UNIVERSIDADE FEDERAL DE GOIÁS – UFG

CAMPUS CATALÃO – CaC

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO – DCC

Bacharelado em Ciência da Computação

Projeto Final de Curso

Curvas Elípticas: Aplicações Criptográficas

Autor: Leonardo Garcia Marques

Orientador: Ivan da Silva Sendin

Catalão - 2007

Leonardo Garcia Marques

Curvas Elípticas: Aplicações Criptográficas

Monografia apresentada ao Curso de
Bacharelado em Ciência da Computação da
Universidade Federal de Goiás Campus Catalão
como requisito parcial para obtenção do título de
Bacharel em Ciência da Computação

Área de Concentração: Teoria da Computação
Orientador: Ivan da Silva Sendin

Catalão - 2007

G. Marques, Leonardo

Curvas Elípticas: Aplicações Criptográficas/ Ivan da Silva Sendin - Catalão - 2007

Número de páginas: 94

Projeto Final de Curso (Bacharelado) Universidade Federal de Goiás, Campus Catalão, Curso de Bacharelado em Ciência da Computação, 2007.

Palavras-Chave: 1. Criptografia Computacional. 2. Complexidade de Algoritmos. 3. Curvas Elípticas

Leonardo Garcia Marques

Curvas Elípticas: Aplicações Criptográficas

Monografia apresentada e aprovada em _____ de _____
Pela Banca Examinadora constituída pelos professores.

_____

Ivan da Silva Sendin – Presidente da Banca

_____

Liliane do Nascimento Vale

_____

Acrísio José do Nascimento Júnior

A Arlindo Honorato dos Santos

In memoriam

AGRADECIMENTOS

Agradeço primeiramente a Deus, que criou as condições necessárias para que eu pudesse desempenhar o meu papel e aprender as coisas que aprendi.

A minha mãe, que nesses anos de estudos aceitou e compreendeu a idéia de conviver com um filho ausente e à minha irmã Adriana, sempre presente e prestativa. Eu nada teria conseguido sem o apoio delas. Agradeço também à família de meu pai (antes tão distante e hoje tão próxima) pelas coisas que me ensinaram.

Aos meus amigos de faculdade Thiago de Paulo, Alex, Brener, Thiago Alves, Osias, Rodiney, Eduardo Castilho. O convívio ao longo desses anos sedimentou as melhores amizades que alguém pode fazer ao longo de um curso universitário. A todos também que fizeram ou ainda fazem parte das turmas de 2003 e 2004. Meus sinceros agradecimentos a todos.

Ao Professor Ivan, meu orientador, que com competência, paciência e amizade me orientou nos trabalhos deste projeto final de curso.

A Lidiane Alves, amiga de longa data, pela fé e pelo apoio que sempre manifestou.

“O que procuraste em ti ou fora de

teu ser restrito e nunca se mostrou,
mesmo afetando dar-se ou se rendendo,
e a cada instante mais se retraindo,

olha, repara, ausculta: essa riqueza
sobrante a toda pérola, essa ciência
sublime e formidável, mas hermética,

essa total explicação da vida,
esse nexo primeiro e singular,
que nem concebes mais, pois tão esquivo

se revelou ante a pesquisa ardente
em que te consumiste... vê, contempla,
abre teu peito para agasalhá-lo.”

(*Carlos Drummond de Andrade*)

RESUMO

Marques, L. Curvas Elípticas: Aplicações Criptográficas. Curso de Ciência da Computação, Campus Catalão, UFG, Catalão, Brasil, 2007, 94p.

Uma das principais formas de garantir a segurança da informação é por meio da utilização de criptografia. Entre os sistemas de chave pública atuais, o sistema de curvas elípticas tem atraído a atenção tanto do meio acadêmico quanto empresarial, dada a sua característica de oferecer um alto nível de segurança, mesmo se utilizando de valores de chave considerados pequenos. Neste trabalho é apresentada uma análise dos aspectos matemáticos e computacionais de tais sistemas. A descrição e a análise de seus principais algoritmos são apresentadas e uma implementação é mostrada no final, a fim de demonstrar os aspectos teóricos tratados ao longo do trabalho.

Palavras-Chaves: Criptografia Computacional, Complexidade de Algoritmos, Curvas Elípticas

Sumário

1 Introdução **1**

1.1 Contexto 1

1.2 Objetivos 2

1.3 Estrutura do trabalho 2

I Conceitos Fundamentais **3**

2 Algoritmos e Aspectos Matemáticos **4**

2.1 Introdução 4

2.2 Sobre algoritmos e suas análises 4

2.2.1 Analisando algoritmos 5

2.3 Noções gerais de álgebra moderna 6

2.3.1 Grupos 6

2.3.2 Anéis 9

2.3.3 Corpos 9

2.4 Considerações finais 13

3 Curvas Elípticas **14**

3.1 Introdução 14

3.1.1 Um Exemplo Prático 14

3.2 Caracterização de uma Curva Elíptica 16

3.2.1 Simplificando a Equação de Weiestrass 17

3.3 Lei de Grupo Sobre um Conjunto de Pontos de uma Curva Elíptica 20

3.3.1 Abordagem Geométrica 21

3.3.2 Abordagem Algébrica 21

3.3.3 A ordem de um grupo de pontos de uma curva elíptica 26

3.4 Multiplicação escalar de pontos de uma curva 27

3.4.1 Uma primeira abordagem: o método binário 27

3.4.2 Forma não-adjacente e sua utilização na multiplicação de pontos 28

3.5 Considerações finais 30

4 Criptografia **31**
4.1 Introdução 31
4.2 Conceitos Fundamentais 32
4.2.1 Entidades de comunicação e o modelo adversário 32
4.2.2 Definições 32
4.3 Sistemas Simétricos 34
4.3.1 Fundamentos 34
4.3.2 Problemas com a Administração de Chaves 35
4.4 Criptografia de Chave Pública 36
4.4.1 Protocolo de Troca de Chaves 36
4.4.2 O RSA 37
4.4.3 El Gamal 38
4.5 Assinatura Digital 39
4.5.1 Descrição 39
4.6 Considerações finais 40

II Aplicações Criptográficas **41**

5 Criptossistemas de Curvas Elípticas **42**
5.1 Introdução 42
5.2 Transformando uma texto em pontos de uma curva 43
5.3 Parâmetros públicos 43
5.4 Protocolo de troca de chaves – ECDH 44
5.5 Analogia ao ElGamal 45
5.6 Considerações finais 45

6 Ataques ao Problema do Logaritmo Discreto **47**
6.1 Introdução 47
6.1.1 Definições preliminares 47
6.2 Algoritmo de Shanks 48
6.3 Método ρ de Pollard 49
6.4 Algoritmo index calculus: descrição matemática e seus efeitos 53
6.4.1 Descrição geral 54
6.4.2 Index calculus aplicado em $\mathbb{Z}_p$ 55
6.4.3 Index Calculus em grupos de pontos de uma curva elíptica 56
6.5 Ataques por isomorfismo 57

6.5.1 Vulnerabilidade de curvas anômalas primas 58
6.6 Considerações finais 58

III Considerações Finais **59**

7 Implementações **60**
7.1 Introdução 60
7.1.1 Detalhes de implementação 60
7.2 Manipulando inteiros grandes 60
7.3 Aritmética dos corpos 61
7.3.1 A escolha do tipo de corpo 61
7.4 Aritmética das curvas 62
7.5 Manipulação de pontos de uma curva 63
7.5.1 Uma comparação entre os métodos gerais de multiplicação escalar . . . 63
7.6 Exemplificação de um protocolo criptográfico 65
7.6.1 Sistema análogo ao ElGamal 65
7.7 Ataque ao problema do logaritmo discreto: método ρ de Pollard 67
7.8 Considerações finais 68

8 Conclusões **69**
8.1 Trabalhos futuros 69

Referências **70**

Apêndices **72**

A Fundamentos Matemáticos **73**
A.1 Relações e Funções 73
A.1.1 Relações 73
A.1.2 Funções 74
A.2 Aritmética Básica 74
A.2.1 Divisibilidade 74
A.2.2 Números Primos 75
A.2.3 Congruência 76

B Código Fonte **78**
B.1 As classe fundamentais 78
B.1.1 A classe `Fp` 78
B.1.2 A classe `EC` 82

B.1.3 A classe `Ponto` 84
B.2 Esquemas de ciframento e deciframento 89
B.2.1 A classe `ElGamal` 89
B.2.2 A classe `ParDeChaves` 90
B.2.3 A classe `Criptograma` 91
B.2.4 A classe `ExemplificaElGamal` 91
B.3 Ataque ao problema do logaritmo discreto 92
B.3.1 A classe `ECDLP` 92
B.3.2 A classe `TestaMetodoRho` 94

Lista de Figuras

3.1 Curva $y^2 = x^3 - x$ 17
3.2 Curva $y^2 = x^3 + \frac{1}{4}x + \frac{5}{4}$. 17
3.3 Soma dos pontos P e Q, em uma curva elíptica: $P + Q = R$. 21
3.4 Duplicação do ponto P em uma curva elíptica: $P + P = R$. 22
3.5 Curva $y^2 = x^3 - 8x + 4$, definida sobre $\mathbb{Z}_{11}$ [BARRETO (1999)]. 22

7.1 $a = 2^{(t-1)W}A[t-1] + \cdots 2^{2W}A[2] + 2^{W}A[1] + A[0]$ 61
7.2 Comparação entre os tempos de execução dos métodos de multiplicação binário e o método binário com a forma não-adjacente para valores do tipo 2^n. Os dois algoritmos apresentam um desempenho semelhante, sendo o que se utiliza da FNA um pouco mais eficiente. 65
7.3 Comparação entre os tempos de execução dos métodos de multiplicação binário e o método binário com a forma não-adjacente para valores do tipo $2^n - 1$. O algoritmo que faz uso da forma não adjacente tem um desempenho muito superior nesse caso. 66
7.4 Tempo comparação entre os tempos de execução dos métodos me multiplicação binário e binário com a forma não-adjacente quando aplicados a inteiros quaisquer. Observa-se o bom desempenho do segundo algoritmo, mesmo no caso médio. 66

Lista de Tabelas

4.1 Alfabeto e seus valores numéricos associados. 34
4.2 Nova configuração do alfabeto para $k = 3$. 34

6.1 Valores dos passo intermediários do Algoritmo 6.2, aplicado ao Exemplo 6.2 [MENEZES et al. (1997)]. 51
6.2 Valores dos passo intermediários do Algoritmo 6.3, aplicado ao Exemplo 6.3 [HANKERSON et al. (2004)]. 54

Lista de Algoritmos

3.1 Método simples de multiplicação escalar 27
3.2 Método binário de multiplicação 28
3.3 Cálculo da forma não-adjacente para um inteiro positivo 29
3.4 Método binário com FNA para multiplicação de pontos 30
4.1 Protocolo de Diffie e Helman 36
4.2 Geração do par de chaves no RSA 37
4.3 Esquema básico de ciframento no RSA 38
4.4 Esquema básico de deciframento no RSA 38
4.5 Geração da chave pública no esquema de ElGamal 39
4.6 Ciframento por ElGamal 39
4.7 Deciframento por ElGamal 39
5.1 ElGamal Elíptico: Ciframento 45
5.2 ElGamal Elíptico: Deciframento 45
6.1 Algoritmo de Shanks 48
6.2 Método ρ de Pollard para o cálculo do logaritmo discreto. 50
6.3 Método ρ de Pollard adaptado para grupos de curvas elípticas. 52
6.4 Index Calculus 55

Capítulo 1

Introdução

1.1 Contexto

O atual estágio de desenvolvimento das telecomunicações tem exigido que mecanismos cada vez melhores sejam propostos, a fim de garantir a segurança da informação. Um dos métodos mais antigos utilizados com esse objetivo é a criptografia.

A criptografia objetiva ocultar o conteúdo semântico de uma dada informação e garantir tanto o sigilo quando a autenticidade de dados transmitidos. Com o advento dos computadores eletrônicos, diversos sistemas foram propostos. O mais revolucionário entre eles é o sistema de chave pública.

Todo sistema de chave pública garante sua segurança em algum problema matemático. O conhecido RSA, por exemplo, baseia sua segurança na aparente dificuldade de se fatorar inteiros grandes. Outro sistema existente é o de curvas elípticas.

Pesquisas relativas a Curvas Elípticas têm sido feitas por estudiosos em Teoria dos Números desde os últimos anos do século XIX. Recentemente, em 1985, dois pesquisadores em Ciência da Computação, Neal Koblitz [KOBLITZ (1987)] e Victor Miller [MILLER (1986)], propuseram de forma independente um esquema criptográfico de chave pública que se baseia no uso de grupos de Curvas Elípticas sobre corpos finitos. Tal esquema é também conhecido como ECC (*Elliptic Curve Cryptography*)

Esse sistema tem atraído a atenção tanto do meio acadêmico quanto do meio empresarial por se revelar uma atraente alternativa a sistemas tradicionais, tais como RSA [STINSON (1995)] e DSA [HANKERSON et al. (2004)]. Isso se deve principalmente ao fato de tais esquemas – cujo artifício matemático empregado para prover segurança é o problema da fatoração de inteiros e o problema do logaritmo discreto, respectivamente – já contarem com algoritmos capazes de encontrar soluções em tempo sub-exponencial, ao passo que sistemas baseados em Curvas Elípticas – que se baseia no Problema do Logaritmo Discreto em Curvas Elípticas (ECDLP) – conta somente com algoritmos exponenciais, em suas soluções mais eficientes

[BONEH & LIPTON (1996)]. Tal característica permite o uso de parâmetros menores, quando comparados aos parâmetros utilizados em algoritmos como o RSA, ainda proporcionando o mesmo nível de segurança. Por exemplo, a segurança fornecida pelo algoritmo RSA, usando uma chave de tamanho 1024 bits é equivalente à segurança fornecida por Sistemas de Curvas Elípticas com chave de 160 bits [HANKERSON et al. (2004)].

Essa característica permite que tal técnica seja usada em meios onde os recursos (poder de processamento, espaço de armazenamento, dentre outros) são limitados. Esses meios podem ser representados por dispositivos como celulares, cartões inteligentes e aparelhos similares.

1.2 Objetivos

Este trabalho tem por objetivo analisar as propriedades matemáticas e computacionais dos sistemas criptográficos de chave pública, concentrando-se nos sistemas de curvas elípticas. Para tanto, são apresentados os principais conceitos teóricos que são de fundamental importância ao bom entendimento do assunto.

As definições formais de sistemas simétricos e assimétricos são apresentados, assim como a formalização da estrutura de curvas elípticas e o modo como essas podem e são utilizadas no campo da criptografia. Os principais ataques são analisados e ao final é apresentada uma implementação, que tem por objetivo ilustrar os conceitos teóricos apresentados ao longo do trabalho.

1.3 Estrutura do trabalho

Este trabalho está dividido em oito capítulos. O Capítulo 2 faz um levantamento das principais propriedades no campo da álgebra moderna que são indispensáveis à compreensão da aritmética das curvas elípticas. Noções de análise e complexidade de algoritmos também são apresentadas. O Capítulo 3 formaliza a definição de curvas elípticas e explica a aritmética de sua lei de grupos. É mostrada também a evolução na confecção de algoritmos eficientes para a operação de multiplicação de pontos por um valor escalar. O Capítulo 4 conceitua a criptografia computacional e analisa os principais sistemas criptográficos. O Capítulo 5 demonstra de que maneira alguns sistemas mais antigos podem ser modificados, a fim de se utilizarem das propriedades de curvas elípticas. No Capítulo 6 são analisadas as principais abordagens de ataque ao problema do logaritmo discreto. No Capítulo 7 são apresentadas algumas implementações que visam ilustrar e analisar as principais propriedades desse criptossistema. Por fim, no Capítulo 8 são apresentadas as conclusões deste trabalho e sugestões para trabalhos futuros. Definições matemáticas mais elementares e o código fonte das implementações encontram-se nos Apêndices A e B, respectivamente.

Parte I

Conceitos Fundamentais

Capítulo 2

Algoritmos e Aspectos Matemáticos

2.1 Introdução

A proposta deste trabalho é descrever matematicamente o sistema criptográfico de curvas elíptica e analisar seus principais algoritmos. Antes de tudo, porém, alguma atenção deve ser dada à noção de algoritmo e aos aspectos matemáticos mais relevantes ao contexto: a álgebra moderna.

Esse capítulo trata desses dois assuntos, oferecendo uma visão geral sobre as estruturas algébricas mais relevantes para o contexto em estudo e discutindo sobre as principais propriedades dos algoritmos.

2.2 Sobre algoritmos e suas análises

Um algoritmo consiste numa seqüencia finita de ações capazes de gerar um resultado [BRASSARD & BRA
Sua existência independe dos computadores — o que pode ser facilmente entendido quando se considera o "algoritmo de Euclides", um método para o cálculo do máximo divisor comum entre dois inteiros desenvolvido alguns séculos antes de Cristo.

Computacionalmente, um algoritmo transforma um valor (ou um conjunto de valores) de entrada em uma saída — ou em um conjunto de valores de saída. Ele pode assim ser visto como uma ferramenta utilizada para resolver problemas computacionais bem definidos, descrevendo não somente a entrada e a saída, mas também o relacionamento entre um e outro [CORMEN et al. (2001)].

Um algoritmo é correto quando retorna uma resposta dentro do conjunto de respostas esperados. Mas, curiosamente, alguns algoritmos que nem sempre apresentam uma resposta inteiramente correta são também úteis em aplicações práticas. É o caso dos algoritmos probabilísticos, muitos dos quais são utilizados para aplicações criptográficas.

2.2.1 Analisando algoritmos

Se o algoritmo executa a tarefa, retornando uma resposta aceitável, ele é considerado eficaz. Outra forma de avaliar um algoritmo é a sua eficiência, que consiste em medir a quantidade de recursos que ele consome para realizar a tarefa para a qual foi designado. Na maioria das vezes, o "tempo de execução" é avaliado, mas a quantidade de processos gerados ou a quantidade de memória consumida podem também serem utilizadas como métrica.

De maneira geral, a avaliação é feita tomando como parâmetro principal o tamanho da entrada. Em algoritmos destinados a ordenar vetores, por exemplo, o tamanho da entrada consiste da quantidade de elementos que o vetor possui. Em criptografia também, o tamanho da entrada consiste na quantidade de bits necessária para representar os números inteiros com os quais se trabalha. Se tal valor é o inteiro n, então a o tamanho da entrada corresponderá a $l = \lfloor \log_2 n \rfloor + 1$.

Quando se analisa um algoritmo, é comum se utilizar da pior situação possível que ele pode assumir. Isso é chamado "pior caso". Por exemplo, se for considerado o tempo gasto para encontrar um fator primo de um inteiro n é, no pior caso, $\sqrt{n}$. Como $\lfloor \log_2 n \rfloor + 1 = l$, então $n = 2^l$. Assim, o tempo corresponde a $\sqrt{2^l}$, o que pode ser reescrito como $2^{l/2}$.

O estudo detalhado do tempo de execução de pode ser tornar extremamente trabalhoso se todos os detalhes referentes ao procedimento em análise for considerado. Na prática, parte desses detalhes pode ser omitidos, utilizando-se os conceitos de "notação assintótica". O comportamento que um algoritmo assume mediante uma entrada fornecida pode ser estudado como uma função. A ferramenta mais convenientemente usada é a notação $\mathcal{O}$, que define um limite inferior.

Definição 2.1 *Uma função $f(n)$ é membro de $\mathcal{O}(g(n))$, o que se indica por $f(n) = \mathcal{O}(g(n))$, se existirem constantes positivas c e n_0 tais que $0 \leq f(n) \leq cg(n)$, para todo $n \geq n_0$.*

Se a análise do comportamento de um algoritmo indicar algo do tipo $f(n) = 5n^2 + 2n + 1$, por exemplo, a notação $\mathcal{O}$ pode ser empregada: as constantes multiplicativas e os termos de menor grau podem ser desprezados e a função é considerada $\mathcal{O}(n^2)$.

De maneira mais simplificada, os problemas podem ser divididos em duas classe principais: os tratáveis e os intratáveis. Os *problemas tratáveis*, que podem ser facilmente resolvidos por uma máquina que emprega a tecnologia atual, e os *problemas intratáveis*, cuja solução é considerada inviável. Esses últimos podem receber ainda uma subdivisão, que os agrupa em problemas *exponenciais* e *sub-exponenciais* (esses últimos possuem um nível de dificuldade um pouco menor).

2.3 Noções gerais de álgebra moderna

O sistema criptográfico estudado neste trabalho efetua suas operações sobre grupos de pontos pertencentes a uma curva elíptica. Tais curvas são definidas sobre corpos finitos. A aritmética desses grupos de pontos será mostrada com maiores detalhes no Capítulo 3. Antes porém, faz-se necessário o entendimento dos conceitos matemáticos de grupos e corpos: suas principais definições e propriedades.

Esse seção faz um levantamento dos temas mais relevantes ao entendimento deste trabalho no que tange à álgebra moderna. Muitos dos resultados elementares de aritmética, omitidos aqui, são encontrados no Apêndice A.

2.3.1 Grupos

Às vezes é útil atribuir aos elementos de um conjunto um comportamento específico. Isso pode ser feito por meio de uma lei de formação. As operações binárias executam esse papel.

Definição 2.2 (Operação Binária) *Dado um conjunto não vazio G, defini-se uma operação binária em G como um função qualquer $*$ de $G \times G$ em G, isto é,*

$$* : G \times G \to G$$
$$(g_1, g_2) \longmapsto g_1 * g_2$$

Quando uma operação binária apresenta certos comportamentos, o par conjunto-operação recebe o nome de grupo, conforme é apresentado na próxima definição:

Definição 2.3 (Grupo) *Defini-se grupo como sendo um conjunto G, juntamante com uma operação binária $*$, onde se verificam as seguintes propriedades:*

1. $\forall a, b, c \in G,\ a * (b * c) = (a * b) * c$ *(propriedade associativa)*
2. $\exists e \in G$ *tal que* $\forall a \in G,\ a * e = a = e * a$ *(existênica do elemento identidade)*
3. $\forall a \in G,\ \exists a' \in G$ *tal que* $a * a' = e = a' * a$ *(elemento inverso)*

Por simplicidade de notação, será dito simplesmente que G é um grupo.

Se o grupo descrito na Definição 2.3 apresentar também a propriedade comutativa, ou seja, $\forall a, b \in G, a * b = b * a$, este grupo é chamado **abeliano**. Este texto foca suas atenções em grupos deste tipo.

O elemento neutro de um grupo abeliano é único. De acordo com a definição de grupo, se e e e' fossem ambos elementos neutros, então valeria a propriedade $e = e * e' = e'$, ou seja, eles

seriam iguais. O inverso de cada elemento é também único e sua unicidade parte também da definição. Se a' e a'' são ambos inversos de um elemento a, então

$$\begin{aligned} a' &= a' * e \\ &= a' * (a * a'') \\ &= (a' * a) * a'' \\ &= e * a'' \\ &= a'', \end{aligned}$$

o que confirma a unicidade do inverso do elemento inverso.

Exemplos bem comuns de grupos são dados a seguir.

Exemplo 2.1 O conjunto $\mathbb{Z}$, dos números inteiros, sob a operação de adição.

Exemplo 2.2 O conjunto $\mathbb{Z}_n$, dos inteiros pertencentes ao intervalo $[0, n-1]$, sob a operação de soma módulo n

Embora os Exemplos 2.1 e 2.2 referenciem conjuntos numérico, isso não é uma exigência. Os Exemplos 2.3 e 2.4 denotam grupos de elementos diversos.

Exemplo 2.3 Se G é um conjunto de cadeias de bits de comprimento de n, onde se aplica uma operação de "ou-exclusivo", este conjunto e esta operação figuram um grupo abeliano, onde a cadeia formada exclusivamente por zeros é o elemento neutro e o inverso de cada cadeia é ela própria [SHOUP (2005)].

Exemplo 2.4 Se G for o conjunto dos naipes de cartas de baralho, ou seja, $G = \{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}$, pode-se estabelecer uma operação $*$, definida por

$*$	$\clubsuit$	$\diamondsuit$	$\heartsuit$	$\spadesuit$
$\clubsuit$	$\clubsuit$	$\diamondsuit$	$\heartsuit$	$\spadesuit$
$\diamondsuit$	$\diamondsuit$	$\spadesuit$	$\clubsuit$	$\heartsuit$
$\heartsuit$	$\heartsuit$	$\clubsuit$	$\spadesuit$	$\diamondsuit$
$\spadesuit$	$\spadesuit$	$\heartsuit$	$\diamondsuit$	$\clubsuit$

Analisando a tábua de operações é possível perceber que ela denota um grupo abeliano.

De maneira geral, a operação $*$ é substituída pelos símbolos $+$ e $\times$, dado origem às notações *aditiva* e *multiplicativa*, respectivamente.

Em notação aditiva, o elemento neutro e é denotado por 0. Além disso, o inverso de a é $-a$, $a - b$ representa $a + (-b)$ e na significa $a + \cdots + a$, onde a aparece n vezes.

De maneira semelhante, em notação multiplicativa tem-se 1 como elemento neutro. O inverso de a é a^{-1}, a/b representa $a \times b^{-1}$ e a^n significa $a \times \cdots \times a$, onde a aparece n vezes.

Definição 2.4 (Subgrupo) *Se G é um grupo, $H \subseteq G$ e H é um grupo quando se aplica a mesma lei de formação de G, então H é chamado subgrupo de G.*

Um grupo pode possuir uma quantidade finita de elementos, situação onde é conhecido por grupo finito, ou infinita, sendo um grupo infinito. Aplica-se assim o conceito de ordem.

Definição 2.5 (Ordem de um Grupo) *Dado um grupo finito G, chama-se ordem (e representa-se por $|G|$) a quantidade de elementos (ou cardinalidade) do conjunto subjacente. A ordem de um elemento $g \in G$ é o menor inteiro $k > 0$ tal que $k * g = e$. O subgrupo gerado por g é denotado por $\langle g \rangle$.*

Sendo k a ordem de um elemento g do grupo G e assumindo (por simplicidade, mas sem perda de generalidade) a notação multiplicativa, verifica-se que

$$g^i = g^i \iff i \equiv j \pmod{k}, \tag{2.1}$$

o que serve de base para a o Teorema de Lagrange.

Teorema 2.1 (Lagrange) *Seja G um grupo finito.*

1. *Seja H um subgrupo de G. Então, a ordem de H divide a ordem de G.*
2. *Seja $g \in G$. Então a ordem de g divide a ordem de G.*

Definição 2.6 *Seja G um grupo. Se todos os elementos de G puderem ser gerados por um elemento $g \in G$, então esse é um grupo cíclico e g recebe o nome de gerador.*

Exemplo 2.5 O comportamento de um grupo finito cíclico pode ser analisado com o grupo descrito no Exemplo 2.4. Quando o elemento $\diamondsuit$ é operado consigo mesmo, observa-se que

$$\begin{aligned}
\diamondsuit &= \diamondsuit \\
\diamondsuit * \diamondsuit &= \spadesuit \\
\diamondsuit * \diamondsuit * \diamondsuit &= \heartsuit \\
\diamondsuit * \diamondsuit * \diamondsuit * \diamondsuit &= \clubsuit \\
\diamondsuit * \diamondsuit * \diamondsuit * \diamondsuit * \diamondsuit &= \diamondsuit \\
\diamondsuit * \diamondsuit * \diamondsuit * \diamondsuit * \diamondsuit * \diamondsuit &= \spadesuit \\
\diamondsuit * \diamondsuit * \diamondsuit * \diamondsuit * \diamondsuit * \diamondsuit * \diamondsuit &= \heartsuit \\
\vdots &= \vdots
\end{aligned}$$

À medida que $\diamondsuit$ é operado consigo mesmo, um elemento diferente do grupo é determinado. Isso define o comportamento de um grupo cíclico, onde o elemento $\diamondsuit$ é seu gerador.

Teorema 2.2 *Se G é um grupo abeliano de ordem prima, então G é cíclico.*

Prova Seja $|G| = p$ e a um elemento qualquer de G. Pelo Teorema 2.1, $|a|$ divide p. Mas, por hipótese, p é primo logo. Assim, há duas possibilidades:

- $|a| = 1$, que ocorre somente se a for o elemento neutro;
- $|a| = p$, ou seja, a gera todos os elementos de G.

