Uma implementação do método das curvas elíticas para fatoração de números inteiros

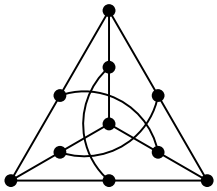
Adriana Betânia de Paula Molgora

Dissertação de Mestrado

Orientação: Prof. Dra. Elisabete Sousa Freitas

Área de Concentração: Teoria dos Números Computacional

Dissertação apresentada ao Departamento de Computação e Estatística (DCT) da Universidade Federal de Mato Grosso do Sul (UFMS) como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.



Departamento de Computação e Estatística
Centro de Ciências Exatas e Tecnologia
Universidade Federal de Mato Grosso do Sul
18 de março de 2006

Dedicatória

Aos meus pais, José Ramos de Paula Filho (in memorian) e Maria Veloso de Paula, meus grandes mestres na escola da vida.

Ao meu esposo, Mário Marcos Molgora e minhas filhas, Flávia Aline e Larissa Beatriz, pelos momentos em que não pude estar presente, fisicamente, mas que estiveram sempre em meu coração e pensamentos.

Agradecimentos

A Deus, fonte de toda sabedoria, por sua constante presença em minha vida.

À minha orientadora e amiga, Profa. Dra. Elisabete Sousa Freitas, pela orientação, dedicação, paciência e, principalmente, pela confiança em mim depositada em todos os momentos.

Aos professores Dr. Edson Norberto Cáceres e Dr. Cícero Fernandes de Carvalho, pela disponibilidade de participarem da Banca Examinadora e pelas valiosas sugestões apresentadas.

A todos os professores do mestrado, pela boa instrução fornecida, contribuindo para a obtenção desse título. Aos funcionários do DCT/UFMS e DMT/UFMS pela atenção a mim dispensada nos momentos em que precisei.

Aos colegas de mestrado, em especial à Cláudia Peviani pela amizade em que cultivamos durante as muitas viagens para Campo Grande. À professora e amiga Gladis, pelas palavras de apoio a mim direcionadas.

À Universidade Estadual de Mato Grosso do Sul, pela concessão do afastamento possibilitando a realização desse trabalho.

Ao meu irmão Fabrício, pela disposição e paciência em tirar minhas muitas dúvidas e também pela revisão preliminar dessa dissertação. Ao meu irmão Nilton pelo auxílio dispensado nos momentos em que precisei. Sou muitíssimo grata a vocês.

À minha mãe e aos meus sogros Aparecida e Milton, que muitas vezes zelaram

por minhas filhas, nos períodos de minha ausência. Agradeço de todo o coração.

A todos meus familiares, em especial à Danieli, Felipe, Elizângela, Juninho, Anderson, Márcia, Sérgio, Francine, Orlando e todos aqueles que, direta ou indiretamente, através de ações ou palavras de apoio, contribuíram para o desenvolvimento desse trabalho.

Em especial, gostaria de deixar registrada a minha gratidão ao meu esposo, que com seu jeito maravilhoso de ser, propiciou-me todo apoio, compreensão e carinho necessários para vencer todos os obstáculos que se apresentaram. Também, um agradecimento especialíssimo às minhas filhas, pelo tesouro que representam em nossas vidas.

Resumo

O problema de fatoração de inteiros tem motivado diversos estudos devido a sua aplicação em sistemas criptográficos, como o RSA, que têm sua segurança baseada na dificuldade de fatorar números grandes. Um dos métodos mais poderosos utilizados na fatoração de inteiros é o método das curvas elíticas. Para implementar esse método é necessário que se realize a análise dos diversos aspectos envolvidos neste processo como o tratamento de cálculos com números muito grandes e os algoritmos aplicados à aritmética das curvas elíticas. Considerando esses fatos, este trabalho apresenta um estudo deste método de fatoração descrevendo os elementos matemáticos envolvidos em seu algoritmo bem como o estudo de uma implementação do mesmo.

Conteúdo

Capítulo 1

Introdução

O problema de fatoração de inteiros tem ocupado um lugar de destaque na teoria dos números. Tal interesse deve-se ao fato de que um algoritmo eficiente de fatoração pode comprometer totalmente a segurança de sistemas criptográficos de chave pública como, por exemplo, o RSA [GS96].

Nas últimas décadas, métodos para fatoração de inteiros tem sido amplamente estudados e, como resultado, diversos algoritmos foram desenvolvidos [Len87, LLMP90, Pol74, Pom85]. Dentre esses métodos, foi criado o método das curvas elíticas (Elliptic Curve Method - ECM) devido a H. W. Lenstra Jr., que se utiliza da estrutura de grupo dos pontos de uma curva elítica para encontrar um fator. Essas curvas têm sido bastante estudadas em teoria dos números e geometria algébrica.

Desde a sua concepção, o método das curvas elíticas tem sido descrito em alguns trabalhos [Bre86, Bre00, Bre89, Car03, Mon94], e estudos têm sido realizados com o objetivo de apresentar melhorias para este método [PO96, CMKM03, ELM03]. No entanto, esses trabalhos não costumam apresentar uma análise detalhada do processo de implementação deste método, reunindo todos os aspectos relacionados com sua implementação cobrindo todo o processo de fatoração.

A implementação do método das curvas elíticas não é uma tarefa trivial, pois vários aspectos devem ser estudados antes de sua realização. Uma implementação deste método envolve conhecimentos tanto matemáticos quanto computacionais.

Dentre as tarefas a serem realizadas para sua implementação podem se destacar: a análise de procedimentos para a realização de cálculos exatos com inteiros de centenas de algarismos, estudos de algoritmos adequados para a aritmética de curvas elíticas e a escolha de curvas apropriadas a serem empregadas no processo de fatoração.

Este trabalho é um estudo teórico e prático do método das curvas elíticas e tem como objetivo principal disponibilizar conhecimentos sobre os aspectos de sua implementação de forma detalhada e didática bem como apresentar uma análise dos resultados obtidos com essa implementação. Para alcançar esse objetivo, foram traçados alguns objetivos específicos que são apresentados na Seção 1.1.

1.1 Objetivos

O primeiro objetivo específico desse trabalho compreende o estudo teórico do método das curvas elíticas. Nesse sentido, buscou-se estudar a fundamentação matemática envolvida nesse método de fatoração, bem como os algoritmos aplicados em seu processo de implementação, cobrindo seu funcionamento, vantagens e desvantagens.

O segundo objetivo específico está relacionado com o estudo prático do método das curvas elíticas. Para isso, buscou-se realizar uma implementação desse método, documentando algumas alternativas de implementação do mesmo e analisando seu desempenho através da realização de testes.

1.2 Organização do texto

Este texto está organizado em seis capítulos como apresentado a seguir:

- Capítulo 2: apresenta, de forma sucinta, o conceito de criptografia e o funcionamento do sistema criptográfico RSA bem como sua relação com o problema de fatoração de números inteiros;

- Capítulo 3: apresenta os conceitos elementares da teoria de curvas elíticas e descreve o algoritmo do método de fatoração que utiliza essas curvas;
- Capítulo 4: apresenta algumas alternativas para a implementação do método das curvas elíticas e os algoritmos fundamentais aplicados nesse processo;
- Capítulo 5: apresenta uma descrição detalhada dos aspectos mais relevantes da implementação do método das curvas elíticas desenvolvida durante este trabalho;
- Capítulo 6: descreve as características dos testes realizados com a implementação e a análise dos resultados experimentais obtidos;
- Capítulo 7: apresenta as considerações finais e as contribuições deste trabalho e relata uma série de trabalhos futuros a serem conduzidos no sentido de evidenciar novas contribuições nesta área.

Buscou-se descrever o conteúdo de forma didática, esperando-se que esse trabalho possa dar um melhor entendimento ao assunto contribuindo para trabalhos futuros nesta área.

Capítulo 2

Criptografia e fatoração de números inteiros

É difícil compreender a importância da fatoração de números inteiros para a criptografia sem antes entender os conceitos básicos utilizados por sistemas criptográficos. De maneira bem resumida, este capítulo apresenta os conceitos fundamentais da criptografia e a relação do problema de fatoração com o sistema criptográfico RSA, cuja segurança tem motivado diversas pesquisas na área.

2.1 Criptografia

Criptografia é a ciência ou a arte de escrever informações secretas em cifras ou códigos de modo que somente seu destinatário legítimo consiga interpretá-las [Cou00]. De forma mais abrangente, a criptografia pode ser considerada como o estudo de técnicas matemáticas visando a segurança da informação.

As técnicas criptográficas implementam as quatro propriedades seguintes, tanto em seus aspectos teóricos quanto práticos [MvOV97]:

- Confidencialidade: impede que pessoas não autorizadas tenham acesso ao conteúdo da informação;
- Integridade: garante que o conteúdo da informação não foi alterado;

- Autenticidade: garante a identidade de quem produziu a informação;
- Não-repúdio: impede que alguém negue o envio ou recepção de uma determinada informação.

Codificar uma mensagem consiste no processo de transformar a mensagem original em uma mensagem cifrada utilizando para isso funções matemáticas e um segredo denominado chave. O processo de reverter a mensagem cifrada para a mensagem original com o uso da chave é denominado decodificação [GS96].

Existem basicamente duas classes de criptografia: a criptografia de chave privada também denominada de criptografia simétrica, e a criptografia de chave pública também denominada de criptografia assimétrica.

Na criptografia de chave privada é utilizada uma única chave tanto para a cifragem como para a decifragem da mensagem. Neste caso, tanto o emissor quanto o destinatário tem acesso a mesma chave devendo mantê-la em segredo.

Na criptografia de chave pública, é utilizada uma chave para cifrar a mensagem, denominada de chave pública, e outra chave para decifrar, denominada chave privada [GS96]. Neste sistema, todos os emissores de mensagem compartilham da chave pública, mas somente o receptor pode decifrar a mensagem, pois é o único proprietário da chave privada.

2.2 Elementos da criptografia

Existem diferentes sistemas criptográficos que utilizam maneiras distintas de cifrar informações através do uso do computador. No entanto, esses sistemas fazem uso de elementos comuns, como [GS96]:

- Algoritmo criptográfico: é a função que executa as tarefas de cifragem e decifragem de dados;
- Chaves criptográficas: compreendem os segredos utilizados pelo algoritmo criptográfico para determinar como os dados serão cifrados ou decifrados;

- Tamanho da chave: as chaves criptográficas tem um tamanho pré-determinado. Geralmente, chaves grandes são mais difíceis de serem descobertas por atacantes do que chaves pequenas;
- Texto: informação a ser cifrada ou texto original;
- Texto cifrado: informação cifrada.

Existem diversos sistemas criptográficos tanto de chave privada quanto de chave pública tendo como objetivo principal a proteção de informações. A maior vantagem dos sistemas de chave privada é a velocidade de cifragem/decifragem, porém sua principal desvantagem é a dificuldade na distribuição e manutenção de chaves. Os sistemas de chave pública apresentam uma desvantagem no desempenho por utilizar sofisticadas funções matemáticas. Entretanto esses sistemas são amplamente utilizados porque não exigem a manutenção de um segredo comum entre as partes. Ou seja, nesses sistemas o fato de saber cifrar uma mensagem não implica em saber decifrar.

Um dos sistemas de chave pública mais conhecidos é o RSA cuja segurança está baseada na dificuldade de fatorar um número composto por dois números primos grandes. A idéia principal desse sistema é apresentada na próxima seção.

2.3 Sistema criptográfico RSA

Antes de apresentar o sistema criptográfico RSA, é interessante lembrar alguns conceitos matemáticos envolvidos em seu funcionamento. Nesse sentido, a Seção 2.3.1 realiza a apresentação desses conceitos para que, em seguida, seja descrito o funcionamento do RSA na Seção 2.3.2.

2.3.1 Teorema da fatoração única

Divisibilidade e divisores

Sejam a e b inteiros. Dizemos que a divide b, se existe um inteiro c tal que $b = a \cdot c$. Quando a divide b, dizemos também que a é divisor de b, ou a é fator de b, ou b é múltiplo de a. Se a divide b, então isto é denotado por $a \mid b$.

Números primos e compostos

Um inteiro $p > 1$ é denominado número **primo** se seus únicos divisores positivos são 1 e o próprio p. Caso contrário, p é denominado número **composto**.

Teorema da fatoração única

TEOREMA 2.3.1 *Todo inteiro positivo $n \geq 2$ pode ser escrito, de modo único, na forma*

$$n = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$$

onde $1 < p_1 < p_2 < \cdots < p_r$ são números primos e $e_1, e_2, \ldots, e_r$ são inteiros positivos [Cou00].

2.3.2 Criptografia RSA

O sistema criptográfico RSA foi inventado em 1978 por R. L. Rivest, A. Shamir e L. Adleman, no Massachussets Institute of Technology (MIT) [RSA77]. Esse sistema foi uma das primeiras soluções para a criptografia de chave pública. Apesar dos contínuos esforços na proposição de novos métodos criptográficos ele ainda representa um importante papel na implementação de mecanismos de autenticação e privacidade nos sistemas de comunicação de dados.

De maneira geral, para se implementar o RSA é necessário o uso de dois parâmetros básicos: dois números primos p e q que constituem o segredo do sistema. A

partir dos primos p e q é calculada a função de Euler:

$$\phi(n) = (p-1)(q-1), \text{ onde } n = p \cdot q$$

Em seguida, obtêm-se um par (n, e) , onde e satisfaz a condição:

$$\text{mdc}(e, \phi(n)) = 1$$

O par (n, e) corresponde a chave pública do sistema que é tornada pública aos emissores para a codificação das mensagens.

Para decodificar a mensagem é necessário o uso da chave privada dada pelo par (n, d) onde d é o inverso de e módulo $\phi(n)$. A chave privada (n, d) é propriedade do receptor da mensagem. É possível o cálculo de d sendo conhecidos os fatores p e q de n.

A escolha dos primos p e q que compõem n é um dos pontos principais da segurança do RSA. A possibilidade de fatorar n implica na obtenção dos fatores p e q e conseqüentemente na decodificação da mensagem. Se p e q forem números pequenos, n poderá ser facilmente fatorado e, portanto, a mensagem poderá ser decifrada. Somente o fato de escolher p e q grandes também não garante a segurança. Os primos p e q, além de serem suficientemente grandes devem ser obtidos a partir de estudos na teoria dos números de forma que a fatoração de n seja bastante improvável.

2.3.3 Complexidade do problema de fatoração de inteiros

Classes de complexidade

A complexidade de tempo de um problema está relacionada com a quantidade de tempo necessária para resolvê-lo. Existem basicamente três classes de problemas [CLRS01]:

- **Classe P:** Consiste nos problemas que podem ser resolvidos em tempo polinomial;
- **Classe NP:** Consiste nos problemas que são "verificáveis" em tempo polinomial;
- **Classe NP-Completo:** consiste nos problemas em NP cuja complexidade é relacionada com a complexidade da classe inteira. Ou seja, possui a propriedade de que, se qualquer problema NP-Completo pode ser resolvido em tempo polinomial, então todo problema em NP tem uma solução em tempo polinomial.

O problema de fatoração de inteiros ainda é considerado "intratável computacionalmente", pois não existe um algoritmo com tempo de execução polinomial que realize a fatoração de qualquer número inteiro. Não se sabe, exatamente, em que classe de complexidade está situado o problema de fatoração de inteiros [np06]. Acredita-se que esse problema seja NP-Completo, no entanto, isso ainda não foi provado.

Números com fatores muito grandes como, por exemplo, uma chave pública do RSA são muito difíceis de serem fatorados. Diante disso, acredita-se que tanto quebrar o sistema RSA quanto fatorar n sejam problemas equivalentes, embora até agora isso não tenha sido demonstrado [Cou00].

A relação direta entre a segurança do RSA e o problema de fatoração, constitui uma boa razão para o estudo dos métodos de fatoração de inteiros. Existem diversos algoritmos de fatoração de inteiros[Pol74, Len87, Pom85, LLMP90]. Dentre os métodos mais poderosos utilizados atualmente, estão o Crivo do Corpo de Números (Number Field Sieve - NFS) [LLMP90] e o Método das Curvas Elíticas (Elliptic Curve Method - ECM) [Len87]. A idéia principal do método de fatoração das curvas elíticas é fazer uso de uma estrutura de grupo dos pontos de uma curva elítica para encontrar um fator primo de um número composto n. Uma melhor descrição deste método é apresentada no próximo capítulo.

Capítulo 3

Método de fatoração das curvas elíticas

O método de fatoração das curvas elíticas foi descoberto por H. W. Lenstra Jr. em 1985. Desde a sua concepção, este método tem sido descrito em alguns trabalhos [Bre86, Bre00, Bre89, Car03, Mon94], e alguns estudos tem sido realizados com o objetivo de apresentar melhorias para este método [PO96, CMKM03, ELM03].

Este capítulo apresenta o algoritmo original de Lenstra. Antes porém, para um melhor entendimento do mesmo, serão introduzidos alguns conceitos elementares da teoria de curvas elíticas baseados em [Car03, Gon79, Sil86, ST92]. Informações adicionais podem ser encontradas nessas mesmas referências.

3.1 Curvas elíticas

As curvas elíticas podem ser definidas sobre um corpo K como, por exemplo, o corpo dos números complexos. Elas são descritas por uma equação cúbica do tipo

$$ax^3 + bx^2y + cxy^2 + dy^3 + ex^2 + fxy + gy^2 + hx + iy + j = 0 \qquad (3.1)$$

onde $a, b, \ldots, j$ são elementos de um corpo K. Fazendo uma mudança apropriada

de variáveis, uma curva elítica geral sobre um corpo de característica diferente de 2 e 3, pode ser escrita na forma normal de Weierstrass:

$$y^2 = f(x) = x^3 + ax + b \tag{3.2}$$

onde

$$4a^3 + 27b^2 \neq 0 \tag{3.3}$$

o que garante que f não tem raízes múltiplas.

Essas curvas podem assumir diversas formas, dependendo dos parâmetros utilizados. As Figuras 3.1, 3.2, 3.3 e 3.4 ilustram alguns exemplos de gráficos de curvas elíticas.

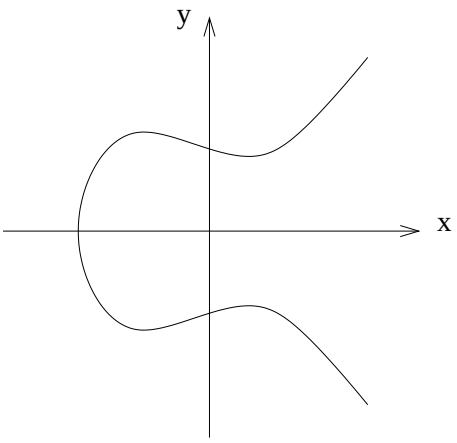


Figura 3.1: Exemplo 1

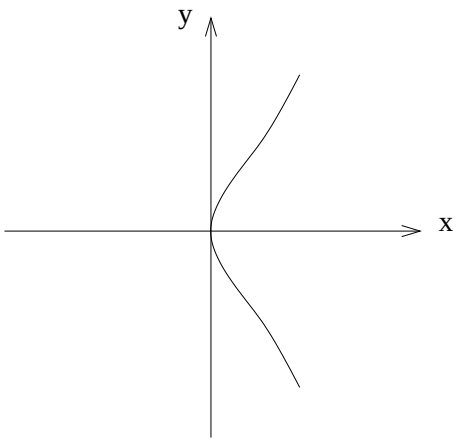


Figura 3.2: Exemplo 2

Um dos fatos importantes da teoria das curvas elíticas é que se pode definir uma estrutura de grupo abeliano sobre essas curvas. Dados dois pontos P e Q sobre uma curva elítica, pode-se obter um terceiro ponto sobre ela traçando-se a reta que passa por P e Q e encontrando-se assim o ponto de intersecção entre a reta e a curva como mostra a Figura 3.5. Tal ponto é denotado por $P * Q$.

