{271}$, $6^{117} \equiv 3 \pmod{271}$ *e* $6^{170} \equiv 5 \pmod{271}$.

Para concluir, calculamos g^{-1}, *o inverso de* g *módulo* 271, *obtendo* $g^{-1} = 226$. *Em seguida, selecionamos aleatoriamente um valor de* t *tal que* $h(g^{-1})^t \bmod p$ *possa ser completamente fatorado com os primos da base* P. *O valor* $t = 102$ *é apropriado, já que* $217.226^{102} \bmod 271 = 128 = 2^7$. *Assim,* $f_1 = 7$ *e* $f_2 = f_3 = 0$. *Logo,* $x = (t + f_1x_1 + f_2x_2 + f_3x_3) \bmod p - 1 = (102 + 7.154) \bmod 270 = 100$.

6.6 Exercícios

1. Descreva formalmente as versões generalizadas dos algoritmos ingênuo, "Baby-Step/Giant-Step", "Rho" e Pohlig-Hellman para o cálculo do PLD em um grupo finito cíclico $\mathcal{G} = (G, *)$ de ordem n qualquer.

2. Resolva o Problema do Logaritmo Discreto $g^x \equiv h \pmod{p}$ utilizando o Algoritmo "Baby-Step/Giant-Step" nos casos abaixo:

   2.1. $g = 155$, $h = 181$, $p = 211$

   2.2. $g = 33$, $h = 123$, $p = 269$

3. Dados os parâmetros públicos $g = 107$ e $p = 223$ e a chave pública $c = 86$ de uma implementação do método de criptografia El Gamal, utilize o Algoritmo "Baby-Step/Giant-Step" para obter a chave privada e decriptar a mensagem $\widehat{m} = (104, 202)$.

4. Resolva o Problema do Logaritmo Discreto $g^x \equiv h \pmod{p}$ utilizando o Algoritmo "Rho" nos casos abaixo:

   4.1. $g = 6$, $h = 2$, $p = 157$

   4.2. $g = 78$, $h = 122$, $p = 193$

5. Dados os parâmetros públicos $g = 66$ e $p = 113$ e a chave pública $c = 29$ de uma implementação do método de criptografia El Gamal, utilize o Algoritmo "Rho" para obter a chave privada e decriptar a mensagem $\widehat{m} = (62, 45)$.

6. Resolva o Problema do Logaritmo Discreto $g^x \equiv h \pmod{p}$ utilizando o Algoritmo de Pohlig-Hellman nos casos abaixo:

   6.1. $g = 59$, $h = 51$, $p = 101$

   6.2. $g = 57$, $h = 13$, $p = 109$

7. Resolva o Problema do Logaritmo Discreto $g^x \equiv h \pmod{p}$ utilizando o Algoritmo do Cálculo de Índices com base de primos $P = [2, 3, 5]$ nos casos abaixo:

   7.1. $g = 43$, $h = 42$, $p = 103$ (sabendo que $g^{53} \equiv 5 \pmod{103}$, $g^{31} \equiv 75 \pmod{103}$, $g^{66} \equiv 30 \pmod{103}$ e $h(g^{-1})^3 \equiv 64 \pmod{103}$)

   7.2. $g = 29$, $h = 30$, $p = 127$ (sabendo que $g^{101} \equiv 6 \pmod{127}$, $g^{21} \equiv 20 \pmod{127}$, $g^{75} \equiv 10 \pmod{127}$ e $h(g^{-1})^{26} \equiv 50 \pmod{127}$)

Bibliografia

[1] ADLEMAN, L. M. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In: KOSARAJU, S. R. (Ed.). *Anais do "20th Annual Symposium on Foundations of Computer Science"*. New York: IEEE, 1979. p. 55–60.

[2] BLAKE, I. F.; SEROUSSI, G.; SMART, N. P. *Elliptic Curves in Cryptography*. Cambridge: Cambridge University Press, 1999.

[3] COUTINHO, S. C. *Números Inteiros e Criptografia RSA*. Segunda edição. Rio de Janeiro: IMPA, 2013.

[4] DIFFIE, W.; HELLMAN, M. E. New directions in cryptography. *IEEE Transactions on Information Theory*, v. 22, n. 6, p. 644–654, 1976.