Portanto, G é cíclico. ■

Teorema 2.3 *Todos os subgrupos de um grupo cíclico são também cíclicos.*

2.3.2 Anéis

Definição 2.7 *Seja o conjunto R, juntamente com duas operações: $+$ (adição) e $\times$ (multiplicação). A terna $(R, +, \times)$ constitui um anel se*

1. *R e a operação $+$ formam um grupo abeliano com identidade representada por 0.*
2. *A operalção $\times$ é associativa.*
3. *A operação $\times$ é distibutiva sobre $+$.*

Se a multiplicação for comutativa, ou seja, $a \times b = b \times a, \forall a, b \in R$, então o anel é chamado comutativo. Caso exista a identidade multiplicativa, isto é, para todo $a \in R$, existe a' tal que $a \cdot a' = 1$, então o anel é dito ser com unidade.

2.3.3 Corpos

Em termos práticos, uma das noções algébricas mais relevantes ao contexto deste trabalho é a noção de corpo. Esta seção o define e discute sua aritmética, com base nos conceitos apresentados até aqui.

Definição 2.8 *Se um anel comutativo onde todos os elementos não nulos possuem inversos multiplicativo, então ele é chamado corpo e será denotado por $\mathbb{F}$.*

Definição 2.9 *O número mímimo para m tal que $\sum_{i=1}^{m}$ (considerando que este cálculo é efetuado sobre um corpo) é chamado característica deste corpo. Caso não exista um inteiro m que satisfaça este somatório, o corpo é dito ter característica 0.*

Exemplo 2.6 O conjunto dos números inteiros é um corpo com característica zero.

A subtração em $\mathbb{F}$ é definida em função da adição, pois se $a, b \in \mathbb{F}$, então $a - b = a + (-b)$, onde $-b$ é o *oposto* de b. De maneira análoga é definida a divisão: se $a, b \in \mathbb{F}$, então $a/b = a \cdot b^{-1}$, onde b^{-1} é o *inverso* de b.

Assim como em grupos, o conceito de ordem aplica-se também a corpos.

Definição 2.10 *A ordem de um corpo $\mathbb{F}$ é seu número de elementos. Isto é denotado por $|\mathbb{F}|$.*

Há um condição de existência para um dado corpo de ordem q

Teorema 2.4 *Existe um corpo de ordem q se, e somente se, $q = p^m$, onde p é um primo e $m > 0$.*

A característica do corpo $\mathbb{F}_q$ corresponde ao valor p. Além disso, dependendo do valor de m, o corpos podem ser classificados em três categorias principais: corpos primos, corpos binários e corpos de extensão.

Corpos Primos

São corpos do tipo $\mathbb{F}_p$, ou seja, onde $m = 1$. O conjunto de $\mathbb{F}_p$ é $\{0, 1, ...p - 1\}$ e as operações (multiplicação e adição) são executadas módulo p.

Exemplo 2.7 Por definição, $\mathbb{F}_{13}$ é um corpo primo. Os elementos de $\mathbb{F}_{13}$ são $\{0, 1, 2, ..., 11, 12\}$. Exemplos de operações aritméticas possíveis são:

- *Adição:* $10 + 4 = 1$, pois $14 \bmod 13 = 1$.
- *Subtração:* $5 - 10 = 8$, pois $-5 \bmod 13 = 8$
- *Multiplicação:* $2 \cdot 8 = 3$, pois $16 \bmod 13 = 3$
- *Inversão:* $3^{-1} = 9$, pois $3 \cdot 9 \bmod 13 = 1$

Corpos Binários

Corpos binários, ou de característica 2 ($\mathbb{F}_{2^m}$, $m > 1$). É usual representar os elementos de $\mathbb{F}_{2^m}$ como um polinômio de grau menor que m e com os coeficientes pertencentes ao corpo $\mathbb{F}_{2^m} = \{0, 1\}$. Logo,

$$\mathbb{F}_{2^m} = \{a_{m-1}z^{m-1} + a_{m-2}z^{m-1} + \cdots + a_2z^2 + a_1z + a_0 : a_i \in \{0, 1\}\}. \tag{2.2}$$

A representação polinomial dos elementos de $\mathbb{F}_{2^m}$ permite que eles sejam reprentados tal qual uma *string* de bits de comprimento máximo m: o polinômio z^2+1, pertencente a $\mathbb{F}_{2^4}$, pode ser representado por (0101). Observa-se, assim, que a adição de dois elementos de um corpo, que corresponde à adição usual de polinômios, mas com toda aritmética de seus coeficientes

sendo executada módulo dois, pode ser executada como uma operação ou-exclusivo sobre as duas *strings* de bits. Observa-se ainda que adição e subtração são equivalentes, uma vez que $-1 = 1 \pmod 2$.

A multiplicação polinomial é também executada da maneira usual, com os devidos cuidados aritméticos, naturalmente. Para que se possa sempre obter polinômios de grau menor que m, é utilizado um polinômio $f(z)$, de grau m e irredutível. Sempre que a multiplicação de dois elementos de $\mathbb{F}_{2^m}$ resultar em um polinômio com grau maior ou igual a m, é determinado o resto da divisão deste com o polinômio redutor $f(z)$, para que o resultado esteja em $\mathbb{F}_{2^m}$.

Exemplo 2.8 O corpo $\mathbb{F}_{2^4}$ possui os seguinte elementos (em notação polinomial):

$$\begin{array}{llll} 0 & z^2 & z^3 & z^3+z^2 \\ 1 & z^2+1 & z^3+1 & z^3+z^2+1 \\ z & z^2+z & z^3+z & z^3+z^2+z \\ z+1 & z^2+z+1 & z^3+z+1 & z^3+z^2+z+1 \end{array}$$

Pode-se estabelecer o polinômio redutor $f(z) = z^4 + z + 1$ para as operações deste corpo. Sejam $z^3+z^2+1, z^2+z+1 \in \mathbb{F}_{2^4}$. Com estes valores, são mostrados exemplos de operações em $\mathbb{F}_{2^m}$:

- *Adição:*

$$\begin{aligned} (z^3+z^2+1)+(z^2+z+1) &= z^3+z^2+z^2+z+1+1 \\ &= z^3+2z^2+z+2 \\ &= z^3+0z^2+z+0 \\ &= z^3+z \end{aligned}$$

- *Subtração:*

$$\begin{aligned} (z^3+z^2+1)-(z^2+z+1) &= z^3+z^2+1-z^2-z-1 \\ &= z^3+z^2-z^2-z+1-1 \\ &= z^3+0z^2-z+0 \\ &= z^3+0z+z+0 \\ &= z^3+z \end{aligned}$$

- *Multiplicação:*

$$\begin{aligned} (z^3+z^2+1)\cdot(z^2+z+1) &= z^5+z^4+z^3+z^4+z^3+z^2+z^2+z+1 \\ &= z^5+2z^4+2z^3+2z^2+z+1 \\ &= z^5+0z^4+0z^3+0z^2+z+1 \\ &= z^5+z+1 \end{aligned}$$

Como o polinômio resultante tem grau igual a 4, deve-se calcular o resto da divisão dele com o polinômio redutor:

$$(z^5 + z + 1) \bmod (z^4 + z + 1) = z^2 + 1$$

- *Inversão:* $(z^3 + z^2 + 1)^{-1} = z^2$, uma vez que

$$(z^3 + z^2 + 1) \cdot z^2 \bmod (z^4 + z + 1) = 1$$

Neste exemplo, é possível constatar a propriedade $-a = a, \forall a \in \mathbb{F}_{2^m}$.

Corpos de Extensão

Se $m > 1$ e p é um primo maior que 2, então o corpo é chamado corpo de extensão. As proprieades básicas desta entidade matemática é similar às vistas em corpos binários. A diferença está na forma com que a aritmética dos coeficientes é definida sobre o corpo — módulo p, no caso.

A representação dos elementos deste tipo de corpo pode também ser representada em notação polinomial:

$$\mathbb{F}_{p^m} = \{a_{m-1}z^{m-1} + a_{m-2}z^{m-1} + \cdots + a_2z^2 + a_1z + a_0 : a_i \in \mathbb{F}_p\}. \tag{2.3}$$

Também neste caso, um polinômio redutor (e irredutível) é definido para cada corpo.

Exemplo 2.9 O corpo de extensão $\mathbb{F}_{251^5}$ pode utilizar o polinômio $f(z) = z^5 + z^4 + 12z^3 + 9z^2 + 7$ como redutor. As operações são todas executadas módulo 251 e o resto da divisão entre o resultado e $f(z)$ é obtido sempre que ocorrer um polinômio com grau maior ou igual a 5. Assim, considerando $123z^4 + 76z^2 + 7z + 4$ e $196z^4 + 12z^3 + 225z^2 + 76$, ambos elementos de $\mathbb{F}_{251^5}$, tem-se:

- *Adição:* $(123z^4 + 76z^2 + 7z + 4) + (196z^4 + 12z^3 + 225z^2 + 76) = 68z^4 + 12z^3 + 50z^2 + 7z + 80$
- *Subtração:* $(123z^4 + 76z^2 + 7z + 4) - (196z^4 + 12z^3 + 225z^2 + 76) = 178z^4 + 239z^3 + 102z^2 + 7z + 80$
- *Multiplicação:* $(123z^4 + 76z^2 + 7z + 4) \cdot (196z^4 + 12z^3 + 225z^2 + 76) = 117z^4 + 151z^3 + 117z^2 + 182z + 217$
- *Inversão:* $(123z^4 + 76z^2 + 7z + 4)^{-1} = 109z^4 + 111z^3 + 250z^2 + 98z + 85$

2.4 Considerações finais

Este capítulo reuniu os principais conceitos necessários ao pleno entendimento deste trabalho. A idéia de grupo é necessária pelo fato de os sistemas criptográficos de curvas elípticas serem definidos sobre grupos de pontos de curvas deste tipo. Tais curvas são definidas sobre corpos finitos, sendo relevante a compreensão de sua aritmética.

Os assuntos tratados aqui são de nível introdutório. Há uma vasta literatura que cobre esse assunto, relacionando-o ao contexto da criptografia. Neal Koblitz, [KOBLITZ (1991)], por exemplo, associa bem os dois assuntos em sua obra. John K. Truss [TRUSS (1998)], Bernard Kolman e Robert C. Busby [KOLMAN & BUSBY (1987)] também trabalham com um enfoque semelhante, dentro da matemática discreta. Em especial, Victor Shoup [SHOUP (2005)] e Henri Cohen [COHEN (1993)] deram uma importante contribuição ao campo de estudo conhecido como *álgebra computacional*.

No que tange a análise de algoritmos, a obra de Thomas Cormen, Ronald L. Rivest (um dos criadores do RSA) e Charles E. Leiserson [CORMEN et al. (2001)] e a série *The art of computer programming*, de Donald Knuth [KNUTH (1997)] fornecem uma boa base ao entendimento de tais assuntos.

O próximo capítulo expande as idéias aqui discutidas, apresentando uma definição formal de curvas elípticas.

Capítulo 3

Curvas Elípticas

3.1 Introdução

O estudo de curvas elípticas tem chamado a atenção dos pesquisadores desde as últimas décadas do século XIX. Há pouco mais de dez anos, suas propriedades permitiram a Andrew Wiles provar o *Último Teorema de Fermat*[1], que ficara alguns séculos em aberto.

Além dos aspectos teóricos, a constatação da aparente dificuldade de se resolver o logaritmo discreto em grupos de pontos de uma curva elíptica tornou esta entidade matemática uma ferramenta útil para aplicações criptográficas, recebendo atenção não somente da comunidade acadêmica, mas também do meio empresarial [LÓPEZ & DAHAB (2000)].

Os assuntos relacionados aos protocolos criptográficos serão mostrados no Capítulo 5. O presente capítulo aborda as definições, as propriedades matemáticas e os resultados mais relevantes ao desenvolvimento de tais criptossistemas.

Antes, porém, é mostrada a resolução de um problema que, embora não esteja diretamente relacionado com criptografia, demonstra a utilização de curvas elípticas e explora algumas das propriedades que serão discutidas posteriormente.

3.1.1 Um Exemplo Prático

Uma certa quantidade de balas de canhão pode ser agrupada de maneira que forme uma pirâmide cuja base seja um quadrado. Por exemplo, pode-se ter uma bola no primeiro nível (topo), quatro no segundo nível, nove no terceiro e assim por diante. Uma questão que pode ser levantada é: será possível desmanchar esta pirâmide e reagrupar estas bolas de maneira que formem um quadrado?

[1] O Último Teorema de Fermat diz que a equação $a^n + b^n = c^2$ não tem solução no conjunto dos números inteiros para $n > 2$. Simon Singh [SINGH (2005)] conta de maneira fascinante toda a trajetória que precedeu à prova deste teorema, desde a morte de Fermat até a apresentação da demonstração por Wiles, passando pelos principais matemáticos que estudaram esse assunto.

Caso a pirâmide tenha quatro nível, ter-se-á $1+4+9+16 = 30$. Logo, com esta quantidade de bolas, não é possível formar um quadrado. É fácil ver, por indução finita, que

$$1^2 + 2^2 + 3^2 + \cdots + x^2 = \frac{x(x+1)(2x+1)}{6} \tag{3.1}$$

Como se deseja que a quantidade total de bolas forme um quadrado, tem-se:

$$y^2 = \frac{x(x+1)(2x+1)}{6} \tag{3.2}$$

A Equação 3.2 representa uma curva elíptica. Sua solução pode ser obtida através do método do método diofantino [WASHINGTON (2003)], que consiste em encontrar o novas soluções a partir de soluções já conhecidas. Nesse caso, identifica-se duas soluções que correspondem aos casos triviais: $(0, 0)$ (uma pirâmide sem nenhuma bola) e $(1, 1)$ (uma pirâmide composta por somente uma bola). Como a reta que contém esses dois pontos é definida por $y = x$, a intersecção entre essa reta e a curva pode ser obtido substituindo esse valor de y na Equação 3.2, obtendo-se:

$$x^2 = \frac{x(x+1)(2x+1)}{6} \tag{3.3}$$

cujo desenvolvimento resulta na igualdade

$$x^3 - \frac{3}{2}x^2 + \frac{1}{2}x = 0 \tag{3.4}$$

Como a Equação 3.4 é um polinômio de terceiro grau, é possível expressá-lo sob a forma fatorada $(x-a)(x-b)(x-c)$, desde que as raízes a, b e c sejam conhecidas.

O desenvolvimento da forma fatorada mostra que $(x-a)(x-b)(x-c) = x^3 - (a+b+c)x^2 + (ab+ac+bc)x - abc$, indicando que quando o coeficiente de x^3 é 1 (conforme acontece na Equação 3.4), o valor de $-(a+b+c)$, ou seja, o negativo da soma das raízes do polinômio, corresponde ao valor do coeficiente de x^2. Aplicando essa propriedade no caso em estudo, tem-se:

$$0 + 1 + x = \frac{3}{2} \Rightarrow \begin{cases} x = \dfrac{1}{2} \\ \\ y = \dfrac{1}{2} \end{cases} \tag{3.5}$$

Como os valores encontrados não correspondem a números inteiros, não podendo portanto serem considerados soluções válidas para o problema. Mas, como $(1/2, -1/2)$ também são pontos da curva (o que pode ser verificado pela simetria do gráfico de uma curva desse tipo, conforme será mostrado nas próximas seções), pode-se repetir o processo com os pontos $(1/2, -1/2)$ e $(1, 1)$. Desta vez, encontra-se $x = 24$ e $y = 70$, o que representa $1^2 + 2^2 + 3^2 + \cdots 24^2 = 70^2$

Uma estratégia semelhante a essa será utilizada no desenvolvimento da definição da lei de grupo sobre um conjunto de pontos de uma curva, que será mostrada ainda nesse capítulo.

3.2 Caracterização de uma Curva Elíptica

Definição 3.1 *Uma* curva elíptica E, sobre um corpo K *é definida pela equação*

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{3.6}$$

onde $a_1, a_2, a_3, a_4, a_6 \in K$. *Essa equação possui um discriminate* Δ *que deve ser diferente de zero e s definido por*

$$\left.\begin{aligned} \Delta &= -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6 \\ d_2 &= a_1^2 + 4a_2 \\ d_4 &= 2a_4 + a_1a_3 \\ d_6 &= a_3^2 + a_6 \\ d_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^4 \end{aligned}\right\} \tag{3.7}$$

Se L *é qualquer corpo de extensão de* K, *então o conjunto de pontos L-racionais em* E *é*

$$E(L) = \{(x, y) \in L \times L : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 = 0\} \cup \{\infty\}$$

onde ∞ *é um ponto no infinito.*

Algumas observações podem ser feitas sobre a Definição 3.1. A Equação 3.6 é conhecida como equação de Weierstrass. Dizer que essa equação está definida sobre um corpo K significa afirmar que seus coeficientes a_1, a_2, a_3, a_4, a_6 são elementos deste corpo. Isso pode ser expresso pela notação $E(K)$.

A definição aponta ainda a necessidade de o discriminante Δ ser diferente de zero. Essa condição garante a não existência de pontos na curva onde haja mais de uma reta tangente distinta. Curvas desta natureza são chamadas "suaves".

Por fim, ainda no que se refere à Definição 3.1, o conjunto de pontos L-racionais sobre uma curva E são aqueles cujas coordenadas x e y pertencem a um corpo de extensão L, de E. É importante enfatizar que o ponto ∞ encontra-se presente em todos os corpos de extensão L de K.

Exemplo 3.1 As curvas elípticas

$$E_1 : y^2 = x^3 - x$$

$$E_2 : y^2 = x^3 + \tfrac{1}{4}x + \tfrac{5}{4}$$

são exemplos de curvas definidas sobre o corpo dos números reais. Seus gráficos são apresentados nas Figuras 3.2 e 3.2, respectivamente [HANKERSON et al. (2004)].

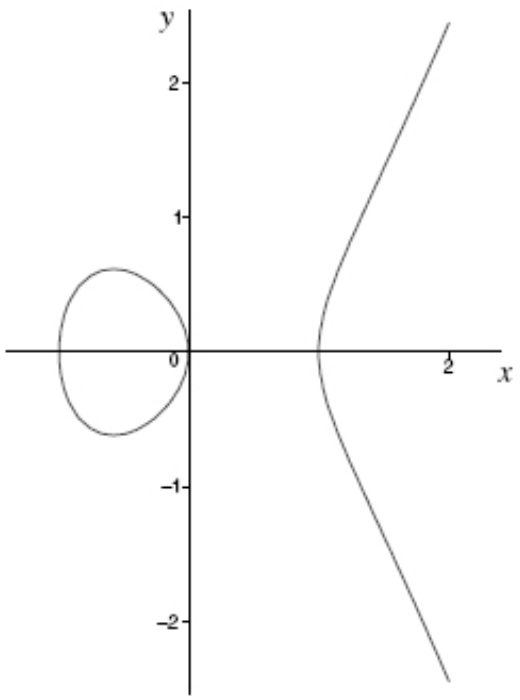

Figura 3.1: Curva $y^2 = x^3 - x$

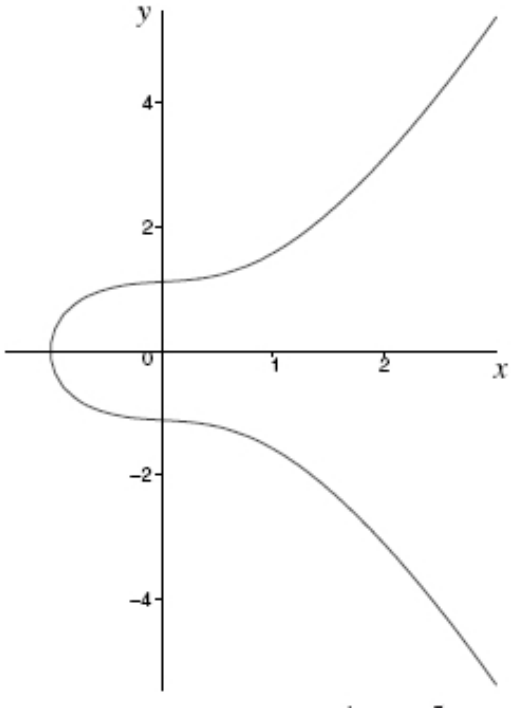

Figura 3.2: Curva $y^2 = x^3 + \frac{1}{4}x + \frac{5}{4}$.

3.2.1 Simplificando a Equação de Weiestrass

Trabalhar diretamente com a Equação 3.6 pode ser um processo trabalhoso. Com o intuito de minimizar esse problema, aplica-se uma mudança de variável, capaz de transformar uma curva E_1 em uma curva isomorfa a ela E_2.

Definição 3.2 *Sejam as curvas E_1 e E_2, definidas sobre o corpo K, tais que*

$$E_1 : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

$$E_2 : y^2 + \bar{a}_1xy + \bar{a}_3y = x^3 + \bar{a}_2x^2 + \bar{a}_4x + \bar{a}_6$$

Tais curvas são chamadas isomorfas caso existam $u, r, s, t \in K, u \neq 0$, tais que a mudança de variáveis

$$(x, y) \rightarrow (u^2x + r, u^3y + u^2sx + r) \tag{3.8}$$

converta E_1 em E_2.

A transformação apresentada na Definição 3.2 é muito usada e assume diferentes formas, conforme a característica do corpo ao qual pertence a curva que se pretende transformar. Assim, dada a curva E definida sobre o corpo K e escrita conforme a Equação 3.6, três casos devem ser observados:

1. Para o caso onde a característica de K é diferente tanto de 2 quanto de 3, a mudança

$$(x, y) \to \left(\frac{x - 3a_1^2 - 12a_2}{36}, \frac{y - 3a_1 x}{216} - \frac{a_1^3 + 4a_1 a_2 - 12a_3}{24} \right)$$

converte E em

$$y^2 = x^3 + ax + b, \text{ com } a, b \in K \tag{3.9}$$

Neste caso, o discriminante Δ passa a valer $-16(4a^3 + 27b^2)$.

2. Quando K possui característica igual a 2, há duas possíveis mudanças

   - Quando $a_1 \neq 0$, obtém-se a curva

   $$y^2 + xy = x^3 + ax^2 + b, \text{ com } a, b \in K \tag{3.10}$$

   que recebe o nome de curva não-supersingular, através da mudança

   $$(x, y) \to \left(a_1^2 x + \frac{a_3}{a_1}, a_1^3 y + \frac{a_1^2 a_4 + a_3^2}{a_1^3} \right)$$

   cujo discriminante passa a ser $\Delta = b$.

   - Quando $a_1 = 0$, a curva obtida, chamada supersingular, é definida por

   $$y^2 + cy = x^3 + ax + b, \text{ com } a, b, c \in K, \tag{3.11}$$

   através da transformação

   $$(x, y) \to (x + a_2, y),$$

   cujo discriminante passa a ser dado por $\Delta = c^4$.

3. Nos casos onde a característica do corpo é 3, há também duas situações específicas:

   - Quando $a_1^2 \neq -a_2$, obtém-se uma curva não-supersingular

   $$y^2 = x^3 + ax + b, \text{ com } a, b \in K, \tag{3.12}$$

   de discriminante $\Delta = -a^3 b$, através da mudança de variável

   $$(x, y) \to \left(x + \frac{a_4 - a_1 a_3}{a_1^2 + a_2}, y + a_1 x + a_1 \frac{a_4 - a_1 a_3}{a_1^2 + a_2} + a_3 \right)$$

   - Quando $a_1^2 = -a_2$, uma equação supersingular

   $$y^2 = x^3 + ax + b, \text{ com } a, b \in K \tag{3.13}$$

   é obtida, sendo o seu discriminate dado por $\Delta = -a^3$, através da mudança

   $$(x, y) \to (x, y + a_1 x + a_3)$$

Exemplo 3.2 Seja a curva elíptica $y^2 = x^3 + 2x + 4$, definida sobre $\mathbb{F}_{11}$. É fácil ver que $\Delta = -16(4 \cdot 2^3 + 27 \cdot 4^2) = -7224 \not\equiv 0 \pmod{11}$, cumprindo a condição para uma curva simplificada de acordo com a Equação 3.9. A obtenção dos pontos de $E(\mathbb{F}_{11})$ é feita iniciando-se com a verificação de quais elementos y, pertencentes a $\mathbb{F}_{11}$ representam resíduos quadráticos:

y	0	1	2	3	4	5	6	7	8	9	10
$z = y^2 \pmod{11}$	0	1	4	9	5	3	3	5	9	4	1

Tais valores são, como pode ser visto, 1, 3, 4, 5 e 9.

O fato de $p + 1 = 11 + 1 = 12$ ser divisível por 4 traz algumas vantagens para os cálculos a serem efetuados. Se z é um resíduo quadrático módulo 11, então $\pm z^{(11+1)/4} \pmod{11} = \pm z^3 \pmod{11}$. Esses valores representam as duas raízes quadradas de z. De fato,

$$\left(z^{\frac{p+1}{4}}\right)^2 = z^{\frac{p+1}{2}} = z^{\frac{p+1+2-2}{2}} = z^{\frac{p-1+2}{2}} = z^{\frac{p-1}{2}+1} = z^{\frac{p-1}{2}} \cdot z \equiv z \pmod{11}.$$

Com essas informações, é possível montar a tabela a seguir:

x	$z = (x^3 + 2x + 4) \pmod{11}$	$y = \pm z^3 \pmod{11}$
0	4	9 e 2
1	7	—
2	5	4 e 7
3	4	9 e 2
4	10	—
5	7	—
6	1	1 e 10
7	9	3 e 8
8	4	9 e 2
9	3	5 e 6
10	1	1 e 10

A primeira coluna contém todos os elementos do corpo em estudo. A segunda coluna corresponde aos valores da coluna 1 aplicada à equação da curva, reduzida módulo 11 e a terceira, os valores de y tais que seus quadrados módulo 11 resultem nos valores correspondentes na segunda coluna. Assim,

- $y = 9 \Rightarrow 9^2 = 81 \bmod 11 = 4$
- $y = 2 \Rightarrow 2^2 = 4 \bmod 11 = 4$
- $y = 4 \Rightarrow 4^2 = 16 \bmod 11 = 5$
- $y = 7 \Rightarrow 7^2 = 49 \bmod 11 = 5$

- $y = 1 \Rightarrow 1^2 = 1 \bmod 11 = 1$
- $y = 10 \Rightarrow 10^2 = 100 \bmod 11 = 1$
- $y = 3 \Rightarrow 3^2 = 9 \bmod 11 = 9$
- $y = 8 \Rightarrow 8^2 = 64 \bmod 11 = 9$
- $y = 5 \Rightarrow 5^2 = 25 \bmod 11 = 3$
- $y = 6 \Rightarrow 6^2 = 36 \bmod 11 = 3$

Dos cálculos efetuados acima, conclui-se que a curva $y^2 = x^3 + 2x + 4$ possui 17 pontos, sendo eles:

$$\begin{array}{lllll} \infty & (0,2) & (7,3) & (7,8) & (2,4) \quad (2,7) \\ (6,10) & (8,2) & (3,9) & (3,2) & (9,5) \quad (9,6) \\ (10,1) & (8,9) & (6,1) & (0,9) & (10,10) \end{array}$$

Exemplo 3.3 Seja E uma curva não-supersingular $y^2 + xy = x^3 + ax^2 + b$, definida sobre $\mathbb{F}_{2^4}$, cujo polinômio redutor é dado por $f(x) = z^4 + z + 1$. Como um elemento deste corpo pode ser representado por uma cadeia de bits do tipo $(a_3a_2a_1a_0)$, o valor (1000), correspondente a z^3, e (1001), correspondente a $z^3 + 1$, podem ser atribuídos aos coeficientes a e b, respectivamente. Assim,

$$E(\mathbb{F}_{2^4}) : y^2 + xy = x^3 + z^3x^2 + (z^3 + 1)$$

Os pontos desta curva são:

$$\begin{array}{llll} \infty & (0011,1100) & (1000,0001) & (1100,0000) \\ (0000,1011) & (0011,1111) & (1000,1001) & (1100,1100) \\ (0001,0000) & (0101,0000) & (1001,0110) & (1111,0100) \\ (0001,0001) & (0101,0101) & (1001,1111) & (1111,1011) \\ (0010,1101) & (0111,1011) & (1011,0010) & \\ (0010,1111) & (0111,1100) & (1011,1001) & \end{array}$$

3.3 Lei de Grupo Sobre um Conjunto de Pontos de uma Curva Elíptica

É possível estabelecer uma estrutura de grupo abeliano sobre um conjunto composto por pontos de uma curva elíptica. Para tanto, é necessário que se estabeleça uma operação binária apropriada e que se defina também as noções de elemento neutro, inverso e negativo (ou oposto), tudo isso referente a operação binária especificada. Esta operação é a adição de pontos, que também pode ser vista como duplicação, conforme será mostrado a seguir.