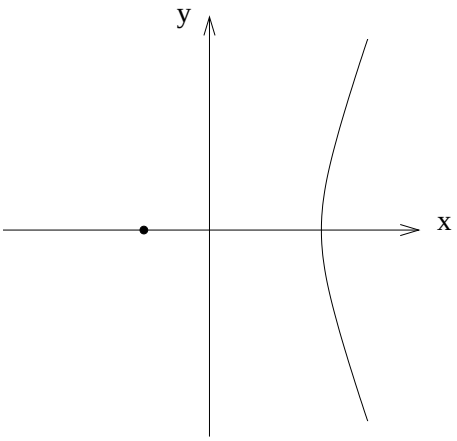


Figura 3.3: Exemplo 3

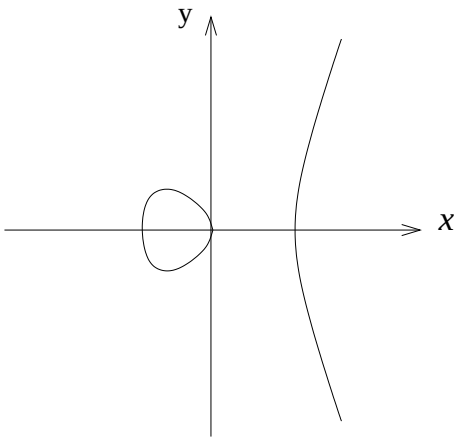


Figura 3.4: Exemplo 4

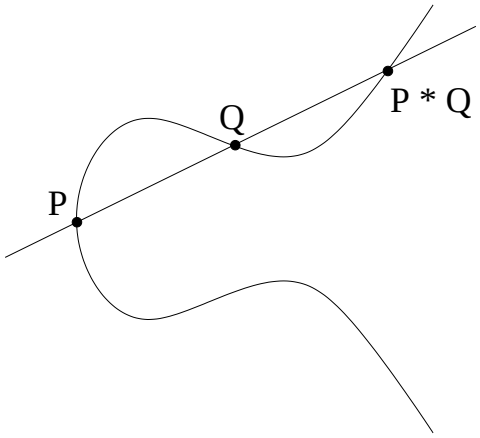


Figura 3.5: Obtenção do ponto $P * Q$.

Se os pontos são iguais, considera-se a reta por P e Q como sendo a reta tangente à curva em P e obtêm-se o terceiro ponto $P * P$, como mostra a Figura 3.6.

Define-se a operação de adição, como segue:

Operação de adição: Seja O um ponto qualquer sobre C. Considere a operação $+$ que a cada par (P, Q) de pontos de C associa o ponto $P + Q$ sobre C definido por $P + Q = O * (P * Q)$. Assim, dados dois pontos P e Q, encontra-se o terceiro ponto $P * Q$ traçando a reta L_1 que passa por P e Q; em seguida traça-se a reta L_2 que passa por O e $P * Q$ e encontra-se o terceiro ponto que é $P + Q$. Ver Figura 3.7.

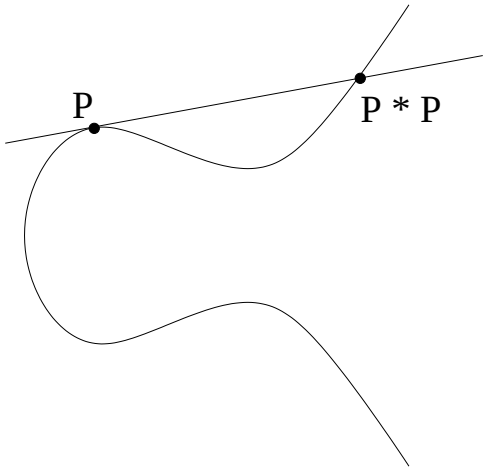


Figura 3.6: Obtenção do ponto $P * P$.

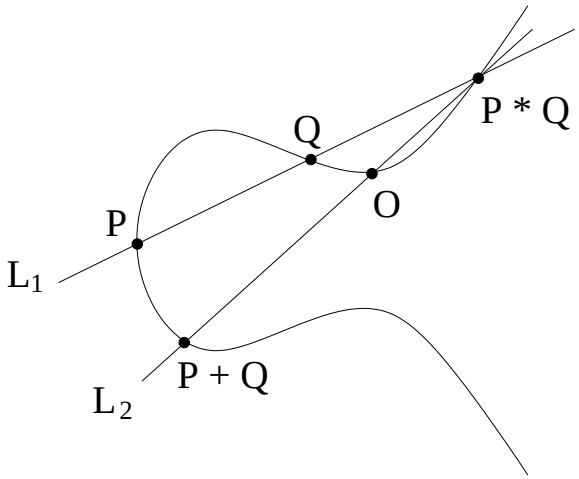


Figura 3.7: Obtenção do ponto $P + Q$.

Considerando-se o ponto O como sendo o ponto no infinito (o ponto onde as retas verticais se intersectam), para se adicionar dois pontos P_1 e P_2 traçamos primeiramente a reta que passa por P_1 e P_2 encontrando o ponto de intersecção entre a reta e a curva. Em seguida traçamos a reta que passa por O e por $P_1 * P_2$ que é exatamente a reta vertical que passa por $P_1 * P_2$. O ponto $P_1 + P_2$ será o ponto simétrico de $P_1 * P_2$. Ver Figura 3.8.

Pode-se mostrar que a operação de adição possui as seguintes propriedades:

Associatividade Quaisquer que sejam os pontos P, Q e R em C tem-se que

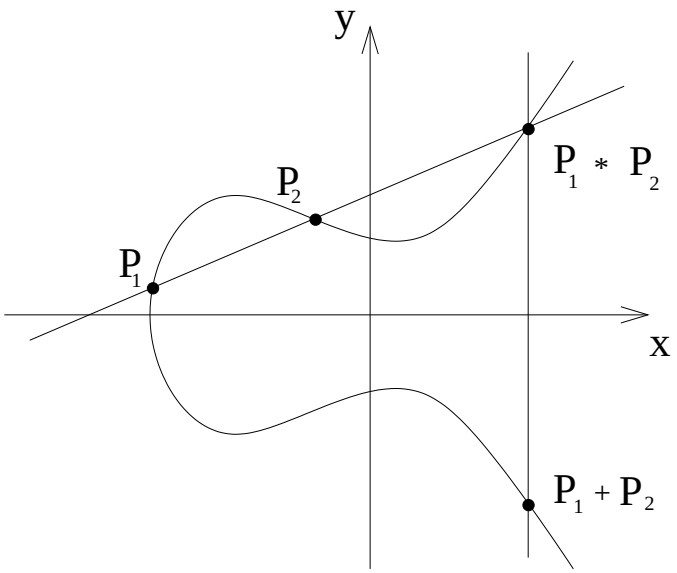


Figura 3.8: Obtenção do ponto $P + Q$ considerando-se o ponto O como sendo o ponto no infinito.

$(P + Q) + R = P + (Q + R)$;

Existência de elemento neutro O ponto $O \in C$ é tal que $P + O = O + P = P, \forall P \in C$, logo O é o elemento neutro da adição;

Existência de elemento simétrico Para cada ponto $P \in C$, existe o ponto $-P \in C$ tal que $P + (-P) = (-P) + P = O$.

Também vale a seguinte propriedade:

Comutatividade Quaisquer que sejam os pontos $P, Q \in C$, tem-se $P + Q = Q + P$.

Ou seja, o conjunto de pontos da curva com a operação de adição possui estrutura de grupo abeliano. Algebricamente, pode-se mostrar que a adição de dois pontos da curva é dada por:

- **Adição dos pontos P_1 e P_2 com $P_1 \neq P_2$ e $x_1 \neq x_2$:** Dados dois pontos $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$ pertencentes à curva $C(K)$ (definida sobre um

corpo K) de equação $y^2 = x^3 + ax + b$, obtêm-se a soma $P_1 + P_2 = (x_3, y_3)$ através das seguintes fórmulas

$$\begin{cases} \lambda = \dfrac{y_2 - y_1}{x_2 - x_1} \\ \\ x_3 = \lambda^2 - x_1 - x_2 \\ \\ y_3 = \lambda(x_1 - x_3) - y_1 \end{cases} \tag{3.4}$$

De forma análoga pode-se deduzir as fórmulas para $P_1 + P_2$ no caso em que $P_1 = P_2$:

- **Duplicação do ponto P_1:** Seja $P_1 = (x_1, y_1)$. O resultado de $2P_1 = P_1 + P_1 = (x_3, y_3)$, é obtido através das fórmulas

$$\begin{cases} \lambda = \dfrac{3x_1^2 + a}{2y_1} \\ \\ x_3 = \lambda^2 - 2x_1 \\ \\ y_3 = \lambda(x_1 - x_3) - y_1 \end{cases} \tag{3.5}$$

- **Adição dos pontos P_1 e P_2 com $P_1 \neq P_2$ e $x_1 = x_2$:** Nesse caso $P_2 = -P_1$ e tem-se $P_1 + P_2 = O$.

A multiplicação de um ponto P de uma curva C por um inteiro $k > 2$ denominada **operação escalar de um ponto** é a principal operação realizada no processo de fatoração pelo método das curvas elíticas. Essa operação pode ser efetuada utilizando-se as fórmulas de adição e duplicação de ponto, ou seja, adicionando o ponto a ele mesmo tantas vezes quanto for o fator de multiplicação:

$$kP = \underbrace{P + P + \cdots + P}_{k \text{ vezes}} \tag{3.6}$$

3.2 Curvas elíticas sobre corpos finitos

A Seção 3.2.1 apresenta a definição de curvas elíticas sobre corpos finitos e a Seção 3.2.2. apresenta a definição de ordem de uma curva elítica e o Teorema de Hasse, que serão posteriormente utilizados na descrição do método de fatoração das curvas elíticas.

3.2.1 Curvas elíticas sobre $\mathbb{Z}_p$

Seja p um número primo. O corpo finito $\mathbb{Z}_p$ consiste do conjunto $\mathbb{Z}_p = \{0, 1, 2, ..., p-1\}$, onde as operações de adição e multiplicação sobre esse conjunto são definidas da seguinte maneira:

Adição: Se $a, b \in \mathbb{Z}_p$, então $a + b = r$, onde r, $0 \leq r \leq p-1$, é o resto da divisão de $a+b$ (adição em $\mathbb{Z}$) por p. Esta operação é denominada **adição módulo p**.

Multiplicação: Se $a, b \in \mathbb{Z}_p$, então $a \cdot b = s$, onde s, $0 \leq s \leq p-1$, é o resto da divisão de $a \cdot b$ (multiplicação em $\mathbb{Z}$) por p. Esta operação é denominada **multiplicação módulo p**.

Uma curva elítica C sobre $\mathbb{Z}_p$ denotada por $C(\mathbb{Z}_p)$ é definida pela equação da forma

$$y^2 = x^3 + ax + b \mod p \tag{3.7}$$

onde $a, b \in \mathbb{Z}_p$ e $4a^3 + 27b^2 \neq 0 \mod p$, juntamente com o ponto no infinito O. O conjunto $C(\mathbb{Z}_p)$ consiste de todos os pontos $(x, y), x, y \in \mathbb{Z}_p$ satisfazendo a equação (3.7).

A operação de adição dos pontos $P_1 + P_2 = P_3$ onde $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ e $P_3 = (x_3, y_3)$ em uma curva elítica sobre $\mathbb{Z}_p$ é obtida por:

$$\begin{cases} \lambda = \dfrac{y_2 - y_1}{x_2 - x_1} \mod p \\ x_3 = \lambda^2 - x_1 - x_2 \mod p \\ y_3 = \lambda(x_1 - x_3) - y_1 \mod p \end{cases} \tag{3.8}$$

para $P_1 \neq P_2$ e $x_1 \neq x_2 \bmod p$.

De forma análoga, a operação de duplicação de um ponto dada por $2P_1 = P_1 + P_1 = (x_3, y_3)$, com $P_1 = (x_1, y_1)$ é dada por:

$$\begin{cases} \lambda = \dfrac{3x_1^2 + a}{2y_1} \mod p \\ x_3 = \lambda^2 - x_1 \mod p \\ y_3 = \lambda(x_1 - x_3) - y_1 \mod p \end{cases} \tag{3.9}$$

Exemplos:

1. Seja $p = 23$ e considere a curva $C : y^2 = x^3 + x + 1$ definida sobre $\mathbb{Z}_{23}$. Observe que $4a^3 + 27b^2 = 4 + 4 = 8 \neq 0 \bmod 23$. Os pontos com coordenadas em $\mathbb{Z}_{23}$ pertencentes a essa curva são (0,1);(0,22); (1,7); (1,16); (3,10); (3,13); (4,0); (5,4); (5,19); (6,4); (6,19); (7,11); (7,12); (9,7); (9,16); (11,3); (11,20); (12,4); (12,19); (13,7); (13,16); (17,3); (17,20); (18,3); (18,20); (19,5); (19,18) e o ponto O.

2. Considere a mesma curva do exemplo anterior e os pontos $P = (3; 10)$ e $Q = (9; 7)$ (note que $P, Q \in \mathbb{Z}_{23}$).

   a) $P + Q = (x_3, y_3)$ é dado por:

$$\begin{cases} \lambda = \dfrac{7-10}{9-3} = \dfrac{-3}{6} = \dfrac{-1}{2} = 11 \\ \\ x_3 = 11^2 - 3 - 9 = 6 - 3 - 9 = -6 = 17 \\ \\ y_3 = 11(3-(-6)) - 10 = 11(9) - 10 = 20 \end{cases}$$

Portanto $P + Q = (3, 10) + (9, 7) = (17, 20)$ em $\mathbb{Z}_{23}$.

b) $2P = P + P = (x_3, y_3)$ é dado por:

$$\begin{cases} \lambda = \dfrac{3(3^2)+1}{20} = \dfrac{5}{20} = \dfrac{1}{4} = 6 \\ \\ x_3 = 6^2 - 6 = 7 \\ \\ y_3 = 6(3-7) - 10 = 18 + 4 - 10 = 12 \end{cases}$$

Portanto $2P = 2(3, 10) = (7, 12)$ em $\mathbb{Z}_{23}$.

3. Considere a curva $C : y^2 = x^3 + x - 1$ definida sobre $\mathbb{Z}_5$ e o ponto P =(0,3) pertencente a curva.

a) A operação de multiplicação escalar kP para $k = 5$ pode ser efetuada através da seguinte sequência de cálculos:

- Obtém-se $P + P = 2P = (0, 3) + (0, 3) = (1, 1)$, através das fórmulas de duplicação de pontos (para $P_1 = P_2$) :

$$\begin{cases} \lambda = \dfrac{3-1}{2 \cdot 3} = \dfrac{-1}{6} = 1 \mod 5 \\ \\ x_3 = 1^2 - 2 \cdot 0 = 1 \mod 5 \\ \\ y_3 = 1(0-1) - 3 = -4 = 1 \mod 5 \end{cases}$$

- Obtém-se $2P + P = 3P = (1,1) + (0,3) = (3,3)$ através das fórmulas de adição de pontos (para $P_1 \neq P_2$):

$$\begin{cases} \lambda = \dfrac{3-1}{0-1} = -2 = 3 \mod 5 \\ x_3 = 3^2 - 1 - 0 = 8 = 3 \mod 5 \\ y_3 = 3(1-3) - 1 = -7 = 3 \mod 5 \end{cases}$$

- Utilizando as mesmas fórmulas para adição de pontos diferentes na curva, calcula-se $3P + P = 4P = (3,3) + (0,3) = (2,2)$:

$$\begin{cases} \lambda = \dfrac{3-3}{0-3} = 0 \mod 5 \\ x_3 = 0^2 - 3 - 0 = -3 = 2 \mod 5 \\ y_3 = 0(3-2) - 3 = 2 \mod 5 \end{cases}$$

- Finalmente obtém-se $4P + P = 5P = (2,2) + (0,3) = (2,3)$ de:

$$\begin{cases} \lambda = \dfrac{3-2}{0-2} = 2 \mod 5 \\ x_3 = 2^2 - 2 - 0 = 2 \mod 5 \\ y_3 = 2(2-2) - 2 = 3 \mod 5 \end{cases}$$

  Portanto $5P = (2,3)$.

b) Ao tentar calcular kP para $k = 9$, considerando os cálculos efetuados no ítem anterior, obtêm-se os seguintes resultados:

$5P + P = 6P = (2,3) + (0,3) = (3,2)$

$6P + P = 7P = (3,2) + (0,3) = (1,4)$

$7P + P = 8P = (1,4) + (0,3) = (0,2)$

$8P + P = 9P = O$

Ou seja, o ponto no $9P$ é igual ao ponto no infinito O, pois no cálculo de $9P$ obteve-se $\lambda = \frac{3}{0}$.

Observação: Nesses exemplos a operação kP foi efetuada de maneira simplificada. Na prática, existem formas otimizadas para realizar essa tarefa que serão apresentadas no Capítulo 4.

3.2.2 Ordem de uma curva elítica e teorema de Hasse

A ordem de uma curva elítica, denotada por $\#C(\mathbb{Z}_p)$, é o número de pontos distintos pertencentes a essa curva incluindo o ponto no infinito O. É muito importante conhecer a ordem de uma curva elítica em aplicações criptográficas, testes de primalidade e métodos de fatoração que utilizam essas curvas. Um importante resultado da teoria de curvas elíticas, conhecido como Teorema de Hasse [Bre89], fornece um intervalo contendo a ordem de uma curva.

TEOREMA 3.2.1 (TEOREMA DE HASSE) *Seja p um número primo e $C(\mathbb{Z}_p)$ uma curva elítica sobre $\mathbb{Z}_p$. Então, $p + 1 - 2\sqrt{p} \leq \#C(\mathbb{Z}_p) \leq p + 1 + 2\sqrt{p}$.*

Exemplo:

1. Considerando a curva $C : y^2 = x^3 + x + 1$ definida sobre $\mathbb{Z}_{23}$. Pelo Teorema de Hasse:

$$p + 1 - 2\sqrt{p} \leq \#C(\mathbb{Z}_p) \leq p + 1 + 2\sqrt{p} \Rightarrow$$

$$23 + 1 - 2\sqrt{23} \leq \#C(\mathbb{Z}_{23}) \leq 23 + 1 + 2\sqrt{23} \Rightarrow$$

$$15 \leq \#C(\mathbb{Z}_p) \leq 34.$$

Como visto no exemplo anterior, essa curva possui 28 pontos distintos, o que confirma o resultado obtido pelo teorema de Hasse: $15 \leq 28 \leq 34$.