[5] EL GAMAL, T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, v. 31, n. 4, p. 469–472, 1985.

[6] EUCLIDES. *The Thirteen Books of the Elements, Volume 2: Books III–IX*: Traduzido por T. L. Heath. Second edition. New York: Dover Publications, 1956.

[7] GAUSS, C. F. *Disquisitiones Arithmeticae*: Traduzido por A. A. Clarke. New Haven: Yale University Press, 1965.

[8] GOLUB, G. H.; VAN LOAN, C. F. *Matrix Computations*. Fourth edition. Baltimore: Johns Hopkins University Press, 2012.

[9] HODGES, A. *Alan Turing: The Enigma*. Centennial edition. Princeton: Princeton University Press, 2012.

[10] HOFFSTEIN, J.; PIPHER, J.; SILVERMAN, J. H. *An Introduction to Mathematical Cryptography*. Heidelberg: Springer, 2008.

[11] KAHN, D. *The Codebrakers*. Revised and updated edition. New York: Scribner, 1996.

[12] KNUTH, D. E. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*. Third edition. Reading: Addison-Wesley, 1997.

[13] KOBLITZ, N. *A Course in Number Theory and Cryptography*. Second edition. Heidelberg: Springer, 1994.

[14] KRAITCHIK, M. *Théorie des Nombres*. Paris: Gauthier-Villars, 1922.

[15] KRAVITZ, D. *Digital signature algorithm*. 1993. US Patent 5,231,668. Disponível em: <http://www.google.com/patents/US5231668>.

[16] MENEZES, A. J.; VAN OORSCHOT, P. C.; VANSTONE, S. A. *Handbook of Applied Cryptography*. New York: CRC Press, 1996.

[17] MILLER, G. L. Riemann's hypothesis and tests for primality. *Journal of Computer and System Sciences*, v. 13, n. 3, p. 300–317, 1976.

[18] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *Digital Signature Standard*. 1993. FIPS 186. Disponível em: <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>.

[19] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *Recommendation for Key Management*. 2012. Special Publication 800-57 Part 1 Revision 3. Disponível em: <http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57_part1_rev3_general.pdf>.

[20] NICÔMACO. *Introduction to Arithmetic*: Traduzido por M. L. D'Ooge. New York: Macmillan Company, 1926.

[21] POHLIG, S. C.; HELLMAN, M. E. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Transactions on Information Theory*, v. 24, n. 1, p. 106–110, 1978.

[22] POLLARD, J. M. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, v. 32, n. 143, p. 918–924, 1978.

[23] RABIN, M. O. Probabilistic algorithm for testing primality. *Journal of Number Theory*, v. 12, n. 1, p. 128–138, 1980.

[24] RIBENBOIM, P. *Números Primos. Velhos Mistérios e Novos Recordes*. Rio de Janeiro: IMPA, 2012.

[25] RIVEST, R. L.; SHAMIR, A.; ADLEMAN, L. M. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, v. 21, n. 2, p. 120–126, 1978.

[26] SHANKS, D. Class number, a theory of factorization, and genera. In: LEWIS, D. J. (Ed.). *Anais do "Symposia in Pure Mathematics"*. Providence: American Mathematical Society, 1971. v. 20, p. 415–440.

[27] UNICODE CONSORTIUM. *Unicode 6.3 Character Code Charts*. 2013. Disponível em: <http://www.unicode.org/charts/>.

[28] YONG, L. L.; SE, A. T. *Fleeting Footsteps: Tracing the Conception of Arithmetic and Algebra in Ancient China*: Tradução de "Sunzi Suanjing". Revised edition. Singapore: World Scientific Publishing, 2004.