3.3.1 Abordagem Geométrica

Nos casos onde a curva é definida sobre o corpo dos números reais, é possível visualizar a lógica geométrica que envolve a definição da operação de adição de pontos. Essa lógica é conhecida como *lei de corda e tangente*, descrita a seguir.

Sempre que se queira somar dois pontos P e Q, traça-se primeiramente uma reta que contenha esses dois pontos, interceptando a curva em um terceiro ponto. A seguir, executa-se uma reflexão em relação o eixo x, encontrando um outro ponto de interceptação R. Esse ponto corresponde à soma $P + Q$. Todo o processo é mostrado na Figura 3.3.

A duplicação de pontos é definida de maneira análoga. Inicialmente, desenha-se a reta tangente ao ponto P, que se deseja duplicar. Essa reta intercepta ao gráfico da curva em um segundo ponto, que ao ser refletido resulta no ponto R, correspondente a $2P$. Isso pode ser visto na Figura 3.4.

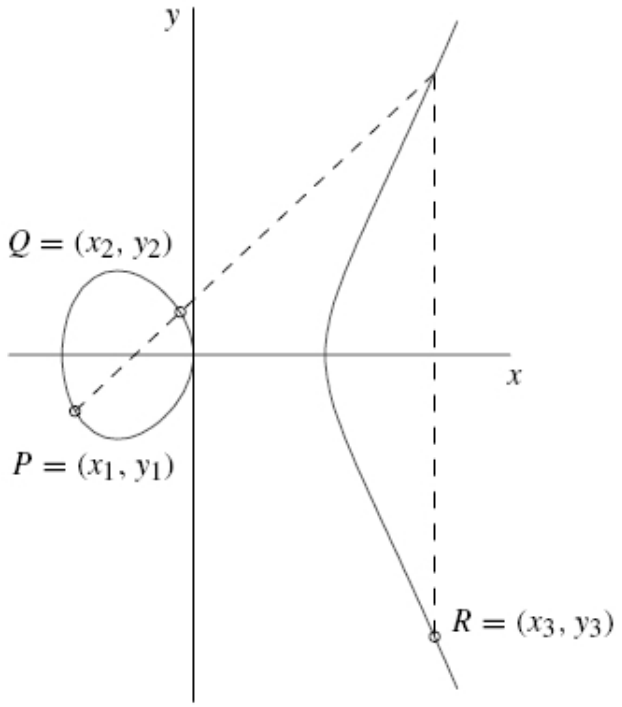

Figura 3.3: Soma dos pontos P e Q, em uma curva elíptica: $P + Q = R$.

3.3.2 Abordagem Algébrica

Na prática, nem sempre é viável utilizar a lei de corda e tangente. Isso é facilmente entendido quando se considera uma curva definida sobre um corpo finito. Neste caso, o gráfico da curva não corresponde a linhas contínuas como na Figura 3.2, e sim a pontos discretos no plano, como pode ser visto na Figura 3.5. Entretanto, utilizando-se desta construção, é possível definir fórmulas gerais, sendo estabelecido um caráter algébrico à abordagem geométrica.

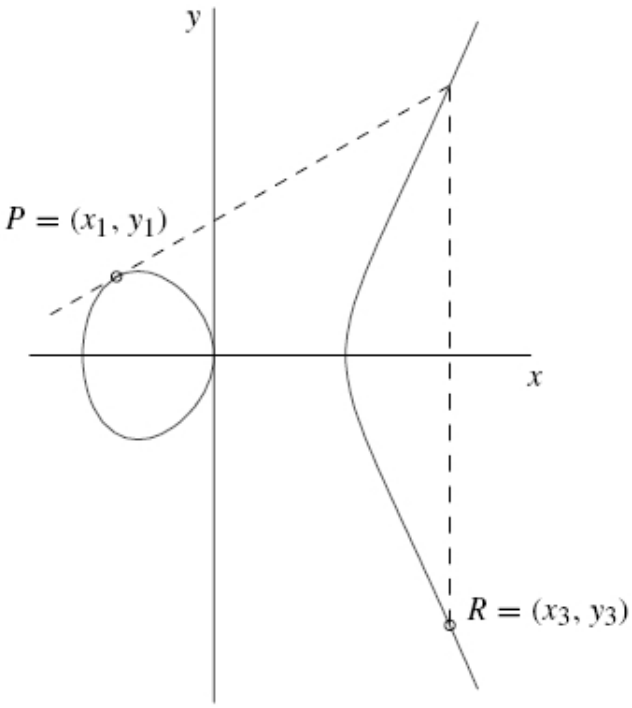

Figura 3.4: Duplicação do ponto P em uma curva elíptica: $P + P = R$.

Obtenção das Fórmulas Gerais para $y^2 = x^3 + ax + b$

A explicação que se segue mostra o comportamento curvas do tipo definidas pela Equação 3.9, ou seja, para curvas definidas sobre corpos cuja características sejam diferentes de 2 e 3. A lógica empregada aqui é a mesma utilizada no exemplo da pirâmide de balas de canhão, mostrado no início deste capítulo.

Sejam os pontos $P = (x_1, y_1), Q = (x_2, y_2) \in E(\mathbb{F}_p)$. Deseja-se calcular $R = P + Q = (x_3, y_3)$, tal que este também seja um ponto de $E(\mathbb{F}_p)$.

Considerando $P \neq Q$ e ambos diferentes do ponto ∞, a inclinação da reta L que os contém

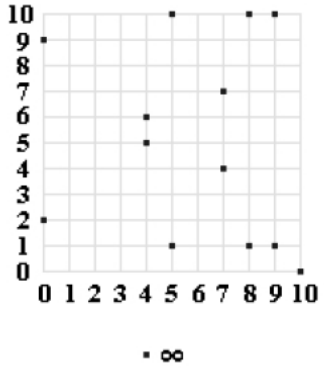

Figura 3.5: Curva $y^2 = x^3 - 8x + 4$, definida sobre $\mathbb{Z}_{11}$ [BARRETO (1999)].

é dada por

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

Por hora, considera-se também $x_1 \neq x_2$, o que evita que L seja vertical. Essa reta é dada pela equação

$$y = m(x - x_1) + y_1$$

O ponto de intersecção entre L e a curva elíptica é obtido pela substituição

$$(m(x - x_1) + y_1)^2 = x^3 + ax + b$$

cujo desenvolvimento leva à cúbica

$$x^3 - m^2x^2 + (2m^2x_1 - 2my_1)x + (b - m^2x_1^2 + 2mx_1y_1 - y_1^2) = 0.$$

As raízes desta equação corresponde aos três pontos de intersecção entre a reta e a curva.

Embora a resolução desta equação não seja, na maioria dos casos, trivial, duas de suas raízes já são conhecidas: x_1 e x_2. Um raciocínio análogo ao do exemplo das balas de canhão pode ser empregado aqui.

Sendo r, s, t as raízes de uma cúbica, tem-se:

$$\begin{aligned} x^3 + ax^2 + bc + c &= (x - r)(x - s)(x - t) \\ &= x^3 - rx^2 - sx^2 - rsx - tx^2 + rtx + stx - srt \\ &= x^3 - (s + r + t)x^2 + (rt + sr + st)x - srt \end{aligned}$$

Donde se conclui que $-s - r - t = a$. No caso em estudo: $-x_1 - x_2 - x = -m^2 \Rightarrow x = m^2 - x_1 - x_2$, que corresponde a x_3. Como $y = m(x - x_1) + y_1$ já é conhecido, basta executar uma reflexão em relação ao eixo y para que se obtenha y_3. Assim,

$$x_3 = m^2 - 2x_1 \text{ e } y_3 = m(x_1 - x_3) - y_1.$$

Mas, se $x_1 = x_2$, com $y_1 \neq y_2$, a reta L é vertical, o que significa que a intersecção encontra-se no infinito. Deste modo, $P + Q = \infty$. O caso onde ocorre também a igualdade entre y_1 e y_2, que corresponde à duplicação de pontos, é analisado a seguir.

Se $P = Q = (x_1, y_1)$, a reta L que os contém é na verdade uma reta tangente a ambos. A inclinação da reta tangente à curva no ponto (x, y) é facilmente obtido pela derivação implícita desta:

$$\begin{aligned} \frac{d}{dx}[y^2 = x^3 + ax + b] &\Rightarrow \frac{d}{dx}[y^2] = \frac{d}{dx}[x^3 + ax + b] \\ &\Rightarrow 2y\frac{dy}{dx} = 3x^2 + a \\ &\Rightarrow \frac{dy}{dx} = \frac{3x^2 + a}{2y}, \end{aligned}$$

que aplicada ao ponto (x_1, y_1), resulta em

$$m = \frac{dy}{dx} = \frac{3x_1^2 + a}{2y_1}.$$

Para os casos onde $y_1 = 0$, define-se $P + Q = \infty$. Caso contrário, a equação da reta é dada por

$$y = m(x - x_1) + y_1.$$

Através de um processo análogo ao empregado para a obtenção dos pontos de intersecção entre a reta e a curva no caso anterior, chega-se ao resultado

$$x_3 = m^2 - 2x_1 \text{ e } y_3 = m(x_1 - x_3) - y_1.$$

Por fim, quando um dos pontos corresponde ao ponto no infinito, por exemplo $Q = \infty$, a reta que intercepta P e Q é uma vertical que passa por P_1, cuja reflexão em relação ao eixo y faz com que esta retorne ao ponto P. Assim,

$$P + \infty = P.$$

Um caso particular é quando $P = Q = \infty$:

$$\infty + \infty = \infty.$$

Em síntese, a lei de grupo para tais curvas é dada por

1. *Identidade*. $P + \infty = P, \forall P \in E(\mathbb{F}_p)$.
2. *Negativos*. Se $P = (x_1, y_1) \in E(\mathbb{F}_p)$, então o negativo de P é dado por $-P = (x, -y)$.
3. *Adição de pontos*. Sejam $P = (x_1, y_1), Q = (x_2, y_2) \in E(\mathbb{F}_p)$, com $P \neq Q$. $P + Q = R = (x_3, y_3)$ é dado por:
$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \text{ e } y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1.$$
4. *Duplicação de pontos*. Seja $P = (x_1, y_1) \in E(\mathbb{F}_p)$. Então, $2P = (x_3, y_3)$ é dado por
$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \text{ e } y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1.$$

Exemplo 3.4 Seja a curva elíptica $E(\mathbb{F}_{11}) : y^2 = x^3 + 2x + 4$, definida no Exemplo 3.2. Exemplos de soma e duplicação de pontos desta curva são dados a seguir.

1. A soma dos pontos $P = (7, 3)$ e $Q = (2, 4)$, ambos pertencentes a esta curva é executada da seguinte maneira:
$$\begin{aligned} t &= \frac{y_2 - y_1}{x_2 - x_1} = \frac{4 - 3}{2 - 7} = -\frac{1}{5} \equiv -\frac{45}{5} = -9 \equiv 2 \pmod{11} \\ x_3 &= t^2 - x_1 - x_2 = 4 - 9 = -5 \equiv 6 \pmod{11} \\ y_3 &= t(x_1 - x_3) - 1 = 2(7 - 6) - 3 = -1 \equiv 10 \pmod{11} \end{aligned}$$
Assim, $(7, 3) + (2, 4) = (6, 10)$

2. A duplicação do ponto $P = (7, 3)$ é obtida por meio dos seguintes cálculos:

$$\begin{aligned} t &= \frac{3x_1^2 + a}{2y_1} = \frac{149}{6} \equiv \frac{6}{6} = 1 \equiv 1 \pmod{11} \\ x_3 &= t^2 - 2x_1 = 1 - 14 = -13 \equiv 9 \pmod{11} \\ y_3 &= t(x_1 - x_3) - y_1 = 1(7 - 9) - 3 = 5 \equiv 6 \pmod{11} \end{aligned}$$

Assim, $2 \cdot (7, 3) = (9, 6)$

Embora não tenha sido definido o cálculo de $3P$, esse valor pode se facilmente obtido através da soma $2P + P$ que, para $P = (7, 3)$, significaria $(9, 6) + (7, 3) = (0, 2)$. Esse método pode ser generalizado para que se possa determinar o ponto $kP, k \in \mathbb{Z}$.

Leis de Grupos em $E(\mathbb{F}_{2^m})$

As operações de soma e duplicação de pontos, assim como as propriedades dos elementos negativos e do elemento identidade, são definidas para curvas sobre corpos binários $E(\mathbb{F}_{2^m})$ de duas formas distintas.

Para uma curva não-supersingular do tipo $E(\mathbb{F}_{2^m}) = y^2 + xy = x^3 + ax^2 + b$, tem-se:

1. *Identidade*. $P + \infty = \infty + P = P, \forall P \in E(\mathbb{F}_{2^m})$.

2. *Negativo*. Se $P = (x, y) \in E(\mathbb{F}_{2^m})$, então $-P = (x, x + y)$

3. *Adição de pontos*. Sejam $P = (x_1, y_1), Q = (x_2, y_2) \in E(\mathbb{F}_{2^m})$, com $P \neq \pm Q$. $P + Q = R = (x_3, y_3)$ é dado por:

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \text{ e } y_3 = \lambda(x_1 + x_3) + x_3 + y_1,$$

   com $\lambda = (x_1 + y_2)/(x_1 + x_2)$.

4. *Duplicação de pontos*. Seja $P = (x_1, y_1) \in E(\mathbb{F}_{2^m})$, onde $P \neq -P$. Então, $2P = (x_3, y_3)$ é dado por

$$x_3 = \lambda^2 + \lambda + a = x_1^2 + \frac{b}{x_1^2} \text{ e } y_3 = x_1^2 + \lambda x_3 + x_3,$$

   com $\lambda = x_1 + y_1/x_1$.

Exemplo 3.5 Seja a curva elíptica descrita no Exemplo 3.3. Se $P = (0001, 0001)$ e $Q = (0010, 1101)$, então o procedimento para o cálculo de $P + Q$ é dado por:

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2} = \frac{(1100)}{(0011)} = \frac{z^3 + z^2}{z + 1} = \frac{z^2(z + 1)}{z + 1} = z^2 = (0100)$$

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a = z^4 + z^3 + z^2 + z + 1 \equiv z^3 + z^2 = (1100) \\ y_3 &= \lambda(x_1 + x_3) + x_3 + y_1 = z^2(1 + z^3 + z^2)z^3 + z^2 + 1 \equiv z^3 + z^2 = (1100). \end{aligned}$$

Assim, $(0001, 0001) + (0010, 1101) = (1100, 1100)$

3.3.3 A ordem de um grupo de pontos de uma curva elíptica

Tendo definido a estrutura geral de um grupo formado por pontos de uma curva elíptica sobre um corpo finito, uma questão que naturalmente surge é a quantidade de elementos que tal grupo poderá comportar. De maneira geral, uma curva definida sobre um $\mathbb{F}_q$ possui uma quantidade de elementos próxima a q.

Verificando a Equação 3.6, constata-se a presença do termo y^2, o que implica que para cada valor de x atribuído à equação, há no máximo dois valores correspondentes para y. Como há q valores possíveis ($x \in \mathbb{F}_q$), a ordem de uma curva, denotada por $\#E(\mathbb{F}_q)$, é um valor presente entre 1 e $2q + 1$. O próximo teorema propõe um limite mais justo para este valor.

Teorema 3.1 (Hasse) *Seja E uma curva elíptica definida sobre o corpo $\mathbb{F}_q$. Pode-se afirmar que*

$$q + 1 - 2\sqrt{q} \leq \#E(\mathbb{F}_q) \leq q + 1 + 2\sqrt{q}.$$

Uma forma alternativa de se representar a ordem de uma curva é através da relação $\#E(\mathbb{F}_q) = q + 1 - t$, onde $|t| \leq 2\sqrt{2}$. O termo t recebe o nome de *traço* de E sobre $\mathbb{F}_q$. O traço e uma curva é particularmente útil no momento de se estabelecer a ordem admissível de uma curva, conforme é mostrado no próximo teorema.

Teorema 3.2 *Seja $q = p^m$, onde p é a característica de $\mathbb{F}_q$. A existência de uma curva definida sobre este corpo com ordem $q + 1 - t$ está condicionada ao comprimento de uma das seguintes condições:*

1. $t \not\equiv 0 \pmod{p}$ *e* $t^2 \leq 4q$
2. m *é ímpar e*
   - (a) $t = 0$, *ou*
   - (b) $t^2 = 2q$ *e* $p = 2$, *ou*
   - (c) $t^2 = 3q$ *e* $p = 3$
3. m *é par e*
   - (a) $t^2 = 4q$, *ou*
   - (b) $t^2 = q$ *e* $p \not\equiv 1 \pmod{3}$, *ou*
   - (c) $t = 0$ *e* $p \not\equiv 1 \pmod{4}$

Isso indica que, para qualquer primo p e inteiro t que satisfaçam $|t| \leq 2\sqrt{p}$, há uma curva sobre $\mathbb{F}_p$ com ordem $p + 1 - t$.

Além da ordem de um grupo de pontos, outro tema de interesse é a ordem de um ponto. Operando um ponto consigo mesmo, obtém-se um subgrupo gerado por este ponto. A quantidade de elementos deste subgrupo define a ordem do ponto que o gerou.

3.4 Multiplicação escalar de pontos de uma curva

Embora a multiplicação de pontos por um número escalar não tenha sido formalmente definida, esta pode ser obtida por meio da adição sucessiva, utilizando as técnicas apresentada na Seção 3.3. A multiplicação escalar eficiente em curvas elípticas é um assunto de grande interesse (tanto prático quanto teórico), tendo sido o foco de artigos e teses defendidas nos últimos anos.

Esta operação domina o tempo de execução de criptossistemas de curvas elípticas [HANKERSON et al. (20
Conforme será mostrado nos próximos capítulos, os esquemas criptográficos fazem usos freqüente
da multiplicação escalar para cumprir seus protocolos.

Por ser uma assunto extremamente amplo, este trabalho não apresentará todos os seus pormenores, apresentando apenas as técnicas básicas necessárias para oferecer uma eficiência razoável para um sistema que a implemente.

3.4.1 Uma primeira abordagem: o método binário

Sejam o ponto $P \in E(\mathbb{F}_q)$ e $k \in \mathbb{Z}$. O cálculo de kP poderia ser facilmente executado por meio da adição de P a si mesmo, repetindo esse processo k vezes. Isso levaria ao Algoritmo 3.1.

Algoritmo 3.1 Método simples de multiplicação escalar

Entrada: $P \in E(\mathbb{F}_q)$ e $k \in \mathbb{Z}$

Saída: $Q = kP$

1: $Q \leftarrow \infty$
2: **para** i de 1 até k **faça**
3: $\quad Q \leftarrow Q + P$
4: **fim para**
5: **retorne** Q

Este método traz claros inconvenientes quando k é um valor muito grande — o que é comum em esquemas criptográficos. Uma forma de melhorá-lo é fazer uso de algumas propriedades da expansão binária de k.

A expansão binária de k corresponde a $(k_{t-1}, ..., k_1, k_0)$, cujo comprimento é $t = \lceil \log k \rceil$. O Algoritmo 3.2 percorre esta expansão, efetuando a adição somente para os casos onde k_i é igual a 1. Se a quantidade de algarismo de um número em sua representação binária for dado por t, então a quantidade esperada de 'uns' é $t/2$. Isso reduz significativamente o tempo de execução deste algoritmo, quando comparado com o anterior, uma vez que são executado (em média) $m/2$ adições e m duplicações.

Há uma maneira de melhorar ainda mais o método binário. No cálculo de kP, com $k = (k_{t-1}, ..., k_1, k_0)_2$, as adições são executadas somente nos casos onde $k_i \neq 0, 0 \leq i < t$, conforme já foi mostrado. Se o inteiro k puder ser expresso de forma a diminuir a quantidade

Algoritmo 3.2 Método binário de multiplicação

Entrada: $k = (k_{t-1}, ..., k_1, k_0)_2$ e $P \in E(\mathbb{F}_q)$.

Saída: $Q = kP$

1: $Q \leftarrow \infty$
2: **para** i de 0 até $t-1$ **faça**
3: **se** $k_i = 1$ **então**
4: $Q \leftarrow Q + P$
5: **fim se**
6: $P \leftarrow 2P$
7: **fim para**
8: **retorne** Q

de elementos diferentes de zero em sua representação, então diminuir-se-á também a quantidade de adições executadas no processo. Essa representação pode ser conseguida pela forma não-adjacente, que merece ser detalhada um pouco mais.

3.4.2 Forma não-adjacente e sua utilização na multiplicação de pontos

Definição 3.3 *A forma não-adjacente, ou NAF (non-adjacent form) de $k \in \mathbb{Z}$, é uma expressão do tipo $\sum_{i=0}^{l-1} k_i 2^i$, onde l é seu comprimento e $k_i \in \{-1, 0, 1\}$, $k_{l-1} \neq 0$, sem que haja dois dígitos consecutivos que diferentes de zero.*

Exemplo 3.6 O inteiro 123456789, cuja representação binária é dada por:

$$(1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1),$$

tem sua forma não adjacente dada por:

$$(1, 0, 0, 0, \mathbf{1}, 0, \mathbf{1}, 0, 0, \mathbf{1}, 0, 0, 0, \mathbf{1}, 0, 1, 0, \mathbf{1}, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1),$$

onde $\mathbf{1} = -1$.

No caso mostrado no Exemplo 3.6, dos 27 dígitos que compõem a representação binária, 16 são diferentes de zero. Na forma não adjacente, esse número cai para 11. Isso tende a ser mais significativo para casos onde há muitos dígitos diferentes de zero, conforme é mostrado no Exemplo 3.7.

Exemplo 3.7 O inteiro 2045, cuja representação binária é dada por $(1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1)$ é expresso na forma não adjacente por $(1, 0, 0, 0, 0, 0, 0, 0, 0, \mathbf{1}, 0, 1)$.

O Teorema 3.3 apresenta interessantes propriedades da forma não-adjacente.

Teorema 3.3 *Seja k um inteiro positivo. A forma não-adjacente obedece às seguintes propriedades:*

1. *A representação de k na forma não-adjacente é única.*
2. *A representação de k na forma não-adjacente apresenta uma menor quantidade de dígitos diferentes de zero de todas as suas outras representações digitais com sinal.*
3. *Se a representação de k na forma não-adjacente tem comprimento l, então $2^l/3 < k < 2^{l+1}/3$*
4. *A densidade média de dígitos diferentes de zero de um número na forma não-adjacente cujo comprimento l é $l/3$.*

As informações apresentadas ilustram a melhoria que pode ser obtida pelo método binário se este utilizar a NFA ao invés da simples representação binária de k. O Algoritmo 3.3 fornece um meio de calculá-la.

Algoritmo 3.3 Cálculo da forma não-adjacente para um inteiro positivo

Entrada: $k \in \mathbb{Z}_+$

Saída: A forma não-adjacente de k

1: $i \leftarrow 0$
2: **enquanto** $k \geq 1$ **faça**
3: **se** k é ímpar **então**
4: $k_i \leftarrow 2 - (k \bmod 4)$
5: $k \leftarrow k - k_i$
6: **senão**
7: $k_i \leftarrow 0$
8: **fim se**
9: $k \leftarrow k/2$
10: $i \leftarrow i + 1$
11: **fim enquanto**
12: **retorne** $k = (k_{l-1}, k_{l-2}, ..., k_1, k_0)$

Agora, o Algoritmo 3.2 pode ser modificado, inserindo-se a etapa do cálculo da FNA, o que reduz consideravelmente o tempo de execução. As modificações são apresentadas no Algoritmo 3.4.

De acordo com o item 4, do Teorema 3.3, a quantidade esperada de dígitos diferentes de zero é $m/3$. Isso significa que, de acordo com o Algoritmo 3.4, o número de adições é, na média, $m/3$ e que são executadas m duplicações; um ganho substancial quando comparado aos demais métodos apresentados até aqui.

Algoritmo 3.4 Método binário com FNA para multiplicação de pontos

Entrada: Um inteiro k, e o ponto $P \in E(\mathbb{F}_q)$

Saída: $Q = kP$

1: Use o Algoritmo 3.3 para calcular a forma não adjacente de k.
2: $Q \leftarrow \infty$.
3: **para** i de $l-1$ até 0 **faça**
4: $\quad Q \leftarrow 2Q$;
5: $\quad$ **se** $k_i = 1$ **então**
6: $\quad\quad Q \leftarrow Q + P$;
7: $\quad$ **fim se**
8: $\quad$ **se** $k_i = -1$ **então**
9: $\quad\quad Q \leftarrow Q - P$
10: $\quad$ **fim se**
11: **fim para**
12: **retorne** Q.

3.5 Considerações finais

Este capítulo reuniu os principais conceitos necessários ao entendimento das entidades matemáticas denominada curvas elípticas. Os aspectos teóricos de álgebra moderna, apresentados no capítulo anterior, serviram de base para tal entendimento visto elas são equações definidas sobre corpos finitos.

Foram apresentadas também os detalhes referentes à forma de se operar pontos de uma curva e o porque do par operação/grupo serem considerados grupos.

Por fim, foi mostrada a evolução lógica para o tratamento da multiplicação de pontos de uma curva por um valor escalar, detalhando as principais técnicas utilizadas para se obter algoritmos mais eficientes.

O exemplo das balas de canhão, apresentado na Seção 3.1.1, foi adaptado de [WASHINGTON (2003)]. Uma descrição mais detalhada das propriedades aritmética de curvas elípticas pode ser encontrada na obra de Joseph Silverman [SILVERMAN (1985)].

O enfoque desse capítulo foi dado às curvas definidas sobre corpos primos, visto que esse foi o tipo de curva escolhido para a implementação feita nesse trabalho. Entretanto, curvas definidas sobre corpos binários apresentam propriedades interessantes quanto à multiplicação escalar. Um maior detalhamento sobre tal assunto pode ser encontrado em [NETO (2006)].

Destacam-se as *curvas de Koblitz*, conhecidas também como *curvas binárias anômalas* [SOLINAS (1997)]. Elas apresentam interessantes aplicações criptográficas, mas não discutidas aqui por constituírem uma assunto demasiadamente amplo.

Capítulo 4

Criptografia

4.1 Introdução

Criptografia é a ciência que trata do estudo e da análise de técnicas matemáticas que possibilitam comunicação segura, ainda que a mesma aconteça em presença de supostos adversários [HANKERSON et al. (2004)]. Esta ciência está estritamente ligada à manipulação de informações e possui aplicações que vão desde a mais simples e informal comunicação entre usuários da internet [TANENBAUM (2003)] até questões de segurança nacional. Com o advento dos computadores digitais, esta ciência ganhou um destaque ainda maior.

Segundo [MENEZES et al. (1997)], sistemas de informação considerados seguros apresentam quatro características principais:

1. *Confidencialidade.* Consiste em manter uma dada informação ao alcance somente daqueles que estão autorizados a manipulá-la. Consiste, portanto, em preservar o sigilo da informação.

2. *Integridade dos dados.* Impede alterações indevidas nos dados — sejam elas intencionais ou não. Inserção, remoção e substituição de trechos da informação (ou da informação inteira) são exemplos de alterações indevidas. Sempre que elas ocorrerem, ambas as partes comunicantes devem ser capazes de detectar tais ocorrência.

3. *Altenticação.* Já na década de setenta, Diffie e Hellman [DIFFIE & HELLMAN (1976)] discutiam sobre a importância da autenticação, tanto no que se refere às partes envolvidas na comunicação quanto a autenticação dos dados manipulados no processo. Segundo eles, a capacidade de se autenticar era o principal pré-requisito para garantir a segurança em transações empresariais, o que se confirmou com o tempo, haja vista a grande difusão atual do comércio eletrônico. A *autenticação de entidade* consiste na capacidade de ambas as partes envolvidas estarem certas da identidade uma da outra. A *autenticação de*

dados consiste na capacidade de uma parte saber se os dados recebidos são os mesmos dados enviados pela outra.