3.3 Método das curvas elíticas

O método de fatoração de inteiros utilizando curvas elíticas, devido a H. W. Lenstra, baseia-se no método $p-1$ de Pollard [Pol74]. O método $p-1$ de Pollard tem como base o fato de que os elementos não nulos de $\mathbb{Z}_p$ formam um grupo multiplicativo $\mathbb{Z}_p^*$ de ordem $p-1$ e a seguinte proposição:

Proposição 3.3.1 Sejam n um número inteiro positivo ímpar composto e p um fator primo de n. Sejam a e k números inteiros tais que $\text{mdc}(a,p)=1$ e $p-1|k$. Então, $p|\text{mdc}(a^k-1,n)$.

Prova: Como $p-1|k$, tem-se que $k=(p-1)\cdot t$, para algum $t\in\mathbb{Z}$. Como p é primo e $\text{mdc}(a,p)=1$, segue pelo teorema de Fermat que $a^{p-1}\equiv 1 \bmod p$. Elevando a t ambos os termos da congruência e usando a relação entre k e t obtêm-se $a^k\equiv 1 \bmod p$, que é equivalente a $p|a^k-1$. Assim, p é fator comum de a^k-1 e n, donde segue que $p|\text{mdc}(a^{k-1},n)$.

Na prática, é escolhido um valor aleatório para k e espera-se que $p-1|k$. Também é escolhido um valor para a e calculado o $\text{mdc}(a^k-1,n)=d$. Uma vez calculado d tem-se três possibilidades:

1. $1<d<n$. Neste caso, foi encontrado um fator não trivial de n.
2. $d=1$. Este caso ocorre quando $p-1$ não divide k. Deve-se aumentar o valor de k e repetir o processo.
3. $d=n$. Deve-se tomar outro valor para a e tentar novamente.

A idéia de Lenstra é substituir o grupo multiplicativo $\mathbb{Z}_p^*$ utilizado no método $p-1$, por uma família dos grupos $C(\mathbb{Z}_p)$ de pontos de curvas elíticas sobre $\mathbb{Z}_p$, e substituir o inteiro a por um ponto $P\in C(\mathbb{Z}_p)$. Existe uma grande vantagem nesta substituição, pois no método $p-1$ o sucesso da fatoração depende apenas de um único número, $\#\mathbb{Z}_p^*=p-1$, enquanto no método das curvas elíticas pode-se contar com uma família dos grupos $C(\mathbb{Z}_p)$ dos pontos de curvas elíticas sobre $\mathbb{Z}_p$. Ou seja, pelo método $p-1$, se a tentativa de fatorar um número n usando o grupo $\mathbb{Z}_p^*$ não for bem sucedida, não existe outra alternativa a não ser desistir. No entanto, pelo

método das curvas elíticas, é possível fazer outras tentativas a partir da escolha de uma nova curva elítica e portanto de novos grupos. Isso, por sua vez, aumenta as chances de sucesso na fatoração.

A idéia do método das curvas elíticas é baseada no método $p-1$ de Pollard. Para encontrar um fator p de n, supõe-se um inteiro k tal que $\#C(\mathbb{Z}_p)|k$. Segue do Teorema de Lagrange [Gon79] que a ordem de qualquer ponto de $C(\mathbb{Z}_p)$ divide k. Em particular $kP = O$. Sendo $kP = (x : y : z)$ onde $(x : y : z)$ é uma terna de coordenadas normalizadas, então $z \equiv 0 \bmod p$, logo $p|z$. Assim p é fator comum de n e z, donde segue que $p|\text{mdc}(z, n)$.

Na prática, é escolhido um valor aleatório para k e espera-se que $\#C(\mathbb{Z}_p)|k$. Também são escolhidos um ponto P e uma curva elítica que contém P e tenta-se obter kP. Como não é possível realizar as operações módulo p, pois p ainda não é conhecido, as operações são realizadas módulo n e, considerando que n não é um número primo, tem-se que $\mathbb{Z}_n$ não é corpo, mas somente um anel. Para o cálculo de kP é necessário que os denominadores de λ dados por $x_2 - x_1$ e $2y_1$, apresentados na Seção 3.1, sejam invertíveis. No entanto, sabe-se que um inteiro c é invertível em $\mathbb{Z}_n$ se, e somente se, $\text{mdc}(c, n) = 1$. Portanto, ao tentar calcular kP tem-se duas possibilidades:

1. Consegue-se calcular kP. Nesse caso, os denominadores de λ utilizados no cálculo de kP são invertíveis em $\mathbb{Z}_n$, e $kP \neq O$. Logo, não obteve-se um fator não trivial de n. Deve-se então aumentar o valor de k ou tomar outra curva e tentar de novo.

2. Não é possível calcular kP. Nesse caso, durante os cálculos surgiu algum denominador c de λ não invertível em $\mathbb{Z}_n$, então $1 <\text{mdc}(c, n) \leq n$. Se $\text{mdc}(c, n) = d < n$ então d é fator não trivial de n, e portanto, conseguiu-se fatorar n. Se $\text{mdc}(c, n) = d = n$, d é fator trivial de n, portanto não foi possível fatorar n; logo, deve-se tomar outra curva e repetir o processo.

A seguir é apresentado o algoritmo original de Lenstra.

3.3.1 Algoritmo original de Lenstra

Seja $n \geq 2$ um inteiro composto para o qual deseja-se achar um fator primo.

1. Verifique que $\text{mdc}(n, 6) = 1$ e que n não tem a forma m^r para algum $r \geq 2$.
2. Escolha inteiros aleatórios a, x_1 e y_1 entre 1 e n.
3. Faça $b = y_1^2 - x_1^3 - ax_1 \bmod n$. (Seja C a curva $y^2 = x^3 + ax + b$ e $P = (x_1, y_1)$ um ponto de C).
4. Verifique que $\text{mdc}(4a^3 + 27b^2, n) = 1$. Se for igual a n, vá para o passo (2) e escolha novo a. Se estiver entre 1 e n, então ele é um fator não trivial de n; pare.
5. Escolha um número k que seja um produto de primos pequenos elevados a potências adequadas[1]. Por exemplo, considere $k = \text{mmc}(1, 2, 3, \ldots, B)$ para algum inteiro B.
6. Tente calcular $kP \bmod n$ pelo método das duplicações sucessivas. Se conseguir (é porque todos λ's têm denominadores invertíveis em $\mathbb{Z}_n$), vá para (5) e aumente o valor de k ou vá para (2) e tome outra curva. Caso contrário (é porque em alguma etapa do cálculo de kP o denominador c de λ é não invertível, i.é., $\text{mdc}(c, n) \neq 1$ vá para (7).
7. Se $\text{mdc}(c, n) < n$, foi encontrado um fator não trivial de n; pare. Se $\text{mdc}(c, n) = n$ vá para (2) e escolha outra curva.

Corretude do algoritmo

Suponha que p e q são os fatores de n tais que k é múltiplo de $\#C(\mathbb{Z}_p)$, ou seja $\#C(\mathbb{Z}_p)|k$ mas $\#C(\mathbb{Z}_q) \nmid k$. Então $kP = O$ em $C(\mathbb{Z}_p)$ e $kP \neq O$ em $C(\mathbb{Z}_q)$ [GB97]. Logo, no cálculo de kP existem $P_1 = (x_1, y_1)$ e $P_2 = (x_2, y_2)$ tais que:

[1] A expressão potências adequadas pode ser interpretada como potências pequenas. Em geral, dado um inteiro B, $k = \prod_{d_i^{a_i} \leq B} d_i^{a_i}$ onde os d_i's são números primos.

$$x_1 \equiv x_2 \mod p \text{ se } P_1 \neq P_2$$

ou

$$y_1 \equiv -y_2 \mod p \text{ se } P_1 = P_2$$

Ou seja,

$$x_1 - x_2 \equiv 0 \bmod p \Rightarrow \text{p}|\ x_1 - x_2 \Rightarrow \text{p}|\text{mdc}(x_1 - x_2, n)$$

ou

$$y_1 + y_2 \equiv 0 \bmod p \Rightarrow \text{p}|\ y_1 + y_2 \Rightarrow \text{p}|\text{mdc}(y_1 + y_2)$$

Assim, o $\text{mdc}(x_1 - x_2, n)$ ou o $\text{mdc}(y_1 + y_2, n)$ é um fator não trivial de n. Os pontos P_1 e P_2 serão encontrados quando $P_1 + P_2 = O$ em $C(\mathbb{Z}_p)$.

Análise assintótica

Como apresentado na Seção 3.3, o sucesso na fatoração pelo método das curvas elíticas depende de uma escolha aleatória de curvas elíticas sobre $\mathbb{Z}_p$. Ou seja, se a fatoração não for bem sucedida utilizando-se uma determinada curva, é possível fazer outras tentativas a partir de uma nova curva elítica aumentando as chances de sucesso na fatoração.

Com base nessa idéia, pode-se obter a complexidade heurística esperada para o método das curvas elíticas. Supõe-se que o número n a ser fatorado é composto, coprimo a 6 e não é potência perfeita. Considerando também que p é o menor fator de n e q o seu outro fator, pode-se afirmar que o tempo esperado para o método das curvas elíticas encontrar o menor fator p de n é de aproximadamente $O(exp(\sqrt{2 \ln p \ln \ln p}))$ operações de grupo [CP02]. Portanto, o método das curvas elíticas apresenta um tempo de execução subexponencial. O pior caso do algoritmo ocorre quando os dois fatores de n são aproximadamente iguais, ou seja, quando $p \approx \sqrt{n}$.

3.4 Otimizando o método das curvas elíticas

O método das curvas elíticas pode ser otimizado, na prática, acrescentando-se uma segunda fase em cada tentativa de fatoração do algoritmo [CP02]. Nessa abordagem, assume-se que cada tentativa de fatoração é subdividida em duas fases apresentadas a seguir:

- **1ª fase:** Esta fase consiste em escolher o ponto P, a curva C e o parâmetro k, conforme apresentado na Seção 3.3.1 e calcular o ponto $Q = kP$. Considerando que o parâmetro k é dado por um produto de primos pequenos elevados a potências adequadas, a fatoração será obtida nesta fase se ocorrer $kP = O$, para k da forma:

$$k = \prod_{d_i^{a_i} \leq B_1} d_i^{a_i}$$

  onde os d_i's são números primos. Portanto, espera-se que a ordem da curva seja formada por fatores primos todos menores ou iguais a B_1. Se essa condição é satisfeita, então a fatoração é realizada. Se a ordem da curva possuir algum fator maior que B_1 então a fatoração não é realizada e, nesse caso, aplica-se a segunda fase do algoritmo.

- **2ª fase:** Nesta fase, assume-se que a ordem da curva é dada por

$$\#C(\mathbb{Z}_p) = r \prod_{d_i^{a_i} \leq B_1} d_i^{a_i}$$

  onde r é um número primo maior que B_1. Nesse caso, utiliza-se um limite B_2, onde B_2 é um número inteiro com $B_1 < B_2$, e verifica-se para cada primo r tal que $B_1 < r \leq B_2$ se $rQ = O$. Se a fatoração não for obtida durante a segunda fase, deve-se repetir o procedimento em uma nova tentativa para outras curvas elíticas e outros pontos iniciais [FKP+05].

  A adição dessa segunda fase ao algoritmo original de Lenstra, apresenta, na prática uma melhora no tempo de execução. No entanto, o tempo de execução assintótico do algoritmo permanece o mesmo [Bre90].

Capítulo 4

Algoritmos para implementação do método das curvas elíticas

A implementação do método das curvas elíticas não é uma tarefa trivial, pois vários aspectos devem ser estudados antes de sua realização. Dentre esses aspectos, devem ser analisados a escolha de um sistema de computação algébrica para cálculos com inteiros muito grandes, o sistema de coordenadas a ser empregado na implementação, os tipos de curvas adequados e, ainda, os algoritmos a serem utilizados.

Este capítulo apresenta algumas alternativas para a implementação do método das curvas elíticas e os algoritmos fundamentais aplicados nesse processo. Procurou-se fazer a descrição dos algoritmos em um pseudocódigo legível por qualquer leitor com um pouco de conhecimento em programação, sem sacrificar a profundidade do enfoque ou o rigor matemático.

4.1 Cálculo do inverso multiplicativo

Uma implementação do método das curvas elíticas exige a utilização de rotinas que executem operações modulares. Dessa forma é necessário o uso de algoritmos eficientes para computar operações de adição, subtração, multiplicação e inverso multiplicativo. Em aplicações de propósito geral é conveniente considerar que essas operações gastam um tempo constante para serem realizadas. No entanto, sabe-se que essas operações levam um tempo maior para serem efetuadas quando trabalha-se com números grandes. Nesse caso, faz-se

necessária uma análise mais criteriosa tornado-se conveniente medir quantas operações de bits um algoritmo exige [CLRS01].

A operação mais custosa utilizada no método das curvas elíticas é a de inverso multiplicativo. Sabe-se que o inverso de um número $a \in \mathbb{Z}_p$ com $a \neq 0$, denotado por $a^{-1} \mod p$ ou simplesmente a^{-1}, é o elemento $x \in \mathbb{Z}_p$ tal que $ax = 1$ em $\mathbb{Z}_p$, isto é, $ax \equiv 1 \mod p$.

Inversos podem ser eficientemente calculados utilizando-se o algoritmo euclidiano estendido para inteiros [HMV04]. Este algoritmo é uma extensão do algoritmo euclidiano, ou algoritmo de Euclides que calcula o máximo divisor comum de dois inteiros.

4.1.1 Algoritmo euclidiano

Sejam $a, b \in \mathbb{Z}_p$ com $a, b \neq 0$. O máximo divisor comum de a e b, denotado por $\text{mdc}(a, b)$ é o maior inteiro c que divide ambos a e b. Algoritmos eficientes para calcular o $\text{mdc}(a, b)$ são baseados no teorema 4.1.1.

TEOREMA 4.1.1 *Sejam a e b inteiros positivos tal que $a = bq + r$, onde b e q são inteiros. Então o* $\text{mdc}(a, b) = \text{mdc}(b, r)$.

O Algoritmo 4.1.1 calcula o mdc de inteiros positivos a e b onde $a \geq b$, com base na idéia apresentada no Teorema 4.1.1. Esse cálculo é realizado através dos seguintes passos:

1. divide-se a por b encontrando o resto r_1;
2. se $r_1 \neq 0$, divide-se b por r_1 encontrando o resto r_2.
3. se $r_2 \neq 0$, divide-se r_1 por r_2 encontrando o resto r_3. E assim por diante. Esse processo é repetido até que se encontre o último resto igual a zero, e então o $\text{mdc}(a, b)$ será o último resto diferente de zero.

ALGORITMO 4.1.1 *Euclidiano*
Entrada: inteiros positivos a e b onde $a \geq b$
Saída: $\text{mdc}(a, b)$
1. $x \leftarrow a$

2. $y \leftarrow b$
3. `enquanto` $(r \neq 0)$ `faça`
4. $\quad r \leftarrow x \bmod y$
5. $\quad x \leftarrow y$
6. $\quad y \leftarrow r$
7. `retorne` x

Exemplo:
Sejam $a = 858$ e $b = 253$. Utilizando o algoritmo euclidiano, o mdc entre entre a e b é obtido fazendo-se:

$$\begin{aligned} 858 &= 3 \cdot 253 + 99 \\ 253 &= 2 \cdot 99 + 55 \\ 99 &= 1 \cdot 55 + 44 \\ 55 &= 1 \cdot 44 + 11 \\ 44 &= 4 \cdot 11 \end{aligned}$$

Portanto o $\text{mdc}(858, 253) = 11$.

O Algoritmo 4.1.1 pode ser estendido para encontrar inteiros x e y tais que $ax + by = d$ onde $d = \text{mdc}(a, b)$. Esse cálculo pode ser realizado através do Algoritmo 4.1.2.

ALGORITMO 4.1.2 *Euclidiano estendido*
Entrada: inteiros positivos a e b, com $a \geq b$.
Saída: $d = \text{mdc}(a, b)$ e inteiros x e y satisfazendo $ax + by = d$.

1. $u \leftarrow a$
2. $v \leftarrow b$
3. $x_1 \leftarrow 1$
4. $y_1 \leftarrow 0$
5. $x_2 \leftarrow 0$
6. $y_2 \leftarrow 1$
7. `enquanto` $(v \neq 0)$ `faça`
8. $\quad q \leftarrow \lfloor u/v \rfloor$
9. $\quad r \leftarrow u - qv$
10. $\quad x \leftarrow x_1 - qx_2$

11. $y \leftarrow y_1 - qy_2$
12. $u \leftarrow v$
13. $v \leftarrow r$
14. $x_1 \leftarrow x_2$
15. $x_2 \leftarrow x$
16. $y_1 \leftarrow y_2$
17. $y_2 \leftarrow y$
18. $d \leftarrow v$
19. $x \leftarrow x_1$
20. $y \leftarrow y_1$
21. `retorne` (d, x, y)

Exemplo:
Sejam $a = 99$ e $b = 78$. Aplicando o Algoritmo 4.1.2, obtém-se:

	u	v	q	x	y	r	d
1ª iteração	99	78	1	1	-1	21	3
2ª iteração	78	21	3	-3	4	15	
3ª iteração	21	15	1	4	-5	6	
4ª iteração	15	6	2	-11	14	3	
5ª iteração	6	3	2	26	-33	0	
6ª iteração	3	0		-11	14		

Logo, $d = 3$, $x = -11$ e $y = 14$ satisfazendo $ax + by = d$.

Supondo p e a inteiros, tais que p é primo e $a \in [1, p-1]$, então $\text{mdc}(a, p) = 1$. Se o algoritmo euclidiano estendido for executado com as entradas (a, p), o último resto diferente de zero encontrado será igual a 1. Consequentemente, u, x_1 e y_1, atualizados nas linhas 12, 14 e 16 do algoritmo, satisfazem $ax_1 + py_1 = u$ com $u = 1$. Portanto, $ax_1 \equiv 1 \bmod p$, isto é $a^{-1} = x_1 \bmod p$. Dessa foram, obtém-se o Algoritmo 4.1.3 que calcula o inverso de números inteiros.

ALGORITMO 4.1.3 *Calcula o inverso*
Entrada: primo p e $a \in [1, p-1]$
Saída: $a^{-1} \bmod p$ e inteiros x e y satisfazendo $ax + by = d$.