Índice

Abel, 62
Adleman, 7, 45, 115
Algoritmo
  Baby-Step/Giant-Step, 104
  Chinês do Resto, 37
  de Assinatura
    DSA, 96
    El Gamal, 89
  de Decriptação El Gamal, 86
  de Divisão Inteira, 14
  de Encriptação El Gamal, 85
  de Fatoração, 45
  de Gauss, 75
  de Geração de Chaves
    da Assinatura DSA, 95
    da Assinatura El Gamal, 88
    da Criptografia El Gamal, 83
  de Pohlig-Hellman, 111
  de Potenciação Modular, 32
  de Verificação da Assinatura
    DSA, 97
    El Gamal, 91
  do Cálculo de Índices, 117
  do Crivo de Eratóstenes, 50
  do Teste de Miller-Rabin, 59
  Euclidiano Estendido, 21
  Ingênuo para o PLD, 102
  Rho, 107
Assinatura Digital, 6, 79
  DSA, 94
    Assinatura, 96
    Geração de Chaves, 95
    Verificação, 97
  El Gamal, 88
    Assinatura, 89
    Geração de Chaves, 88
    Verificação, 91

César, 3
Carmichael, 53
Chave, 2
  Efêmera, 85
  Pública, 6
  Privada, 2
Cifra de César, 3
Classe de Equivalência, 25
Congruência Módulo n, 27
Conjunto Quociente, 26
Criptografia
  com Curvas Elípticas, 82, 118
  de Chave Pública, 6, 78
  de Chave Privada, 2
  El Gamal, 82
    Decriptação, 86
    Encriptação, 85
    Geração de Chaves, 83
Crivo de Eratóstenes, 50

Decriptação, 2
  El Gamal, 86
Diffie, 6
Dividendo, 13
Divisão Inteira, 13
Divisor, 13, 17
  Próprio, 17
  Primo, 42
DSA, 94
  Assinatura, 96
  Geração de Chaves, 95
  Verificação, 97

El Gamal, 7, 82
  Método de Assinatura Digital, 88
    Assinatura, 89
    Geração de Chaves, 88
    Verificação, 91
  Método de Criptografia, 82
    Decriptação, 86
    Encriptação, 85
    Geração de Chaves, 83
Encriptação, 2
  El Gamal, 85
  Probabilística, 85
Enigma, 4

Eratóstenes, 46
Euclides, 18, 43, 62
Euler, 51, 61

Fator, 17
    Próprio, 17
    Primo, 42
Fatoração em Primos, 44
Fermat, 51, 62
$\phi(n)$, 64
Forma Reduzida Módulo n, 29
Função
    ϕ de Euler, 64
    Hash, 89
        Criptograficamente Segura, 89

Galois, 61
Gauss, 61, 74
Geração de Chaves
    da Assinatura DSA, 95
    da Assinatura El Gamal, 88
    da Criptografia El Gamal, 83
Gerador, 70
Grupo, 62
    Abeliano, 62
    Cíclico, 70
    Comutativo, 62
    Finito, 63
    $U(n)$, 64

Hellman, 6, 109

Inteiros Módulo n, 29
Inverso Multiplicativo Modular, 33

Lagrange, 61, 67
Leibniz, 51
Lema Chave, 72

Máximo Divisor Comum, 17
Múltiplo, 17
Miller, 56

Número
    Composto, 41
    de Carmichael, 53
    P-Suave, 115
    Primo, 41
Nicômaco, 46

Ordem
    de um Elemento, 69
    de um Grupo, 63

Pohlig, 109
Pollard, 105
Potenciação Modular, 31
Primorial, 43
Problema do Logaritmo Discreto, 70, 101
Produto Modular, 30
Propriedade
    Fundamental dos Primos, 43
Pseudoprimo, 53
    de Fermat, 53
    Forte, 57

Quociente, 13

Rabin, 56
Raiz Primitiva, 70
Relação de Equivalência, 24
Resto, 13
Rivest, 7, 45
RSA, 45

Shamir, 7, 45
Shanks, 103
Soma Modular, 29
Subgrupo, 66
    Cíclico, 70
    Próprio, 67
Subtração Modular, 30
Sunzi (Sun Tzu), 35

Teorema
    da Inversão Modular, 34
    Chinês do Resto, 35
    da Infinidade dos Primos, 43
    da Raiz Primitiva, 74
    da Unicidade da Fatoração, 44
    de Fermat, 51
    de Lagrange, 67
    Fundamental da Aritmética, 44
Teste
    de Fermat, 52
    de Miller-Rabin, 59
Turing, 5

$U(n)$, 64
Unicidade
    da Fatoração, 44
    do Quociente e Resto, 14