4. *Não-repúdio.* Esta característica é também muito útil em casos que envolvem contratos comerciais [DIFFIE & HELLMAN (1976)]. Em uma negociação entre duas ou mais partes, uma parte não deve ser capaz de negar algo que tenha previamente alegado.

Este capítulo oferece uma visão geral da criptografia computacional. São descritos aqui os aspectos fundamentais, como as principis notações e nomenclaturas, os sistemas simétricos e as razões que levaram à criação dos sistemas assimétrico. O protocolo de troca de chave e os criptossistemas RSA e ElGamal também são descritos nesse capítulo. As características específicas dos criptossistemas de curvas elípticas serão melhor detalhadas no Capítulo 5.

4.2 Conceitos Fundamentais

4.2.1 Entidades de comunicação e o modelo adversário

Tradicionalmente, a literatura representa as entidades participantes da comunicação por A (Alice) e B (Bob). Eles desejam se comunicar de forma segura. Entretanto, um terceiro participante E (Eve), também está presente, representando um adversário que possui "poderes" consideráveis. Eve pode ser capaz de escutar o canal de comunicação sem revelar sua presença e adulterar dados que trafegam por esse canal, apagando trechos da mensagem original ou inserindo seus próprios dados na mesma. Eve conhece também todos os mecanismos, protocolos e algoritmos utilizados no canal de comunicação, desconhecendo apenas os dados que A e B mantém em segredo.

Essas entidades podem assumir diversas formas. Alice e Bob podem ser, por exemplo, dois funcionários de uma empresa que desejam trocar informação confidenciais via rede — interna ou externa — enquanto Eve pode ser um espião, que invade a rede desejoso de conseguir algum segredo industrial. Em outra situação, A pode representar o navegador de um usuário que faz compras pela internet. Neste caso, B seria o site que processa a venda e E, alguém interessado em obter informações sigilosas sobre o cartão de crédito de A.

As situações descritas acima correspondem ao chamado *modelo adversário* e tem sido usado com freqüência no estudo da criptografia. O principal objetivo dos criptógrafos é desenvolver mecanismos que sejam capazes de manter a segurança diante de adversários cada vez mais poderosos [HANKERSON et al. (2004)]

4.2.2 Definições

A princípio, é necessário definir alfabeto, espaço de mensagem e espaço de texto cifrado.

Alfabeto é um conjunto finito e não vazio de símbolos. Aqui, ele será denotado por $\mathcal{A}$. Este conjunto pode ser composto por letras do alfabeto latino, caracteres ASCII, UNICODE ou, de maneira geral, por dígitos binários. $\mathcal{M}$ denota o espaço de mensagens: uma seqüencia de símbolos de $\mathcal{A}$ que compõem a mensagem original, conhecida por *mensagem clara* ou *texto claro*. Após passar pelo processo de ciframento, $\mathcal{M}$ tranforma-se em um elemento de $\mathcal{C}$, que é o espaço de textos cifrados.

O *espaço de chaves* é denotado por $\mathcal{K}$. Bijeções do tipo $f : \mathcal{M} \rightarrow \mathcal{C}$ são definidas para cada elemento $e \in \mathcal{K}$. Tal função é representada por $f = E_e$, chamada *função de ciframento*. Analogamente, defini-se sua inversa $D_d : \mathcal{C} \rightarrow \mathcal{M}$, com $d \in \mathcal{K}$, chamada *função de deciframento*. Assim, se $m \in \mathcal{M}$, então $D_d(C_e(m)) = m$. Nisto se baseiam as funções criptográficas.

As entidades participantes de uma comunicação foram descritas na Seção 4.2.1. Tudo se passa em um *canal de comunicação*. Entende-se canal de comunicação como sendo o meio por onde trafegam as informações trocadas entre remetente e destinatário. Em geral, esses canais não oferecem qualquer tipo de proteção aos dados que por ele trafegam.

A criptografia pressupõe que os conjuntos $\mathcal{M}$, $\mathcal{C}$, $\mathcal{K}$ e as funções E_e e D_d sejam de domínio público. Assim, a segurança fica por conta do sigilo do par de chaves (e, d). Se em um dado sistema for comprovadamente impossível se obter o texto claro a partir do texto cifrado sem o prévio conhecimento das chaves, então tal sistema é dito ser *inquebrável* [SHANNON (1949)].

É importante também considerar o tempo que uma informação deve permanecer em sigilo. Isso está diretamente relacionado ao tempo que o adversário levaria para decodificar uma mensagem sem o prévio conhecimento da chave de decodificação. Ainda que ele consiga reaver a mensagem, quando isso acontecer ela pode não ser mais tão importante.

Uma maneira simples (porém, ineficiente) de se reaver uma mensagem sem o conhecimento prévio da chave é a *busca exaustiva*. Este processo consiste em percorrer todo o conjunto $\mathcal{K}$, testando todas as possibilidades, até que se obtenha um texto que se identifique com algum idioma conhecido. Embora o resultado seja sempre encontrado[1], o tempo gasto para tanto pode tornar esta atividade inviável.

Questões referentes ao estudo de técnicas matemáticas que visam encontrar "fraquezas" em sistemas criptográficos são o foco de estudo da criptoanálise. Dos principais tipos de ataques, pode-se destacar o *ataque somente ao texto cifrado*, onde o criptoanalista tem acesso somente ao texto cifrado, o *ataque com texto conhecido*, situação em que o critpoanalista possui, além do texto cifrado, uma parte deste texto "traduzido" e o *ataque com texto claro escolhido* onde o adversário cifra um texto que ele mesmos determinou e realiza uma análise comparativa entre o texto claro e o texto cifrado, procurando identificar algum padrão. [DIFFIE & HELLMAN (1976)].

Existem dois tipos de cifradores: os simétricos e o assimétricos. As próximas seções fazem

[1]É preciso enteder que este processo é determinístico. Desta maneira, embora o número de testes seja suficientement alto para manter o adversário ocupado por "algumas centenas de anos", em um dado momento a chave certa será encontrada.

uma descrição destes dois sistemas.

4.3 Sistemas Simétricos

4.3.1 Fundamentos

O termo simétrico advém do fato de as chaves de codificação e e de decodificação d serem iguais. São também conhecidos por criptossistemas de chave secreta. A Definição 4.1, adaptada de [MENEZES et al. (1997)], explica melhor este esquema.

Definição 4.1 *Esquemas criptográficos simétricos são aqueles onde, para todo par de chaves* (e, d)*, é computacionalmente "fácil" determinar* e *a partir de* d *e* d *a partir de* e*. Na maioria dos casos,* $e = d$*.*

Um exemplo clássico de sistemas simétricos é a cifra de César. Ela consiste em atribuir uma valor para cada letra do alfabeto em uso e efetuar uma soma modular em cada letra, executando um deslocamento k. A Tabela 4.1 mostra a configuração inicial do alfabeto e a Tabela 4.2, a nova configuração para $k = 3$. A cifra de César enquadra-se na categoria de cifras de deslocamento.

A	B	C	D	E	F	G	H	I	J	K	L	M
0	1	2	3	4	5	6	7	8	9	10	11	12
N	O	P	Q	R	S	T	U	V	X	Y	W	Z
13	14	15	16	17	18	19	20	21	22	23	24	25

Tabela 4.1: Alfabeto e seus valores numéricos associados.

A	B	C	D	E	F	G	H	I	J	K	L	M
3	4	5	6	7	8	9	10	11	12	13	14	15
N	O	P	Q	R	S	T	U	V	X	Y	W	Z
16	17	18	19	20	21	22	23	24	25	0	1	2

Tabela 4.2: Nova configuração do alfabeto para $k = 3$.

Embora seja simples de se entender e de se implementar, as cifras de deslocamento sucumbem facilmente ao ataque de força bruta, dado o pequeno tamanho do espaço de chaves. É fácil ver que a cifra exemplificada possui apenas 25 valores de chave valor este relacionado com o comprimento do alfabeto. De maneira geral, o ataque é linear no tamanho do alfabeto.

Um alternativa é a *cifra de substituição*. Neste caso, ao invés de utilizar um simples deslocamento, uma tabela de permutação é utilizada. Desta forma se $|\mathcal{A}| = n$, então haverá $n! \approx 4 \times 10^{26}$ possibilidades de chave.

De maneira geral, sistemas simétricos possuem algoritmos rápidos e simples de se implementar. A eficiência pode ser conseguida tanto em implementações em software quanto em hardware. Entretanto, a manipulação de chaves pode levar alguns inconvenientes, conforme será visto na Seção 4.3.2. Além disso, não é possível aplicar o não repudio, visto que não há meios de se distinguir Alice de Bob.

Há bons cifradores simétricos em uso atualmente e um exemplo deles é o AES, cujo algoritmo é tomado como um padrão mundial e que foi desenvolvido para substituir o DES, quebrado no fim da década de 90. Entretanto, esses sistemas não serão detalhados aqui, visto que eles não são o foco deste trabalho. O principal intuito desta seção é mostrar as situações que levaram à criação de um novo enfoque para a construção de criptossistemas.

Em um sistema simétrico, se A e B desejam se comunicar, haverá antes a necessidade de ambos combinarem a chave que será utilizada pelo cifrador. Para que A envie a mensagem, uma chave k deverá ser utilizada como argumento da função de ciframento. Analogamente, B deverá se utilizar do mesmo valor k para reaver a mensagem clara. Esse processo incorre em algumas complicações e algumas das quais serão mostradas a seguir.

4.3.2 Problemas com a Administração de Chaves

Uma questão particularmente complexa de se tratar é o conhecido problema de distribuição de chaves. Se A e B desejam trocar uma mensagem utilizando-se do sistema simétrico, então é necessário que eles combinem a chave a ser usada antes da troca propriamente dita acontecer. Para tanto, é necessário que um canal seguro seja utilizado, o que raramente é possível.

A situação se agrava ainda mais quando o número de usuários é demasiadamente grande — usuários de internet, por exemplo. Isso se deve ao fato de que, existindo n usuários, se cada um desejar se comunicar sigilosamente com todos os outros, cada um deverá ter as $n - 1$ chaves dos demais, além de sua própria chave. Dessa maneira haverá $n(n - 1)/2$ chaves a serem administradas ao longo da rede. Para $n = 1.000$, por exemplo, existiriam 499.500 chaves para se manipular.

Algumas soluções foram propostas. Uma delas é utilizar-se de uma terceira parte T que seja da confiança tanto de A quando de B. Nessa situação, se A desejar enviar uma mensagem para B, então A cifra a mensagem com sua chave k_A e envia a mensagem para T, que, possuindo todas as chaves existentes, decifra a mensagem de A e a cifra novamente com a chaves k_B de B, enviando-a para B. Desta forma, T deverá armazenar todas as n chaves.

Embora o método descrito no parágrafo tenha sido adaptado por sistemas renomados, como o Kerberos, ele nem sempre é praticável. Assim, surge nos fins dos anos setenta o conceito de criptografia de chave pública.

4.4 Criptografia de Chave Pública

Em 1976, Whitfield Diffie e Martin E. Hellman [DIFFIE & HELLMAN (1976)] propuseram um novo esquema criptográfico que ficou conhecido por *sistema assimétrico* ou *criptografia de chave pública*.

Esse esquema propõe o uso de duas chaves: uma chave e, para ciframento e uma chave d, para deciframento. A escolha do par (e, d) é feita de tal forma que a obtenção de d a partir de e seja computacionalmente impraticável. e pode ser tornada de conhecimento público, enquanto d é mantida em sigilo.

4.4.1 Protocolo de Troca de Chaves

O pioneirismo do trabalho de Diffie e Hellman [DIFFIE & HELLMAN (1976)] concentra-se no protocolo de troca de chaves. Como foi visto na Seção 4.3.2, o uso de sistemas simétricos torna-se problemático quando se considera a dificuldade de se combinar e se administrar chaves secretas.

A solução apresentada baseia-se na aparente dificuldade de se calcular o logaritmo discreto sob corpos primos[2] [DIFFIE & HELLMAN (1976)]. O Algoritmo 4.1 demonstra o funcionamento do protocolo.

Algoritmo 4.1 Protocolo de Diffie e Helman

Entrada: Um grupo $\mathbb{Z}_p$ e um valor $y \in \mathbb{Z}_p$, ambos de conhecimento público.

Saída: A chave secreta k.

1: A escolhe $x_a \in \{1, \cdots, p-1\}$ e o mantém secreto.
2: B escolhe $x_b \in \{1, \cdots, p-1\}$ e o mantém secreto.
3: A calcula $y^{x_a} \pmod p$ e envia este valor para B.
4: B calcula $y^{x_b} \pmod p$ e envia este valor para A.
5: A calcula $(y^{x_b})^{x_a} = y^{x_a x_b} \pmod p$.
6: B calcula $(y^{x_a})^{x_b} = y^{x_a x_b} \pmod p$.
7: **retorne** $k = y^{x_a x_b}$

Para que seja possível determinar a chave tendo conhecimento apenas dos valores que são de conhecimento público, é necessário que se resolva o *problema do logaritmo discreto*, o que, como foi dito, é impraticável.

O protocolo de troca de chaves possibilitou o aumento da segurança em ambientes onde sejam utilizados cifradores simétricos. Porém, muitos pesquisadores estenderam esta abordagem para proporem criptosistemas que exploram melhor a técnica de chave pública. Dentre eles,

[2]Embora ainda não se tenha encontrado qualquer método capaz de calcular este valor de maneira eficiente, não há também a certeza de que ele não exista.

destacam-se o RSA [RIVEST et al. (1978)], e o ElGamal [GAMAL (1985)], descritos rapidamente nas Seções 4.4.2 e 4.4.3, respectivamente.

4.4.2 O RSA

Criado por R. Rivest, A. Shamir e L. Adleman, o RSA foi o primeiro algoritmo de ciframento de chave pública bem sucedido a ser inventado [BLAKE et al. (2000)]. Sua utilização é amplamente difundida, sendo o padrão em diversos sistemas de comunicação [STINSON (1995)].

Como foi mencionado, sistemas assimétricos baseiam sua segurança em problemas matemáticos cuja solução é considerada intratável. O RSA utiliza-se da aparente dificuldade em se fatorar números inteiros grandes. Este é o conhecido problema da fotaração de inteiros [MENEZES et al. (1997)].

Geração do par de chaves

No RSA, o par de chaves pública a privada consiste nos pares de inteiros positivos (e, n) e (d, n), respectivamente. n é chamado *módulo RSA* e é gerado a partir do produto de dois primos de mesmo tamanho. O Algoritmo 4.2 demonstra a política de geração de chaves. Encontrar d a partir de d e n é, em termos computacionais, equivalente a resolver o *problema de fatoração de números interios* [HANKERSON et al. (2004)].

Algoritmo 4.2 Geração do par de chaves no RSA

Entrada: Parâmetro de segurança l

Saída: As chaves (n, e) (pública) e d (privada)

1: Escolha randomicamente os primos p e q de mesmo comprimento (em bits) de $l/2$
2: Calcule $n = p \cdot q$ e $\phi = (p-1) \cdot (q-1)$
3: Escolha um e tal que $1 < e < \phi$ e $\text{mdc}\,(e, \phi) = 1$
4: Econtre d que satisfaça $1 < d < \phi$ e $e \cdot d \equiv 1 \pmod{\phi}$
5: **retorne** (n, e, d)

Exemplo 4.1 Alice escolhe os primos $p = 2357$ e $q = 2551$ e calcula $p \cdot q = 6012707$ e $\phi = (p-1) \cdot (q-1) = 6007800$. Alice escolhe também $e = 3674911$ e $d = 422191$, visto que $e \cdot d \equiv 1 \pmod{\phi}$. Assim, a chave pública será $(3674911, 6012707)$ e a chave privada, $(422191, 6012707)$

Esquemas de Ciframento e Decifrамento

Devido à natureza dos inteiros d e e, pode-se observar o seguinte comportamento:

$$m^{ed} \equiv m \pmod{n}, \forall m \in \mathbb{Z}. \tag{4.1}$$

Os procedimentos de ciframento e deciframento são derivados da expressão 4.1. Uma mensagem representada pelo inteiro m, por exemplo, pode ser cifrada aplicando-se $c = m^e \pmod{n}$. Assim, o deciframento é feito aplicando $c^d = (m^e)^d \equiv m \pmod{m}$. Esses procedimentos podem ser vistos nos Algoritmos 4.3 e 4.4.

Algoritmo 4.3 Esquema básico de ciframento no RSA

Entrada: Chave pública (e, n) e a mensagem clara $m \in [0, n-1]$

Saída: Texto cifrado c

1: Compute $c = m^e \bmod n$

2: **retorne** c

Algoritmo 4.4 Esquema básico de deciframento no RSA

Entrada: O par (d, n) e o texto cifrado c

Saída: O texto claro m

1: Calcule $m = c^d \bmod n$

2: **retorne** m

Exemplo 4.2 Considerando o par de chaves gerado no Exemplo 4.1, é demonstrado aqui os processos de ciframento e deciframento. Se Bob quiser enviar para Alice a mensagem $m = 5234673$, ele realiza os cálculos

$$5234673^{3674911} \bmod 6012707 = 3650502,$$

enviando o resultado para Alice. Para Alice reaver a mensagem, ela aplica as operações

$$3650502^{422191} \bmod 6012707 = 5234673.$$

4.4.3 El Gamal

ElGamal, assim como o esquema de troca de chaves de Diffie e Hellman, é conhecido por ser um sistema de logaritmo discreto. Isso significa que sua segurança reside na dificuldade de se calcular o logaritmo discreto sobre corpos finitos. Foi originalmente proposto em meados da década de 1980 [GAMAL (1985)] e é hoje amplamente usado de forma adaptada.

A geração dos parâmetros públicos é resumidamente descrita no Algoritmo 4.5. Este é o procedimento que Alice deve executar para obter sua chave pública. Caso Bob queira cifrar uma mensagem e enviá-la para Alice, ele deverá proceder conforme o descrito no Algoritmo 4.6. De maneira semelhante o deciframento é feito de acordo com o Algoritmo 4.7.

A observação dos algoritmos mostra que os processos de ciframento e deciframento são baseados na propriedade aritmética 4.2.

$$\gamma^{-a} \cdot \delta \equiv \alpha^{-ak} m \alpha^{ak} \equiv m \pmod{p} \tag{4.2}$$

Algoritmo 4.5 Geração da chave pública no esquema de ElGamal

Saída: Os parâmetros públicos (p, α, α^a)

1: É gerado um primo aleatório p e é encontrado o inteiro α, gerador de $\mathbb{Z}_p^*$

2: O inteiro a, $1 \leq a \leq p-2$, é randomicamento selecionado

3: É calculado $\alpha^a \bmod p$

4: **retorne** (p, α, α^a)

Algoritmo 4.6 Ciframento por ElGamal

Entrada: Os parâmetros públicos (p, α, α^a) e o texto claro

Saída: O texto cifrado c

1: Represente o texto claro como um inteiro $m \in [0, p-1]$

2: Selecione randomicamente o inteiro $k \in [1, p-2]$

3: Compute $\gamma = \alpha^k \bmod p$ e $\delta = m \cdot (\alpha^a)^k \bmod p$

4: **retorne** $c = (\gamma, \delta)$

4.5 Assinatura Digital

Contratos, transações bancárias e correspondências e outros tipos de documentos eletrônicos que envolvam algum tipo de responsabilidade civil são exemplos de situações onde não somente a privacidade é necessária, mas também a autenticidade. Quando um documento de papel é assinado à mão, há uma garantia de legitimidade de seu proprietário: somente ele é capaz de produzir a assinatura e qualquer um é capaz de vê-la e reconhecê-la.

Essa Seção introduz o conceito de assinatura digital e mostra como ela é aplicada utilizando os principais esquemas de chave pública conhecidos.

4.5.1 Descrição

Definição 4.2 *A assinatura digital de uma mensagem é um valor numérico que depende de alguma informação secreta conhecida somente por seu proprietário. Ela pode conter também o conteúdo da mensagem que está sendo assinada.*

Esquemas de assinatura digital podem ser facilmente implementados baseando-se nos conceitos de criptografia de chave pública. Seja d a chave privada de Alice, que deseja assinar uma mensagem m A única coisa que Alice dever fazer é cifrar a m utilizando d. Como as funções

Algoritmo 4.7 Deciframento por ElGamal

Entrada: Os parâmetros públicos (p, α, α^a) e o texto cifrado c

Saída: O texto claro m

1: Calcule $m = \gamma^{p-1-a} \bmod p$

2: **retorne** m

de ciframento e deciframento são inversas, é possível reaver m submetendo-a à chave pública e de Alice. Isso cumpre o papel da assinatura: somente Alice é capaz de assinar uma mensagem (somente ela te acesso à sua chave privada) que qualquer um é capaz de verificar a autenticidade da assinatura (visto que e é de conhecimento público).

Uma mensagem com sua assinatura verificada oferece três garantias:

1. *Integridade da mensagem*: a mensagem não foi alterada enquanto estava a caminho do destinatário;

2. *Origem da mensagem*: a mensagem foi realmente enviada por Alice;

3. *Não-repúdio*: Alice não pode alegar não ter enviado a mensagem.

Todos os esquemas de chave pública mostrados nesse trabalho implementam algum tipo de assinatura digital.

4.6 Considerações finais

Este capítulo apresentou os principais aspectos referentes aos sistemas criptográficos, enfatizando os sistemas de chave pública mais amplamente utilizados hoje. As discursões a cerca de sistemas de curvas elípticas são tratadas no Capítulo 5.

Apenas as questões teóricas foram tratadas aqui. Uma descrição sobre os principais sistemas de segurança implementados hoje pode ser encontrada na obra de William Stallings [STALLINGS (2002)], que reúne diversas informações relativas a segurança em ambientes de redes de computadores.

A próxima parte deste trabalho trata das aplicações criptográficas de curvas elípticas.

Parte II

Aplicações Criptográficas

Capítulo 5

Criptossistemas de Curvas Elípticas

5.1 Introdução

Com a descoberta da dificuldade de se resolver o problema do logaritmo discreto em grupos de pontos de curvas elípticas, estas se tornaram uma atraente alternativa para a criação de criptossitemas de chave pública. Muitos protocolos já conhecidos, que baseiam sua segurança no problema do logaritmo discreto em grupos de ordem prima, como é o caso do protocolo de Diffie e Helman, do sistema de ElGamal e do DSA foram revistos e adaptados para agirem sobre essa nova categoria de grupos.

O problema do logaritmo discreto em curvas elípticas é o tema do Capítulo 6, onde se ele é definido e seus principais ataques são detalhados. Entretanto, como ele será amplamente citado e utilizado nesse capítulo, é razoável que sua definição formal seja dada também aqui:

Definição 5.1 (O problema do logaritmo discreto em curvas elípticas (PLDCE)) *Dada uma curva elípticas E, definida sobre um corpo finito $\mathbb{F}_q$, um ponto $P \in E(\mathbb{F}_q)$ (de ordem n) e um ponto $Q \in \langle P \rangle$, o problema do logaritmo discreto em curvas elípticas consiste em encontrar o inteiro $l \in [0, n-1]$ tal que $Q = lP$. O inteiro l é chamado logaritmo discreto de Q na base P, sendo representado por $l = \log_P Q$.*

Embora a definição original do problema do logaritmo discreto tenha sido elaborada para grupos multiplicativos e não para grupos aditivos, como é o caso dos grupos de pontos de uma curva, as propriedades entre eles são equivalentes.

A dificuldade de se resolver esse problema quando aplicado a curva elípticas traz o benefício de se poder usar parâmetros criptográficos menores (quando comparados a outros sistemas) e ainda assim manter o mesmo nível de segurança, podendo ser empregado em meios onde há limitação de recursos — como memória e poder de processamento [MIRANDA (2002)]. Um bom exemplo desse tipo de aplicação é a segurança em *smart cards* [PIETILäINEN (2000)] e o uso da tecnologia Java Card [CALEGARI (2002)].

Este capítulo descreve a versão elíptica destes dois protocolos, detalhando seus algoritmos e como eles podem ser implementados. Inicialmente é discutido o procedimento para se representar mensagens no formato de pontos de uma curva. A seguir, são definidos os parâmetros públicos que serão utilizados pelos criptossistemas. Por fim, a versão elíptica do esquema de Diffie e Helman é apresentada assim como a versão do sistema de ElGamal.

5.2 Transformando uma texto em pontos de uma curva

Todo processo de ciframento nos criptossistemas baseados em curvas elípticas são feitos sobre pontos, conforme será mostrado nos algoritmos deste capítulo. Dessa maneira, é necessário que seja estabelecida um método que possibilite transformar unidades de um texto em pontos de uma curva, para que os procedimentos criptográficos possam ser aplicados. O método descrito nessa seção foi discutido por [SARTORI (2006)].

Seja uma curva elíptica E, definida sobre um corpo finito $\mathbb{F}_q$ e um inteiro k. Considera-se que cada unidade do texto possa ser representada por um inteiro m, cujo valor seja menor que um certo valor M — por exemplo, caracteres ASCII, cujos valores numéricos para os caracteres correspondentes não passam de $M = 255$.

A ordem q do corpo deve obedecer à relação $q > Mk$. O procedimento consiste em escrever inteiros de 1 a Mk na forma $mk + j$, com $1 \leq j \leq k$. Isso gera um correspondência (sob a forma de uma função injetora) dos elementos desse conjunto aos elementos de $\mathbb{F}_q$.

Os pontos de uma curvas são pares que satisfazem a uma equação do tipo

$$y^2 = x^3 + ax + b \tag{5.1}$$

Dessa forma, para cada unidade de mensagem m do texto original, obtém-se uma valor dentro de $\mathbb{F}_q$ que corresponda a uma inteiro $mk + j$, $1 \leq j \leq k$. Esse inteiro corresponde ao x da Equação 5.1. Substituindo essa valor na equação, obtendo-se uma igualdade do tipo $y^2 = t$, tenta-se calcular a raiz quadrada de t. Caso essa não exista, um novo valor para j é escolhido e o processo é repetido, até que se obtenha uma valor que possibilite gerar o par (x, y).

Para os algoritmos mostrados nesse capítulo, considera-se que as unidades de mensagem já estejam codificadas na forma de pontos da curva subjacente.

5.3 Parâmetros públicos

Conforme foi mostrado no Capítulo 4, os sistemas assimétricos mantém um conjunto de parâmetros que é de conhecimento público. No protocolo de troca de chaves de Diffie e Helman, por exemplo, o inteiro y e a ordem n do grupo são de conhecimento público. Fato análogo acontece em sistemas de curvas elípticas.

Definição 5.2 (Parâmetros Públicos) *Os parâmetros públicos do esquema criptográfico de curvas elípticas é dado por uma sétupla* (q, FR, a, b, G, n, h)*, onde*

1. q *é um número que define uma potência prima* ($q = p^m$)*;*
2. FR *é a representação usada para os elementos de* $\mathbb{F}_q$*;*
3. a *e* b *são elementos pertencentes a* $\mathbb{F}_q$ *e correspondem aos coeficientes da equação da curva;*
4. G *é um ponto pertencente a* $E(\mathbb{F}_q)$*;*
5. n *é a ordem do subgrupo gerado por* G*;*
6. $h = \#E(\mathbb{F}_q)/n$*, chamado de cofator.*

Uma escolha adequada de tais parâmetros é crucial para o bom desempenho desempenho desse esquema criptográfico, sendo n o principal dentre eles [CALEGARI (2002)]. De fato, n deve ser um primo suficientemente grande, para que seja o sistema seja imune aos principais ataques existentes[1]. Estudos apontam o valor 2^{160} como adequado para tanto. É importante também que $\#E(\mathbb{F}_q) \neq q$, para que se evite ataques por isomorfismo.

Embora haja técnicas adequadas para gerar esses parâmetros públicos, eles já são fornecidos por instituições normativas internacionais, tais como o NIST[2]

5.4 Protocolo de troca de chaves – ECDH

O protocolo de troca de chaves é análogo ao apresentado na Seção 4.4.1 e detalhado no Algoritmo 4.1. Tal qual foi apresentado, aqui também Alice a Bob querem combinar uma chave secreta, para que possam posteriormente se comunicar utilizando-a em um cifrador clássico.