1. $u \leftarrow a$
2. $v \leftarrow p$
3. $x_1 \leftarrow 1$
4. $x_2 \leftarrow 0$
5. `enquanto` $(u \neq 1)$ `faça`
6. $\quad q \leftarrow \lfloor v/u \rfloor$
7. $\quad r \leftarrow v - qu$
8. $\quad x \leftarrow x_2 - qx_1$
9. $\quad v \leftarrow u$
10. $\quad u \leftarrow r$
11. $\quad x_2 \leftarrow x_1$
12. $\quad x_1 \leftarrow x$
13. `retorne` $x_1 \bmod p$

Exemplo

Sejam $a = 8$ e $p = 11$. Aplicando-se o Algoritmo 4.1.3, obtém-se a seguinte sequência de resultados:

	u	v	x_1	x_2	q	r	x
1ª iteração	8	11	1	0	1	3	-1
2ª iteração	3	8	-1	1	2	2	3
3ª iteração	2	3	3	-1	1	1	-4
4ª iteração	1	2	-4	3			

Portanto $x_1 \bmod p = -4 \bmod 11 = 7$. Logo $a^{-1} \equiv 7 \bmod 11$, isto é, o inverso de 8 é igual 7 em $\mathbb{Z}_{11}$.

Os tempos de execução dos algorimos 4.1.1, 4.1.2 e, consequentemente, do Algoritmo 4.1.3 são iguais. Se aplicados a dois números de β bits , serão executadas $O(\beta)$ operações aritméticas e $O(\beta^2)$ operações de bits [CLRS01]. O inconveniente desses algoritmos é a exigência de operações de divisões na linha 6 do algoritmo, que são consideradas computacionalmente caras. Uma maneira de melhorar o custo da operação de inversão é utilizar a versão binária do Algoritmo 4.1.1 para realizar essa operação.

4.1.2 Algoritmo binário de inversão

Na década 1960, R. Silver and J. Terzian [Knu81] observaram que o algoritmo utilizado para calcular o mdc poderia ser executado de forma binária com base na idéia do Teorema 4.1.2 apresentado a seguir.

TEOREMA 4.1.2 *Para inteiros x e y:*

a) Se x e y são ambos pares, então $\text{mdc}(x, y) = 2\ \text{mdc}(x/2, y/2)$.

b) Se x é par e y é ímpar então o $\text{mdc}(x, y) = \text{mdc}(x/2, y)$.

c) $\text{mdc}(x, y) = \text{mdc}(x - y, y)$.

d) Se x e y são ambos ímpares, então $|x - y|$ é par.

O Algoritmo 4.1.4 calcula o mdc de forma binária. Antes da linha 9, u ou v é ímpar, logo a divisão por 2 nas linhas 10 ou 12 não altera o valor do mdc. Depois de executar as linhas 10 e 12, ambos u e v são ímpares e, portanto, um dos dois será par no final da linha 16. Assim, cada iteração no laço da linha 8, reduz o número de bits de u ou v em no mínimo 1 bit. Segue daí que o número total de iterações deste laço é no máximo $2k$ onde k é o número máximo de bits de a e b.

ALGORITMO 4.1.4 *Calcula o* mdc *de forma binária*
Entrada: inteiros positivos a e b
Saída: $\text{mdc}(a, b)$

1. $u \leftarrow a$
2. $v \leftarrow b$
3. $e \leftarrow 1$
4. `enquanto` u é par `e` v é par `faça`
5. $\quad u \leftarrow u/2$
6. $\quad v \leftarrow v/2$
7. $\quad e \leftarrow 2e$
8. `enquanto` $(u \neq 0)$ `faça`
9. $\quad$ `enquanto` u é par `faça`
10. $\quad\quad u \leftarrow u/2$

11. `enquanto` v é par `faça`
12. $v \leftarrow v/2$
13. `se` $u \geq v$ `então`
14. $u \leftarrow u - v$
15. `senão`
16. $v \leftarrow v - u$
17. `retorne` $e \cdot v$

O Algoritmo 4.1.5, apresentado a seguir, calcula o inverso de a em $\mathbb{Z}_p$ de forma binária utilizando a idéia apresentada no Algoritmo 4.1.4.

ALGORITMO 4.1.5 *Calcula o inverso binário*
Entrada: primo p e $a \in [1, p-1]$
Saída: $a^{-1} \bmod p$

1. $u \leftarrow a$
2. $v \leftarrow p$
3. $x_1 \leftarrow 1$
4. $x_2 \leftarrow 0$
5. `enquanto` $(u \neq 1)$ `e` $(v \neq 1)$ `faça`
6. `enquanto` u é par `faça`
7. $u \leftarrow u/2$
8. `se` x_1 é par `então`
9. $x_1 \leftarrow x_1/2$
10. `senão`
11. $x_1 \leftarrow (x_1 + p)/2$
12. `enquanto` v é par `faça`
13. $v \leftarrow v/2$
14. `se` x_2 é par `então`
15. $x_2 \leftarrow x_2/2$
16. `senão`
17. $x_2 \leftarrow (x_2 + p)/2$
18. `se` $u \geq v$ `então`
19. $u \leftarrow u - v$
20. $x_1 \leftarrow x_1 - x_2$
21. `senão`

22. $v \leftarrow v - u$
23. $x_2 \leftarrow x_2 - x_1$
24. `se` $u = 1$ `então`
25. `retorne` x_1 mod p
26. `senão`
27. `retorne` x_2 mod p

Na prática, a versão binária do algoritmo de Euclides é frequentemente mais rápida do que a versão original, embora a complexidade seja a mesma [CP02]. Outras melhorias propostas para o método de Euclides podem ser encontradas em [Web95].

4.2 Algoritmos para testar a entrada n

Antes de iniciar o processo de fatoração pelo método das curvas elíticas, é interessante que sejam realizados alguns testes a respeito do número n que se deseja fatorar, tais como:

1. Verificar se n é composto.
2. Verificar se n é uma potência perfeita.
3. Verificar se $\text{mdc}(n, 6) = 1$. Em caso contrário, n é um número trivialmente divisível por 2 ou 3.

4.2.1 Verificando se n é composto

Para execução desse passo, os procedimentos mais utilizados são o teste de pseudoprimalidade e o teste de Miller-Rabin. Ambos os testes, utilizam um algoritmo que calcula a operação de exponenciação modular $a^b \text{mod } n$. Esse cálculo pode ser executado de forma eficiente utilizando a representação binária de b, ou seja, $\langle b_k, b_{k-1}, \ldots, b_1, b_0 \rangle$ com $k+1$ bits de comprimento, onde b_k é o bit mais significativo e b_0 o menos significativo. O algoritmo 4.2.1 calcula a^c mod n, à medida que c é aumentado por duplicações e incrementos desde 0 até b.

ALGORITMO 4.2.1 *Exponenciação modular*
Entrada: a, b, n
Saída: $a^b \bmod n$
1. $c \leftarrow 0$
2. $d \leftarrow 1$
3. seja $\langle b_k, b_{k-1}, \ldots, b_1, b_0 \rangle$ a representação binária de b
4. `para` $i \leftarrow k$ `até` 0 `faça`
5. $\quad c \leftarrow 2c$
6. $\quad d \leftarrow (d \cdot d) \bmod n$
7. $\quad$ `se` $b_i = 1$ `então`
8. $\quad\quad c \leftarrow c + 1$
9. $\quad\quad d \leftarrow (d \cdot a) \bmod n$
10. `retorne` d

Exemplo:
Suponha que se deseja calcular $3^{27} \bmod 1215$. Nesse caso $a = 3$, $b = 27$ e $n = 1215$. Representando b na base 2 tem-se que $27_{10} = 11011_2$ e utilizando o algoritmo de exponenciação modular acima obtêm-se a seguinte sequência de resultados:

	i	b_i	c	d
1ª iteração	4	1	1	3
2ª iteração	3	1	3	27
3ª iteração	2	0	6	729
4ª iteração	1	1	13	243
5ª iteração	0	1	27	0

Portanto $3^{27} \bmod 1215 = 0$.

Esse algoritmo exige um total de $O(\beta)$ operações aritméticas e $O(\beta^3)$ operações de bits, considerando as entradas a, b e n como números de β bits [CLRS01].

O teste de pseudoprimalidade, apresentado no Algoritmo 4.2.2, tem como base o seguinte teorema:

TEOREMA 4.2.1 (PEQUENO TEOREMA DE FERMAT) *Sejam p um número primo e a um número inteiro. Se* $\mathrm{mdc}(a, p) = 1$, *então* $a^{p-1} \equiv 1 \bmod p$.

Ou seja, para testar se n é composto, basta fazer $a = 2$, por exemplo, e calcular $2^{n-1} \bmod n$, como no procedimento a seguir.

ALGORITMO 4.2.2 *Teste de pseudoprimalidade*
Entrada: n
Saída: "Composto" ou "Provavelmente Primo"
1. `se` *Exponenciação modular*$(2, n-1, n) \neq 1 \bmod n$ `então`
2. `retorne` "Composto"
3. `senão`
4. `retorne` "Provavelmente Primo"

Esse teste pode produzir erros ao retornar que n é primo, mas estará sempre correto se retornar que n é composto. Isso acontece pois existem números compostos para os quais o algoritmo afirma serem provavelmente primos. Esses números são denominados pseudoprimos de base 2. Não existe uma maneira de eliminar todos os erros apenas utilizando outros valores de a no cálculo de $a^{n-1} \bmod n$, pois existem números compostos que satisfazem a equação $a^{n-1} \equiv 1 \bmod n$ para todo $a \in \mathbb{Z}_n^*$. Esses números são denominados números de Carmichael.

Uma maneira de melhorar esse teste de forma que ele não seja enganado pelos números de Carmichael é utilizar o teste aleatório do caráter primo de Miller-Rabin, que supera os problemas do teste de pseudoprimalidade com duas alterações:

- vários valores de a são gerados aleatoriamente e testados;
- durante o cálculo da exponenciação modular, verifica se existe uma raiz quadrada não trivial de 1 módulo n, ou seja, testa se um número x satisfaz a equação $x^2 \equiv 1 \bmod n$, mas não é equivalente a nenhuma das duas raízes "triviais": 1 ou -1, módulo n.

O teste utilizado para verificar se existe uma raiz quadrada não trivial de 1 módulo n tem como base o teorema apresentado a seguir.

TEOREMA 4.2.2 *Sejam p um número primo e $x \geq 1$ tal que $x^2 \equiv 1 \bmod p$. Então $x \equiv 1 \bmod p$ ou $x \equiv p - 1 \bmod p$.*

Demonstração: Se $x^2 \equiv 1 \bmod p \Rightarrow x^2 - 1 \equiv 0 \bmod p \Rightarrow (x-1)(x+1) \equiv 0 \bmod p \Rightarrow (x-1) \equiv 0 \bmod p$ ou $(x+1) \equiv 0 \bmod p \Rightarrow x \equiv 1 \bmod p$ ou $x \equiv p-1 \bmod p$, o que completa a demonstração.

Portanto, se $x^2 \equiv 1 \bmod p$ para $x \neq 1$ e $x \neq -1$, foi encontrada uma raiz não trivial de 1 módulo n. Logo, n é composto.

O algoritmo para o teste de Miller-Rabin utiliza dois procedimentos auxiliares para sua execução:

- o gerador de números aleatórios `Random` que retorna um inteiro $1 \leq a \leq n-1$ aleatório;
- o procedimento `Testemunha` dado no algoritmo 4.2.3 que retorna "Verdadeiro" se a é uma testemunha do caráter múltiplo de n.

Na prática, a maioria dos ambientes de programação oferece um gerador de números pseudo-aleatórios, ou seja, um algoritmo determinístico que retorna números que "parecem" ser estatisticamente aleatórios. No entanto, para implementar o algoritmo 4.2.3, assume-se que $n-1 = 2^t u$, onde $t \geq 1$ e u é ímpar. Logo $a^{n-1} \bmod n$ pode ser escrito como $a^{n-1} \equiv ((a^u)^{2^t} \bmod n$, o que pode ser calculado primeiramente calculando-se $a^u \bmod n$ e elevando-se o resultado ao quadrado t vezes sucessivamente.

ALGORITMO 4.2.3 *Testemunha*
Entrada: uma base a e um número ímpar n, cujo caráter primo será testado
Saída: "Verdadeiro" ou "Falso"

1. seja $n-1 = 2^t u$, onde $t \geq 1$ e u é ímpar
2. `se` $x_0 \leftarrow$ *Exponenciação modular*(a, u, n)
3. `para` $i \leftarrow 1$ `até` t `faça`
4. $\quad x_i \leftarrow x_{i-1}^2$
5. $\quad$ `se` $x_i = 1$ `e` $x_{i-1} \neq 1$ `e` $x_{i-1} \neq n-1$ `então`
6. $\quad\quad$ `retorne` "Verdadeiro"
7. `se` $x_t \neq 1$ `então`
8. $\quad$ `retorne` "Verdadeiro"
9. `retorne` "Falso"

Com base neste algoritmo, a rotina do teste de Miller-Rabin é dada a seguir, supondo que n é um inteiro ímpar maior que 2.

ALGORITMO 4.2.4 *Miller-Rabin*
Entrada: um número ímpar n cujo caráter primo será testado e a quantidade s de valores aleatórios para a.
Saída: "Composto" ou "Provavelmente Primo"

```
1. para j ← 1 até s faça
2.     a ← Random(1, n − 1)
3.     se Testemunha(a, n) então
4.         retorne "Composto"
5. retorne "Provavelmente Primo"
```

Exemplo:
Para ilustrar o funcionamento do algoritmo 4.2.4, seja o número de Carmichael $n = 278545$. O teste de pseudoprimalidade apresentado no algoritmo 4.2.2 retorna "provavelmente primo" para a entrada n, pois $2^{278544} \bmod 278545 = 1$. No entanto, se for aplicado o teste de Miller-Rabin tem-se que $n - 1 = 278544 = 2^4 \cdot 17409$ e supondo que $a = 3$ o procedimento `Testemunha` calcula $x_0 \equiv 2^{17409} \equiv 149773$ e produz a seguinte sequência de resultados:

$x_1 \equiv 165589$
$x_2 \equiv 25666$
$x_3 \equiv 263176$
$x_4 \equiv 1$

Então $a = 3$ é uma testemunha do caráter múltiplo de n. O procedimento `Testemunha` retorna "verdadeiro" e o algoritmo 4.2.4 retorna "composto".

Para qualquer inteiro ímpar $n > 2$ e inteiro positivo s, a probabilidade do teste de Miller-Rabin errar é no máximo 2^{-s} e sua complexidade é de $O(s\beta)$ operações aritméticas e $O(s\beta^3)$ operações de bits.

4.2.2 Verificando se n é potência perfeita

Dado um inteiro $n \geq 2$, pretende-se determinar se n é uma potência perfeita, isto é, se $n = d^e$ para inteiros d e e, ambos maiores do que 1. Esse problema pode ser facilmente resolvido considerando que se d e e existirem, então $2^e \leq n$, logo devem ser testados todos os possíveis valores candidatos de e desde 2 até $\lfloor \log_2 n \rfloor$. Supondo n um número de k-bits, isto é, $2^{k-1} \leq n \leq 2^k$, então $2^{(k-1)/e} \leq n^{1/e} \leq 2^{k/e}$. Assim toda raiz e-ésima do inteiro n deve estar no conjunto $\{u, \ldots, v-1\}$, onde $u = 2^{\lfloor (k-1)/e \rfloor}$ e $v = 2^{\lceil k/e \rceil}$. Usando u e v como valores iniciais, pode-se testar se n é potência perfeita, através do Algoritmo 4.2.5 cuja complexidade é de $O(k^3/e)$ [Sho05a].

ALGORITMO 4.2.5 *Potência perfeita*
Entrada: um número ímpar n
Saída: "n é potência perfeita" ou "n não é potência perfeita"

1. `enquanto` $u < v$ `faça`
2. $w \leftarrow \lfloor (u+v)/2 \rfloor$
3. $z \leftarrow w^e$
4. `se` $z = n$ `então`
5. `retorne` "n é potência perfeita"
6. `senão`
7. `se` $z < n$ `então`
8. $u \leftarrow w + 1$
9. `senão` $v \leftarrow w$ `retorne` "n não é potência perfeita"

4.2.3 Verificando se mdc(n, 6)=1

Para realizar este teste, basta utilizar o Algoritmo 4.1.1 que que calcula o máximo divisor comum entre dois números inteiros. Se algoritmo retornar 1, n é co-primo a 6 e a fatoração pode ser iniciada. Caso contrário, n é trivialmente divisível por 2 ou 3.

4.3 Obtenção da curva elítica

O desempenho do método de fatoração das curvas elíticas depende também, da escolha de curvas adequadas para sua implementação. Alguns autores, como Atkin e Morain em [AM92], apresentam um estudo nesse contexto descrevendo uma família infinita de curvas definidas sobre $\mathbb{Q}$, bem como a construção dessas curvas. Outros estudos afirmam que o método ECM pode ser otimizado através do uso de curvas de ordem divisível por 12 ou 16 [Bre00].

Outro aspecto que deve, também, ser considerado no processo de implementação do método das curvas elíticas, refere-se ao sistema de coordenadas a ser utilizado [CP02]. A implementação deste método pode ser realizada utilizando-se coordenadas afim ou coordenadas projetivas. Em cada uma dessas abordagens tem-se:

- **Coordenadas afins:** Essa abordagem, utiliza de forma direta as operações de adição e duplicação de pontos apresentadas no Capítulo 3, envolvendo uma operação de inversão para cada adição e duplicação de ponto da curva.
- **Coordenadas projetivas:** Nessa abordagem, as operações de grupo são realizadas utilizando-se coordenadas projetivas $(x : y : z)$. Quando $z \neq 0$, $(x : y : z)$ corresponde ao ponto afim $(x/z, y/z)$ na curva. O ponto $(0 : 1 : 0)$ é o ponto no infinito O. Nesse caso, não são realizadas operações de inversão. Maiores detalhes sobre coordenadas projetivas podem ser encontrados em [CP02].

Qual dessas abordagens é melhor depende de vários aspectos. A operação do inverso multiplicativo é bastante cara, pois é baseada no algoritmo de Euclides, porém o uso de coordenadas afins pode ser adequado considerando a utilização de um método eficiente para execução dessa operação. A representação por coordenadas projetivas dispensa o cálculo do inverso multiplicativo, entretanto demanda um maior número de multiplicações. Segundo Menezes [Men93], o uso de coordenadas projetivas proporciona ganhos em velocidade, por dispensar a custosa operação de inverso multiplicativo, mas perde-se em memória de programa e de dados pois necessita-se de mais operações e mais variáveis temporárias para se implementar a soma e a duplicação de pontos.

Para obtenção de uma curva elítica aleatória $y^2 = x^3 + ax + b$ e de um ponto (x, y) pertencente a essa curva, utilizando coordenadas afins, a maneira mais prática é escolher aleatoriamente $x, y, a \in [0, n-1]$ através de um gerador de números aleatórios oferecido pelo ambiente de programação e calcular $b = (y^2 - x^3 - ax) \bmod n$.

Conforme apresentado no Capítulo 3, uma curva elítica $y^2 = x^3 + ax + b$ é bem definida se $4a^3 + 27b^2 \neq 0$. Como no método das curvas elíticas são utilizadas curvas sobre $\mathbb{Z}_n$, para saber se a curva obtida é adequada basta verificar se $\text{mdc}(4a^3 + 27b^2, n) = 1$ utilizando o algoritmo Euclidiano. Se o resultado for 1, então a curva é utilizada no processo de fatoração. Se o resultado for igual a n, deve-se escolher outra curva. Porém, se o resultado estiver entre 1 e n, encontrou-se um fator de n encerrando a execução do algoritmo.

4.4 Escolhendo *k*

Como apresentado no Capítulo 3, o parâmetro k deve ser determinado como um produto de primos pequenos elevados a potências adequadas. Ou seja, k deve ser dado por

$$k = \prod_{d_i^{a_i} \leq B} d_i^{a_i}.$$

onde os d_i's são números primos.

É evidente que para a obtenção de k deve-se estabelecer em primeiro lugar, o limite B. Existem diferentes abordagens para a resolução desse problema, no entanto, essa escolha consiste mais em arte do que em ciência pois, em geral, o valor ótimo de B é obtido de forma empírica, baseada em experimentação [CP02].

Alguns autores, como em [BL95, SJ93] propõem parâmetros ótimos para a escolha de B, com base no tamanho do fator que se queira encontrar. Em [SJ93] os parâmetros são obtidos através de uma análise estatística aplicada a algoritmos aleatórios. Nesse estudo, os autores sugerem valores para os limites B_1 e B_2 a serem utilizados. O limite B_1 é usado na primeira fase do algoritmo e o limite B_2 na segunda fase.

A Tabela 4.1 apresenta os parâmetros B_1 e B_2 sugeridos em [SJ93], de acordo com o número de dígitos decimais de p. Em geral, esses valores são aproximados para o número primo mais próximo. Por exemplo, supondo que se queira encontrar um fator de 5 dígitos decimais e aproximando-se os valores de $B_1 = 16$ e $B_2 = 800$, tem-se $B_1 = 17$ e $B_2 = 797$.

Depois de estabelecido o limite inicial B, a obtenção de k como o produto de primos menores que B elevados a potências adequadas, pode ser realizada através do cálculo do mínimo múltiplo comum entre todos os números menores ou iguais a B, ou seja calculando-se:

$$k = \text{mmc}(1, 2, 3, ..., B)$$

O parâmetro k também pode ser obtido efetuando-se:

$$k = 2^{\lfloor \ln B / \ln 2 \rfloor} \cdot 3^{\lfloor \ln B / \ln 3 \rfloor} \cdot ... \cdot B$$

Tabela 4.1: Parâmetros Ótimos

número de dígitos de p	parâmetro B_1	parâmetro B_2
5	16	800
7	53	2650
9	156	7176
11	405	19440
13	962	42328
15	2240	103017
17	4778	215010
19	9004	405180
21	18437	792791
23	34155	1.40e+6
25	66596	2.66e+6
27	133297	5.33e+6
29	247988	9.99e+6
31	374990	1.54e+7
33	649996	2.66e+7
35	1170924	4.80e+7
37	1967442	8.13e+7
39	3276490	1.31e+8
41	5249667	2.01e+8

Exemplo:
Seja $n = 1469558737$ um número composto de fatores com 5 dígitos decimais. De acordo com a Tabela 4.1, o valor de B aproximado para o primo mais próximo é 17. Efetuando-se $k = \text{mmc}(1, 2, 3, ..., B)$ obtém-se $k = 2^4{\cdot}3^2{\cdot}5{\cdot}7{\cdot}11{\cdot}13{\cdot}17$. Se k for calculado utilizando-se a expressão $k = 2^{\lfloor \ln B/\ln 2 \rfloor} \cdot 3^{\lfloor \ln B/\ln 3 \rfloor} \cdot ... \cdot B$, também obtém-se $k = 2^4 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17$.

4.5 Calculando *kP*

Nessa etapa do algoritmo pretende-se "tentar" realizar uma das operações mais importantes do método das curvas elíticas, que é a operação de multiplicação escalar

$$kP = \underbrace{P + P + \cdots + P}_{k \text{ vezes}} \tag{4.1}$$

O cálculo de kP constitui uma das etapas mais importantes no método das curvas elíticas. Uma das maneiras de realizar este cálculo é utilizar o método das duplicações sucessivas que é apresentado a seguir.

4.5.1 Método das duplicações sucessivas

O método das duplicações sucessivas utiliza a representação de k em termos de sua expansão binária para o cálculo de kP. A obtenção de kP através deste método pode ser realizada através dos seguintes passos:

1. Seja $k = k_0 + 2k_1 + 2^2k_2 + ... + 2^rk_r$, onde $k_i = 0$ ou $k_i = 1$, $1 \leq i \leq r$ e $r \leq \lg k$, a representação de k em termos de sua expansão binária;

2. Calcule os pontos 2^iP com $1 \leq i \leq r$, fazendo

$$P_0 = P$$
$$P_1 = 2P_0 = 2P$$
$$P_2 = 2P_1 = 2^2P$$

$$P_3 = 2P_2 = 2^3P$$
$$\vdots$$
$$P_r = 2P_{r-1} = 2^rP;$$

3. Adicione os pontos P_i's para os quais $k_i = 1$.

Dessa forma calcula-se kP com um número de passos menor do que $2\lg k$, onde cada passo é uma adição ou uma duplicação de pontos.

Exemplo:
Seja $P = (1, 1)$ um ponto sobre a curva elítica $y^2 = x^3 + x - 1$ em $\mathbb{Z}_n$ para $n = 3457380169$ e seja $k = 27720$. Para calcular kP pelo método das duplicações sucessivas em primeiro lugar obtêm-se a expansão binária de k, ou seja

$$k = 2^3 + 2^6 + 2^{10} + 2^{11} + 2^{13} + 2^{14}$$

A partir dessa representação de k, são calculados os pontos 2^iP, para $1 \leq i \leq 14$, conforme mostra a Tabela 4.2.

Tabela 4.2: Duplicação de Pontos

i	$2^iP \mod 5$
0	$\mathrm{P}_0 = 2^0P = (1, 1)$
1	$\mathrm{P}_1 = 2^1P = 2P_0 = (2, 3457380166)$
2	$\mathrm{P}_2 = 2^2P = 2P_1 = (1056421719, 1712683695)$
3	$\mathrm{P}_3 = 2^3P = 2P_2 = (1012278507, 377993791)$
4	$\mathrm{P}_4 = 2^4P = 2P_3 = (1692970624, 993407508)$
5	$\mathrm{P}_5 = 2^5P = 2P_4 = (983801888, 753048190)$
6	$\mathrm{P}_6 = 2^6P = 2P_5 = (2055169165, 2026559068)$
7	$\mathrm{P}_7 = 2^7P = 2P_6 = (769327054, 2082141828)$
8	$\mathrm{P}_8 = 2^8P = 2P_7 = (194117548, 3423237911)$
9	$\mathrm{P}_9 = 2^9P = 2P_8 = (3429217964, 2293865171)$
10	$\mathrm{P}_{10} = 2^{10}P = 2P_9 = (3208768166, 905312802)$
11	$\mathrm{P}_{11} = 2^{11}P = 2P_{10} = (1927141887, 3147511082)$
12	$\mathrm{P}_{12} = 2^{12}P = 2P_{11} = (65815309, 715879671)$
13	$\mathrm{P}_{13} = 2^{13}P = 2P_{12} = (1385673311, 3369844450)$
14	$\mathrm{P}_{14} = 2^{14}P = 2P_{13} = (799596392, 333488497)$

Em seguida, são calculadas as somas parciais dos valores $2^iP \mod 5$, para os quais $k_i = 1$. Ou seja,

$((((2^3P + 2^6P) + 2^{10}P) + 2^{11}P) + 2^{13}P) + 2^{14}P \Rightarrow$
$((((893867983, 2672742174) + 2^{10}P) + 2^{11}P) + 2^{13}P) + 2^{14}P \Rightarrow$
$(((847823675, 1510523570) + 2^{11}P) + 2^{13}P) + 2^{14}P \Rightarrow$
$((35388288, 383148084) + 2^{13}P) + 2^{14}P \Rightarrow$
$(2667997643, 3103205478) + 2^{14}P \Rightarrow$
$(2130795921, 439182547) = kP$

4.6 Executando a segunda fase do algoritmo

Como visto na Seção 3.4, a segunda fase do algoritmo consiste em verificar para cada primo r tal que $B_1 < r \leq B_2$, se $rQ = O$. Essa verificação pode ser realizada com base em alguns métodos sugeridos em [Bre86, CP02, Mon87].

Uma das maneiras de realizar essa verificação é calcular, a partir do ponto $Q = kP$ obtido na primeira fase, os pontos

$$[r_0]Q, [r_0 + \Delta_0]Q, [r_0 + \Delta_0 + \Delta_1]Q, [r_0 + \Delta_0 + \Delta_1 + \Delta_2]Q, ...$$

onde r_0 é o menor primo maior que B_1, e Δ_i são as diferenças entre os primos subsequentes maiores que r_0. Se, em algum desses cálculos, existir uma operação de inversão impossível de ser realizada, então um fator não trivial de n pode ser encontrado [CP02].

Exemplo :
Considere $n = 143$, $B_1 = 3$, $B_2 = 13$ e a curva $y^2 = x^3 + 118x + 1$. Supõe-se que na execução da primeira fase do algoritmo não foi possível realizar a fatoração de n e obteve-se o ponto $kP = Q = (64, 67)$. Nesse caso, é executada a segunda fase do algoritmo verificando para cada um dos números primos $r_0 = 5$, $r_1 = 7$, $r_2 = 11$ e $r_3 = 13$ se $r_iQ = O$. Essa verificação pode ser obtida através da seguinte sequência de cálculos:

$r_0Q = (134, 29)$

$\Delta_0 = r_1 - r_0 = 2$

$\Delta_0 Q = (16, 72)$

$[r_0 + \Delta_0]Q = (134, 29) + (16, 72) = (9, 43)$

$\Delta_1 = r_2 - r_1 = 4$

$\Delta_1 Q = (37, 55)$

$[r_0 + \Delta_0 + \Delta_1]Q = (9, 43) + (37, 55) = (68, 136)$

$\Delta_2 = r_3 - r_2 = 2$

$\Delta_2 Q = (16, 72)$

Como as operações são realizadas módulo 143, ao tentar calcular $[r_0+\Delta_0+\Delta_1+\Delta_2]Q = (68, 136) + (16, 72)$ verifica-se que não existe o inverso de $x_1 - x_2 = 52 \mod 143$. Calculando o máximo divisor comum entre 52 e 143 obtém-se o fator 13 e, portanto, a fatoração é realizada.

4.7 Algumas considerações

Neste capítulo foram apresentados, de maneira geral, alguns métodos e algoritmos básicos que podem ser utilizados na implementação do método das curvas elíticas. Estudos mais específicos sobre otimizações de operações como inversão e multiplicação escalar podem ser encontrados em [BHLM01, CMKM03, CP02, ELM03]. Para estudos em paralelização dos algoritmos de fatoração pode-se recorrer a [Bre99, DL93].

Para a manipulação de inteiros muito grandes é imprescindível utilizar um conjunto de operações de multiprecisão de inteiros que possibilite realizar operações aritméticas em inteiros de tamanho arbitrário ou tamanho fixo suficientemente grande. O presente trabalho não apresenta um estudo aprofundado dessas rotinas básicas,

pois um estudo desta natureza dependeria de algumas decisões de um projeto, como por exemplo, das características do hardware que se pretende usar. Porém, existe uma variedade de bibliotecas livres que implementam essas primitivas. Algumas delas são: GMP, LiDIA, CLN, PARI, NTL [HK05, Gra05, Gro04, Sho05b, CB04]. A seguir, é apresentada uma descrição sucinta dessas bibliotecas:

- **GMP**(*GNU Multiple Precision arithmetic library*): É uma biblioteca escrita em C, para artimética de precisão arbitrária, realizando cálculos com números inteiros, racionais e números de ponto flutuante. É direcionada principalmente para aplicações criptográficas, sistemas algébricos e pesquisas em álgebra computacional. É uma das bibliotecas mais eficientes que implementa os algoritmos mais rápidos assintoticamente. Foi escrita por Törbjorn Granlund, por quem é ainda mantida. Vários grupos e organizações também tem contribuído para o aprimoramento dessa biblioteca;

- **LiDIA**(*A C++ Library for Computational Number Theory*): É uma biblioteca orientada a objeto que fornece uma coleção de implementações otimizadas de algoritmos e vários tipos de dados de multi-precisão. Inclui rotinas para fatoração de inteiros, determinação de logaritmos em corpos finitos, contagem de pontos em curvas elíticas e etc. Permite a utilização de pacotes diferentes de multi-precisão de inteiros como, por exemplo, GMP e CLN. Foi desenvolvida na Universidade de Tecnologia de Darmstadt;

- **CLN**(*Class Library for Numbers*): Permite cálculos com números inteiros (com precisão ilimitada), racionais, números de ponto flutuante, complexos, inteiros modulares e polinomials. CLN é uma biblioteca escrita em C++ que executa funções aritméticas, lógicas e transcendentais (não polinomiais) elementares. Foi escrita por Bruno Haible e é mantida atualmente por Richard Kreckel;

- **PARI**: É uma biblioteca escrita em C que foi desenvolvida em Bordeaux, França, por uma equipe conduzida por Henri Cohen. Ela executa computações simbólicas ou numéricas como uma calculadora poderosa. Suas características principais incluem a computação numérica com precisão arbitrária, cálculos simbólicos e operações de matrizes e vetores. Pari também fornece uma relação de funções para cálculos em teoria dos números. Sua vantagem principal está em sua velocidade;

- **NTL**(*Library for doing Number Theory*): É uma biblioteca de alta performance, escrita em C++ que fornece estruturas de dados e algoritmos para inteiros de tamanho arbitrário, vetores, matrizes, polinômios sobre inteiros e corpos finitos e para a aritmética de precisão arbitrária de ponto flutuante. É escrita e mantida principalmente por Victor Shoup.

Capítulo 5

Implementação

Este capítulo apresenta uma descrição detalhada dos aspectos mais relevantes da implementação do método das curvas elíticas desenvolvida durante este trabalho. A Seção 5.1 apresenta o ambiente de desenvolvimento da implementação e a Seção 5.2 descreve as etapas da implementação. Na Seção 5.3 são descritos os tipos de dados utilizados e a Seção 5.4 apresenta as principais funções utilizadas para efetuar as operações aritméticas necessárias. Nas seções 5.5, 5.6, 5.7 e 5.8 são descritas as implementações de cada passo do algoritmo de fatoração.

5.1 Ambiente de desenvolvimento

O algoritmo do método das curvas elíticas, apresentado no Capítulo 3, foi implementado na linguagem C em uma plataforma Pentium IV 2,40 GHz, 480 MB de RAM, usando o sistema operacional Linux Fedora Core 2.

Para a representação de números grandes de tamanho arbitrário foi utilizada a biblioteca GMP [Gra05], versão 4.1.4, disponibilizada no site http://www.swox.com/gmp.

5.2 Etapas da implementação

O método das curvas elíticas foi implementado em duas versões:

- **1ª versão:** Foi implementado o algoritmo original de Lenstra, apresentado na Seção 3.3.1. Essa implementação era constituída apenas da primeira fase do algoritmo;
- **2ª versão:** Foi acrescentada à implementação inicial a segunda fase do algoritmo, conforme apresentado na Seção 4.6.

Foram realizados testes com as duas implementações, cujos resultados são mostrados no Capítulo 6.

5.3 Tipos de dados e inicialização

Na biblioteca GMP, os dados do tipo inteiro de tamanho arbitrário são representados pela sintaxe `mpz_t` e as funções são iniciadas com o prefixo `mpz_`. As principais variávies inteiras de tamanho arbitrário utilizadas na implementação do método das curvas elíticas são apresentadas a seguir:

- `n`: número a ser fatorado;
- `a` e `b`: parâmetros da curva;
- `x1` e `y1`: coordenadas dos pontos;
- `k`: multiplicador da operação escalar;
- `gcd`: utilizado nos cálculos de mdc;

Essas essas variáveis foram inicializadas através da função `mpz_init` de acordo com seguinte procedimento:

```
// Inicializa variaveis do tipo mpz_t
void inicializa_mpz(mpz_t n, mpz_t a, mpz_t b, mpz_t x1,
                    mpz_t y1, mpz_t k, mpz_t gcd)
{
  mpz_init(n);
  mpz_init(a);
  mpz_init(b);
  mpz_init(x1);
  mpz_init(y1);
  mpz_init(k);
  mpz_init(gcd);
}
```

5.4 Operações aritméticas

Assumindo-se que todas as variáveis foram inicializadas, as operações aritméticas exigidas pelo algoritmo, foram realizadas a partir das funções disponibilizadas pela biblioteca GMP, como:

- `mpz_add`: retorna o resultado da adição de dois operandos;
- `mpz_sub`: retorna o resultado da subtração de dois operandos;
- `mpz_mul`: retorna o resultado da multiplicação de dois operandos;
- `mpz_neg`: realiza a mudança de sinal em um operando;
- `mpz_mod`: calcula o módulo de um operando;
- `mpz_pow`: retorna o resultado da potência de um operando;
- `mpz_root`: retorna o resultado da raíz *n-ésima* de um operando.

Nessa biblioteca as operações aritméticas são realizadas passo-a-passo pois, em geral, as funções só permitem fazer operações com apenas 2 operandos de cada vez. Por exemplo para realizar a operação $c = 4a^3 + 27b^2$, pode-se proceder da seguinte maneira:

Função	Operação
`mpz_pow_ui(a, a, 3)`	$a \leftarrow a^3$
`mpz_mul_ui(a, a, 4)`	$a \leftarrow 4a^3$
`mpz_pow_ui(b, b, 2)`	$b \leftarrow b^2$
`mpz_mul_ui(b, b, 27)`	$b \leftarrow 27b^2$
`mpz_add(c, a, b)`	$c \leftarrow 4a^3 + 27b^2$

5.5 Testando a entrada n

Como visto no Capítulo 4, antes de iniciar a fatoração de n é interessante que seja verificado se n é um número possivelmente primo ou uma potência perfeita ou, ainda, coprimo a 6.

Nesta implementação foi realizado o teste de primalidade de n através da função `mpz_probab_prime_p`, disponibilizada pela biblioteca GMP. Esta função realiza o teste probabilístico de primalidade de Miller-Rabin de acordo com um número de tentativas definido pelo programador. Foi estabelecido um limite de 10 tentativas para verificar a primalidade de n, reduzindo as chances de um número composto retornar "provavelmente primo".

O teste de potência perfeita foi realizado através da função `mpz_perfect_power_p` também disponível na biblioteca GMP. Finalmente, para verificar se n é coprimo a 6, calculou-se o $mdc(n, 6)$ utilizando-se a função `mpz_gcd_ui`.