Tendo ambos conhecimento dos parâmetros (q, FR, a, b, G, n, h), Alice escolhe de maneira aleatória um inteiro $a \in [1, n-1]$ e calcula o ponto $P_A = aG \in \langle G \rangle$, mantendo o inteiro a em sigilo. O ponto P_A pode ser enviado para Bob por um canal provavelmente inseguro. Bob, por sua vez, escolhe um inteiro b no mesmo intervalo que Alice e, mantendo-se secreto, determina o ponto $P_B = bG \in \langle G \rangle$, enviando esse valor para Alice por meio de um canal também não necessariamente seguro. Alice realiza o cálculo de $aP_B = a(bG) = (ab)G$ e Bob, $bP_A = b(aG) = (ab)G$. Esse ponto em comum é agora utilizado como chave.

Se os valores a ou b puderem ser encontrados a partir de P_A ou P_B, então o todo o esquema poderá ser quebrado. Entretanto, isso implicaria em resolver o problema do logaritmo discreto

[1] Tais ataques estão detalhados no Capítulo 6.

[2] NIST: *National Institute of Standards and Technology* ("Instituto Nacional de Padrões e Tecnologia"). Agência do governo norte americano que promove padronizações e que realiza no campo da segurança. Mais detalhes em `http://www.nist.org`.

em curvas elípticas. Estudos realizados por diversos pesquisadores (em [BONEH & LIPTON (1996)], por exemplo) apontam a impossibilidade de se resolver esse problema de maneira eficiente, nem mesmo em tempo sub-exponencial.

5.5 Analogia ao ElGamal

Como foi visto no Capítulo 4, o esquema de ElGamal para cifrамento é uma construção que segue diretamente do protocolo de troca de chaves. Assim como ele foi apresentado para trabalhar sobre grupos multiplicativos $\mathbb{Z}_p^*$, sua adaptação para a abordagem elíptica é quase que imediata.

O par de chaves é conseguido de maneira simples. Alice, por exemplo, tendo conhecimento dos parâmetros públicos, escolhe aleatoriamente um inteiro d no intervalo $[1, n-1]$. Esse inteiro é sua chave secreta. A seguir, Alice determina o ponto $Q = dG$, que é usado como chave pública. Os esquemas de cifrамento e decifrамento do ElGamal elíptico estão indicados nos Algoritmos 5.1 e 5.2, respectivamente.

Algoritmo 5.1 ElGamal Elíptico: Cifrамento

Entrada: Os parâmetros públicos (q, FR, a, b, G, n, h), a chave pública Q e o texto claro

Saída: O texto cifrado c

1: Represente o texto claro como um ponto $M \in E(\mathbb{F}_q)$
2: Selecione um inteiro k qualquer no intervalo $[1, n-1]$
3: Compute $C_1 = kG$
4: Compute $C_2 = M + kQ$
5: **retorne** $c = (C_1, C_2)$

Algoritmo 5.2 ElGamal Elíptico: Decifrамento

Entrada: Os parâmetros públicos (q, FR, a, b, G, n, h), a chave privada d e o texto cifrado c

Saída: O texto claro m

1: Calcule $M = C_2 - dC_1$
2: Volte M para sua representação m original
3: **retorne** m

5.6 Considerações finais

Este capítulo apresentou a maneira pela qual um grupos de pontos de uma curva elíptica pode ser utilizado em aplicações criptográficas. É importante notar que toda a segurança do

sistema está baseada na dificuldade de se resolver o problema do logaritmo discreto, definido nesses grupos.

O próximo capítulo discute as principais abordagem de ataque ao problema do logaritmo discreto.

Capítulo 6

Ataques ao Problema do Logaritmo Discreto

6.1 Introdução

Alguns sistemas criptográficos baseiam sua segurança na aparente dificuldade de se resolver o problema do logaritmo discreto. Os mais conhecidos são o protocolo de troca de chave de Diffie e Hellman [DIFFIE & HELLMAN (1976)], o sistema de Massey-Omura [KOBLITZ (1991)] e o sistema de ElGamal [GAMAL (1985)]. O criptosistema de curvas elípticas é um deles e possui a vantagem de apresentar ainda mais dificuldade para solucionar este problema que nos sistemas anteriores.

Este capítulo apresenta o problema do logaritmo discreto. Em um primeiro momento, este problema é definido para grupos quaisquer, sendo apresentados alguns dos seus mais eficientes métodos de resolução (ou ataques). A seguir, serão mostrados os mesmos ataques adaptados para grupos específicos, formados por pontos de uma curva, o que é de maior interesse para esse trabalho. Por fim, é apresentado um ataque específico, que explora as propriedades de isomorfismo entre grupos: o emparelhamento.

6.1.1 Definições preliminares

Para grupos quaisquer, o problema em estudo pode ser caracterizado por meio da definição a seguir.

Definição 6.1 *Sejam G um conjunto ao qual se associa uma operação binária que obedecem às propriedade de um grupo finito de ordem n e $b \in G$. Se a pertence ao subgrupo gerado por b, ou seja, $a \in \langle b \rangle$, então o problema do logaritmo discreto consiste em encontrar o inteiro $x \in [0, n-1]$, tal que $a = b^x$. Este valor é denotado por $x = \log_b a$*

Um abordagem intuitiva para a resolução deste problema é a busca exaustiva, que consiste em percorrer todo o conjunto G a procura do elemento x que satisfaça a relação $x = \log_b a$. A quantidade de passos executados no pior caso é de aproximadamente n passos e, segundo [HANKERSON et al. (2004)], $n/2$ passos no caso médio. Isso se torna uma situação problemática quando a ordem do do grupo é muito elevada (algo do tipo $n \geq 2^{20}$), caracterizando assim um problema computacionalmente intratável. Os métodos algoritmos aplicados a esta questão têm por objetivo melhorar esse limite assintótico $\mathcal{O}(n)$ [DULLIUS (2001)].

A Definição 6.1 é aplicada tomando por base um grupo multiplicativo qualquer. No Capítulo 3, foi mostrado que um grupo de pontos de uma curva é do tipo aditivo. Porém, a definição acima é facilmente adaptada para este novo cenário:

Definição 6.2 (O problema do logaritmo discreto em curvas elípticas (PLDCE)) *Dada uma curva elípticas E, definida sobre um corpo finito $\mathbb{F}_q$, um ponto $P \in E(\mathbb{F}_q)$ (de ordem n) e um ponto $Q \in \langle P \rangle$, o problema do logaritmo discreto em curvas elípticas consiste em encontrar o inteiro $l \in [0, n-1]$ tal que $Q = lP$. O inteiro l é chamado logaritmo discreto de Q na base P, sendo representado por $l = \log_P Q$.*

6.2 Algoritmo de Shanks

Este método encontra a solução para o logaritmo discreto em tempo $\mathcal{O}(\sqrt{n})$, onde n é a ordem do grupo [BARRETO (1999)]. Sua estratégia consiste na construção de duas tabelas,

Algoritmo 6.1 Algoritmo de Shanks

Entrada: O elemento $a \in \langle b \rangle$, com $| \langle b \rangle | = n$ e $m = \lceil \sqrt{n} \rceil$

Saída: $x = \log_b a$, com $x \in [0, n-1]$

1: **para** i de 0 até $m - 1$ **faça**
2: calcule ab^{-i}
3: **fim para**
4: Construa o conjunto S formado pelos pares ordenados (i, ab^{-i})
5: **para** j de 0 até m **faça**
6: calcule b^{mj}
7: **fim para**
8: Construa o conjunto T, formado pelos pares ordenados (j, b^{mj})
9: Encontre $(i, ab^{-i}) \in S$ e $(j, b^{mj}) \in T$, tais que $ab^{-i} = b^{mj}$ e destaque os inteiros i e j
10: **retorne** $mj + i$

O funcionamento do Algoritmo 6.1 é explicado através do O Algoritmo da Divisão (ver Apêndice A), que garante a existência dos inteiros j e i que obedecem à relação $x = mj + i$, com $0 \leq i < m$. De fato, $a = b^x = b^{mj+i} = b^{mj}b^i$. Desta forma, $ab^{-i} = b^{mj}$, com $0 \leq i < m$

e $0 \leq j < m$. Isso garante que $(i, ab^{-i}) \in S, (j, b^{mj}) \in T$, com $ab^{-i} = b^{mj}$, sempre existem. Das igualdades apresentadas, conclui-se que $\log_b a = mj + i \pmod{n}$.

Exemplo 6.1 Seja o grupo $\mathbb{Z}_{53}$. Como 53 é primo, então o grupo multiplicativo está bem definido. Assim, $\lceil \sqrt{53} \rceil = 8$. Deseja-se calcular $x = \log_2 45$, ou seja, $a = 2$ e $b = 45$. O cálculo de $ab^{-i} \bmod 53$, para $0 \leq i \leq 7$ é dada por

$$S = \{(0, 45), (1, 49), (2, 51), (3, 52), (4, 26), (5, 13), (6, 33), (7, 43)\}.$$

Já o cálculo de $b^{mj} \bmod 53$, para $0 \leq j \leq 53$ fornece

$$T = \{(0, 1), (1, 44), (2, 28), (3, 13), (4, 42), (5, 46), (6, 10), (7, 16), (8, 15)\}.$$

A comparação entre S e T mostra que os pares procurados são $(5, 13) \in S$ e $(3, 13) \in T$. Portanto, $i = 5$ e $j = 13$, donde se pode calcular $8 \cdot 3 + 5 = 29$. Portanto, $\log_2 45 = 29 \pmod{53}$.

A ato de determinar a tabela S é conhecido na literatura por "passo de bebê"e a determinação de T, por "passo de gigante". Segundo [BUCHMANN (2002)], a comparação entre os elemento dos dois conjuntos torna-se mais eficiente quando os elementos de S são armazenados em uma tabela de *hash* [CORMEN et al. (2001)].

6.3 Método ρ de Pollard

Na seção anterior, foi mostrado que o Algoritmo de Shanks é capaz de encontrar o valor do logaritmo discreto, mas para isso necessita de espaço adicional para o armazenamento das tabelas S e T. De fato, a análise deste método é de mero interesse teórico, visto que o algoritmo descrito nesta seção, proposto na década de 1970 por J.M. Pollard [POLLARD (1978)], apresenta o mesmo tempo de execução de Shanks, mas traz a vantagem de não demandar recursos de espaço para armazenamento de dados [DULLIUS (2001)].

O método ρ de Pollard é uma técnica randomizada que pode ser aplicada a um grupo cíclico qualquer. Assim, seja o grupo G um conjunto que, juntamente com uma operação binária bem definida, forma um grupo cíclico de ordem prima n. G é inicialmente dividido em três partições S_1, S_2 e S_3, de cardinalidades aproximadamente iguais. Uma condição para tal divisão é que $1 \notin S_2$.

O próximo passo consiste no cálculo da seqüencia $x_0, x_1, x_2, ...$, composta por elementos de G, e com $x_0 = 1$ e

$$x_{i+1} = f(x_i) = \begin{cases} \beta \cdot x_i, & \text{se } x_i \in S_1, \\ x_i^2, & \text{se } x_i \in S_2, \\ \alpha \cdot x_i, & \text{se } x_i \in S_3, \end{cases} \tag{6.1}$$

para $i \geq 0$. Esta seqüencia é na verdade duas outras seqüências $a_0, a_1, a_2, ...$, e $b_0, b_1, b_2, ...$, que satisfazem à relação $x_i = \alpha^{a_i}\beta^{b_i}$, para $i \geq 0$, com $a_0 = b_0 = 0$,

$$a_{i+1} = \begin{cases} a_i, & \text{se } x_i \in S_1, \\ 2a_i \bmod n, & \text{se } x_i \in S_2, \\ a_i + 1 \bmod n, & \text{se } x_i \in S_3, \end{cases} \tag{6.2}$$

e

$$b_{i+1} = \begin{cases} b_i + 1 \bmod n, & \text{se } x_i \in S_1, \\ 2b_i \bmod n, & \text{se } x_i \in S_2, \\ b_i, & \text{se } x_i \in S_3. \end{cases} \tag{6.3}$$

Utiliza-se a seguir o algoritmo "Floyd's cycle-finding" [KNUTH (1997)] que, dado o par (x_i, x_{2i}), computa $(x_{i+1}, x_{2i+2}) = (f(x_i), f(f(x_{2i})))$. Desta maneira é possível encontrar dois elementos do grupo tais que $x_i = x_2 i$. Desta forma, $\alpha^{a_i}\beta^{b_i} = \alpha^{a_{2i}}\beta^{b_{2i}}$, o que implica em $\beta^{b_i - b_{2i}} = \alpha^{a_{2i} - a_i}$

$$\begin{aligned} \alpha^{a_i}\beta^{b_i} = \alpha^{a_{2i}}\beta^{b_{2i}} &\Rightarrow \beta^{b_i - b_{2i}} = \alpha^{a_{2i} - a_i} \\ &\Rightarrow \log_\alpha \beta^{b_i - b_{2i}} = \log_\alpha \alpha^{a_{2i} - a_i} \\ &\Rightarrow (b_i - b_{2i}) \log_\alpha \beta = a_{2i} - a_i \\ &\Rightarrow \log_\alpha \beta = \frac{a_{2i} - a_i}{b_i - b_{2i}}, \end{aligned}$$

devendo-se enfatizar que todas as operações ocorrem módulo n. Desde que $b_i \not\equiv b_{2i} \pmod n$, então o logaritmo discreto estará calculado.

Algoritmo 6.2 Método ρ de Pollard para o cálculo do logaritmo discreto.

Entrada: Um gerador α de um grupo cíclico de conjunto G de ordem prima n e um elemento $\beta \in G$.

Saída: O logaritmo discreto $x = \log_\alpha \beta$

1: $x_0 \leftarrow 1, a_0 \leftarrow 0, b_0 \leftarrow 0$
2: **para** $i = 1, 2, ...$ **faça**
3: calcule x_i, a_i, b_i e x_{2i}, a_{2i}, b_{2i} utilizando as equações (6.1), (6.2) e (6.3), respectivamente
4: **se** $x_i = x_{2i}$ **então**
5: $r \leftarrow b_i - b_{2i} \bmod n$
6: **se** $r = 0$ **então**
7: **retorne** "falha"
8: **fim se**
9: **retorne** $x = r^{-1}(a_{2i} - a_i) \bmod n$
10: **fim se**
11: **fim para**

Quando a linha 7 é executada, ou seja, quando o algoritmo falha, o procedimento pode ser tentando novamente, utilizando-se desta vez novos valores para a_0, b_0, escolhidos aleatoriamente no intervalo $[1, n-1]$.

A função f comporta-se como se gerasse valores aleatórios. De fato, isso é válido para os primeiros resultados retornados por elas, mas após um certo número de cálculos, observa-se que f se torna periódica. O nome do método deriva daí: a parte não periódica da função é a "calda" da letra grega ρ, enquanto a parte periódica é a curva que se forma.

Exemplo 6.2 Seja $\alpha = 2 \in \mathbb{Z}^*_{383}$, gerador de um subgrupo G com $n = 191$ elementos, e $\beta = 228$. G é dividido em três partições, que consistem em $x \in S_1$ se $x \equiv 1 \pmod 3$, $x \in S_2$ se $x \equiv 0 \pmod 3$ e $x \in S_3$ se $x \equiv 2 \pmod 3$, para todo $x \in G$. Executando o Algoritmo 6.2 com esses valores de entrada, os resultados intermediários são dados na Tabela 6.1.

i	x_i	a_i	b_i	x_{2i}	a_{2i}	b_{2i}
1	228	0	1	279	0	2
2	279	0	2	184	1	4
3	92	0	4	14	1	6
4	184	1	4	256	2	7
5	205	1	5	304	3	8
6	14	1	6	121	6	18
7	28	2	6	144	12	38
8	256	2	7	235	48	152
9	152	2	8	72	48	154
10	304	3	8	14	96	118
11	372	3	9	256	97	119
12	121	6	18	304	98	120
13	12	6	19	121	5	51
14	144	12	38	144	10	104

Tabela 6.1: Valores dos passo intermediários do Algoritmo 6.2, aplicado ao Exemplo 6.2 [MENEZES et al. (1997)].

Na tabela, observa-se que $x_{14} = x_{28} = 144$. Assim, calcula-se $r = (b_{14} - b_{28}) \bmod 191 = 125$, $r^{-1} = 125^{-1} \bmod 191 = 136$ e $r^{-1}(a_{28} - a_{14}) \bmod 191 = 110$. Portanto, $\log_2 228 = 110$

Até neste momento, foi visto o método para grupos cíclicos quaisquer. Considera-se agora a versão para o problema no contexto de curvas elípticas, de acordo com a Definição 6.2. O algoritmo de Pollard deve encontrar os pares (c', d') e (c'', d'') tais que

$$c'P + d'Q = c''P + d''Q,$$

onde todas as operações ocorrem módulo n. Esta fórmula pode ser reescrita da seguinte forma:

$$(c' - c'')P = (d'' - d')" = (d'' - d')lP,$$

de onde se conclui que

$$(c' - c'') \equiv (d'' - d')l \pmod{n}.$$

Multiplicando ambos os lados da equivalência pelo inverso de $(d'' - d')$, obtém-se o logaritmo discreto $l = log_P Q$:

$$l = (c' - c'')(d'' - d')^{-1} \bmod n. \tag{6.4}$$

Uma primeira abordagem para a escolha dos pares (c', d') e (c'', d'') seria escolher c e d aleatoriamente no intervalo $[0, n-1]$ e armazenar os valores c, d e $cP + dQ$ em uma linha de uma tabela, repetido este procedimento até que seja encontrada uma linha com o valor de $cP + dQ$ igual a alguma linha anterior. De acordo com o paradoxo do aniversário, essa situação ocorrerá em aproximadamente $\sqrt{\pi n/2} \approx 1,2533\sqrt{n}$ passos, o que levaria a um tempo de execução e armazenamento próximos ao algoritmo de Shanks.

Algoritmo 6.3 Método ρ de Pollard adaptado para grupos de curvas elípticas.

Entrada: O ponto $P \in E(\mathbb{F}_p)$, de ordem prima n, e $Q \in \langle P \rangle$.

Saída: O logaritmo discreto $l = \log_P Q$.

1: Selecione um valor para L
2: Selecione uma função partição $H : \langle P \rangle \to \{1, 2, ..., L\}$
3: **para** j de 1 até L **faça**
4: Selecione randomicamente $a_j, b_j \in [0, n-1]$
5: Calcule $R_j = a_j P + b_j Q$
6: **fim para**
7: Selecione aleatoriamente c', d' de intervalo $[0, n-1]$ e calcule $X' = c'P + d'Q$
8: $X'' \leftarrow X, c'' \leftarrow c', d'' \leftarrow d'$
9: **repita**
10: $j \leftarrow H(X')$
11: $X' \leftarrow X' + R_j, c' \leftarrow c' + a_j \bmod n, d' \leftarrow d' + b_j \bmod n$
12: **para** i de 1 até 2 **faça**
13: $j = H(X'')$
14: $X'' \leftarrow X'' + R_j, c'' \leftarrow c'' + a_j \bmod n, d'' \leftarrow d'' + b_j \bmod n$
15: **fim para**
16: **até** $X' = X''$
17: **se** $d' = d''$ **então**
18: **retorne** *falha*
19: **fim se**
20: **retorne** $l = (c' - c'')(d'' - d') \bmod n$

Quando se adapta o algoritmo de Pollard para solucionar esta situação, encontrando os pares (c', d') e (c'', d'') sem a necessidade de armazená-los, deve-se inicialmente estabelecer

uma função f que seja aparentemente aleatória. Neste caso, $f : \langle P \rangle \rightarrow \langle P \rangle$, onde dados $X \in \langle P \rangle$ e $X = cP + dQ$, com $c, d \in [0, n-1]$, f deve ser capaz de retornar iterativamente, $\bar{X} = \bar{c}P + \bar{d}Q$, com $\bar{c}, \bar{d} \in [0, n-1]$. A seguir, $\langle P \rangle$ é dividido em L conjuntos $\{S_1, S_2, ..., S_L\}$, de cardinalidades aproximadamente iguais. Por fim, estabelece-se a função partição, dada por $H(X) = j$, que indica a partição que um dado elemento X pertence. Assim, a função $f(X)$ fica melhor definida por

$$f(X) = X + a_j P + b_j Q,$$

onde $j = H(X)$ e a_j e b_j são randomicamente escolhidos em $[0, n-1]$.

Assim como no caso geral, as colisões são calculadas utilizando-se também do algoritmo "Floyd's cycle-finding", que determina o par (X_i, X_{2i}) até que seja encontrado um par em que $X_i = X_{2i}$. Segundo [HANKERSON et al. (2004)], sendo f uma função randômica, o número médio de testes executados para se encontrar uma colisão deste tipo é de $1,0308\sqrt{n}$ e o número de operações é de cerca de $3\sqrt{n}$. Todo o processo pode ser acompanhado no Algoritmo 6.3.

Exemplo 6.3 Seja a curva $E(\mathbb{F}_{229}) : y^2 = x^3 + x + 44$. Como 229 é primo, o grupo de pontos desta curva é cíclico, tendo assim ordem $n = 229$. Seja $P = (2, 116) \in E(\mathbb{F}_{229})$ e $Q = (155, 166) \in \langle P \rangle$. Este exemplo mostra as etapas necessárias para o cálculo de $l = \log_P Q$, seguindo o Algoritmo 6.3.

Inicialmente, determina-se $L = 4$ e a função $H : \langle P \rangle \rightarrow \{1, 2, 3, 4\}$, dada por

$$H(x, y) = (x \bmod 4) + 1.$$

Executando o laço da linha 6, são gerados os seguintes valores:

$$[a_1, b_1, R_1] = [79, 163, (135, 117)]$$
$$[a_2, b_2, R_2] = [206, 19, (96, 97)]$$
$$[a_3, b_3, R_3] = [87, 109, (84, 62)]$$
$$[a_4, b_4, R_4] = [219, 68, (72, 134)],$$

o que gera os valores intermediários apresentados na Tabela 6.2. O valor adequado é encontrado na 12ª iteração. Através desses valores, calcula-se

$$l = (192 - 213) \cdot (104 - 24) \bmod 239 = 176,$$

que corresponde ao valor do logaritmo discreto l.

6.4 Algoritmo index calculus: descrição matemática e seus efeitos

Estudos mostram que a técnica index calculus é a mais poderosa abordagem para o tratamento do problema do logaritmo discreto. Este algoritmo é processado em três etapas, duas das

i	c'	d'	X'	c''	d''	X''
-	54	175	(39,159)	54	175	(39,159)
1	34	4	(160, 9)	113	167	(130,182)
2	113	167	(130,182)	180	105	(36, 97)
3	200	37	(27, 17)	0	97	(108, 89)
4	180	105	(36, 97)	46	40	(223,15)
5	20	29	(119,180)	232	127	(167, 57)
6	0	97	(108, 89)	192	24	(57,105)
7	79	21	(81,168)	139	111	(185,227)
8	46	40	(223,153)	193	0	(197, 92)
9	26	108	(9, 18)	140	87	(194,145)
10	232	127	(167, 57)	67	120	(223,153)
11	212	195	(75,136)	14	207	(167, 57)
12	192	24	(57,105)	213	104	(57,105)

Tabela 6.2: Valores dos passo intermediários do Algoritmo 6.3, aplicado ao Exemplo 6.3 [HANKERSON et al. (2004)].

quais são independentes do grupo onde o logaritmo é definido (as primeiras) e podem ser executadas uma única vez, com seus resultados utilizados para o cálculo de diversos logaritmos sob uma mesma base. A terceira e última etapa se utiliza de particularidades do grupo subjacente.

Esta seção descreve o processo, mostrando seu bom desempenho quando aplicado em grupos quaisquer e elucidando o porque de sua ineficiência quando utilizado em grupos de pontos de uma curva elíptica.

6.4.1 Descrição geral

Conforme foi dito no início desta seção, o algoritmo index calculus é dividido em três etapas, a saber:

1. Seleção dos fatores base,
2. Cálculo dos logaritmos dos elementos encontrados na etapa anterior,
3. O cálculo do logaritmo desejado.

Na descrição que se segue, sejam G um grupo multiplicativo e $b \in G$, que gera um subgrupo cíclico de ordem prima n. Para $a \in \langle b \rangle$, deseja-se calcular x tal que

$$x \equiv \log_b a \pmod{n}.$$

O Algoritmo 6.4 detalha a metodologia a ser seguida. A primeira etapa inicia-se na linha 1 e termina na linha 4. Ela deixa clara o fato deste método não ser geral, visto que diferentes caminhos podem ser tomados para se resolver a congruência aí apresentada, dependendo do

Algoritmo 6.4 Index Calculus

Entrada: $b \in G$, gerador de um subgrupo cíclico de ordem n prima e $a \in \langle b \rangle$ e os elementos $p_1, p_2, ..., p_m \in \langle b \rangle$

Saída: $x = \log_b a \pmod{n}$

1: **repita**
2: Escolha aleatoriamente um elemento $g_i \in \mathbb{N}$ e determine $r_i = b^{g_i}$
3: **até** r_i puder ser fatorado como um produto de $(p_1, p_2, ..., p_m)$
4: Obtenha a relação $b^{g_i} = \prod_{j=1}^{m} p_j^{a_{ij}}$
5: Resolva o sistema de congruências lineares dado por $g_i \equiv \sum_{j=1}^{m} a_{ij} \log_b p_j \pmod{n}$
6: **repita**
7: Escolha aleatoriamente $e_t \in \mathbb{N}$ e calcule $l_t = ab^{e_t}$
8: **até** l_t puder ser fatorado como um produto de $(p_1, p_2, ..., p_m)$.
9: Obtenha $l_t = \prod_{j=1}^{m} p_j^{u_j}$
10: **retorne** $x = \log_b a \equiv \sum_{j=1}^{m} u_j \log p_j - e_t \pmod{n}$

grupo em que se trabalha. A segunda etapa é executada na linha 5. A última inicia-se na linha 6.

Para um maior esclarecimento, a próxima seção mostra o método aplicado a um grupo multiplicativo $\mathbb{Z}_p$.

6.4.2 Index calculus aplicado em $\mathbb{Z}_p$

Seja o grupo $\mathbb{Z}_p$, com p primo e b seu gerador. Conforme foi visto no Capítulo 2, esse grupo corresponde a um corpo finito de ordem $n = p - 1$.

Seguindo os passos estabelecidos na seção anterior, primeiro se escolhe aleatoriamente um elemento g_i, tal que $1 \leq g_i \leq n$ e calcula-se $r_i \leq b^{g^i} \pmod{p}$. O conjunto $\{p_1, p_2, ..., p_m\}$ escolhido é composto pelos m primeiros números primos. Este conjunto é usado na tentativa de fatorar r_i e, quando da fatoração é obtida, este pode ser escrito como

$$b^{g_i} = \prod_{j=1}^{m} p_j^{a_{ij}} \tag{6.5}$$

A equação 6.5 é então reescrita por meio da utilização das propriedades de logaritmos:

$$\begin{aligned} b^{g_i} &= \prod_{j=1}^{m} p_j^{a_{ij}} \\ \log_b b^{g_i} &= \log_b \prod_{j=1}^{m} p_j^{a_{ij}} \\ g_i &= \sum_{j=1}^{m} \log_b p_j^{a_{ij}} \end{aligned}$$

$$g_i \;=\; \sum_{j=1}^{m} a_{ij} \log_b p_j,$$

que equivale ao sistema de congruências lineares

$$g_i \equiv \sum_{j=1}^{m} a_{ij} \log_b p_j \pmod{n}, \tag{6.6}$$

que é resolvido na segunda etapa do processo.

Inicia-se então o cálculo do logaritmo discreto propriamente dito. Tendo escolhido o valor e_t que possibilite sua representação como um produto dos m primeiros números primos, determina-se

$$l_t = ab^{e_t} \pmod{p}. \tag{6.7}$$

Como $l_t = \prod_{j=1}^{m} p_j^{u_j}$, a equação 6.7 pode ser trabalhada da seguinte maneira:

$$\begin{aligned}
ab^{e_t} &= \prod_{j=1}^{m} p_j^{u_j} \\
a &= \prod_{j=1}^{m} p_j^{u_j} \cdot (b^{e_t})^{-1} \\
\log_b a &= \log_b \prod_{j=1}^{m} p_j^{u_j} \cdot (b^{e_t})^{-1} \\
&= \log_b \prod_{j=1}^{m} p_j^{u_j} - \log_b b^{e_t} \\
&= \sum_{j=1}^{m} \log_b p_j^{u_j} - e_t \log_b b \\
&= \sum_{j=1}^{m} u_j \log_b p_j - e_t
\end{aligned}$$

Portanto,

$$\log_b a \equiv \sum_{j=1}^{m} u_j \log_b p_j - e_t \pmod{n}. \tag{6.8}$$

A análise do tempo de execução deste algoritmo é demasiadamente complicada e foge aos objetivos deste texto. Porém, as análises apresentadas por [MENEZES et al. (1997)], [ODLYZKO (1999)] e [DULLIUS (2001)] apontam um tempo de execução sub-exponencial, quando aplicado a grupos finitos como os mostrados nessa seção. Este é o melhor desempenho de todos os algoritmos mostrados neste capítulo até agora.

6.4.3 Index Calculus em grupos de pontos de uma curva elíptica

Conforme foi visto, o index calculus apresenta um bom desempenho quando aplicado a grupos de números inteiros. Analisa-se agora o caso onde o grupo subjacente é composto de pontos de uma curva elíptica. Esta seção se baseia nos resultados apresentados por [HANKERSON et al. (2004)].

Seja $E(\mathbb{F}_p) : y^2 = x^3 + ax + b$, com $\#E(\mathbb{F}_p)$ prima (um grupo cíclico) e com gerador $\langle P \rangle$. Antes de se aplicar o index calculus é necessário transformar a curva E em outra curva $\widetilde{E}$, definida sobre $\mathbb{Q}$. Isso equivale a $\widetilde{E} : y^2 = x^3 + \widetilde{a}x + \widetilde{b}$, com $\widetilde{a}, \widetilde{b} \in \mathbb{Q}$, sendo $a = \widetilde{a} \bmod p$ e $b = \widetilde{b} \bmod p$. Observa-se que na seção anterior, elementos de $\mathbb{Z}_p$ formam representados como elementos de $\mathbb{Z}$ na primeira etapa de index calculus. Analogamente, na versão elíptica um ponto $R \in E(\mathbb{F}_p)$ é representado como um ponto $\widetilde{R} \in \widetilde{E}(\mathbb{Q})$.

Desta forma, $\widetilde{kP}$ pode ser escrito como uma combinação linear de pontos em $\widetilde{E}(\mathbb{Q})$:

$$\widetilde{kP} = c_1\widetilde{P}_1 + c_2\widetilde{P}_2 + \cdots + c_r\widetilde{P}_r,$$

que pode ser reescrita como

$$kP = c_1P_1 + c_2P_2 + \cdots + c_rP_r$$

em $E(\mathbb{F}_p)$.

Ainda não se conhece um procedimento eficiente para se transformar pontos de $E(\mathbb{F}_p)$ em pontos de $\widetilde{E}(\mathbb{Q})$. Além disso, foi provado que a quantidade de pontos em $\widetilde{E}(\mathbb{Q})$ que apresentam propriedades de interesse neste método é muito pequeno, o que representa uma pequena fração de pontos em $E(\mathbb{F}_p)$. Isso torna o problema do logaritmo discreto sobre curvas elípticas seguro contra o index calculus.

6.5 Ataques por isomorfismo

Até o momento, pode-se perceber a dificuldade de se resolver o problema do logaritmo discreto quando este é definido sobre um grupo de pontos de uma curva elíptica. Entretanto, há casos onde um grupo de pontos de uma curva é isomorfo a outro grupo, cuja solução do logaritmo discreto seja mais eficiente. Isso constitui os ataques por emparelhamento ou por isomorfismo.

Seja uma curva E, definida sobre um corpo $\mathbb{F}_q$ e um ponto $P \in E(\mathbb{F}_q)$, cujo subgrupo gerado $\langle P \rangle$ tem ordem prima n. Assim, $\langle P \rangle$ é cíclico, sendo isomorfo a outro grupo de conjunto G, de mesma ordem n. Isso pode ser representado por

$$\psi : \langle P \rangle \rightarrow G \tag{6.9}$$

Instâncias do problema da logaritmo discreto em curva elípticas podem ser resolvidas por meio da utilização de outros grupo, por meio da função de isomorfismo mostrada na Equação 6.9, ou seja

$$\log_P Q = \log_{\psi(P)} \psi(Q) \tag{6.10}$$

Se o grupo de uma curva puder ser eficientemente reduzido a outro grupo cuja resolução do logaritmo discreto seja mais eficiente, então um ataque em tempo sub-exponencial (ou, em certos casos, mais eficiente) poderá ser efetuado. As próximas subseções apresentam as três principais abordagens para o emparelhamento.

6.5.1 Vulnerabilidade de curvas anômalas primas

Definição 6.3 *Seja E uma curva definida sobre o corpo primo $\mathbb{F}_p$. Se $\#E(\mathbb{F}_p) = p$, então E é denominada curva anômala prima.*

Curvas deste tipo apresentam seus grupos de pontos isomorfos a grupos aditivos $\mathbb{Z}_p$, ou seja, o conjunto de inteiros no intervalo $[0, p-1]$, munidos da adição módulo p. Aplicando o isomorfismo da Equação 6.10, a equação $kP = Q$ pode ser substituída por $la \equiv b \pmod{p}$, que pode ser facilmente resolvido por meio do cálculo do inverso de a. Para tanto, usa-se o algoritmo de Euclides estendido.

Por esses motivos, curvas anômalas são inadequadas para aplicações criptográficas.

6.6 Considerações finais

Este capítulo apresentou as principais metodologias de ataque ao problema do logaritmo discreto. Até o momento, não se tem notícia de qualquer ataque que seja realmente bem sucedido para curvas bem escolhidas. Entretanto, diversas pesquisas têm sido realizadas nesse sentido.

Parte III

Considerações Finais

Capítulo 7

Implementações

7.1 Introdução

Até este ponto, foram fornecidos os fundamentos teóricos necessários à compreensão de curvas elípticas e seu uso em criptografia. A aritmética de corpos finitos e das curvas elípticas, os métodos gerais de multiplicação, os principais protocolos e os principais ataques ao problema do logaritmo discretos foram analisados sob o ponto de vista teórico.

O objetivo desse capítulo é ilustrar o funcionamento computacional de todos esse conceitos. As classes utilizadas serão descritas aqui tomando por base seus comportamentos mais relevantes. O código fonte completo da implementação está disponível no Apendice B.

7.1.1 Detalhes de implementação

Os testes e a implementação foram executados utilizando-se um computador equipado com um processador Pentium (Dual Core, 2.8GHz), com 1024MB e memória RAM. O sistema operacional em uso é o Windows XP Professional e a linguagem de programação utilizada foi o Java (JDK 1.6).

7.2 Manipulando inteiros grandes

Um dos principais desafios encontrados quando se trabalha com implementações criptográficas é a manipulação de números inteiros de elevado valor. A dificuldade justifica-se baseada na constante necessidade necessidade de se trabalhar com parâmetros criptográficos seguros, os quais podem atingir o comprimento de centenas de bits em suas representações binárias. A título de exemplo, o NIST recomenda corpos de ordem prima igual a $p = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$, o que não pode ser representado por tipos primitivos da maioria das principais linguagens de programação hoje em uso.

Há, porém, mecanismos para solucionar esse problema: a utilização de uma vetor de palavras. Se um inteiro p, cujo comprimento de sua representação binária é $m = \lceil \log_2 p \rceil$ e cada inteiro é capaz de armazenar algarismos com W bits, então é possível distribuir esse valor ao longo de um vetor de inteiros de comprimento $t = \lceil m/W \rceil$. Dessa forma, um elemento $a \in \mathbb{F}_p$ pode ser representado conforme o mostrado na Figura 7.1. Essa nova representação implica em

$A[t-1]$	$\cdots$	$A[2]$	$A[1]$	$A[0]$

Figura 7.1: $a = 2^{(t-1)W}A[t-1] + \cdots 2^{2W}A[2] + 2^{W}A[1] + A[0]$

uma nova abordagem ao se trabalhar com números inteiros, o que se reflete na maneira como as operação básicas como adição, subtração são executadas.

A linguagem de programação Java já oferece uma interessante ferramenta para manipular inteiros grandes: a classe `BigInteger`. Sendo uma classe disponível no pacote `java.math`, seu funcionamento interno é semelhante ao que foi descrito no parágrafo anterior. Essa classe é dotada de métodos que tratam a maioria das rotinas essenciais de maneira otimizada, oferecendo também suporte à aritmética modular e a testes de primaridade — ambos muito úteis aos trabalhos em criptografia. As implementações feitas para este trabalho utilizaram-na amplamente.

7.3 Aritmética dos corpos

7.3.1 A escolha do tipo de corpo

O corpo escolhido para servir de base para a implementação apresentada é do tipo primo. Essa escolha justifica-se nas facilidades de se manipular inteiros grandes de forma modular, como foi discutido na seção anterior.

```
    public class Fp {
        private Fp(BigInteger p){ ... }
        public Fp(BigInteger a, BigInteger p){ ... }
        public Fp getRandomico(SecureRandom rand){ ... }
        public Fp sub(Fp b){ ... }
        public Fp soma(Fp b){ ... }
        public Fp incr(){ ... }
        public Fp decr(){ ... }
        public Fp negativo(){ ... }
        public Fp mult(Fp b){ ... }
        public Fp multEscalar(int n){ ... }
        public Fp quadrado(){ ... }
        public Fp inverso() throws ArithmeticException{ ... }
        public Fp div(Fp b){ ... }
        public BigInteger getP(){ ... }
        public boolean equals(Fp b){ ... }
        public String toString(){ ... }
        public boolean isZero(){ ... }
        public Fp valorDe(int n){ ... }
        public BigInteger getA(){ ... }
        public Fp raiz() { ... }
}
```

A classe corpo recebeu o nome de `Fp`. É construída mediante a passagem de dois objetos BigInteger, que representa o elemento a ser representado e o número primo que define o corpo, respectivamente, ou com a passagem de uma primo somente. Nesse último caso, o elemento que o corpo representa é o zero.

As operações de adição, subtração, e multiplicação foram implementadas de maneira mais simples: a operação é realizada e o resultado é reduzido módulo p. O código para a multiplicação, por exemplo, é dado por:

```
public Fp mult(Fp b){
    return new Fp(a.multiply(b.a).mod(p), p);
}
```

7.4 Aritmética das curvas

A aritmética das curvas segue diretamente de sua definição matemática. Sendo uma curva do tipo $y^2 = x^3 + ax + b$, ela é construída a partir de de dois objetos da classe `Fp`.Após a construção do objeto EC, o valor de Δ pode ser verificado, a fim de se assegurar a existência da curva.

```
public class EC {
    public EC(Fp a, Fp b) { ... }
    public boolean pertence(Ponto Q){ ... }
    public Fp delta(){ ... }
    public Fp getY(Fp x){ ... }
    public  Ponto codifica(char c){ ... }
    public char decodifica(Ponto c){ ... }
    public Fp getA(){ ... }
    public Fp getB(){ ... }
    public String toString(){ ... }
}// fim da classe EC
```

Nesse trabalho foi implementado um método de ciframento, conforme será mostrado ainda nesse capítulo. Entretanto tal ciframento é feito sobre pontos de uma curva não sobre caracteres. Dessa maneira, foi necessário estabelecer uma relação bionívoca que associasse cada letra ou símbolo a um único ponto da curva. Com esse objetivo, foi implementado o método `codifica`, mostrado abaixo.

```
public  Ponto codifica(char c){
    int m = (int) c;
    Fp x, y;

    for(int j = 0; j < k; j++){
        x = a.valorDe(m*k + j);
        y = x.quadrado();
        if(y != null)
            return new Ponto(x, y , this);
    }

    return null;
}
```

Deve-se ressaltar que esse método não realiza o ciframento; ele apenas fornece uma nova representação ao caracter ao qual se deseja trabalhar.

Este método trabalha sobre a tabela de caracteres ASCII. Como cada caracter está associado a um inteiro m, cujo valor varia de 0 a 127. Empregando o procedimento descrito na Seção 5.2, tal ponto é facilmente calculado.

Da mesma forma, foi implementado o método `decodifica`, que extrai de cada ponto o seu caracter associado. Esse método é mostrado a seguir.

```
public char decodifica(Ponto c){
    int ch = c.getX().getA().divide(bigK).mod(bigM).intValue();
    return (char) ch;
}
```

7.5 Manipulação de pontos de uma curva

Uma das classe de maior interesse na implementação realizada é que manipula pontos de uma curva. Para cada objeto dessa classe instanciado, são atribuídos dois elementos de um corpo, correspondentes às coordenadas x e y, além de uma referência à curva ao qual ele pertence. Os métodos dessa classe são mostrados correspondem a:

```
public class Ponto { ... }
    public Ponto(){
    public Ponto(Fp x, Fp y, EC c) {
    public boolean infinito(){
    public Ponto negativo(){
    public Fp getX(){
    public Fp getY(){
    public Ponto dup(){
    public Ponto soma(Ponto b){
    public Ponto sub(Ponto b){
    public boolean equals(Ponto b){
    public Ponto multSimples(BigInteger n){
    public Ponto multtBin(BigInteger n){
    public Ponto multNFA(BigInteger n){
    public  void fna(BigInteger k, int[] s){
    public String toString(){
}
```

Quando o construtor padrão é utilizado, o objeto criado corresponde ao ponto no infinito. A variável booleana inf define se ele é ou não um ponto desse tipo.

É nesta classe que estão definidas as operações da lei de grupo. Além da adição e da duplicação de pontos, outra operação de crucial interesse criptográfico está presente aqui: a multiplicação de um ponto por um escalar, cujos métodos são analisados na próxima seção.

7.5.1 Uma comparação entre os métodos gerais de multiplicação escalar

Três métodos de multiplicação escalar de pontos foram apresentados no Capítulo 3. O primeiro parte da definição: para encontrar um ponto P multiplicado pelo inteiro k, executa-se k adições para chegar a esse valor.

Na tentativa de melhorar este método, diversos métodos foram propostos. Os mais comuns (estudados nesse trabalho) exploram a expansão binária do inteiro a ser usado no cálculo, a fim de diminuir a quantidade de adições e duplicações envolvidas no processo. Dessa forma, foram analisados o método binário e sua variação, que se utiliza da forma não-adjacente.

Para a análise, foi adotada uma curva descrita por:

$$\begin{aligned} p &= 2^{130} + 169, \\ a &= 3, \\ b &= 1043498151013573141076033119958062900890, \\ \#E(\mathbb{F}_p) &= 136112946768375385380880778449568887423 7, \end{aligned}$$

uma curva de ordem prima. Um ponto P pertencente a essa curva foi aleatoriamente escolhido e operado com um inteiro K. Três instâncias de entrada foram testadas: inteiros do tipo 2^n, $2^n - 1$ e inteiros quaisquer.

- *Inteiros do tipo* 2^n. São caracterizados por apresentarem em sua representação binária apenas um dígito diferente de zero, que corresponde ao algarismo mais significativo. Para esse caso, ambos os algoritmos apresentam um bom resultado. O gráfico contendo tais resultados é apresentado na Figura 7.2

- *Inteiros do tipo* $2^n - 1$. São caracterizados por não apresentarem algarismo zeros na representação binária. Em casos como esse, o método binário não consegue executar eficientemente, visto que todas as operações devem ser feitas. O método binário com FNA, por outro lado, obteve bons resultados, visto que um pré-processamento elimina o excesso de dígitos diferentes de zero na representação binária. Isso está ilustrado na Figura 7.3

- *Inteiros quaisquer*. Testes para inteiros sem uma propriedade específica também foram realizados. Tais casos (prática, mais comuns) serviram para mostrar qual dos métodos é, em média, mais eficiente. As conclusões relativas à eficiência da método que se utiliza da forma não-adjacente já haviam foram apresentadas no Capítulo 3. Esse bom desempenho foi comprovada e é mostrada na Figura 7.4. Para esse teste, foram utilizados inteiros com 58 bits e quantidades quaisquer de dígitos zeros e uns.

Devido aos resultados apresentados, a o método `multNFA` se tornou o método padrão de multiplicação nas atividades realizadas.

O código completo dos testes realizados encontram-se também no Apêndice B.

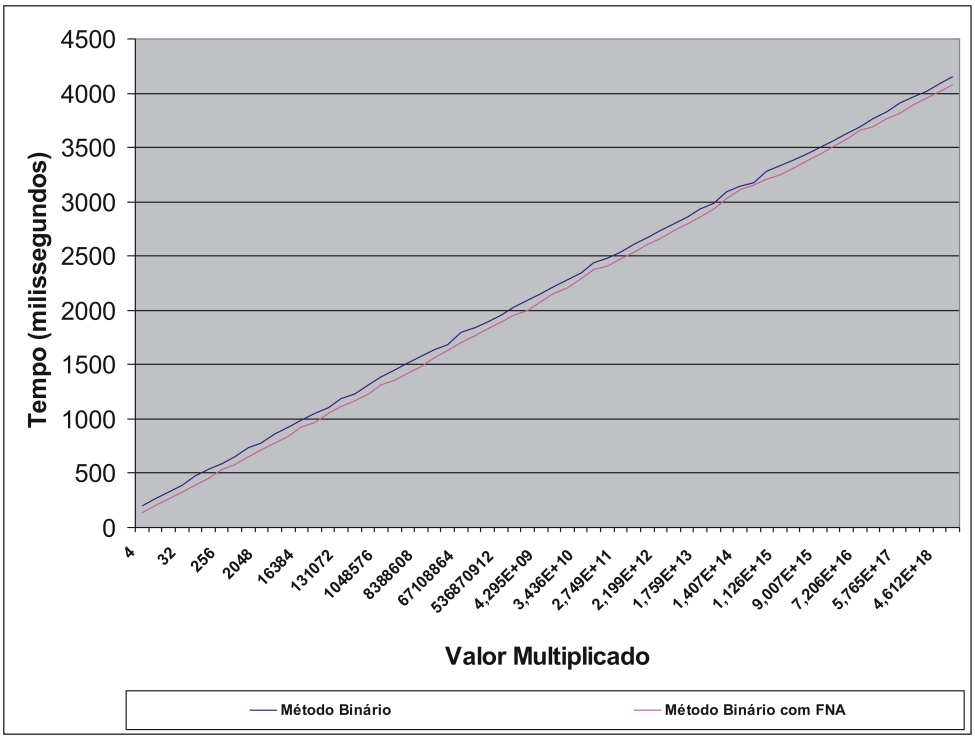

Figura 7.2: Comparação entre os tempos de execução dos métodos de multiplicação binário e o método binário com a forma não-adjacente para valores do tipo 2^n. Os dois algoritmos apresentam um desempenho semelhante, sendo o que se utiliza da FNA um pouco mais eficiente.

7.6 Exemplificação de um protocolo criptográfico

7.6.1 Sistema análogo ao ElGamal

Tendo implementado a aritmética dos corpos finitos e das curvas elípticas e ainda adotado um método padrão de multiplicação de pontos, foi elaborado uma programa para simular o funcionamento de um esquema análogo ao de ElGamal. Um exemplo completo de todas as etapas no processo criptográfico, gerado a partir de um programa desenvolvido, demonstra as etapas intermediárias do processo.

Conforme pode ser visto, os comportamentos da classe `ElGamal` seguem diretamente da definição do problema.

```
public class ElGamal { ... }
    public ElGamal(BigInteger p, EC E, Ponto P, BigInteger n) { ... }
    public ParDeChaves gerarChaves(){ ... }
    public Criptograma cifrar(Ponto M, Ponto Q){ ... }
    public Ponto decifrar(Criptograma c, BigInteger d){ ... }
    private BigInteger randN(){ ... }
}
```

A curva utilizada no exemplo possui os seguintes parâmetros:

$$p = 2^{130} + 169,$$

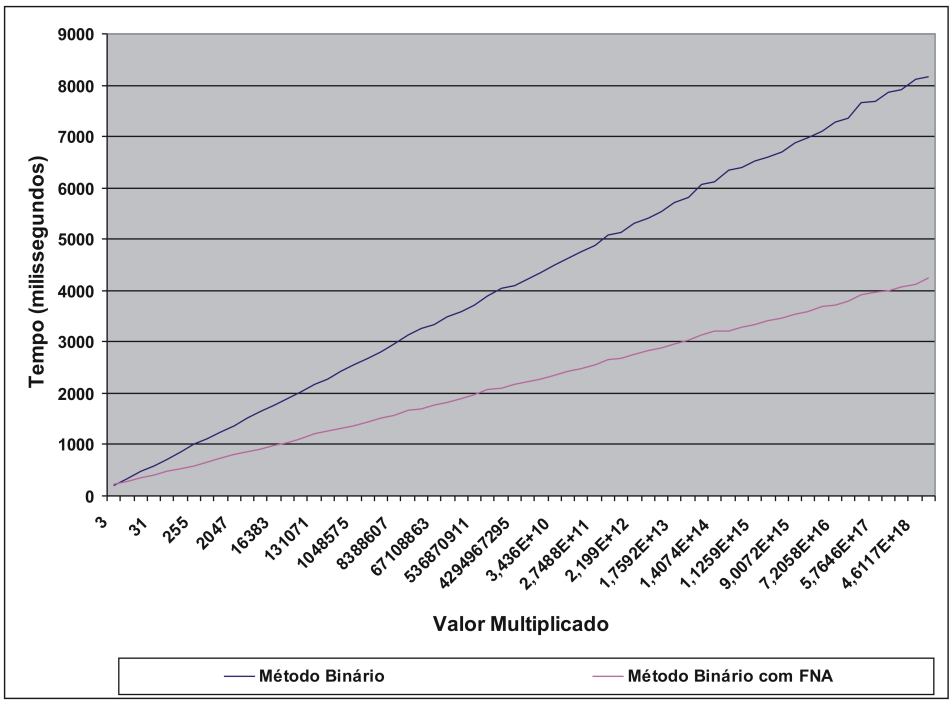

Figura 7.3: Comparação entre os tempos de execução dos métodos de multiplicação binário e o método binário com a forma não-adjacente para valores do tipo $2^n - 1$. O algoritmo que faz uso da forma não adjacente tem um desempenho muito superior nesse caso.

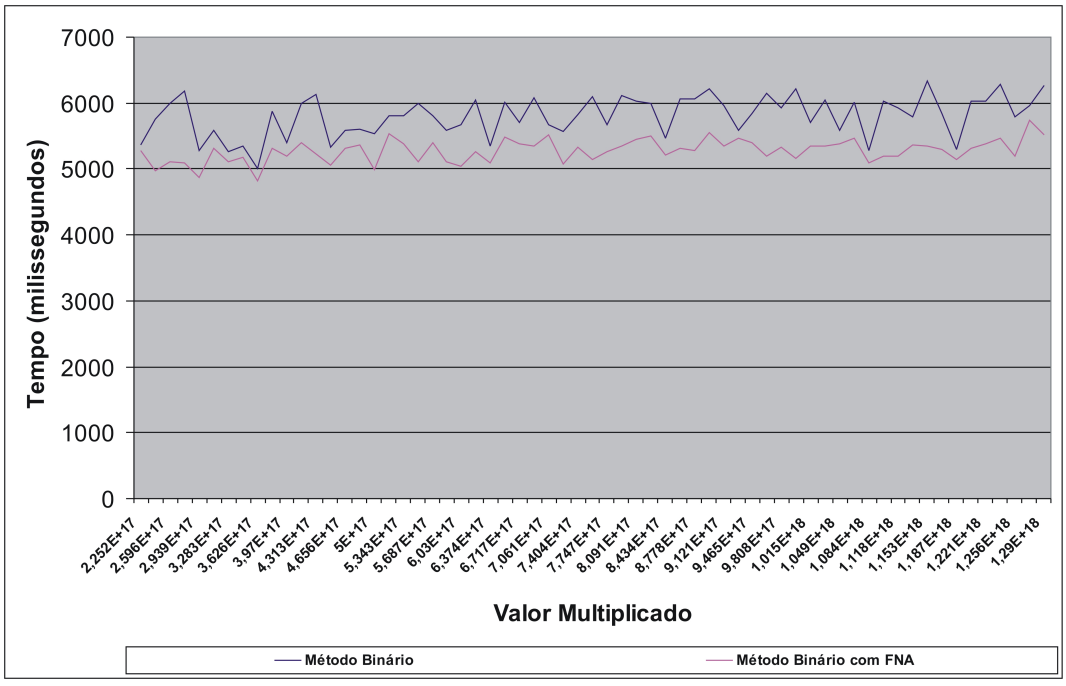

Figura 7.4: Tempo comparação entre os tempos de execução dos métodos me multiplicação binário e binário com a forma não-adjacente quando aplicados a inteiros quaisquer. Observa-se o bom desempenho do segundo algoritmo, mesmo no caso médio.

$$
\begin{aligned}
a &= 1, \\
b &= 1230929586093851880935564157041535079194, \\
\#E(\mathbb{F}_p) &= 136112946768375385384606053116008589648 3.
\end{aligned}
$$

Observa-se que este é um grupo de ordem prima.

O processo descrito no programa exemplo consiste em instanciar os objetos ec (Curva), e gamal (ElGamal), exibir a mensagem (original e sob a forma de pontos), seus criptogramas, o texto cifrado e, por fim, exibir o texto decifrado.

Eis o resultado da execução do programa exemplo:

```
Chave pública: (647544344024037808074176243714978788857,
                710096677407200699924970368600796104781)
Chave privada: 1159860682947512121025412873406089286916

Mensagem a ser cifrada: Lidiane

Mensagem representada sob a forma de pontos:
(4788,22924944),(6615,43758225),(6300,39690000),(6615,43758225),
(6111,37344321),(6930,48024900),(6363,40487769),

Criptogramas [C1,C2] que serão enviados por um canal inseguro:

[(13827231084352335027609556764769131861 1,761026618125772906355326935519624056 35),
(99997047333200912706902770176662005072 0,790297398421415877206609624467193770739)]
[(38923071495521682698648656056945446843 5,901676726785207594241018955010585565245),
(10079152929382100877819178313961791796 71,112938263549519913585801297111077330917 6)]
[(34433329706751699875398158417017640511 8,127795065767326484872083955141386360477 6),
(21133735270229932432679500306148588620 5,225968246085299502931686343293120965336)]
[(10218613493151335033877279704289129130 07,107637338990447315017921466328396425106 4),
(11619882957009125565480306193062613908 67,267175621964891807580572599324804201 66)]
[(79260075181638607099317819179680396564 3,719329370507037067733357863222813457 04),
(68931502945856050321854243894493205264 4,905771923722507474860706804254885027046)]
[(31733638077817251116976852829482857222 4,109907120736306127735312039587344368081 5),
(13095599696432323215895382704176848389 61,121041626438303835771938469656024905298 1)]
[(12194009057375337316283949852552821874 73,222487276605227970478719476613882923088),
(46357456947493130265520588324927924195 6,814509862767068886048435239725729151257)]

Esses critptogramas equivalema aos caracteres:   ?7g$t/{

Texto após o deciframento: Lidiane
```

7.7 Ataque ao problema do logaritmo discreto: método ρ de Pollard

A versão elíptica do método ρ, de Pollard foi desenvolvida, com o intuito de demonstrar um tipo de ataque ao logaritmo discreto. Um contador foi adicionado, de modo que possa ser exibida a quantidade de iterações executadas até que o resultado procurado seja obtido.

Por ser um problema intratável, o programa-exemplo é mostrado para um curva de de ordem pequena

$$E(\mathbb{F}_{29}) : y^2 = x^3 + 4x^2 + 20, \tag{7.1}$$

onde $\#E(\mathbb{F}_{29}) = 37$, que é primo.

O programa seleciona aleatoriamente um ponto P desta curva para ser a base do logaritmo discreto e depois seleciona aleatoriamente cinco outros pontos da curva, a fim de calcular o logaritmo e exibir a quantidade de iterações executadas em seu laço principal.

```
Ponto escolhido aleatoriamente para ser a base: (8,10).

Cálculo de log (15,27) na base (8,10)
Número de iterações: 6
Valor encontrado: 19.
De fato, (8,10)x19 = (15,27).

Cálculo de log (17,10) na base (8,10)
Número de iterações: 3
Valor encontrado: 27.
De fato, (8,10)x27 = (17,10).

Cálculo de log (10,25) na base (8,10)
Número de iterações: 4
Valor encontrado: 6.
De fato, (8,10)x6 = (10,25).

Cálculo de log (2,23) na base (8,10)
Número de iterações: 3
Valor encontrado: 30.
De fato, (8,10)x30 = (2,23).

Cálculo de log (4,10) na base (8,10)
Número de iterações: 5
Valor encontrado: 9.
De fato, (8,10)x9 = (4,10).
```

Observa-se que, diferentemente do que aconteceria com a busca exaustiva, não há a necessidade de se testar todos os valores para se encontrar o logaritmo discreto, visto que esta é uma técnica probabilística.

O método, porém, mostrou-se ineficiente no tratamento de curvas com boas aplicações criptográficas, de acordo com testes feitos com curvas recomendadas pelo NIST.

7.8 Considerações finais

Este capítulo demonstrou uma implementação das principais rotinas criptográficas para curvas elípticas. O objetivo foi apresentar de forma clara como os conceitos teóricos podem ser aplicados.

Constatou-se a superioridade do método de multiplicação escalar que se utiliza da forma não-adjacente. Além disso, as implementações possibilitaram testar o método de Pollard para curvas com boas propriedades criptográficas, atestando-se a segurança do sistema

Embora as implementações sobre corpos primos seja de fácil entendimento, consegue-se um melhor desempenho quando se emprega a aritmética de corpos binários [HANKERSON et al. (2000)]. Michael Rosing [ROSING (1999)] e reúne em seu livro todos os algoritmos (em liguagem C).

Capítulo 8

Conclusões

O atual estágio de desenvolvimento das telecomunicações tem exigido que mecanismos cada vez melhores sejam propostos, a fim de garantir a segurança da informação. Um dos métodos mais antigos utilizados com esse objetivo é a criptografia.

Este trabalho analisou as principais características matemáticas e computacionais do sistema criptográfico de chave pública conhecido como curvas elípticas. Suas definições, propriedades e algoritmos foram analisados tanto sob o ponto de vista teórico quando prático.

As informações obtidas ao longo do trabalho atestam a eficácia de tal sistema. Isso pode ser percebido tanto na comparação da dificuldade de se resolver os seu problema base (o logaritmo discreto definido sobre um grupo de pontos), quando comparado a outros sistemas vigentes quando nas implementações e nos testes realizados ao logo de todo o desenvolvimento desse estudo.

8.1 Trabalhos futuros

Esse trabalho procurou realizar de maneira geral as principais metodologias de aplicação das curvas elípticas. Uma proposta de trabalho (mais direcionadas) são a análise e a implementação eficiente de sistemas de curva elípticas (tanto em software quanto em hardware) de maneira que ela aproveita ao máximo os benefícios trazidos pelas tecnologias emergentes.

Outra proposta é fazer um análise do impacto que a tecnologia de sistemas computacionais quânticos podem trazer aos criptosisstemas atuais, de maneira a buscar novos tipos de sistemas que sejam resistentes a tal tecnologia.

Referências

BARRETO, P. S. L. M. (1999). Curvas Elípticas e Criptografia: Conceitos e Algoritmos. Artigo não publicado.

BLAKE, I. F., SEROUSSI, G., & SMART, N. P. (2000). *Elliptic curves in cryptography*. Cambridge University Press, Cambridge.

BONEH, D. & LIPTON, R. J. (1996). Algorithms for Black-Box Fields and their Application to Cryptography (Extended Abstract). In *CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, pages 283–297, London, UK. Springer-Verlag.

BRASSARD, G. & BRATLEY, P. (1996). *Fundamentals of algorithmics*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

BUCHMANN, J. A. (2002). *Introdução à Criptografia*. Berkeley.

CALEGARI, D. T. (2002). Uma implementação de criptografia de curvas elípticas no java card. Tese de Mestrado, Universidade Estatual de Campinas.

COHEN, H. (1993). *A course in computational algebraic number theory*. Springer-Verlag New York, Inc., New York, NY, USA.

CORMEN, T. H., RIVEST, R. L., & LEISERSON, C. E. (2001). *Introduction to Algorithms*. McGraw-Hill Higher Education.

DIFFIE, W. & HELLMAN, M. E. (1976). New Directions in Cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654.

DULLIUS, M. M. (2001). O Problema do Logaritmo Discreto. Tese de Mestrado, Universidade Federal do Rio Grande do Sul.

GAMAL, T. E. (1985). A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA. Springer-Verlag New York, Inc.

HANKERSON, D., HERNANDEZ, J. L., & MENEZES, A. (2000). Software Implementation of Elliptic Curve Cryptography over Binary Fields. In *CHES '00: Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, pages 1–24, London, UK. Springer-Verlag.

HANKERSON, D., MENEZES, A., & VANSTONE, S. (2004). *Guide to Elliptic Curve Cryptography*. Spring-Verlag.

KNUTH, D. E. (1997). *The art of computer programming, volume 2 (3rd ed.): seminumerical algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

KOBLITZ, N. (1987). Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209.

KOBLITZ, N. (1991). *A course in number theory and cryptography*. Springer-Verlag, New York.

KOLMAN, B. & BUSBY, R. C. (1987). *Discrete Mathematical Structures for Computer Science*. Prentice-Hall International, Inc, USA.

LÓPEZ, J. & DAHAB, R. (2000). An Overview of Elliptic Curve Cryptography. Technical Report IC-00-10.

MENEZES, A. J., Van Oorschot, P. C., & VANSTONE, S. A. (1997). *Handbook of applied cryptography*. The CRC Press series on discrete mathematics and its applications. CRC Press.

MILLER, V. S. (1986). Uses of Elliptic Curves in Cryptography. In WILLIAMS, H. C., editor, *Advances in cryptology — CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer-Verlag.

MIRANDA, R. A. (2002). Criptossistemas Baseados em Curvas Elípticas. Tese de Mestrado, Universidade Federal de Campinas.

NETO, A. M. (2006). Multiplicação escalar eficiente em curvas elípticas. Tese de Mestrado, Universidade de São Paulo – Instituto de Matemática e Estatística.

ODLYZKO, A. M. (1999). Discrete logarithms: The past and the future. 17. To appear.

PIETILäINEN, H. (2000). Elliptic curve cryptography on smart cards. Tese de Mestrado, Helsinki University of Technology.

POLLARD, J. M. (1978). Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 13:918–924.

RIVEST, R. L., SHAMIR, A., & ADLEMAN, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126.

ROSING, M. (1999). *Implementing elliptic curve cryptography*. Manning Publications Co., Greenwich, CT, USA.

SANTOS, J. P. O. (2005). *Introdução à Teoria dos Números*. IMPA, Rio de Janeiro, RJ, Brasil.

SARTORI, K. K. (2006). Curvas Elípticas: algumas aplicações em Criptografia e em Teoria dos Números. Tese de Mestrado, Universidade Estadual de Campinas, *Instituto de Matemática, Estatística e Computação Científica*, Departamento de Matemática.

SHANNON, C. E. (1949). Communication Theory of Secrecy Systems. *Bell Systems Technical Journal*, 28:656–715.

SHOUP, V. (2005). *A computational introduction to number theory and algebra*. Cambridge University Press.

SILVERMAN, J. H. (1985). *The Arithmetic of Elliptic Curves (Graduate Texts in Mathematics)*. Springer.

SINGH, S. (2005). *O Último Teorema de Fermat*. Editora Record.

SOLINAS, J. A. (1997). An Improved Algorithm for Arithmetic on a Family of Elliptic Curves. *Lecture Notes in Computer Science*, 1294:357–371.

STALLINGS, W. (2002). *Cryptography and Network Security: Principles and Practice*. Pearson Education.

STINSON, D. R. (1995). *Cryptography Theory and Practice*. CRT Press.

TANENBAUM, A. S. (2003). *Redes de Computadores*. Elsevier, Rio de Janeiro, quarta edition.

TRUSS, J. K. (1998). *Discrete Mathematics for Computer Scientists*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

WASHINGTON, L. C. (2003). *Elliptic Curves: Number Theory and Cryptography*. CRC Press, Inc., Boca Raton, FL, USA.

Apêndice A

Fundamentos Matemáticos

A.1 Relações e Funções

Os aspectos referentes a relações e funções são de suma importância em sistemas criptográficos. Esta seção...

Diferentemente da noção de conjunto $\{a, b\}$, um par ordenado (a, b) é uma representação onde a ordem com que os elemento a e b aparecem é levada em consideração. Desta forma, $(a, b) \neq (b, a)$.

Definição A.1 *Dados dois conjunto A e B, o conjunto composto por todos os possível pares ordenados formados (a, b), onde $a \in A$ e $b \in B$ recebe o nome de produto cartesiano. Em símbolos:*

$$A \times B = \{(a, b) | a \in A, b \in B\} \tag{A.1}$$

A.1.1 Relações

Definição A.2 *Dados A e B conjuntos não vazios, defini-se relação (denotada por R) como sendo um subconjunto do produto cartesiano entre A e B. $(a, b) \in R$ pode ser denotado por aRb (a relaciona-se com b através de R).*

Exemplo A.1 Se $A = \{1, 4, 6\}$ e $B = \{p, q\}$, pode-se definir $R = \{(1, p), (1, q), (6, q)\}$.

Exemplo A.2 Se $A = B = \mathbb{R}$, pode-se estabelecer uma relação R onde aRb se, e somente se, a e b satisfazem à equação $\frac{a^2}{16} + \frac{b^2}{9} = 1$

Em uma relação, o conjunto A recebe o nome de *domínio* e o conjunto B, de *imagem* ou *contradomínio*. É comum o estabelecimento de relações onde domínio e imagem são iguais, ou seja, com $A = B$, $R \in A \times A$. Neste caso, é dito simplesmente que uma relação está estabelecida sobre o conjunto A.

Entre as relações, destacam-se as chamadas *relações de equivalência*

Definição A.3 *Uma relação R sobre o conjunto A é chamada* **relação de equivalência** *se as seguintes propriedades forem observadas:*

- *R é* **reflexiva:** *aRa, para todo $a \in A$;*
- *R é* **simétrica:** *se aRb, então bRa, para todo $a, b \in A$;*
- *R é* **transitiva:** *se aRb e bRc, então aRc, para todo $a, b, c \in A$.*

Uma relação de equivalência sobre o conjunto A o separa em **classes de equivalência**, que são chamadas **partições**. Uma partição é um subconjunto de A que é disjunto a todos os demais subconjuntos de A. Além disso, a união de todas as partições de A resulta em A

Há relações que se diferenciam por apresentarem características peculiares e diversas aplicações práticas. Estas relações recebem o nome de *funções*.

A.1.2 Funções

Definição A.4 *Dados os conjuntos A e B, define-se função como uma lei de formação f que associa a cada elemento de A a um único elementos de B. A imagem de f é dada por $b = f(a)$. A notação padrão é $f : A \to B$.*

A.2 Aritmética Básica

Esta seção apresenta o que há de mais fundamental no campo da Teoria dos Números, cobrindo os principais resultados sobre o conjunto $\mathbb{Z} = \{... - 2, -1, 0, 1, 2, ...\}$ dos números inteiros. Divisibilidade, números primos, congruências, e outros conceitos fundamentais ao entendimento deste trabalho são apresentados.

A.2.1 Divisibilidade

O primeiro conceito mostrado é o que define a divisibilidade em $\mathbb{Z}$.

Definição A.5 *Sejam $a, b \in \mathbb{Z}$. Diz-se que a divide b, o que é denotado por $a|b$, se existir um inteiro c tal que $b = ac$. Neste caso, pode-se dizer que a é divisor de b ou ainda que b é multilpo de a. O fato de a não dividir b é denotado por $a \nmid b$.*

Exemplo A.3 $5|10$, pois $10 = 5 \cdot 2$. Por outro lado, $2 \nmid 15$, pois não existe inteiro k tal que se verifique $15 = 2 \cdot k$.

De acordo com o Teorema de Eudoxius [SANTOS (2005)], para $a, b \in \mathbb{Z},\ b \neq 0$, tem-se que a é múltiplo de b ou se encontra entre dois múltiplos consecutivos de b. Isso implica a existência de um inteiro q tal que

$$qb \leq a < (q+1)b, \text{ para } b > 0 \tag{A.2}$$

$$qb \leq a < (q-1)b, \text{ para } b < 0 \tag{A.3}$$

Com tais resultados, é possível entender o próximo Teorema:

Teorema A.1 (Algoritmo da Divisão) *Sejam $a, b \in \mathbb{Z}$, com $b > 0$, existe um único par de inteiros q e r tais que*

$$a = qb + r, \ 0 \leq r < b \tag{A.4}$$

No Teorema A.1, q é chamado quociente e r, resto. Como consequência, $b|a$ se, e somente se, $r = 0$.

Definição A.6 (Máximo Divisor Comum) *Dados dois interios a e b — a ou b não nulos — defini-se máximo divisor comum, denotado por $\text{mdc}\,(a, b)$, como o maior inteiro d que divide a e b simultaneamente.*

Teorema A.2 *Sejam $r_0 = a$ e $r_1 = b$ inteiros não nulos com $b \neq 0$. Se o algoritmo da divisão for aplicado de maneira que*

$$r_j = q_{j+1} r_{j+1} + r_{j+2}, \ 0 \leq r_{j+2} < r_{j+1} \tag{A.5}$$

onde $j = 0, 1, 2, ..., n-1$ e $r_{n+1} = 0$, então, $\text{mdc}\,(a, b) = r_n$.

Definição A.7 *Dois números a e b são primos entre si (ou relativamente primos) se $\text{mdc}\,(a, b) = 1$.*

A.2.2 Números Primos

Algumas características próprias dos números primos têm atraído a atenção dos matemáticos há séculos. Tal interesse cresceu ainda mais com o surgimento do criptosistema RSA [RIVEST et al. (1978)], em 1978, que baseia sua segurança na dificuldade de se fatorar inteiros de grande magnitude. O criptosistemas baseado em curvas elípticas também utiliza algumas propriedades destes números: por questões de segurança, curvas definidas sobre subgrupos de ordem prima são utilizadas, conforme será explicado.

A seguir, são apresentados os resulados mais relevantes ao contexto estudado.

Definição A.8 *Um número interio $n > 1$ é dito primo se este possuir somente dois divisores, a saber: 1 e n. Um número não primo é dito composto.*

Teorema A.3 (Teorema Fundamental da Aritmérica) *Qualquer inteiro maior que 1 é um primo ou é um produto de potências de primos. Assim,* $n > 1$ *pode ser escrito como*

$$n = p_1^{e_1} \cdots p_r^{e_r} \tag{A.6}$$

onde p_i *são primos distintos e os expoentes* e_i *são inteiros positivos.*

Teorema A.4 *A sequênica de números primos é infinita.*

A.2.3 Congruência

Definição A.9 *Sejam* $a, b, m \in \mathbb{Z}$*, com* $m > 0$*.* a *é congruente a* b *módulo* m*, e denota-se por* $a \equiv b \pmod{m}$*, se* $m|(a-b)$ *ou, equivalentemente, existir* $k \in \mathbb{Z}$ *tal que* $a = b + km$*. Caso* $m \not|(a-b)$*,* a *é dito ser incongruente a* b *módulo* m*, o que se representa por* $a \not\equiv b \pmod{m}$*.*

A seguir será mostrado que a congruência é uma relação de equivalência, separando o conjunto dos inteiros em partições.

Dados os conjuntos X e Y, o conjunto contendo todos os pares ordenados (x, y), onde $x \in X$ e $y \in Y$ é chamado **produto cartesiano** de X e Y, denotado por $X \times Y$. Uma **relação binária** R é um subconjunto de $X \times Y$ e pode ser representada por xRy. Assim, $x_k R y_k$ existe sempre que $(x_k, y_k) \in R$. Uma relação R definida sobre o conjunto A (ou seja, $A \times A$) é chamada **relação de equivalência** se R preservar as seguintes propriedades:

- **reflexiva:** aRa, para todo $a \in A$;
- **simétrica:** se aRb, então bRa, para todo $a, b \in A$;
- **transitiva:** se aRb e bRc, então aRc, para todo $a, b, c \in A$.

Uma relação de equivalência sobre o conjunto A o separa em **classes de equivalência**, que são chamadas **partições**. Uma partição é um subconjunto de A que é disjunto a todos os demais subconjuntos de A. A união de todas as partições de A resulta em A [TRUSS (1998)].

Proposição A.1 *A relação de congruência é uma relação de equivalência.*

Prova Basta mostrar que a congruência preserva as propriedades simétricas, reflexiva e transitiva. Para tanto, sejam $a, b, c, m \in \mathbb{Z}$, $m > 0$,

- Como $m|0$, então $m|(a-a)$. Assim, $a \equiv a \pmod{m}$;
- Se $a \equiv b \pmod{m}$, então $a = b + k_1 m,\ k_1 \in \mathbb{Z}$. Logo, $b = a - k_1 m$, o que implica $b \equiv a \pmod{m}$;

- Se $a \equiv b \pmod{m}$ e $b \equiv c \pmod{m}$ então $a - b = k_1 m$ e $b - c = k_2 m$, para $k_1, k_2 \in \mathbb{Z}$. Somando as igualdades membro a membro, obtém-se $a - c = km, k = k_1 + k_2$, o que implica $\equiv c \pmod{m}$.

Portanto, a relação de congruência é uma relação de equivalência. ■

Algumas operações são permitidas utilizando congruências:

Teorema A.5 *Se $a, b, c, m \in \mathbb{Z}$, tais que $a \equiv b \pmod{m}$ e $c \equiv d \pmod{m}$, então*

1. $a + c \equiv b + d \pmod{m}$
2. $a - c \equiv b - d \pmod{m}$
3. $ac \equiv bd \pmod{m}$

A exponenciação é também permitida

Proposição A.2 *Se $a, b, k, m \in \mathbb{Z},\ k > 0$, e $a \equiv b \pmod{m}$, então $a^k \equiv b^k \pmod{m}$.*

O próximo Teorema, cuja gereralização será dada logo a seguir, é um resultado que associa congruências e números primos.

Teorema A.6 (Pequeno Teorema de Fermat) *Se p é um primo e $p \nmid a$, então*

$$a^{p-1} \equiv 1 \pmod{p} \tag{A.7}$$

O Teorema de Euler generaliza este resulado para números não necessariamente primos e utiliza a função ϕ.

Definição A.10 *Dado um inteiro n, a função ϕ de Euler, denotada por $\phi(n)$, é o número de inteiros no intervalo $[0, n]$ que são relativamente primos com n.*

Teorema A.7 (Euler) *Sejam m um inteiro positivo e a um inteiro relativamente primo com m.*

$$a^{\phi(m)} \equiv 1 \pmod{m} \tag{A.8}$$

A afirmação de que o Teorema A.7 é uma generalização de Teorema A.6 parte do fato de $\phi(p) = p - 1$, sempre que p é primo.

Apêndice B

Código Fonte

B.1 As classe fundamentais

B.1.1 A classe `Fp`

```
1  package aritmetica;
2
3  import java.math.BigInteger;
4  import java.security.SecureRandom;
5
6  /**
7   * Esta classe utiliza-se dos métodos da classe BigInteger, da linguagem Java
8   * para implementar a aritmética em corpos primos.
9   *
10  * @author Leonardo Garcia Marques
11  */
12 public class Fp {
13     // Constantes utilizadas
14     private static final BigInteger ZERO = BigInteger.ZERO,
15             UM = BigInteger.ONE,
16             DOIS = BigInteger.valueOf(2),
17             TRES = BigInteger.valueOf(3);
18     private static final int CERTEZA_PRIMARIDADE = 50;
19
20     private BigInteger p; // base prima do corpo finito
21     private BigInteger a; // valor do elemendo do corpo
22
23
24     /**
25      * Cria uma nova instância de um elemento do corpo, com o valor zero
26      * @param p número primo que define o corpo
27      */
28     public Fp(BigInteger p){
29         if(!p.isProbablePrime(CERTEZA_PRIMARIDADE) || p.compareTo(TRES) <= 0)
30             throw new ArithmeticException("O valor de p deve ser um primo maior que 3");
31         this.p = p;
32         this.a = ZERO;
33     }
34
35     /**
36      * Cria um elemento específico de um corpo
37      * @param  a  o elemento do corpo
38      * @param  p  o primo que define o corpo
39      */
40     public Fp(BigInteger a, BigInteger p) throws ArithmeticException {
41         if(!p.isProbablePrime(CERTEZA_PRIMARIDADE) || p.compareTo(TRES) <= 0)
42             throw new ArithmeticException("O valor de p deve ser um primo maior que 3");
43         this.p = p;
44         this.a = a;
45     }
46
```

```
47      /**
48       * Cria um elemento aleatório do corpo de ordem p
49       * @param  p  a ordem do corpo
50       * @param  rand  semente para a criação do elemento randômico
51       */
52      public Fp(BigInteger p, SecureRandom rand){
53          this(p);
54          a = new BigInteger(p.bitLength(), rand).mod(p);
55      }
56
57      /**
58       * Retorna um elemento randômico do corpo de ordem p
59       * @param  rand  semente para a criação do elemento
60       */
61      public Fp getRandomico(SecureRandom rand){
62          return new Fp(p, rand);
63      }
64
65      /**
66       * Subtrai this com o elemento passado
67       * @param  b  o outro elemento de um corpo Fp
68       * @return  A diferênça entre this e b
69       */
70      public Fp sub(Fp b){
71          return new Fp(a.subtract(b.a).mod(p), p);
72      }
73
74      /**
75       * Soma this com o elemento passado como parâmetro
76       * @param b outro elemento de Fp
77       * @return A soma entre this e b
78       */
79      public Fp soma(Fp b){
80          return new Fp(a.add(b.a).mod(p), p);
81      }
82
83      /**
84       * Realiza o incremento do corpo em uma unidade
85       * @return this + 1 mod p
86       */
87      public Fp incr(){
88          return new Fp(a.add(UM).mod(p), p) ;
89      }
90
91      /**
92       * Realiza o decremento do corpo em uma unidade
93       * @return this - 1 mod p
94       */
95      public Fp decr(){
96          return new Fp(a.subtract(UM).mod(p), p);
97      }
98
99      /**
100      * Calcula o negativo de this
101      * @return O negativo de this
102      */
103     public Fp negativo(){
104         return new Fp(a.negate().mod(p), p);
105     }
106
107     /**
108      * Multiplica this com outro elemento do corpo
109      * @param b outro elemento de Fp
110      * @return O elemento correspondente a soma entre this e b
111      */
112     public Fp mult(Fp b){
113         return new Fp(a.multiply(b.a).mod(p), p);
114     }
115
116     public Fp multEscalar(int n){
117         BigInteger esc = new BigInteger(n+"");
118         return new Fp(a.multiply(esc).mod(p), p);
119     }
120
```

```
121     /**
122      * Eleva this ao quadrado
123      * @return O quadrado de this
124      */
125     public Fp quadrado(){
126         return new Fp(a.multiply(a).mod(p), p);
127     }
128 
129     /**
130      * Determina o inverso de this
131      * @return O inverso de this
132      */
133     public Fp inverso() throws ArithmeticException{
134         return new Fp(a.modInverse(p), p);
135     }
136 
137     /**
138      * Divide this pelo elemento passado como parâmetro
139      * @param b  elemento de Fp que será o divisor
140      * @return O quociente entre this e b
141      */
142     public Fp div(Fp b){
143         return new Fp(a.multiply(b.a.modInverse(p)).mod(p), p);
144     }
145 
146     /**
147      * Fornece o primo que define o corpo
148      * @return p
149      */
150     public BigInteger getP(){
151         return p;
152     }
153 
154     /**
155      * Verifica se this é igual a outro corpo, passa do por parametro
156      * @return true se this = b; false caso contrario
157      */
158     public boolean equals(Fp b){
159         return (a.compareTo(b.a) == 0) && (p.compareTo(b.p) == 0);
160     }
161 
162     /**
163      * Fornece uma string formatada do elemento que define o corpo
164      * @return string formatada
165      */
166     public String toString(){
167         return a.toString();
168     }
169 
170     /**
171      * Verifica se o corpo representa o elemento zero
172      * @return true, se this = 0
173      */
174     public boolean isZero(){
175         return this.a.compareTo(ZERO) == 0;
176     }
177 
178     /**
179      * Fornece o valor inteiro do elemento que defiene o corpo
180      * @return n valor inteiro de this
181      */
182     public Fp valorDe(int n){
183         return new Fp(BigInteger.valueOf(n), p);
184     }
185 
186     /**
187      * Fornece ao objeto BigInteger que define o corpo
188      * @return a
189      */
190     public BigInteger getA(){
191         return a;
192     }
193 
194     /**
```

```
195         * Calcula o deslocamento à esquerda em n posicooes dos bits que definem this
196         * @param   n  quantidad de bits deslocados
197         * @return  this << n (mod p)
198         */
199        public Fp shiftLeft(int n) {
200            return (n <  0) ? shiftRight(-n) : (n == 0) ? this : new Fp(a.shiftLeft(n).mod(p), p);
201        }
202
203        /**
204         * Calcula o deslocamento à direita em n posicooes dos bits que definem this
205         * @param   n  quantidad de bits deslocados
206         * @return  this >> n (mod p)
207         */
208        public Fp shiftRight(int n) {
209            return (n <  0) ? shiftLeft(-n) : (n == 0) ? this  :
210                new Fp(a.multiply(UM.shiftLeft(n).modInverse(p)).mod(p), p);
211        }
212
213        /**
214         * Calcula a raiz quadrada do corpo, caso exista.
215         * Algoritmo adaptado de  Algorithm P1363 A.2.5 - Finding Square Roots Modulo a Prime
216         * @return raiz(n) (mod p); null, caso a raiz não exista
217         */
218        public Fp raiz() {
219            if (this.isZero()) {
220                return this;
221            }
222
223            // 1ª Possibiliade : q mod 4 == 3
224            if (p.testBit(1)) {
225                Fp z = new Fp(a.modPow(p.shiftRight(2).add(UM), p), p);
226                // test solution:
227                return z.quadrado().equals(this) ? z : null;
228            }
229            // 2ª Possibilidade: q mod 8 == 5
230            if (p.testBit(2)) {
231                BigInteger twog     = a.shiftLeft(1).mod(p);
232                BigInteger gamma    = twog.modPow(p.shiftRight(3), p);
233                BigInteger i        = twog.multiply(gamma).multiply(gamma).mod(p);
234                Fp z = new Fp(a.multiply(gamma).multiply(i.subtract(UM)).mod(p),p);
235                // test solution:
236                return z.quadrado().equals(this) ? z : null;
237            }
238            // 3ª Possibilidade: q mod 4 == q mod 8 == 1
239            BigInteger k = p.add(UM).shiftRight(1);
240            for (BigInteger P = UM; ; P = P.add(UM)) {
241                BigInteger[] VQ0 = lucas(P, a, k);
242                BigInteger halfV = (VQ0[0].testBit(0) ? VQ0[0].add(p) : VQ0[0]).shiftRight(1);
243                Fp z = new Fp(halfV, p);
244                if (z.quadrado().equals(this)) {
245                    return z;
246                }
247                if (VQ0[1].compareTo(UM) > 0 && VQ0[1].compareTo(p.subtract(UM)) < 0) {
248                    return null;
249                }
250            }
251        }
252
253        /**
254         * Calcula a sequência de Lucas (V[k] mod q) and (Qˆ{Floor[k]/2} mod q),
255         * utilizada no algoritmo de cálculo da raiz quadrada modular
256         * Adaptado de P1363 A.2.4 - Generating Lucas Sequences
257         * @param   P   valor inicial da sequencia U
258         * @param   Q   valor inicial da sequencia V
259         * @param   k   índice do elemento que deve ser calculado
260         *
261         * @return  um vetor de BigInteger
262         */
263        private BigInteger[] lucas(BigInteger P, BigInteger Q, BigInteger k) {
264            BigInteger v_0 = DOIS;
265            BigInteger v_1 = P;
266            BigInteger q_0 = UM;
267            BigInteger q_1 = UM;
268
```

```
269          for (int i = k.bitLength() - 1; i >= 0; i--) {
270              q_0 = q_0.multiply(q_1).mod(p);
271              if (k.testBit(i)) {
272                  q_1 = q_0.multiply(Q).mod(p);
273                  v_0 = (v_0.multiply(v_1).subtract(P.multiply(q_0))).mod(p);
274                  v_1 = (v_1.multiply(v_1).subtract(q_1.shiftLeft(1))).mod(p);
275              } else {
276                  q_1 = q_0;
277                  v_1 = (v_0.multiply(v_1).subtract(P.multiply(q_0))).mod(p);
278                  v_0 = (v_0.multiply(v_0).subtract(q_0.shiftLeft(1))).mod(p);
279              }
280          }
281
282          return new BigInteger[] {v_0, q_0};
283      }
284
285  }// fim da classe Fp
```

B.1.2 A classe EC

```
1   package aritmetica;
2
3   import java.math.BigInteger;
4   import java.security.SecureRandom;
5
6   /**
7    * Esta classe implemeta as principais funções de uma curva elíptica do definida
8    * sobre uma corpo primo:*
9    *             E(Fp) : y^2 = x^3 + ax + b
10   * Seus coeficientes são elementos da classe Fp.
11   *
12   * @author Leonardo Garcia Marques
13   */
14  public class EC {
15      // coeficientes da curva
16      private Fp a, b;
17      // constantes utilizadas para converter um caracter em um ponto da curva
18      private static final int M = 128, k = 63;
19      private static final BigInteger bigK = BigInteger.valueOf(k);
20      private static final BigInteger bigM = BigInteger.valueOf(M);
21
22      /**
23       * Construtor da curva
24       *
25       * @param a coeficiente a
26       * @param b coeficiente b
27       */
28      public EC(Fp a, Fp b) {
29          if(a.getP().compareTo(b.getP()) != 0)
30              throw new ArithmeticException("Os coeficiente devem pertencer ao mesmo corpo.");
31          this.a = a;
32          this.b = b;
33      }
34
35      /**
36       * Obtém um ponto aleatório pertencente à mesma curva
37       */
38      public  Ponto getPontoRandomico(){
39          Fp x, y;
40
41          do{
42              x = a.getRandomico(new SecureRandom());
43              y = getY(x);
44          }while(y == null);
45
46          return new Ponto(x, y, this);
47      }
48
49      /**
50       * Verifica se um ponto passado como argumento faz parte do grupo de pontos
51       * da curva definida.
52       *
```

```
53          * @param Q ponto a ser comparado
54          * @return true caso pertença, false caso contrário
55          */
56         public boolean pertence(Ponto Q){
57             Fp x = Q.getX();
58             Fp y2 = Q.getY().quadrado();
59 
60             Fp c = x.mult(x.quadrado());
61             c = c.soma(a.mult(x));
62             c = c.soma(b);
63 
64             return y2.equals(c);
65         }
66 
67         /**
68          * Retorna o discrimiante da curva
69          * @return -16(4a^3 + 27b^2)
70          */
71         public Fp delta(){
72             Fp del = a.quadrado().mult(a).multEscalar(4);
73             return del.soma(b.quadrado().multEscalar(27)).multEscalar(16).negativo();
74         }
75 
76         /**
77          * Calcula (quando existe) a ordenada y de um ponto da curva, quando a
78          * abicissa x é passada como argumento.
79          *
80          * @param x abcissa
81          * @return y ordenada (null, se não existir)
82          */
83         public Fp getY(Fp x){
84             Fp y2 = x.mult(x.quadrado()).soma(a.mult(x)).soma(b);
85             return y2.raiz();
86         }
87 
88         /**
89          * Transforma um caracter ASCII em um ponto da curva.
90          *
91          * @param c caracter a ser codificado
92          * @return P o ponto correspondente ao caracter
93          */
94         public  Ponto codifica(char c){
95             int m = (int) c;
96             Fp x, y;
97 
98             for(int j = 0; j < k; j++){
99                 x = a.valorDe(m*k + j);
100                y = x.quadrado();
101                if(y != null)
102                    return new Ponto(x, y , this);
103            }
104 
105            return null;
106        }
107 
108        /**
109         * Calcula o caracter correspondente ao ponto passado por parâmetro
110         *
111         * @param c ponto que representa o caracter
112         * @return caracter correspondente
113         */
114        public char decodifica(Ponto c){
115            int ch = c.getX().getA().divide(bigK).mod(bigM).intValue();
116            return (char) ch;
117        }
118 
119        /**
120         * Fornece o coeficiente a da curva
121         *
122         * @return a
123         */
124        public Fp getA(){
125            return a;
126        }
```

```
127
128         /**
129          * Fornece o coeficiente a da curva
130          *
131          * @return a
132          */
133         public Fp getB(){
134             return b;
135         }
136
137         /**
138          * Cria uma instância aleatoria de uma curva definida sobre um corpo definido
139          * por um primo qualquer de comprimento (em bits) k.
140          *
141          * @param k comprimento do primo que define Fp.
142          * @return c instancia aleatoria de EC.
143          */
144         public static EC getRandomico(int k){
145             BigInteger p = new BigInteger(k, new SecureRandom()).nextProbablePrime();
146             Fp elem = new Fp(p);
147             Fp a, b;
148             EC c;
149
150             do{
151                 a = elem.getRandomico(new SecureRandom());
152                 b = elem.getRandomico(new SecureRandom());
153                 c = new EC(a, b);
154             }while(c.delta().isZero());
155
156             return c;
157         }
158
159         /**
160          * Cria uma instância aleatoria de uma curva definida sobre um corpo definido
161          * por um primo p.
162          *
163          * @param p primo que define Fp.
164          * @return c instancia aleatoria de EC.
165          */
166         public static EC getRandomico(BigInteger p){
167             Fp elem = new Fp(p);
168             Fp a, b;
169             EC c;
170
171             do{
172                 a = elem.getRandomico(new SecureRandom());
173                 b = elem.getRandomico(new SecureRandom());
174                 c = new EC(a, b);
175             }while(c.delta().isZero());
176
177             return c;
178         }
179
180         /**
181          * Produz uma string formatada representando a curva
182          *
183          * @return string E(Fp) : y^2 = x^3 + ax + b
184          */
185         public String toString(){
186             return String.format("E(F%s) : y^2 = x^3 + %sx + %s",a.getP(), a, b);
187         }
188     }// fim da classe EC
```

B.1.3 A classe `Ponto`

```
1   package aritmetica;
2
3   import java.math.BigInteger;
4   import java.security.SecureRandom;
5
6   /**
7    * Esta classe definie as operação de um ponto de uma curva elíptica definida
```

```
8   * sobre um corpo primo.
9   *
10  * @author Leonardo Garcia Marques
11  */
12 public class Ponto {
13     private final BigInteger big0 = BigInteger.ZERO,
14             big1 = BigInteger.ONE,
15             big2 = BigInteger.valueOf(2);
16     private Fp x, y;
17     private EC c;
18     private boolean inf = false;
19     private Fp _0;
20
21     /**
22      * Construtor sem argumentos. Constrói um ponto no infinito
23      */
24     public Ponto(){
25         inf = true;
26         x = new Fp(BigInteger.ZERO, new BigInteger("7"));
27         y = new Fp(BigInteger.ZERO, new BigInteger("7"));
28     }
29
30     /**
31      * Construtor usado quando se deseja um ponto diferente do infinito
32      *
33      * @param x coordenada x do ponto
34      * @param y coordenada y do ponto
35      * @param c curva elítptica ao qual o ponto pertence
36      */
37     public Ponto(Fp x, Fp y, EC c) {
38         this.x = x;
39         this.y = y;
40         this.c = c;
41         _0 = new Fp(BigInteger.ZERO, x.getP());
42     }
43
44     /**
45      * Verifica se o ponto está no infinito
46      * @return true, caso esteja
47      */
48     public boolean infinito(){
49         return inf;
50     }
51
52     /**
53      * Calcula o negativo de this, que corresponde a uma reflexão do ponto em
54      * relação ao eixo x
55      * @return (x, -y)
56      */
57     public Ponto negativo(){
58         return new Ponto(x, y.negativo(), c);
59     }
60
61     /**
62      * Fornece a coordenada x do ponto
63      * @param x
64      */
65     public Fp getX(){
66         return x;
67     }
68
69     /**
70      * Fornece a coordenada y do ponto
71      * @param y
72      */
73     public Fp getY(){
74         return y;
75     }
76
77     /**
78      * Fórmula geral para duplicação de pontos
79      * @return 2*this
80      */
81     public Ponto dup(){
```

```
82          if(inf || y.equals(_0))
83              return new Ponto();
84
85          Fp lambda = x.quadrado().multEscalar(3).soma(c.getA()).div(y.multEscalar(2));
86          Fp x3 = lambda.quadrado().sub(x.multEscalar(2));
87          Fp y3 = lambda.mult(x.sub(x3)).sub(y);
88
89          return new Ponto(x3, y3, c);
90      }
91
92      /**
93       * Executa a soma conforme a definição padrao para curvas desse tipo
94       *
95       * @param b outro ponto
96       * @return this + b
97       */
98      public Ponto soma(Ponto b){
99          if(this.equals(b))
100             return dup();
101
102         if(inf)
103             return b; // inf + b = b
104
105         if(b.inf)
106             return this; // this + inf = this
107
108         if(this.negativo().equals(b) || this.x.equals(b.x))
109             return new Ponto(); // this + (-this) = inf
110
111         Fp x2 = b.getX();
112         Fp y2 = b.getY();
113         Fp lambda = y2.sub(y).div(x2.sub(x));
114
115         Fp x3 = lambda.quadrado().sub(x).sub(x2);
116         Fp y3 = lambda.mult(x.sub(x3)).sub(y);
117
118         return new Ponto(x3, y3, c);
119     }
120
121     /**
122      * Definie a subtração de pontos, a partir da adição
123      */
124     public Ponto sub(Ponto b){
125         return soma(b.negativo());
126     }
127
128     /**
129      * Constrói e um clone de this
130      * @return this
131      */
132     public Ponto clone(){
133         return new Ponto(x, y, c);
134     }
135
136     /**
137      * Verifica a igualdade entre this e um ponto passado por parâmetro
138      * @return true, se forem iguais
139      */
140     public boolean equals(Ponto b){
141         return this.x.equals(b.x) && this.y.equals(b.y);
142     }
143
144     /*
145      * MÉTODOS GERAIS DE MULTIPLICAÇÃO DE PONTOS
146      */
147
148     /**
149      * Método "ingênuo" de multiplicação. Para o cálculo de n multiplicado por
150      * this, ele é somado a si mesmo n vezes
151      */
152     public Ponto multSimples(BigInteger n){
153         BigInteger _1 = BigInteger.ONE;
154         BigInteger i = BigInteger.ZERO;
155         Ponto soma = new Ponto();
```

```
156
157             while(i.compareTo(n) < 0){
158                 soma = soma.soma(this);
159                 i = i.add(_1);
160             }
161             return soma;
162         }
163
164
165         /**
166          * Método binário de multiplicação. Essa técnica explora as propriedades
167          * da expansão binária de n
168          */
169         public Ponto multtBin(BigInteger n){
170             int t = n.bitLength();
171             Ponto soma = new Ponto();
172             Ponto P = clone();
173
174             for(int i = 0; i < t; i++){
175                 if(n.testBit(i))
176                     soma = soma.soma(P);
177                 P = P.dup();
178             }
179
180             return soma;
181         }
182
183         /**
184          * Método binário utilizando a forma não adjacente
185          */
186         public Ponto multNFA(BigInteger n){
187             int[] k = new int[n.bitLength()+1];
188             fna(n, k);
189             Ponto Q = new Ponto();
190
191             for(int i = 0; i < k.length; i++){
192                 Q = Q.dup();
193                 if(k[i] == 1)
194                     Q = Q.soma(this);
195                 else if(k[i] == -1)
196                     Q = Q.soma(this.negativo());
197             }
198
199             return Q;
200         }
201
202         /**
203          * Método para o cálculo da forna não adjacente de uma inteiro
204          */
205         public  void fna(BigInteger k, int[] s){
206             BigInteger _1 = BigInteger.ONE,
207                     _0 = BigInteger.ZERO,
208                     _2 = BigInteger.valueOf(2),
209                     _4 = BigInteger.valueOf(4);
210             int i = k.bitLength(), x;
211             while(k.compareTo(BigInteger.ZERO) > 0){
212                 if(k.mod(_2).compareTo(_0) != 0){
213                     x = 2 - k.mod(_4).intValue();
214                     k = k.subtract(BigInteger.valueOf(x));
215                 } else
216                     x = 0;
217                 s[i] = x;
218                 i--;
219                 k = k.divide(_2);
220             }
221         }
222
223
224         public String toString(){
225             return inf ? "INF" : String.format("(%s,%s)", x, y);
226         }
227     }
```

A classe `TestaMultiplicacoes`

```
1  package testes;
2
3  import java.security.SecureRandom;
4  import aritmetica.*;
5  import java.math.BigInteger;
6  import java.util.Random;
7  /**
8   * Essa classe faz a contagem e do tempo de execução dos dois principais algoritmos
9   * de multiplicação estudados nesse trabalho. Os dados obtidos são usados na geração
10  * de gráficos.
11  *
12  * @author Leonardo Garcia Marques
13  */
14 public class TestaMultiplicacoes {
15
16     public static void main(String[] args) {
17         BigInteger big2 = BigInteger.valueOf(2);
18         BigInteger big1 = BigInteger.ONE;
19
20         BigInteger p = big2.pow(130).add(BigInteger.valueOf(169));
21         Fp a = new Fp(new BigInteger("3"), p);
22         Fp b = new Fp(new BigInteger("1043498151013573141076033119958062900890"), p);
23
24         EC ec = new EC(a, b);
25         BigInteger n = new BigInteger("136112946768375385380880778449568887423 7");
26
27         Ponto P = ec.getPontoRandomico();
28         System.out.println(P);
29         System.out.println(ec.pertence(P));
30
31         BigInteger k = new BigInteger("2");
32
33         long inicio, fim;
34         long tempBin, tempFNA;
35
36         // mede e exibe os tempos de execução para o primeira caso
37         for(int i = 2; i <= 64; i++){
38             inicio = System.currentTimeMillis();
39             P.multtBin(k);
40             fim = System.currentTimeMillis();
41             tempBin = fim - inicio;
42
43             inicio = System.currentTimeMillis();
44             P.multNFA(k);
45             fim = System.currentTimeMillis();
46             tempFNA = fim - inicio;
47
48             System.out.printf("%s\t%d\t%d\n",k, tempBin, tempFNA);
49             k = big2.pow(i);
50         }
51
52         // mede e exibe os tempos de execução para o primeira caso
53         k = new BigInteger("2");
54         for(int i = 2; i <= 64; i++){
55             inicio = System.currentTimeMillis();
56             P.multtBin(k);
57             fim = System.currentTimeMillis();
58             tempBin = fim - inicio;
59
60             inicio = System.currentTimeMillis();
61             P.multNFA(k);
62             fim = System.currentTimeMillis();
63             tempFNA = fim - inicio;
64
65             System.out.printf("%s\t%d\t%d\n",k, tempBin, tempFNA);
66             k = big2.pow(i).subtract(big1);
67         }
68
69         // mede e exibe os tempos de execução para o primeira caso
70         k = new BigInteger(58, new Random());
71         BigInteger inc = new BigInteger(58, new Random());
72         for(int i = 2; i <= 64; i++){
```

```
73              inicio = System.currentTimeMillis();
74              P.multtBin(k);
75              fim = System.currentTimeMillis();
76              tempBin = fim - inicio;
77
78              inicio = System.currentTimeMillis();
79              P.multNFA(k);
80              fim = System.currentTimeMillis();
81              tempFNA = fim - inicio;
82
83              System.out.printf("%s\t%d\t%d\n",k, tempBin, tempFNA);
84              k = k.add(inc);
85          }
86      }
87  }
```

B.2 Esquemas de cifrамento e decifrамento

B.2.1 A classe `ElGamal`

```
1   package aritmetica;
2
3   import java.math.BigInteger;
4   import java.util.Random;
5
6   /**
7    * Essa classe implementa um criptosistema análogo ao de ElGamal para curvas
8    * elípticas.
9    *
10   * @author Leonardo
11   */
12  public class ElGamal {
13      private EC E;
14      private BigInteger p;
15      private Ponto P;
16      private BigInteger n;
17
18      private static final BigInteger ZERO = BigInteger.ZERO,
19              UM = BigInteger.ONE,
20              DOIS = BigInteger.valueOf(2),
21              TRES = BigInteger.valueOf(3);
22
23      /**
24       * Construtor
25       *
26       * @param p numero primo
27       * @param E curva sob a qual se está trabalhando
28       * @param P gerador
29       * @param n ordem do subgrupo gerado por P
30       */
31      public ElGamal(BigInteger p, EC E, Ponto P, BigInteger n) {
32          this.p = p;
33          this.E = E;
34          this.P = P;
35          this.n = n;
36      }
37
38      /**
39       * Determina o par de chaves pública e privada
40       *
41       * @return par de chaves
42       */
43      public ParDeChaves gerarChaves(){
44          BigInteger d = randN();
45          return new ParDeChaves(P.multNFA(d), d);
46      }
47
48      /**
49       * Cifra um ponto M a partir da chave pública Q
50       *
```

```
51          * @param M ponto a ser cifrado
52          * @param Q chave pública
53          * @return (c1,c2) criptograma
54          */
55         public Criptograma cifrar(Ponto M, Ponto Q){
56             BigInteger k = randN();
57             Ponto c1 = P.multNFA(k);
58             Ponto c2 = M.soma(Q.multNFA(k));
59             return new Criptograma(c1, c2);
60         }
61 
62         /**
63          * Decifra um criptograma por meio da chave privada
64          *
65          * @param c criptograma
66          * @param d chave privada
67          * @return P ponto decifrado
68          */
69         public Ponto decifrar(Criptograma c, BigInteger d){
70             return c.getC2().sub(c.getC1().multNFA(d));
71         }
72 
73         /**
74          * Gera um elemento aleatória difetende de zero
75          *
76          * @return n aleatório diferente de zero.
77          */
78         private BigInteger randN(){
79             BigInteger n;
80 
81             do{
82                 n = new BigInteger(p.bitLength(), new Random()).mod(p);
83             }while(n.compareTo(ZERO) == 0);
84 
85             return n;
86         }
87     }
```

B.2.2 A classe `ParDeChaves`

```
1   package aritmetica;
2 
3   import java.math.BigInteger;
4 
5   /**
6    * Classe utilizada para armazenar um par de chaves utilizado no esquema ElGamal
7    * Elíptico
8    *
9    * @author Leonardo Garcia Marques
10   */
11  public class ParDeChaves {
12      private Ponto Q;
13      private BigInteger d;
14 
15      public ParDeChaves(Ponto Q, BigInteger d){
16          this.Q = Q;
17          this.d = d;
18      }
19 
20      public Ponto getChPublica(){
21          return Q;
22      }
23 
24      public BigInteger getChPrivada(){
25          return d;
26      }
27 
28      public String toString(){
29          return String.format("Chave pública: %s\nChave privada: %s", Q, d);
30      }
31  }
```

B.2.3 A classe `Criptograma`

```
1  package aritmetica;
2
3  /**
4   * Classe usada para o armazenamento temporário dos criptogramas gerados pelo
5   * esquema de ElGamal.
6   *
7   * @author Leonardo Garcia Marques
8   */
9  public class Criptograma {
10
11      private Ponto c1, c2;
12
13     public Criptograma(Ponto c1, Ponto c2) {
14         this.c1 = c1;
15         this.c2 = c2;
16     }
17
18     public Ponto getC1(){
19         return c1;
20     }
21
22     public Ponto getC2(){
23         return c2;
24     }
25
26     public String toString(){
27         return String.format("[%s,%s]", c1, c2);
28     }
29
30 }
```

B.2.4 A classe `ExemplificaElGamal`

```
1  package testes;
2
3  import aritmetica.*;
4  import java.math.BigInteger;
5  /**
6   * Esta classe exemplifica o protoclo de ciframento e deciframento de mensagens
7   * através de um ciframento com curvas elípticas análogo ao ElGamal
8   *
9   * @author Leonardo Garcia Marques.
10  */
11 public class ExemplificaElGamal {
12
13     public static void main(String[] args) {
14         // inicias os parâmetros da curva
15         BigInteger p = BigInteger.valueOf(2).pow(130).add(BigInteger.valueOf(169));;
16         Fp a = new Fp(new BigInteger("1"), p);
17         Fp b = new Fp(new BigInteger("1230929586093851880935564157041535079194"), p);
18         BigInteger n = new BigInteger("1361129467683753853846060531160085896483");
19         // instanciação um objeto curva
20         EC ec = new EC(a, b);
21         // escolha de um ponto aletório
22         Ponto P = ec.getPontoRandomico();
23         // instanciação de um objeto ElGamal, que efetuará o ciframento
24         ElGamal gamal = new ElGamal(p, ec , P, n);
25         // geração do par de chaves
26         ParDeChaves par = gamal.gerarChaves();
27         // exibe o par de chaves selecionado
28         System.out.println(par);
29         // mensagem a ser cifrada
30         String m = "Lidiane";
31
32         System.out.println("Mensagem a ser cifrada: "+m);
33
34         Ponto[] pontos = new Ponto[m.length()];
35         // codificação da mensagem em pontos de uma curva
36         for(int i = 0; i < pontos.length; i++)
37             pontos[i] = ec.codifica(m.charAt(i));
```

```
38
39          System.out.print("\n\nMensagem representada sob a forma de pontos: ");
40          for(int i = 0; i < pontos.length; i++)
41              System.out.printf("%s,",pontos[i]);
42
43          Criptograma[] cripto = new Criptograma[pontos.length];
44
45          // criação dos criptogramas, que conterão os pontos cifrados
46          for(int i = 0; i < pontos.length; i++)
47              cripto[i] = gamal.cifrar(pontos[i], par.getChPublica());
48
49          System.out.println("\n\nCriptogramas [C1,C2] que serão enviados por um canal inseguro");
50          for(int i = 0; i < cripto.length; i++)
51              System.out.printf("%s\n",cripto[i]);
52
53          // exibição da mensagem cifrada
54          System.out.print("\n\nEsses critptogramas equivalema aos caracteres:");
55          for(int i = 0; i < cripto.length; i++)
56              System.out.printf("%c", ec.decodifica(cripto[i].getC2()));
57
58          // deciframento dos pontos
59          Ponto[] pontosDecifrados = new Ponto[pontos.length];
60          for(int i = 0; i < cripto.length; i++)
61              pontosDecifrados[i] = gamal.decifrar(cripto[i], par.getChPrivada());
62
63          // exibição da mensagem decifrada
64          System.out.print("\n\nTexto após o deciframento: ");
65          for(int i = 0; i < pontosDecifrados.length; i++)
66              System.out.printf("%c", ec.decodifica(pontosDecifrados[i]));
67
68      }
69
70  }
```

B.3 Ataque ao problema do logaritmo discreto

B.3.1 A classe `ECDLP`

```
1   package aritmetica;
2
3   import java.math.BigInteger;
4   import java.util.Random;
5   /**
6    * Esta classe contém métodos estáticos para o cálculo do logaritmo discreto
7    * definido sobre um grupo de pontos de uma curva elíptica. Os médodos implementados
8    * correspondem ao algoritmo de busca euxaustiva (força bruta) e o método rho de
9    * Pollard.
10   *
11   * @author Leonardo Garcia Marques
12   */
13  public class ECDLP {
14      private static final BigInteger UM = BigInteger.ONE;
15      public static int L = 8;
16      public static BigInteger BigL = BigInteger.valueOf(L);
17
18      /**
19       * Classe interna utilizada para armazenar os dados temporários
20       */
21      private static class Dados{
22          BigInteger a, b;
23          Ponto R;
24      }
25
26      /**
27       * Tabela auxiliar, utulizada pelo método rho
28       */
29      private static Dados[] tab;
30
31      /**
32       * Método rho de Pollard adaptado para o cálculo do logaritmo discreto sobre
```

```
33          * um grupo de pontos de uma curva elíptica definida sobre um corpo finito.
34          *
35          * @param P ponto gerador do subgrupo <P>
36          * @param n ordem de <P>
37          * @param Q ponto do qual se deseja extrair o logarimo discreto
38          * @return l = log_P Q, o logaritmo discreto, caso exista.
39          */
40         public static BigInteger rho(Ponto P, BigInteger n, Ponto Q){
41             tab = new Dados[L];
42
43             for(int i = 0; i < L; i++){
44                 tab[i] = new Dados();
45                 tab[i].a = new BigInteger(n.bitLength(), new Random());
46                 tab[i].b = new BigInteger(n.bitLength(), new Random());
47                 tab[i].R = P.multNFA(tab[i].a).soma(Q.multNFA(tab[i].b));
48             }
49
50             BigInteger c1 = new BigInteger(n.bitLength(), new Random());
51             BigInteger d1 = new BigInteger(n.bitLength(), new Random());;
52             Ponto X1 = P.multNFA(c1).soma(Q.multNFA(d1));
53
54             Ponto X2 = X1;
55             BigInteger c2 = c1;
56             BigInteger d2 = d1;
57
58             int cont = 0;
59
60             int j;
61             do{
62                 j = H(X1);
63                 X1 = X1.soma(tab[j].R);
64                 c1 = c1.add(tab[j].a).mod(n);
65                 d1 = d1.add(tab[j].b).mod(n);
66
67                 for(int i = 0; i < 2; i++){
68                     j = H(X2);
69                     X2 = X2.soma(tab[j].R);
70                     c2 = c2.add(tab[j].a).mod(n);
71                     d2 = d2.add(tab[j].b).mod(n);
72                 }
73                 cont ++;
74             }while(!X1.equals(X2));
75             if(d1.compareTo(d2) == 0)
76                 return null;
77             System.out.print("\nNúmero de iterações: "+cont);
78             return c1.subtract(c2).multiply(d2.subtract(d1).modInverse(n)).mod(n);
79         }
80
81         /**
82          * Método auxiliar ao método rho. Distribui os pontos em partições.
83          *
84          * @param Ponto
85          * @return particao
86          */
87         private static int H(Ponto ponto){
88             return ponto.getX().getA().mod(BigL).intValue();
89         }
90
91         /**
92          * Cálculo do logaritmo discreto por meio da busca exaustiva
93          *
94          * @param P ponto gerador do subgrupo <P>
95          * @param n ordem de <P>
96          * @param Q ponto do qual se deseja extrair o logarimo discreto
97          * @return l = log_P Q, o logaritmo discreto, caso exista.
98          */
99         public static BigInteger buscaExaustiva(Ponto P, BigInteger n, Ponto Q){
100            BigInteger k = UM;
101
102            do{
103                if(P.multNFA(k).equals(Q))
104                    return k;
105                k = k.add(UM);
106            }while(k.compareTo(n) < 0);
```

```
107
108             return null;
109         }
110     }// fim da classe ECDLP
```

B.3.2 A classe `TestaMetodoRho`

```
1   package testes;
2
3   import aritmetica.*;
4   import java.math.BigInteger;
5   /**
6    * Essa classe demostra a utilização do algoritmo de Pollard para o cálculo do
7    * logaritmo discreto.
8    *
9    * @author Leonardo Garcia Marques.
10   */
11  public class TestaMetodoRho {
12
13      public static void main(String[] args) {
14          // Construção da curva
15          BigInteger p = new BigInteger("29");
16          Fp a = new Fp(new BigInteger("4"), p);
17          Fp b = new Fp(new BigInteger("20"), p);
18          EC ec = new EC(a, b);
19          BigInteger n = new BigInteger("37");
20
21          // Escolhe-se um ponto aleatorio desta curva
22          Ponto P = ec.getPontoRandomico();
23           System.out.printf("\nPonto escolhido aleatoriamente para ser a base: %s. ",P);
24          Ponto Q;
25          BigInteger k;
26
27          // É calculado o logaritmo discreto para alguns pontos da curva.
28          for(int i = 0; i < 5; i++){
29              Q = ec.getPontoRandomico();
30              System.out.printf("\n\nCálculo de log %s na base %s", Q, P);
31              k = ECDLP.rho(P, n, Q);
32              System.out.printf("\nValor encontrado: %s.", k);
33              System.out.printf("\nDe fato, %sx%s = %s. ", P, k, P.multNFA(k));
34          }
35      }
36
37  }
```