Essa sequência de testes realizados com a entrada n pode ser verificada no código apresentado a seguir.

```
//Pre-condicao: n >= 2
if( mpz_cmp_ui(n, 2) < 0 )
{
  gmp_printf("Numero invalido (n < 2)\n");
  return 0;
}

// Teste de pseudo-primalidade pelo Miller-Rabin usando 10 tentativas
if( mpz_probab_prime_p(n, 10) > 0 )
{
```

```
  gmp_printf("Numero possivelmente primo\n");
  return 0;
}
else
  gmp_printf("Iniciando fatoracao de numero composto\n");

// Verificar se mdc(n,6)!=1 ou se n e da forma m^r
mpz_gcd_ui(gcd, n, 6); if( mpz_cmp_ui(gcd, 1) != 0 ||
mpz_perfect_power_p(n) )
{
  gmp_printf("Numero divisivel por 2 ou 3 e/ou potencia
  perfeita\n");
  return 0;
}
```

5.6 Obtenção da curva elítica

Neste trabalho, o método das curvas elíticas foi implementado utilizando-se o sistema de coordenadas afins para os pontos da curva. Os pontos (x, y) e o parâmetro a da curva, foram gerados aleatoriamente limitados por n através da função `mpz_urandomm`, que gera um inteiro aleatório no intervalo de 0 a $n - 1$, inclusive.

```
// Escolhe x1 e y1 aleatorios
void escolha_ponto(mpz_t x1, mpz_t y1, const mpz_t n)
{
  mpz_urandomm(x1, randstate, n);
  mpz_urandomm(y1, randstate, n);
}
```

Uma vez gerados os pontos $(x1, y1)$, é gerado o parâmetro a e, logo em seguida, é calculado o parâmetro b da curva dado por $b = y_1^2 - x_1^3 - ax_1 \bmod n$. Os parâmetros a e b devem satisfazer $mdc(4a^3 + 27b^2, n) = 1$, como apresentado a seguir.

```
// Escolhe a aleatorio e calcula b, ate que satisfaca as condicoes do
```

```
// algoritmo. Retorna em gcd o mdc(4a^3 + 27b^2, n).
void escolha_a(mpz_t gcd, mpz_t a, mpz_t b, const mpz_t x1,
 const mpz_t y1,
               const mpz_t n)
{
  mpz_t tmp1;
  mpz_t tmp2;
  mpz_init(tmp1);
  mpz_init(tmp2);

  do
  {
    // Escolhe a aleatorio.
    mpz_urandomm(a, randstate, n);

    // Calcula b = y1*y1 - x1*x1*x1 - a*x1
    mpz_pow_ui(b, y1, 2);
    mpz_pow_ui(tmp1, x1, 3);
    mpz_sub(b, b, tmp1);
    mpz_mul(tmp1, a, x1);
    mpz_sub(b, b, tmp1);

    // Calcula 4a^3 + 27b^2 e verifica se mdc(4a^3 + 27b^2, n) = 1.
    mpz_pow_ui(tmp1, a, 3);
    mpz_mul_ui(tmp1, tmp1, 4);
    mpz_pow_ui(tmp2, b, 2);
    mpz_mul_ui(tmp2, tmp2, 27);
    mpz_add(tmp1, tmp1, tmp2);

    // calcula mdc e armazena em gcd.
    mpz_gcd(gcd, tmp1, n);

  }while( mpz_cmp(gcd, n) == 0 );
}
```

5.7 Calculando *k*

O multiplicador k deve ser tomado como um produto de primos pequenos elevados a potências adequadas, conforme determina o algoritmo do método de fatoração. Nesta implementação, optou-se por calcular k fazendo-se:

$$k = 2^{\lfloor \ln B / \ln 2 \rfloor} \cdot 3^{\lfloor \ln B / \ln 3 \rfloor} \cdot \ldots \cdot B$$

No Capítulo 6 são apresentados os diversos valores atribuídos ao parâmetro B. Depois de estabelecido este parâmetro, o cálculo de k foi executado utilizando-se a função `mpz_nextprime` que usa um algoritmo probabilístico para identificar primos. Através desta função foram tomados os números primos de 2 até o parâmetro B e, em seguida, esses primos foram elevados à potências adequadas. Esses procedimentos podem ser verificados a seguir.

```
while( mpz_cmp(primo, B) <= 0 )
{
  expoente = log(mpz_get_d(B))/log(mpz_get_d(primo));
  mpz_pow_ui(potencia, primo, (unsigned) expoente);
  mpz_mul(k, k, potencia);
  mpz_nextprime(primo, primo);
}
```

5.8 Calculando *kP*

A implementação da operação kP, de adição de pontos na curva, foi realizada através dos 3 seguintes passos:

1. Primeiramente foi obtida a representação binária de k através da função `mpz_sizeinbase` da biblioteca GMP;
2. Com base nesta representação, foram calculados os pontos 2^iP, $1 \leq i \leq r$, como apresentado em 4.5.1, através das fórmulas de adição de pontos dadas no Capítulo 3. Esses pontos foram armazenados na tabela `tabela`. A seguir é mostrado o código que implementa as operações realizadas nesse passo.

```
// Tenta calcular kP. Caso consiga, retorna 1. Caso nao consiga,
// gcd contera o mdc entre inv(1/2y1) e n, e retorna 0.
unsigned calcula_kP(mpz_t gcd, const mpz_t k, const mpz_t x1,
                    const mpz_t y1, const mpz_t a, const mpz_t n)
```

```
{
  unsigned i;
  mpz_t lambda, tmp1, tmp2;
  mpz_t soma_ponto[2];

  mpz_init(lambda);
  mpz_init(tmp1);
  mpz_init(tmp2);
  mpz_init(soma_ponto[0]);
  mpz_init(soma_ponto[1]);

  // Constru\c{c}\~{a}o da tabela: inicializa primeiro ponto
  mpz_set_ui(tabela[0][0], x1);
  mpz_set_ui(tabela[0][1], y1);

  // Constru\c{c}\~{a}o da tabela: calcula demais pontos
  for(i=1; i<mpz_sizeinbase(k,2); i++)
  {
    // calcula lambda
    mpz_pow_ui(lambda, tabela[i-1][0], 2); // lambda: x1^2
    mpz_mul_ui(lambda, lambda, 3);         // lambda: 3x1^2
    mpz_add(lambda, lambda, a);            // lambda: 3x1^2 + a
    mpz_mul_ui(tmp1, tabela[i-1][1], 2);   // tmp: 2y1

    // Se nao existe inverso...
    if( mpz_invert(tmp1, tmp1, n) == 0 )
    {
      mpz_gcd(gcd, tmp1, n);
      mpz_clear(lambda);
      mpz_clear(tmp1);
      mpz_clear(tmp2);
      mpz_clear(soma_ponto[0]);
      mpz_clear(soma_ponto[1]);
      return 0;
    }
    //lambda:(3x^2+a)*inverso_mod_n(2y1)
    mpz_mul(lambda, lambda, tmp1);

    // calcula x atual
    mpz_pow_ui(tabela[i][0], lambda, 2);
    mpz_mul_ui(tmp1, tabela[i-1][0], 2);
    mpz_sub(tabela[i][0], tabela[i][0], tmp1);
```

```
    mpz_mod(tabela[i][0], tabela[i][0], n);

    // calcula y atual
    mpz_sub(tmp1, tabela[i-1][0], tabela[i][0]);
    mpz_mul(tabela[i][1], tmp1, lambda);
    mpz_sub(tabela[i][1], tabela[i][1], tabela[i-1][1]);
    mpz_mod(tabela[i][1], tabela[i][1], n);
}
```

3. Em seguida, foi feita uma varredura na tabela `tabela`, obtida no passo anterior, e calculadas as somas parciais dos valores 2^iP, para os quais $k_i = 1$, conforme apresentado em 4.5.1.

```
// Inicializa soma pontos: atribui o primeiro ponto significativo
for(i=0; i<mpz_sizeinbase(k,2); i++)
{
  if( mpz_tstbit(k, i) == 0 )
    continue;
  mpz_set(soma_ponto[0], tabela[i][0]);
  mpz_set(soma_ponto[1], tabela[i][1]);
  break;
}
// Soma os demais pontos significativos
for(i=i+1; i<mpz_sizeinbase(k,2); i++)
{
  if( mpz_tstbit(k, i) == 0 )
    continue;

  mpz_sub(lambda, tabela[i][1], soma_ponto[1]); // lambda: y2 - y1
  mpz_sub(tmp1, tabela[i][0], soma_ponto[0]);   // tmp1: x2 - x1

  // Se nao existe inverso...
  if( mpz_invert(tmp1, tmp1, n) == 0 )
  {
    mpz_gcd(gcd, tmp1, n);
    mpz_clear(lambda);
    mpz_clear(tmp1);
    mpz_clear(tmp2);
    mpz_clear(soma_ponto[0]);
```

```
      mpz_clear(soma_ponto[1]);
      return 0;
    }
    mpz_mul(lambda, lambda, tmp1); // lambda: y2 - y*inverso_mod_n(x2 - x1)

    mpz_set(tmp1, soma_ponto[0]); // guardo x1 da itera\c{c}\~{a}o anterior
    mpz_set(tmp2, soma_ponto[1]); // guardo y1 da itera\c{c}\~{a}o anterior

    // calcula x atual: lambda^2 - x1 - x2
    mpz_pow_ui(soma_ponto[0], lambda, 2);
    mpz_sub(soma_ponto[0], soma_ponto[0], tmp1);
    mpz_sub(soma_ponto[0], soma_ponto[0], tabela[i][0]);
    mpz_mod(soma_ponto[0], soma_ponto[0], n);

    // calcula y atual
    mpz_sub(soma_ponto[1], tmp1, soma_ponto[0]);
    mpz_mul(soma_ponto[1], soma_ponto[1], lambda);
    mpz_sub(soma_ponto[1], soma_ponto[1], tmp2);
    mpz_mod(soma_ponto[1], soma_ponto[1], n);
  }
```

Se, em alguma etapa do cálculo de kP não for possível a obtenção do inverso multiplicativo de algum número, então é calculado o máximo divisor comum entre esse número e a entrada n para verificar se foi encontrado um fator não trivial de n. Em caso afirmativo, esse fator é retornado e encerra-se a execução do algoritmo. Caso contrário, é gerado aleatoriamente um outro ponto (x, y) e uma nova curva, a partir dos quais é realizada uma nova tentativa de fatoração de n.

Capítulo 6

Testes e análise dos resultados

Foram realizados alguns testes com a implementação do método das curvas elíticas visando a obtenção de resultados experimentais. Os testes realizados tiveram por objetivo principal verificar o tempo médio de execução do algoritmo e o número de curvas utilizadas no processo de fatoração. Este capítulo apresenta os testes e resultados experimentais como segue. A Seção 6.1 faz algumas observações gerais sobre os testes realizados. A Seção 6.2 apresenta os testes e resultados obtidos com a implementação.

6.1 Testes realizados

Como apresentado na Seção 5.2, o método das curvas elíticas foi implementado em duas versões. A 1ª versão foi realizada com base no algoritmo original de Lenstra e a 2ª versão implementava o método das curvas elíticas com a adição da segunda fase. Foram realizados testes com essas duas implementações, a partir dos quais buscou-se analisar o tempo de execução e o número de curvas utilizadas na fatoração.

Na 1ª versão os testes foram realizados a partir de diversos valores atribuídos ao parâmetro B, utilizado para determinar k. As técnicas usadas para a obtenção de

B, durante os testes, são apresentadas a seguir:

1. valores de B_1 da Tabela 4.1, sugeridos em [SJ93] como valores iniciais[1] para o parâmetro B;
2. $B = \lfloor \sqrt[6]{n} \rfloor$;
3. $B = \lfloor \sqrt[7]{n} \rfloor$;
4. $B = \lfloor \sqrt[8]{n} \rfloor$;
5. $B = \lfloor \sqrt[9]{n} \rfloor$;
6. $B = \lfloor \sqrt[10]{n} \rfloor$.

Os testes foram realizados para cada um desses 6 conjuntos de experimentos. Foram testados números compostos por apenas 2 fatores, ambos com o mesmo número de dígitos decimais. O número de dígitos do fator p variou de 1 a 20. Observe que p com 20 dígitos decimais implica na fatoração de um número composto de, aproximadamente, 40 dígitos decimais. Não foram realizados testes para fatores com um maior número de dígitos, pois a implementação da operação de adição pontos na curva envolvia a construção de uma tabela, conforme apresentado no Capítulo 5, e essa tabela se tornava muito grande à medida que aumentava os dígitos dos fatores. Isso ocasionou uma demanda muito grande de memória RAM, dificultando a realização de testes para fatores com um maior número de dígitos.

Foram realizados 20 testes para cada número de dígitos de p e para cada técnica de cálculo de B. A partir desses testes, foram obtidas as médias aritméticas dos tempos de execução do algoritmo e do número de curvas utilizadas no processo de fatoração. Finalmente, obteve-se a média geral dos resultados para cada cada conjunto de experimentos.

Na 2ª versão da implementação, os testes foram realizados utilizando-se os parâmetros B_1 e B_2 sugeridos em [SJ93] e apresentados na Tabela 4.1. Também foram

[1]Esses parâmetros são sugeridos para uso em implementações que contenham a segunda fase do algoritmo. Como a 1ª versão da implementação apresentada neste trabalho é constituída apenas da primeira fase, esses parâmetros foram utilizados a título ilustrativo.

realizados 20 testes para cada número de dígitos de p, com p variando de 1 a 20 dígitos.

6.2 Análise dos resultados obtidos

6.2.1 Tempo de execução para a 1ª versão

A Tabela 6.1 mostra a média dos tempos de execução em segundos, obtidos em cada conjunto de testes realizados com cada técnica de cálculo de B. A primeira coluna da tabela representa o número de dígitos decimais do fator p e as demais colunas representam as médias dos tempos de execução, em segundos, obtidos com a implementação.

Tabela 6.1: Tempo de Execução

número de dígitos	$B = \sqrt[6]{n}$	$B = \sqrt[7]{n}$	$B = \sqrt[8]{n}$	$B = \sqrt[9]{n}$	$B = \sqrt[10]{n}$	parâmetros ótimos
1	0,000040	0,000157	0,000190	0,000179	0,000038	0,000225
2	0,000056	0,000235	0,000243	0,000258	0,000039	0,000426
3	0,000204	0,000351	0,000767	0,000499	0,000300	0,000361
4	0,002865	0,003943	0,005195	0,004006	0,019127	0,000792
5	0,009863	0,006625	0,015747	0,032378	0,030294	0,007471
6	0,023593	0,019900	0,033082	0,060472	0,126919	0,098641
7	0,033994	0,028839	0,061225	0,160872	0,391117	0,028273
8	0,082990	0,076389	0,104050	0,205673	0,623792	0,258750
9	0,153361	0,114778	0,309545	0,593408	0,765260	0,280019
10	0,404592	0,236918	0,380860	0,827553	1,716420	0,275494
11	0,655230	0,547447	1,062723	1,327398	2,036762	0,822507
12	0,797420	1,092006	1,136158	2,844900	2,753942	2,169142
13	2,353221	1,836695	3,326787	4,262546	6,040799	2,870683
14	7,731692	6,285641	4,138208	4,507227	15,855650	3,141925
15	12,990800	10,016875	7,812391	8,945571	31,308923	10,869936
16	79,393228	24,739222	21,364283	26,054348	30,659625	44,673430
17	100,323392	54,965488	35,693913	42,998321	74,0216107	44,675435
18	148,709236	80,317565	67,300312	94,571107	81,065693	158,068560
19	-	195,423160	132,891916	146,856867	264,702145	208,241863
20	-	311,117788	202,686862	281,584010	310,675445	313,300167

Não foram apresentados resultados com $B = \lfloor \sqrt[6]{n} \rfloor$ para fatores de 19 e 20 dígitos decimais, pois esses testes exigiam a utilização demasiada de memória RAM,

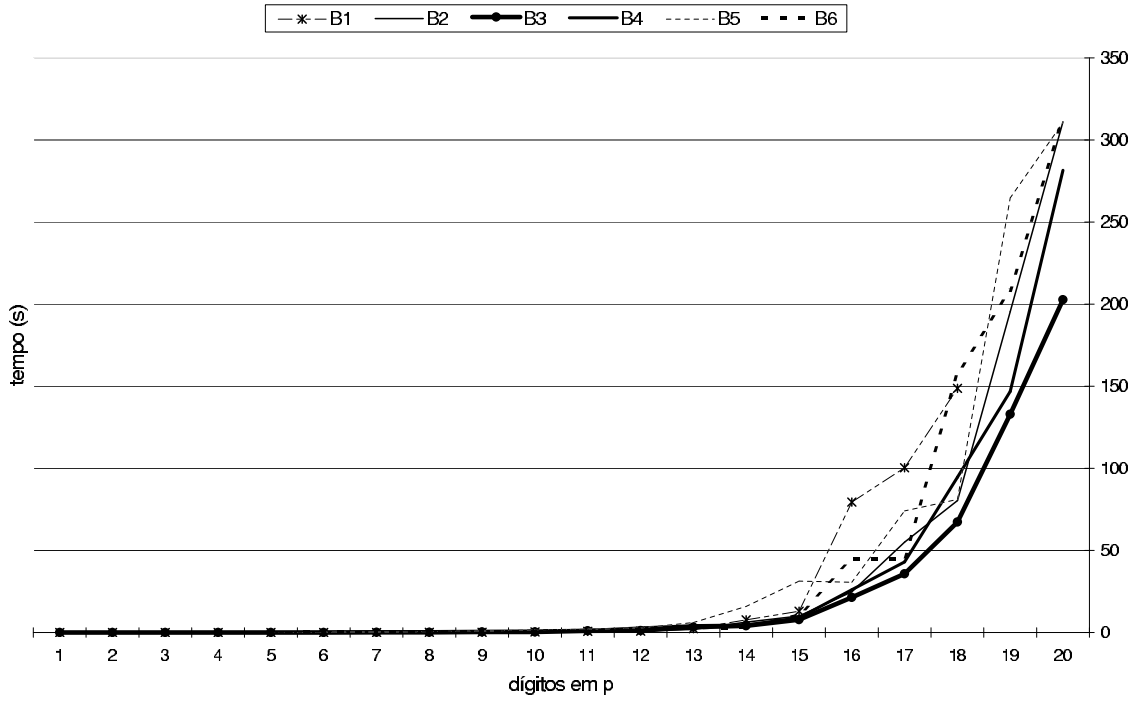


Figura 6.1: Comparativo entre os tempos de execução.

tornando inviável sua execução.

Os resultados mostrados na Tabela 6.1, também podem ser visualizados na Figura 6.1. Para fins práticos, nas figuras os conjuntos de testes realizados para cada técnica de cálculo de B serão denotados da seguinte maneira:

- $B = \lfloor \sqrt[6]{n} \rfloor$ por $B1$;
- $B = \lfloor \sqrt[7]{n} \rfloor$ por $B2$;
- $B = \lfloor \sqrt[8]{n} \rfloor$ por $B3$;
- $B = \lfloor \sqrt[9]{n} \rfloor$ por $B4$;
- $B = \lfloor \sqrt[10]{n} \rfloor$ por $B5$;
- B dado pelos parâmetros ótimos por $B6$.

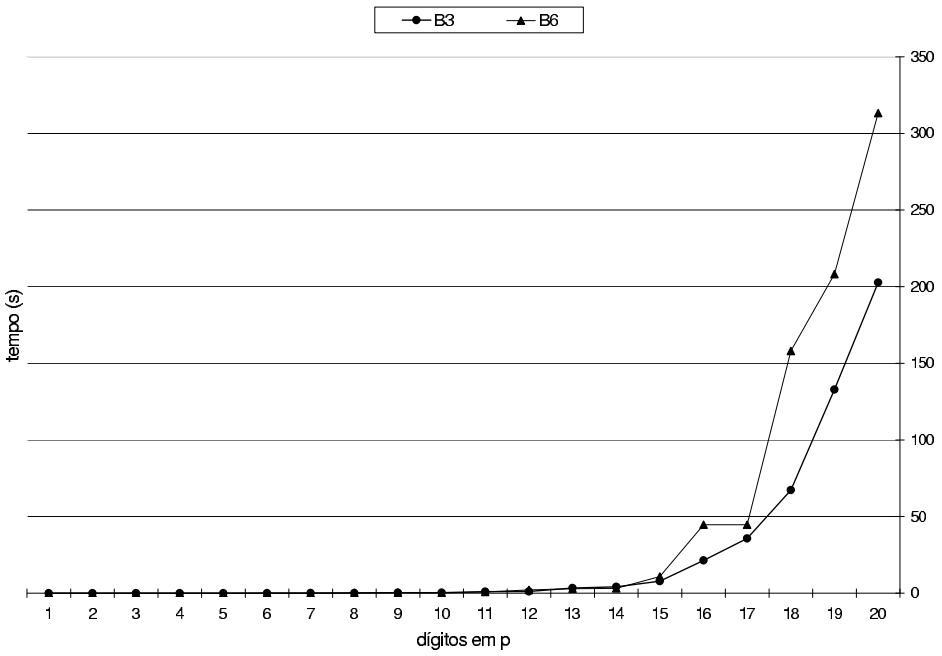


Figura 6.2: Comparativo entre os tempos de execução em $B = \lfloor \sqrt[8]{n} \rfloor$ e B dado pelos parâmetros ótimos.

Nessa figura pode-se observar que dentre as técnicas testadas para o cálculo de B, nessa implementação, o melhor tempo de execução do algoritmo ocorreu para o parâmetro $B = \lfloor \sqrt[8]{n} \rfloor$. Também é notável que o tempo de execução mais lento dentre as técnicas testadas ocorreu para $B = \lfloor \sqrt[6]{n} \rfloor$.

Um comparativo entre o tempo de execução obtido em $B = \lfloor \sqrt[8]{n} \rfloor$ e B dado pelos parâmetros ótimos é apresentado na Figura 6.2. Nessa figura, não é possível identificar o comportamento dos gráficos até 14 dígitos, porém nota-se que o tempo de execução para $B = \lfloor \sqrt[8]{n} \rfloor$ é menor acima de 15 dígitos.

Uma análise mais detalhada do comportamento desses gráficos pode ser feita a partir da Figura 6.3 que mostra os gráficos de $B = \lfloor \sqrt[8]{n} \rfloor$ e de B dado pelos parâmetros ótimos plotados sobre um eixo logarítmico. A partir dessa figura nota-se que o tempo de execução entre os dois gráficos é bastante variado até 14 dígitos, não possibilitando identificar qual o menor tempo de execução nessa faixa. No entanto,

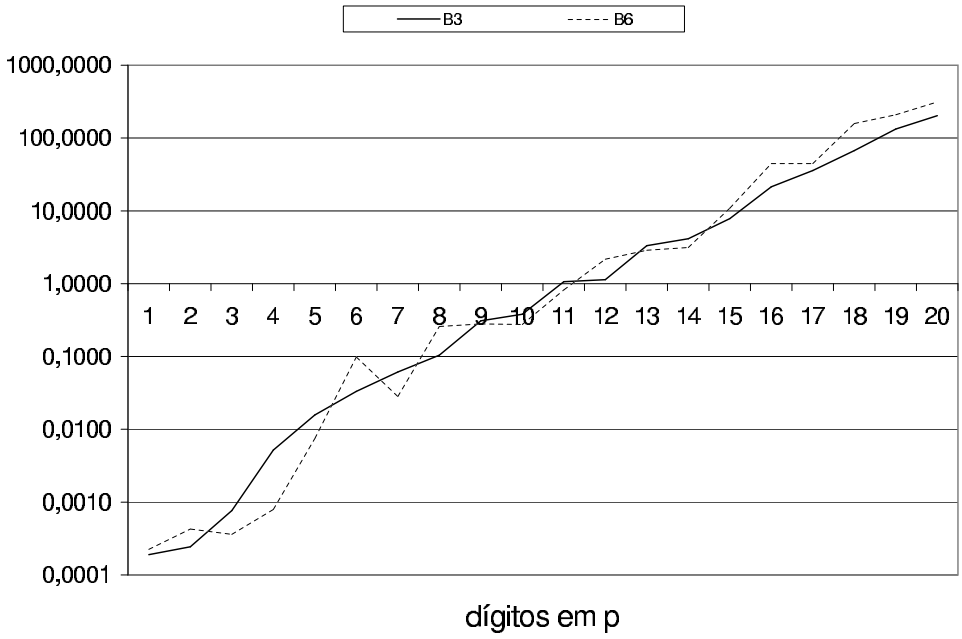


Figura 6.3: Gráficos dos tempos de execução obtidos para $B = \lfloor \sqrt[8]{n} \rfloor$ e B dado pelos parâmetros ótimos.

a partir de 15 dígitos o tempo de execução é sempre menor para $B = \lfloor \sqrt[8]{n} \rfloor$.

Calculando a média geral dos tempos de execução de $B = \lfloor \sqrt[8]{n} \rfloor$ e B dado pelos parâmetros ótimos obtém-se 23,916223 segundos no primeiro conjunto de experimentos e 39,489205 segundos no segundo conjunto. Isso representa uma economia média de 39,44% no tempo de execução do algoritmo para $B = \lfloor \sqrt[8]{n} \rfloor$.

Como apresentado no Capítulo 3, o tempo de execução estimado para o método das curvas elíticas é subexponencial sendo definido por $T \approx e^{(\sqrt{2}+c_2)\sqrt{\ln p \ln \ln p}}$. A Figura 6.4 mostra o gráfico do tempo de execução obtido com esta implementação para $B = \lfloor \sqrt[8]{n} \rfloor$ e o polinômio aproximado para este tempo de execução. Observa-se que para um fator p de até 20 dígitos, o tempo de execução obtido pode ser bem aproximado pelo polinômio de grau 5 dado por $y = 0,001x^5 - 0,0398x^4 + 0,5569x^3 - 3,3974x^2 + 8,4951x - 6,2987$.

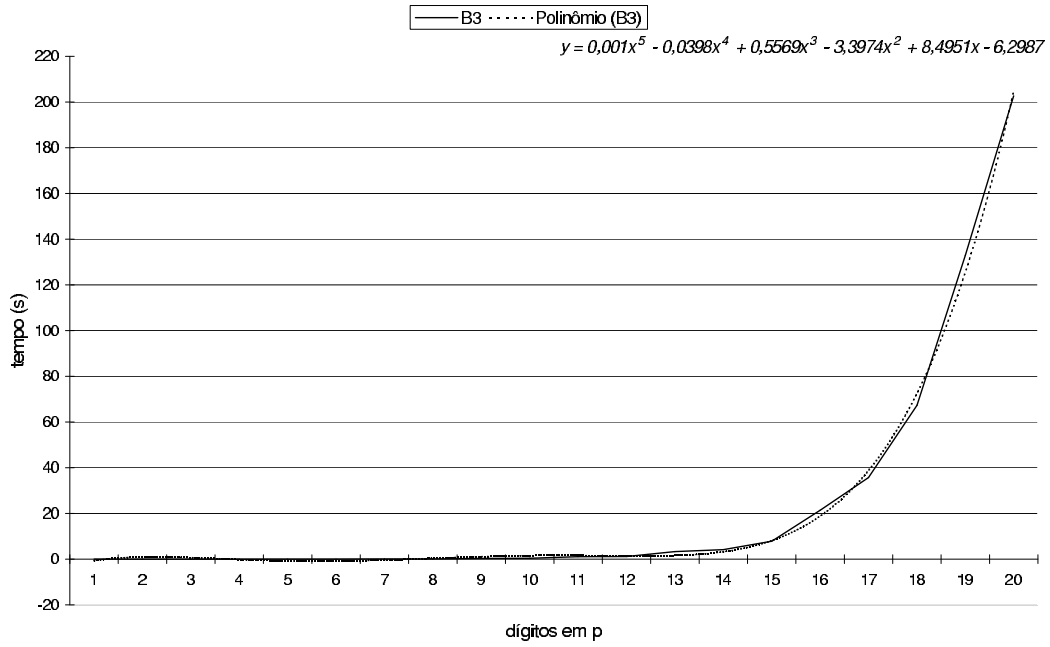


Figura 6.4: Aproximação polinomial para o tempo de execução em $B = \sqrt[8]{n}$.

6.2.2 Número de curvas para 1ª versão

Em relação ao número de curvas utilizadas no processo de fatoração, a Tabela 6.2 apresenta a média do número de curvas utilizadas em cada conjunto de experimentos realizados com cada técnica de cálculo de B. A primeira coluna da tabela representa o número de dígitos decimais do fator p e as demais colunas representam o número médio de curvas utilizadas com a implementação.

Os gráficos dos resultados mostrados na Tabela 6.2 são apresentados na Figura 6.5. Essa figura mostra resultados bastante variados. Isso acontece pelo fato de as curvas serem geradas aleatoriamente, o que pode significar um número variado de iterações para gerar uma curva adequada para a fatoração de n. No entanto, percebe-se uma menor quantidade curvas utilizadas pelo algoritmo para $B = \lfloor \sqrt[6]{n} \rfloor$ e a maior quantidade para $B = \lfloor \sqrt[10]{n} \rfloor$.

O número de curvas utilizado na fatoração não constitui um dado importante se

Tabela 6.2: Número de Curvas

dígitos	$B = \sqrt[6]{n}$ (s)	$B = \sqrt[7]{n}$ (s)	$B = \sqrt[8]{n}$	$B = \sqrt[9]{n}$	$B = \sqrt[10]{n}$	parâm. ótimos
1	3,5	2,7	3,3	2,8	3,2	1,4
2	3,9	5,4	6,0	17,4	4,6	1,1
3	6,1	4,2	22,1	18,2	17,1	1,6
4	9	13,7	20,8	66,3	276,3	3,6
5	10,3	21,1	56,2	314,7	277,6	40,1
6	8,3	19,2	73,7	223,25	1041,2	432,3
7	10	19,0	93,3	434,5	1575,4	31,4
8	14,5	31,5	81,9	268,35	1333,7	309
9	10,7	18,2	122,5	422,7	848,7	107
10	15,8	26,1	93,9	394,25	1379,1	102
11	10,9	29,2	135,5	330	897,6	105
12	6,4	30,7	82,5	432,2	743,9	265,7
13	8,6	26,5	131,1	373,55	990,4	139,7
14	11,6	40,8	81,8	205,3	1457,6	145,1
15	10,3	37,6	95,2	269,85	1945,7	204,3
16	16,4	29,4	93,2	314,75	824,4	738,6
17	9,8	33,6	86,6	301,75	1229,9	333,3
18	7	26,8	95,3	413,75	877,4	1148,1
19		28,3	90,3	331,5	1562	770,6
20		24,7	80,4	387,55	1151,1	1018,6

apresentado isoladamente. Ou seja, não é interessante que o algoritmo execute com um número mínimo de curvas se o tempo de execução não é satisfatório. Embora os resultados obtidos tenham mostrado que para $B = \lfloor \sqrt[6]{n} \rfloor$ houve um número mínimo de curvas utilizadas, esse mesmo parâmetro apresentou o maior tempo de execução como visto na Seção 6.2. Nesse sentido é interessante analisar o número de curvas utilizadas para os tempos de execução satisfátórios.

A Figura 6.6, apresenta o número de curvas para $B = \lfloor \sqrt[8]{n} \rfloor$ e para B dado pelos parâmetros ótimos, que foram os melhores tempos de execução obtidos com a implementação. Nessa figura percebe-se uma menor quantidade curvas utilizadas pelo algoritmo em $B = \lfloor \sqrt[8]{n} \rfloor$. Pode-se notar também que para B dado pelos parâmetros ótimos o número de curvas utilizadas aparenta ter um crescimento maior à medida que aumenta o número de dígitos decimais de p. Em contrapartida, os resultados obtidos para $B = \lfloor \sqrt[8]{n} \rfloor$ apresentam-se mais estáveis nessa faixa de valores.

A média geral do número de curvas utilizadas em $B = \lfloor \sqrt[8]{n} \rfloor$ foi de 73,3 curvas enquanto para B dado pelos parâmetros ótimos, foram utilizadas em média 294,9 curvas. Ou seja, em $B = \lfloor \sqrt[8]{n} \rfloor$ houve uma diminuição no número de curvas em

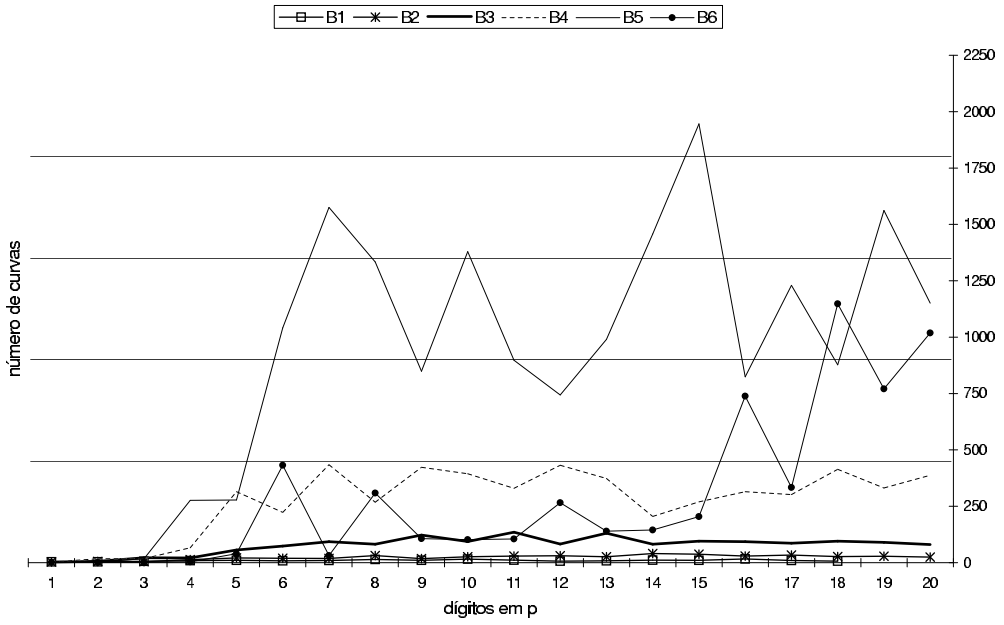


Figura 6.5: Comparativo entre números de curvas obtidos.

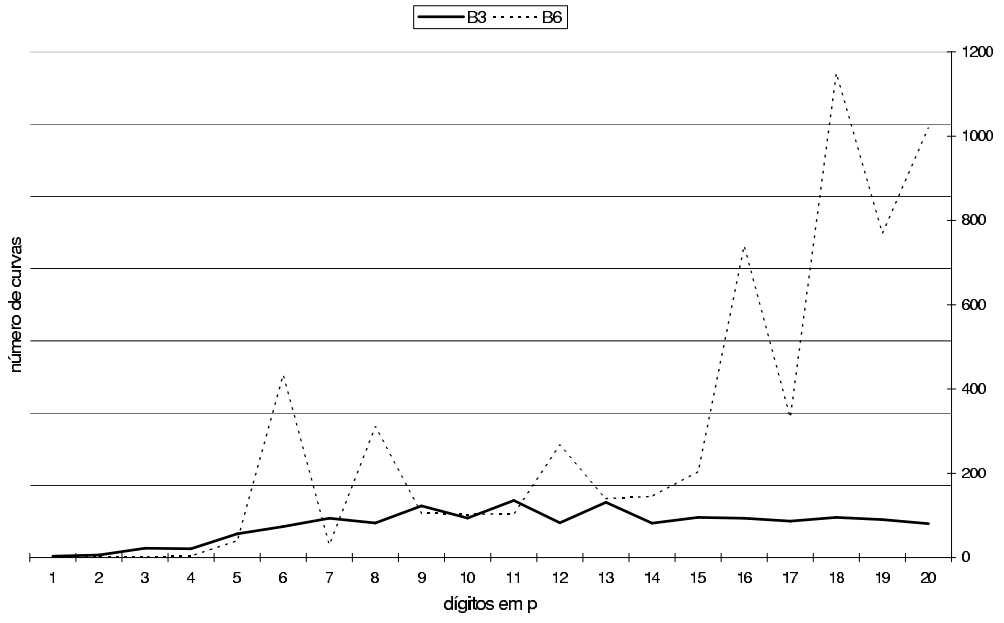


Figura 6.6: Comparativo entre números de curvas obtidos para $B = \sqrt[8]{n}$ e $B =$ parâmetros ótimos.

73,79%, em média.

6.2.3 Tempo de execução para a 2ª versão

A Tabela 6.3 mostra a média dos tempos de execução obtidos com os testes realizados a partir da 2ª versão da implementação. A primeira coluna da tabela representa o número de dígitos decimais do fator p e a segunda coluna representa as médias dos tempos de execução em segundos. Esses resultados também podem ser visualizados na Figura 6.7.

Tabela 6.3: Tempo de Execução da 2ª versão

número de dígitos	Tempo de execução (s)
1	0,00004725
2	0,0000709
3	0,00011545
4	0,00139635
5	0,0123731
6	0,04225665
7	0,036611
8	0,08908115
9	0,2279938
10	0,4690377
11	0,4137298
12	1,40812815
13	1,60939465
14	9,68466645
15	9,5926857
16	32,75541075
17	70,78618165
18	96,95439455
19	273,1362305
20	312,6259522

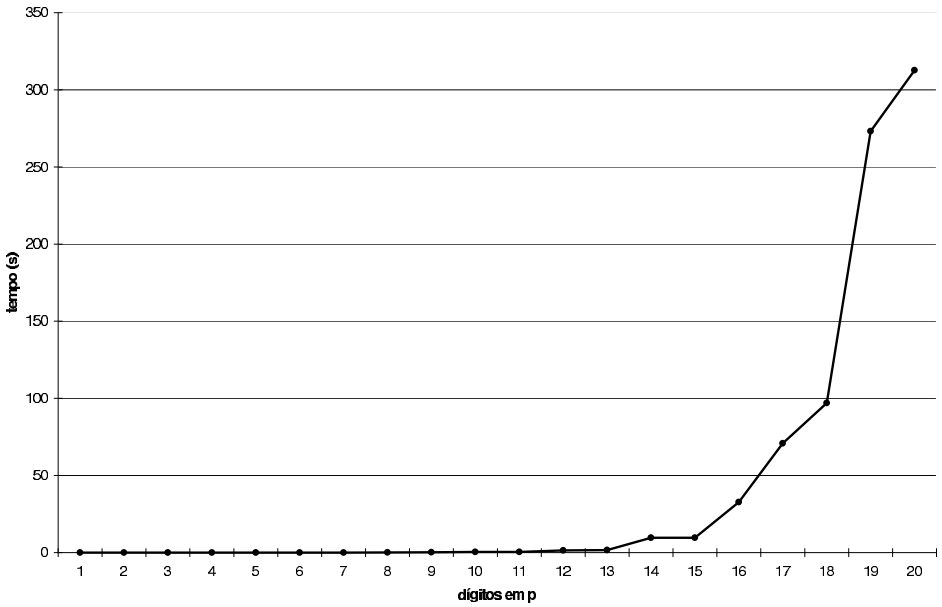


Figura 6.7: Tempo de execução obtido na 2ª versão da implementação.

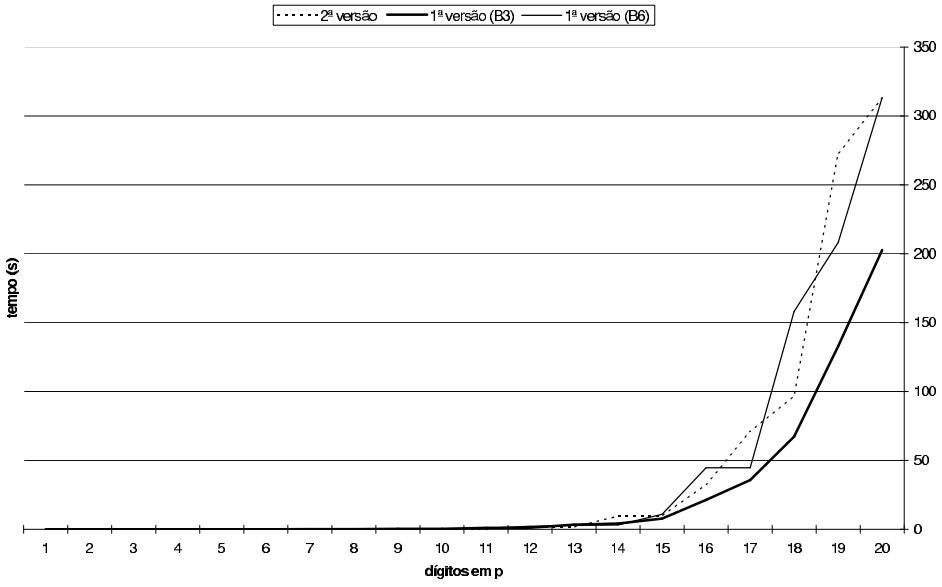


Figura 6.8: Comparativo entre os tempos de execução obtidos nas duas versões da implementação.

A Figura 6.8 apresenta um comparativo entre os tempos de execução obtidos nas duas versões da implementação. Da 1ª versão, são ilustrados os gráficos dos tempos obtidos em $B3$ para $B = \lfloor \sqrt[8]{n} \rfloor$ e $B6$ para B dado pelos parâmetros ótimos iniciais, conforme apresentado em 6.2.1.

Alguns estudos afirmam que a adição da segunda fase ao algoritmo original de Lenstra apresenta, na prática, uma melhora no tempo de execução [Bre90]. De acordo coma a Figura 6.9, não houve melhora no tempo de execução do algoritmo na 2ª versão (que implementa a segunda fase do algoritmo) em relação à 1ª versão. Uma justificativa para tal pode estar ligada a fato de que neste trabalho não foram implementadas todas as melhorias sugeridas para o método das curvas elíticas. Além disso, este estudo considera fatores de, no máximo, 20 dígitos decimais.

6.2.4 Número de curvas para a 2ª versão

A Tabela 6.4 mostra a média do número de curvas utilizadas com os testes realizados a partir da 2ª versão da implementação. Esses resultados também podem ser visualizados na Figura 6.9.

Tabela 6.4: Número de Curvas da 2ª versão

número de dígitos	Número de curvas
1	1,1
2	1,2
3	1,1
4	1,1
5	2,3
6	8,2
7	3,1
8	7,1
9	7,5
10	14,6
11	4,8
12	14,0
13	7,5
14	39,2
15	15,6
16	50,4
17	50,4
18	67,4
19	98,3
20	98,8

A Figura 6.10 apresenta um comparativo entre o número de curvas utilizadas nas duas versões da implementação. Da 1ª versão, são ilustrados os gráficos dos tempos obtidos em $B3$ para $B = \lfloor \sqrt[8]{n} \rfloor$ e $B6$ para B dado pelos parâmetros ótimos iniciais. Através dessa figura, nota-se que o número médio de curvas utilizadas na 2ª versão do algoritmo é bem menor se comparado com os resultados obtidos em $B3$ e $B6$ da 1ª versão. Mais precisamente, calculando a média geral do número de curvas utilizadas obtém-se 24,7 curvas na 2ª versão, 77,3 curvas em $B3$ e 294,9 curvas em $B6$.

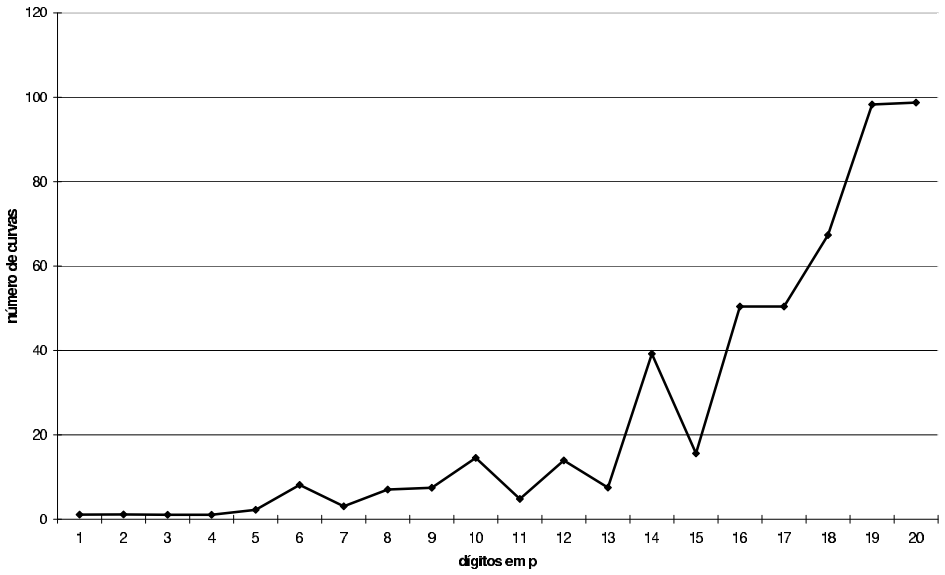


Figura 6.9: Número de curvas utilizadas na 2ª versão da implementação.

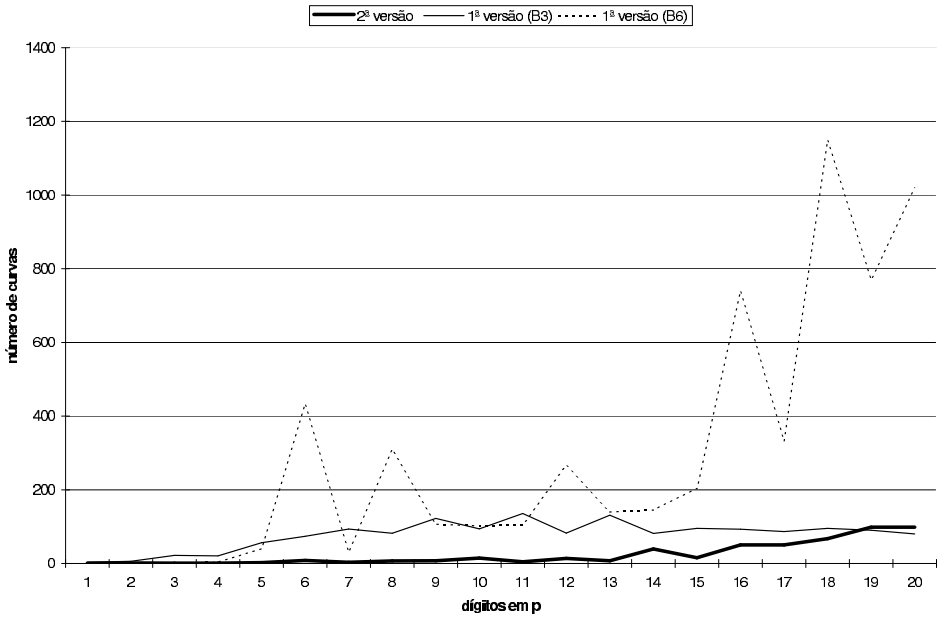


Figura 6.10: Comparativo entre o número de curvas utilizadas nas duas versões da implementação.

Capítulo 7

Considerações finais

O estudo de métodos de fatoração de inteiros merece destaque devido, principalmente, a sua aplicação em sistemas criptográficos de chave pública, cuja segurança apoia-se na ineficiência desses métodos. O método de fatoração das curvas elíticas apresenta-se como um dos mais rápidos da atualidade e, portanto, estudos sobre esse método são indispensáveis.

Este trabalho apresentou as principais escolhas a serem realizadas no processo de implementação do método das curvas elíticas, em cada passo do algoritmo. Foram implementadas duas versões do método das curvas elíticas. Na 1ª versão, o método de fatoração foi implementado com base no algoritmo original de Lenstra e, na 2ª versão foi adicionada uma segunda fase em cada tentativa de fatoração do método.

Foram realizados diversos testes com a implementação e apresentada uma análise comparativa entre o tempo de execução do algoritmo e o número de curvas utilizadas no processo de fatoração. Na 1ª versão, a implementação foi testada para diferentes valores do parâmetro B utilizado para calcular o valor do multiplicador k da operação kP. Os testes foram realizados em números compostos por 2 fatores com até 20 dígitos decimais. Nesta versão, observou-se que o melhor tempo de execução ocorreu para $B = \lfloor \sqrt[8]{n} \rfloor$.

Na 2ª versão, os testes foram realizados também com números compostos por 2

fatores com até 20 dígitos decimais, considerando os parâmetros ótimos dados por [SJ93]. Os resultados dos testes mostraram um menor número de curvas utilizadas na fatoração na 2ª versão do algoritmo, quando comparado com a 1ª versão. No entanto, não foi observada melhoria no tempo de execução obtido na 2ª versão em relação à 1ª versão. Uma justificativa para tal pode estar ligada ao fato de que neste trabalho não foram implementadas todas as melhorias sugeridas para o método das curvas elíticas.

Não foram realizados testes para fatores com mais de 20 dígitos, pois o algoritmo utilizado para a adição de pontos na curva envolvia a construção de uma tabela muito grande utilizando muita mémória RAM.

Este trabalho traz consigo algumas contribuições e, dentre essas, pode-se citar a descrição detalhada do processo de implementação do método das curvas elíticas. Além disso, essa pesquisa propiciou a descoberta do parâmetro $B = \lfloor \sqrt[8]{n} \rfloor$ através do qual foi obtido o melhor desempenho da implementação realizada nesse trabalho. Isso, por sua vez, confirma a importância de estudos sobre parâmetros a serem utilizados no processo de fatoração pelo método das curvas elíticas.

Para trabalhos futuros devem ser realizados testes para fatores com um maior número de dígitos. Nesse sentido, devem ser estudados outros algoritmos para realizar a operação de adição de pontos na curva que não demande a construção de tabelas, evitando o problema de utilização de memória. Para uma implementação com fins em otimização do algoritmo, deve ser considerada a utilização de coordenadas projetivas na implementação e devem ser estudados métodos para a escolha de curvas apropriadas a serem usadas no processo de fatoração.

Referências Bibliográficas

[AM92] A. O. L. Atkin e F. Morain. Finding suitable curves for the elliptic curve method of factorization. *Mathematics of Computation*, 60, páginas 399–405, 1992.

[BHLM01] M. Brown, D. Hankerson, J. Lopez e A. Menezes. Software implementation of the nist elliptic curves over prime fields. Em *Topics in Cryptology - CT-RSA'2001*, páginas 250–265, 2001.

[BL95] W. Bosma e A. K. Lenstra. An implementation of the elliptic curve integer factorization method. *Computational Algebra and Number Theory*, 8, páginas 119–136, 1995.

[Bre86] R. P. Brent. Some integer factorization algorithms using elliptic curves. *Australian Computer Science Communications*, 8, páginas 149–163, 1986.

[Bre89] D. M. Bressoud. *Factorization and Primality Testing*. Springer-Verlag, New York, 1989.

[Bre90] R. P. Brent. Parallel algorithms for integer factorisation. *London Mathematical Society Lecture Note Series*, 154, páginas 26–37, 1990.

[Bre99] R. P. Brent. Some parallel algorithms for integer factorisation. Em *Euro-Par'99 Parallel Processing*, volume 1685, páginas 1–22, 1999.

[Bre00] R. P. Brent. Recent progress and prospects for integer factorisation algorithms. Em *COCOON 2000*, volume 1685, páginas 3–22, 2000.

[BSS99] I. Blake, G. Seroussi e N. Smart. *Elliptic Curves in Cryptography*. Cambridge University Press, 1999.

[Car03] C. Cardoso. *Fatoração de Números Inteiros Usando Curvas Elíticas*. Dissertação de Mestrado, Departamento de Computação e Estatística, Universidade Federal de Mato Grosso do Sul, 2003.

[CB04] H. Cohen e K. Belabas. PARI/GP Library. Disponível *on-line* em março de 2006 na URL http://pari.math.u-bordeaux.fr, 2004.

[CLRS01] T.H. Cormen, C.E. Leiserson, R.L. Rivest e C. Stein. *Introduction to Algorithms*. MIT Press, 2ª edição, 2001.

[CMKM03] M. Ciet, J. Marc, L. Kristin e P.L. Montgomery. Trading inversions for multiplications in elliptic curve cryptography. Kluwer Academic Publishers, 2003.

[Cou00] S. Coutinho. *Números Inteiros e Criptografia RSA*. IMPA-SBM, 2000.

[CP02] R. Crandall e C. Pomerance. *Prime Numbers - A Computational Perspective*. Springer-Verlag, 2002.

[DL93] B. Dixon e A. K. Lenstra. Massively parallel elliptic curve factoring. Em *Eurocrypt '92*, volume 658, páginas 183–193, 1993.

[DL00] R. Dahab e J. Lopez. An overview of elliptic curve cryptography. Relatório técnico, IC-00-10, UNICAMP, 2000.

[ELM03] K. Eisentraeger, K. Lauter e P.L. Montgomery. Fast elliptic curve arithmetic and improved weil pairing evaluation. Em *Topics in Cryptology - CT-RSA'2003*, páginas 343–354, 2003.

[FKP+05] J. Franke, T. Kleijung, C. Paar, J. Pelzl, C. Priplata, M. Simka e C. Stahlke. An efficient hardware architecture for factoring integers with the elliptic curve method. Em *Workshop on Special-purpose Hardware for Attacking Cryptographic Systems - SHARCS 2005*, 2005.

[GB97] S. Goldwasser e M. Bellare. Lectures notes on cryptography. Summer Course "Cryptography and Computer Security"at MIT, 1997.

[Gon79] A. Gonçalves. *Introdução à Álgebra*. IMPA, 1979.

[Gra05] T. Granlund. GMP: The GNU Multiple Precision Arithmetic Library. Disponível *on-line* em março de 2006 na URL http://www.swox.com/gmp, 2005.

[Gro04] LiDIA Group. LiDIA: a C++ library for computational number theory. Disponível *on-line* em março de 2006 na URL http://www.informatik.tu-darmstadt.de/TI/LiDIA, 2004.

[GS96] S. Garfinkel e G. Spafford. *Practical Unix and Internet Security*. O'Reilly and Associates, 1996.

[HK05] B. Haible e R. Kreckel. CLN: Class Library for Numbers. Disponível *on-line* em março de 2006 na URL http://www.ginac.de/CLN, 2005.

[HMV04] D. Hankerson, A. J. Menezes e S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[Knu81] D. Knuth. Seminumerical algorithms. Em *The Art of Computer Programming*, volume 2. Addison-Wesley, 2ª edição, 1981.

[Len87] H. Lenstra. Factoring integers with elliptic curves. Em *Annals of Mathematics*, volume 126, páginas 649–673, 1987.

[LLMP90] H. K. Lenstra, J. H. W. Lenstra, M. S. Manasse e J. M. Pollard. The number field sieve. Em *Proc. 22nd Annual ACM Symp. On Theory of Computing*, páginas 564–572, 1990.

[Mao03] W. Mao. *Modern Cryptography: Theory and Practice*. Prentice Hall PTR, 2003.

[Men93] A. J. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.

[Mon87] P. L. Montgomery. Speeding the pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48, páginas 243–264, 1987.

[Mon94] P. L. Montgomery. A survey of modern integer factorization algorithms. *CWI Quarterly*, 7, páginas 337–366, 1994.

[MvOV97] A.J. Menezes, P.C. van Orschot e S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[np06] Wikipedia: The free encyclopedia. Disponível *on-line* em março de 2006 na URL http://en.wikipedia.org/wiki/Integer_factorization, 2006.

[PO96] R. Peralta e E. Okamoto. Faster factoring of integers of a special form. *IEICE Trans. Fundamentals*, 4, páginas 489–493, 1996.

[Pol74] J. Pollard. Theorems on factorization and primality testing. Em *Proceedings of the Cambridge Philosophical Society*, volume 76, páginas 521–528, 1974.

[Pom85] C. Pomerance. The quadratic sieve factoring algorithm. Em *Eurocrypt'84*, volume 209, páginas 169–182, 1985.

[Rei60] G. Reitwiesner. Binary arithmethic. *Advances in Computer*, 1, páginas 231–308, 1960.

[RSA77] R. L. Rivest, A. Shamir e L. Adleman. On digital signatures and public key cryptosystems. Relatório técnico, MIT/LCS/TR-212, MIT Lab. for Computer Science, 1977.

[Sch96] B. Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley Sons, 1996.

[Sho05a] V. Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2005.

[Sho05b] V. Shoup. NTL: A Library for Doing Number Theory. Disponível *on-line* em março de 2006 na URL http://www.shoup.net/ntl, 2005.

[Sil86] J. H. Silverman. *The Arithmetic of Elliptic Curves*. Springer-Verlag, 1986.

[SJ93] R.D. Silverman e S.S. Wagstaff Jr. A practical analysis of the elliptic curve factoring algorithm. *Mathematics of Computation*, 61, páginas 445–462, 1993.

[ST92] J. H. Silverman e J. Tate. *Rational Points on Elliptic Curves*. Springer-Verlag, 1992.

[Web95] K. Weber. The accelerated gcd algorithm. *ACM Transactions on Mathematical Software*, páginas 111–122, 1995.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo