UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE MATEMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM MATEMÁTICA APLICADA

O PROBLEMA DO LOGARITMO DISCRETO

por

Maria Madalena Dullius

Dissertação submetida como requisito parcial
para a obtenção do grau de

Mestre em Matemática Aplicada

Prof. Dr. Vilmar Trevisan
Orientador

Prof. Dr. Claus Haetinger
Co-orientador

Porto Alegre, dezembro de 2001

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Dullius, Maria Madalena

O PROBLEMA DO LOGARITMO DISCRETO / Maria Madalena Dullius.—Porto Alegre: PPGMAp da UFRGS, 2001

76 p.: il.

Dissertação (mestrado) —Universidade Federal do Rio Grande do Sul, Programa de Pós-Graduação em Matemática Aplicada, Porto Alegre, 2001.
Orientador: Trevisan, Vilmar; Co-orientador: Haetinger, Claus

Dissertação: Matemática Aplicada
Modelo, Dissertação

O PROBLEMA DO LOGARITMO DISCRETO

por

Maria Madalena Dullius

Dissertação submetida ao Programa de Pós-Graduação em Matemática Aplicada do Instituto de Matemática da Universidade Federal do Rio Grande do Sul, como requisito parcial para a obtenção do grau de

Mestre em Matemática Aplicada

Linha de Pesquisa: Algoritmos Numéricos e Algébricos

Orientador: Prof. Dr. Vilmar Trevisan

Co-orientador: Prof. Dr. Claus Haetinger

Banca examinadora:

Profª. Drª. Ada Maria de Souza Döering
Programa de Pós-Graduação em Matemática-UFRGS

Profª. Drª. Cyntia Feijó Segatto
Programa de Pós-Graduação em Matemática Aplicada-UFRGS

Prof. Dr. Álvaro Luiz De Bortoli
Programa de Pós-Graduação em Matemática Aplicada-UFRGS

Dissertação apresentada e aprovada em 21 de dezembro de 2001

Prof. Dr. Vilmar Trevisan
Coordenador

AGRADECIMENTO

Ao professor Vilmar Trevisan, a orientação, as contribuições neste trabalho, especialmente a compreensão demonstrada ao longo desta caminhada.

Ao colega professor Claus Haetinger, a co-orientação, os conhecimentos transmitidos e o incentivo demonstrado durante o trabalho.

Aos colegas do curso a caminhada juntos, especialmente à colega Eliamar a amizade, a convivência, o apoio e a motivação.

Ao Colégio João de Deus que disponibilizou-me tempo para o mestrado, em especial à direção que sempre ajustou meus horários de acordo.

Ao meu marido Volmir e aos meus filhos Bruno, Artur e Julia por todo amor, carinho, companheirismo e compreensão.

Aos meus pais e meus irmãos a participação em todos os momentos da minha vida.

À UNIVATES, em especial aos colegas Ingo, Marli e Márcia e aos funcionários do CPD. Marli e Márcia a amizade, o apoio e o incentivo. Ao meu professor e hoje colega Ingo a confiança e o estímulo para continuar esta caminhada. À equipe do CPD, a quem devo todas as instalações e configurações do LaTeX.

RESUMO

Existem muitos sistemas de criptografia cuja segurança é baseada na dificuldade em resolver logaritmos discretos. Neste trabalho descrevemos alguns métodos para calcular logaritmos discretos, a saber: Algoritmo Shanks, Algoritmo Pollard, Algoritmo Silver-Pohlig-Hellman e o Algoritmo Index Calculus. Também são relatadas questões de complexidade computacional e os últimos recordes alcançados para resolver logaritmos discretos.

ABSTRACT

There are many cryptosystems whose security is based on the difficulty of solving the discrete logarithm. In this work, we describe some methods to calculate discrete logarithms: Shanks's Algorithm, Pollard's Algorithm, Silver-Pohlig-Hellman's Algorithm and the Index Calculus Algorithm. We also relate computation complexity issues and the last records that have been obtained on the discrete logarithm problem.

SUMÁRIO

ALGUMAS NOTAÇÕES FREQÜENTEMENTE USADAS

$\mathbb{R}$	corpo dos números reais
$\mathbb{N}$	conjunto dos números naturais incluindo o zero
$GF(q)$	corpo de Galois de q elementos
$\langle a \rangle$	grupo cíclico gerado pelo elemento a
$\log_b a$	logaritmo de a na base b
$\subset, \subseteq$	usado indistintamente para inclusão
$\subsetneq$	inclusão estrita
$a \equiv b \pmod{n}$	a congruente a b módulo n
$\mathbb{Z}_n$	anel dos inteiros módulo n
$mdc(a, b)$	máximo divisor comum entre a e b
(a, b)	par ordenado
$\|G\|$	ordem do grupo G
$a \mid b$	a divide b
$F \setminus \{0\}$	diferença entre conjuntos
S_n	grupo das permutações de n elementos
$w(a)$	número de *bits* para representar $a \in \mathbb{Z}$ em um registro
$p(I)$	probabilidade de ocorrência do *input* I
D_n	conjunto de *inputs* de tamanho n
$t(I)$	número de operações básicas realizadas pelo algoritmo no *input* I
$max(A)$	maior valor dentre os elementos do conjunto A
$W(n)$	número máximo de operações básicas realizadas pelo algoritmo em qualquer *input* de tamanho n
$A(n)$	número médio de operações básicas realizadas pelo algoritmo em cada *input* de tamanho n
$\mathbb{R}^+$	números não negativos
$\lceil a \rceil$	menor inteiro maior que $a \in \mathbb{R}$
$\lfloor a \rfloor$	parte inteira de $a \in \mathbb{R}$

1 APRESENTAÇÃO

Algebricamente, o logaritmo é um expoente. Mais precisamente, se $1 \neq b > 0$ é um número real, então para valores positivos de $a \in \mathbb{R}$, o logaritmo de a na base b é denotado por $\log_b a$ e é definido como sendo aquele expoente ao qual b deve ser elevado para produzir a.

No nosso estudo abordamos os logaritmos discretos. A palavra *discreto* distingue a situação de grupo finito da situação clássica (contínua) citada acima.

O objetivo principal deste trabalho consiste no estudo do logaritmo discreto, os principais métodos para resolvê-lo e seu considerável significado na criptografia. Os algoritmos conhecidos para resolver os logaritmos discretos podem ser classificados em três categorias:

- A primeira é formada por algoritmos para grupos arbitrários, isto é, aqueles que não exploram qualquer propriedade específica do grupo. O capítulo quatro apresenta dois algoritmos com estas características: o Algoritmo Shanks e o Algoritmo Pollard.

- A segunda consiste de algoritmos para grupos finitos cuja ordem não tem fatores primos grandes, ou seja, possuem ordem suave. No capítulo cinco apresentamos o método Silver-Pohlig-Hellman, que se enquadra nesta categoria, e analisamos seu desempenho assintótico.

- Finalmente, a terceira categoria é de algoritmos que exploram métodos para representar elementos de um grupo como produto de elementos de um conjunto relativamente pequeno. Para esta categoria, apresentamos, no capítulo seis, o Algoritmo Index-Calculus na versão básica e também alguns melhoramentos que foram feitos. São abordadas algumas questões técnicas importantes na análise do desempenho do algoritmo.

Para compreender melhor estes métodos, relatamos no capítulo dois, um breve histórico do surgimento dos logaritmos e apresentamos alguns sistemas de criptografia, cuja segurança é baseada na dificuldade de resolver logaritmos discretos. Primeiro abordamos o sistema de troca de chave Diffie-Hellman. A seguir apresentamos o sistema de transmissão de informações de Massey-Omura. Finalmente, apresentamos o sistema de transmissão de informações de ElGamal.

No capítulo três abordamos os conceitos mínimos da Teoria dos Números necessários para o prosseguimento da leitura e fazemos uma abordagem da Teoria da Complexidade para podermos analisar a eficiência dos algoritmos estudados. Para tal, na primeira seção, são lembrados os conceitos de grupo, anél e corpo; na segunda abordamos os números inteiros; na terceira definimos congruências e algumas de suas propriedades; na quarta caracterizamos os corpos finitos e na quinta é feita uma abordagem da complexidade computacional.

E, finalmente, no capítulo sete fazemos uma análise dos métodos que são mais usados no momento, suas implicações na criptografia e os últimos recordes alcançados no cálculo do logaritmo discreto.

2 INTRODUÇÃO

2.1 Relato histórico

Os primeiros registros de logaritmos foram feitos no início do século XVII, quando já era clara a necessidade de facilitar os grandes cálculos trigonométricos da Astronomia e da Navegação. A idéia básica era substituir operações mais complicadas, como multiplicação e divisão, por operações mais simples, como adição e subtração. Isto é possível devido às bem conhecidas propriedades aritméticas dos logaritmos. Historicamente, os primeiros logaritmos a serem estudados foram os de base 10, chamados logaritmos comuns ou logaritmos decimais. Até por volta de 1960, eles eram usados como instrumento de cálculo manual. Mas essa função dos logaritmos decimais foi perdendo importância há algumas décadas com o advento dos computadores eletrônicos, dos microcomputadores e das calculadoras de bolso, bem como também dos programas de computação, cada vez mais eficazes, que facilitaram muito os cálculos. Mais recentemente, os logaritmos de base dois desempenharam papel importante na Ciência da Computação, uma vez que surgiram naturalmente em sistemas numéricos binários. Porém, os logaritmos mais usados nas aplicações são os logaritmos naturais, os quais têm uma base irracional denotada pela letra e.

Nosso estudo aborda os logaritmos discretos. O problema de resolução de logaritmos discretos em grupos finitos adquiriu grande importância recentemente devido à sua aplicação em criptografia. Muitos dos sistemas de criptografia de chave pública mais populares são baseados na exponenciação discreta, e se tornariam inseguros se um algoritmo eficiente de logaritmos discretos fosse descoberto. Poderíamos falar sobre o problema do logaritmo discreto num semigrupo arbitrário, mas na maior parte das aplicações o interesse está em exemplos específicos de grupos cíclicos finitos, por isso vamos considerar que nosso grupo G é cíclico, com $G = \langle b \rangle$.

Seja G um grupo multiplicativo, para $b \in G$ denotamos por $\langle b \rangle$ o subgrupo cíclico gerado por b. Então dado $a \in \langle b \rangle$, definimos o logaritmo discreto de a na base b como sendo o menor inteiro não negativo x tal que $a = b^x$. Denotamos este logaritmo discreto por $\log_b a = x$. O $\log_b a$ é determinado módulo a ordem de b.

A propriedade fundamental dos logaritmos contínuos continua válida para logaritmos discretos, ou seja: $\log_b xy = \log_b x + \log_b y$, para todo $x, y \in GF(q)^*$. Em conseqüência desta propriedade temos: $\log_b xy^{-1} = \log_b x - \log_b y$. E também $\log_b x^n = n \log_b x$, para todo $n \in \mathbb{N}$.

Os logaritmos discretos têm uma longa história na Teoria dos Números. Inicialmente foram usados, principalmente, para cálculos em corpos finitos. O interesse pelos logaritmos discretos começou a crescer no século XX, quando mais cálculos foram feitos e quando questões algorítmicas começaram a ser investigadas mais intensamente. Começaram a representar um importante papel na criptografia já por volta de 1950, muito tempo antes dos sistemas de chave pública aparecerem no cenário, quando criptosistemas baseados em mudar seqüências de registros substituíam os baseados em máquinas que usavam rotores. Logaritmos discretos ocorriam naturalmente naquele contexto como ferramenta para encontrar onde havia um bloqueio particular numa mudança de seqüência de registros.

Atualmente, a principal razão para o grande interesse em logaritmos discretos é que muitos criptosistemas de chave pública, para a sua segurança, dependem da hipótese de que, pelo menos para grupos adequadamente escolhidos, seja difícil calcular os logaritmos discretos.

O principal impulso, no entanto, originou-se a partir do método Diffie-Hellman (DH, por brevidade). O algoritmo DH foi a primeira técnica prática de chave pública publicada e é amplamente usado ainda hoje.

W.Diffie e M.E.Hellman propuseram o problema do logaritmo discreto como uma boa fonte para uma função de caminho único. Dados X, Y conjuntos, podemos pensar uma função de caminho único como uma função $f : X \to Y$

tal que seja fácil calcular $Im(f)$, mas seja difícil encontrar a pré-imagem de y, para a maioria dos valores de $y \in Y$. Em seus trabalhos de criptografia, Diffie e Hellman propuseram como um candidato natural a função baseada na exponenciação módulo um número primo p: $f(x) \equiv b^x (\mathrm{mod}\, p)$, sendo p um primo grande e b uma raiz primitiva módulo p, isto é, um gerador do grupo multiplicativo $GF(p)^*$. Esta função é relativamente fácil de calcular. Invertendo a função f claramente precisamos resolver um problema de logaritmo discreto em $GF(p)^*$, e isto, em geral, é extremamente difícil para primos grandes.

2.2 Sistemas de Criptografia Baseados em Logaritmos Discretos

Originalmente, em muitos sistemas de computação, as senhas usadas eram armazenadas em uma pasta especial. Isto tem a desvantagem de que qualquer pessoa que tiver acesso a esta pasta é capaz de se fazer passar por algum usuário legítimo. Podemos eliminar a necessidade de algum segredo com a eliminação da armazenagem de senhas. Ao invés disso, utiliza-se uma função f cuja pré-imagem seja difícil de se obter, e cria-se uma pasta contendo pares $(i, f(p_i))$, onde i denota o nome do usuário e p_i a respectiva senha. Esta pasta pode então ser feita pública. A segurança deste projeto claramente depende da dificuldade da obtenção da pré-imagem da função f. Um primeiro candidato natural para tal função foi a exponenciação discreta: dado um corpo $GF(p)^*$ e escolhido e publicado um elemento primitivo $b \in GF(p)^*$, então para um inteiro x, definimos $f(x) = b^x$. O indivíduo que tentar ter acesso ao computador fingindo ser o usuário i precisará encontrar p_i conhecendo apenas o valor de b^{p_i}, isto é, precisará resolver o logaritmo discreto no grupo $GF(p)^*$.

2.2.1 Diffie-Hellman

W. Diffie e M.E. Hellman ([7]) desenvolveram um sistema de troca de chave baseado na exponenciação em grupos finitos (1976). Este aparentemente foi o primeiro criptosistema de chave pública proposto. Neste, um grupo finito $GF(p)^*$ e um elemento primitivo $b \in GF(p)^*$ são escolhidos e publicados.

Usuários A e B, que quiserem se comunicar, escolherão aleatoriamente inteiros a e g, respectivamente, com $2 \leq a, g \leq p-2$. Então o usuário A transmite b^a módulo p para B através de um canal público, enquanto o usuário B transmite b^g módulo p para A. A chave comum será então b^{ag} módulo p, a qual A poderá calcular elevando o recebido b^g à potência a (que somente ele conhece), e que B forma elevando b^a à potência g. Um terceiro usuário conhece as chaves públicas b^a e b^g. É claro que um algoritmo eficiente de logaritmo discreto tornará este projeto inseguro, pois depois de b^a publicamente transmitido será possível para um analista de criptografia determinar a, e então determinar a chave usada por A e B. De fato, considerando $u \equiv b^a \pmod{p}$, com b e p públicos, temos que $a \equiv \log_b u \pmod{p}$.

O sistema Diffie-Hellman é baseado no problema do logaritmo discreto e no fato de que é praticamente impossível calcular b^{ag}, conhecendo apenas b^a e b^g, sem calcular a ou g. Isto ainda não foi provado de modo eficiente, e assim não podemos excluir a possibilidade de haver algum caminho para gerar b^{ag} conhecendo somente b^a e b^g, sem calcular a ou g, muito embora não pareça ser provável que tal método exista.

A seguir apresentamos um exemplo deste método.

Exemplo: Dado o grupo finito $(\mathbb{Z}_{53}^*, \cdot)$, com $2 \in \mathbb{Z}_{53}$ um elemento primitivo, sejam eles tornados públicos. Suponhamos que usuários A e B queiram se comunicar e escolham aleatoriamente os números inteiros 29 e 19, respectivamente, com $2 \leq 29, 19 \leq 51$. Então o usuário A transmite $2^{29} \equiv 45 \pmod{53}$ para B através de um canal público, enquanto o usuário B transmite $2^{19} \equiv 12 \pmod{53}$

para A. A chave comum é então $2^{29\times 19} \equiv 21 \pmod{53}$, a qual A pode calcular elevando o recebido 2^{19} à potência 29 (que somente ele conhece), e que B obtém elevando 2^{29} à potência 19. Como A publica 45 que é equivalente a 2^a, está claro que calculando $\log_2 45$ módulo 53 obtemos o valor de a que é 29 e da mesma forma $g \equiv \log_2 12 \equiv 19 \pmod{53}$.

2.2.2 Massey-Omura

J.L. Massey e Omura (ver, por exemplo, [18]) criaram um sistema que usa exponenciação em grupos finitos para transmitir informações, baseado nas idéias de A. Shamir ([19]).

Por exemplo, suponhamos que o usuário A deseje enviar uma mensagem m dada (que podemos considerar como um elemento de um grupo $GF(p)^*$, logo não nulo) para o usuário B. Então o usuário A escolhe aleatoriamente um inteiro c, com $1 \leq c \leq p-1$ e $mdc(c, p-1) = 1$, e transmite $x \equiv m^c \pmod{p}$ para B. Em seguida, o usuário B escolhe aleatoriamente um inteiro d, com $1 \leq d \leq p-1$ e $mdc(d, p-1) = 1$, e transmite $y \equiv x^d \equiv m^{cd} \pmod{p}$ para A. O usuário A agora forma $z \equiv y^{c'} \pmod{p}$, onde $cc' \equiv 1 \pmod{p-1}$, e transmite z para o usuário B, uma vez que $z \equiv y^{c'} \equiv m^{cdc'} \equiv m^d \pmod{p}$. Então o usuário B somente precisa calcular $z^{d'}$ módulo p para recuperar m, onde $dd' \equiv 1 \pmod{p-1}$, visto que $z^{d'} \equiv m^{dd'} \equiv m \pmod{p}$.

Neste sistema está novamente claro que com um método eficiente para calcular logaritmo discreto sobre $GF(p)^*$ seria simples recuperar a mensagem. Pois, de $y = x^d$, temos $d = \log_x y$. Como x e y são públicos, um analista de criptografia pode obter d. De $z = m^d$, temos $d = \log_m z$ que equivale a $d = logz \cdot \log m^{-1}$. Portanto $\log m = \log z \cdot d^{-1}$ e, considerando $\log m = \log_b m$, onde b é uma base qualquer, e $u = \log z \cdot d^{-1}$, obtemos $\log_b m = u$, donde $m = b^u$.

Exemplo: Suponhamos que o usuário A deseje enviar uma mensagem $m = 2$ de um grupo conhecido $GF(53)^*$ para o usuário B. Então A escolhe

aleatoriamente um inteiro $c = 29$, $1 \leq 29 \leq 52$, $mdc(29, 52) = 1$ e transmite $x = 2^{29} \equiv 45 \pmod{53}$ para B. O usuário B escolhe aleatoriamente um inteiro $d = 19$, $1 \leq 19 \leq 52$, $mdc(19, 52) = 1$ e transmite $y = 45^{19} \equiv 21 \pmod{53}$ para A. O usuário A agora forma $z = 21^9 \equiv 12 \pmod{53}$ onde 9 é o inverso de 29 módulo 52, e transmite $z = 12$ para B. O usuário B calcula $m = 12^{11} \equiv 2 \pmod{53}$, onde 11 é o inverso de 19 módulo 52. Como A publica 45 que é equivalente a 2^c, fica claro que calculando $\log_2 45$ módulo 53 obtemos o valor de c que é 29 e da mesma forma $d \equiv \log_{45} 21 \equiv 19 \pmod{53}$.

2.2.3 ElGamal

Outro sistema de transmissão de informações foi proposto por T. ElGamal ([8]). Essencialmente é uma variação do projeto de distribuição de chave de Diffie-Hellman.

O usuário A publica uma chave pública $b^a \in GF(p)^*$, onde o grupo $GF(p)^*$ e uma raiz primitiva b são conhecidos (ou são também publicados por A, ou ainda, são usados por todos num dado sistema), mas mantém a secreto. O usuário B, que deseja enviar $m \in GF(p)^*$ para A, escolhe aleatoriamente um inteiro k, $1 \leq k \leq p-2$, (para cada m escolhe-se um k diferente) e transmite o par (b^k, mb^{ak}) para A. O usuário A conhece a e, portanto, pode calcular $b^{ak} = (b^k)^a$ e recuperar m dividindo o valor recebido mb^{ak} pelo valor calculado b^{ak}. Um algoritmo eficiente de logaritmo discreto permitirá que um analista de criptografia calcule a ou k, e poderá portanto tornar este projeto inseguro também. Considerando $u = b^a$, temos $a = \log_b u$, onde b e u são públicos.

T. ElGamal também propôs um novo projeto de assinatura digital que usa exponenciação em grupos $GF(p)^*$, p um primo. O usuário A, que deseja sinal de mensagem eletrônica, publica um primo p, uma raiz primitiva b módulo p e um inteiro y, $1 \leq y \leq p-1$, o qual é gerado com a escolha aleatória de um inteiro a, que é mantido secreto, e dado $y = b^a$ (para todos os usuários do sistema p e b podem ser os mesmos. Neste caso somente y é especial para o usuário A). Para

receber uma mensagem m, $1 \leq m \leq p-1$, o usuário A fornece um par de inteiros (r, s), $1 \leq r, s \leq p-1$ tal que $b^m \equiv y^r r^s \pmod{p}$. Para gerar r e s, o usuário A escolhe aleatoriamente um inteiro k com $mdc(k, p-1) = 1$ e calcula $r = b^k$. Visto que $y = b^a$, então s satisfaz $b^m \equiv b^{ar+ks} \pmod{p}$, o que é equivalente a $m \equiv ar + ks \pmod{p-1}$. Como $mdc(k, p-1) = 1$, então existe uma única solução para $m \equiv ar + ks \pmod{p-1}$, esta solução é fácil de achar para o usuário A (que conhece a, r, k). Um algoritmo eficiente de logaritmo discreto poderá tornar este sistema inseguro, porque permitirá a um analista de criptografia calcular a a partir de y. Parece não haver nenhum outro caminho até o momento para interromper este sistema sem a necessidade de calcular logaritmos discretos. Assim este método torna-se bastante atrativo.

Alguns grupos nos quais o problema do logaritmo discreto tem sido considerado são $\mathbb{Z}_p^*$, o grupo multiplicativo dos inteiros módulo um primo p, e mais geralmente, $GF(p^k)^*$, o grupo dos elementos não nulos de um corpo finito de p^k elementos *(Galois Field)*, com $k \in \mathbb{N}$. Também têm sido considerados grupos de curvas elípticas de ordem prima, grandes subgrupos cíclicos de $\mathbb{Z}_n^*$ onde n é composto, e grandes subgrupos cíclicos de classes de grupos de corpos de números quadráticos imaginários.

Na discussão de algoritmos para o cálculo de logaritmos discretos, podemos fazer uma distinção entre algoritmos que são designados para serem práticos e outros que são estruturados de tal forma que permitem uma prova rigorosa do seu comportamento. Considerando o caso do cálculo de logaritmo discreto para $GF(2^k)$, para este problema, o algoritmo com o tempo de execução assintótico mais rápido é devido a D.Coppersmith ([5]), mas este é baseado numa análise heurística, e permenecem algumas questões abertas para analisar rigorosamente seu tempo de execução.

Todos os algoritmos rápidos para logaritmos discretos em grupos finitos dependem de encontrar elementos suaves, ou seja, que podem ser expressos como produto de outros que em algum sentido são pequenos. Para inteiros ordinários,

suavidade significa que os elementos pequenos são primos pequenos. Quando os elementos do grupo são representados como polinômios sobre um corpo finito, os elementos pequenos são polinômios irredutíveis de baixo grau.

3 FUNDAMENTOS

Este capítulo contém alguns pré-requisitos da Teoria dos Números necessários à compreensão do estudo dos logaritmos discretos, e alguns conceitos básicos da Teoria da Complexidade.

3.1 Grupos, Anéis e Corpos

Vamos inicialmente relembrar alguns conceitos da Teoria dos Grupos.

Definição 3.1.1 *Um grupo é um conjunto não vazio G munido de uma operação binária interna $*$, associativa. Além disso, existe um elemento neutro em G e todo elemento de G possui um inverso. Se o grupo $(G, *)$ satisfizer ainda a propriedade comutativa, então o grupo é dito abeliano.*

Dizemos que um grupo multiplicativo G é cíclico se existe um elemento $a \in G$ tal que para todo $b \in G$ existe algum inteiro j com $b = a^j$. Tal elemento a é chamado um gerador do grupo cíclico, e escrevemos $G = \langle a \rangle$ (grupo cíclico gerado por a).

Um grupo é chamado finito se contém apenas um número finito de elementos distintos. O número de elementos distintos de um grupo finito é chamado de ordem do grupo . Escreveremos $|G|$ para indicar a ordem de um grupo G.

Definição 3.1.2 *Um anel $(R, +, \cdot)$ é um conjunto não vazio R com duas operações internas, $+$ e $\cdot$, usualmente chamadas adição e multiplicação, onde $(R, +)$ é um grupo abeliano, a operação $\cdot$ é associativa e $\cdot$ é distributiva em relação a $+$, tanto à direita quanto à esquerda.*

Um anel $(R, +, \cdot)$ é chamado anel com unidade se existe um elemento neutro para $(R, \cdot)$.

Um anel $(R, +, \cdot)$ é chamado anel comutativo se a operação $\cdot$ satisfaz a propriedade comutativa.

Um anel $(R, +, \cdot)$ comutativo com unidade é chamado domínio de integridade se R não possui divisores próprios de zero. Um elemento não nulo $b \in R$ é dito um divisor próprio de zero se existir um elemento não nulo $a \in R$ tal que $a \cdot b = 0$.

Se $(R, +, \cdot)$ é um domínio de integridade, com $0 \neq 1$ e todo elemento não nulo de $(R, +, \cdot)$ tem inverso multiplicativo, então o anel é chamado de corpo.

3.2 Números Inteiros

Lembraremos algumas definições bastante conhecidas sobre o anel dos inteiros.

Dados inteiros a e b, dizemos que a divide b (ou b é divisível por a) se existe algum inteiro d tal que $b = ad$ (notação: $a \mid b$). Nesse caso dizemos também que a é um divisor de b ou que b é múltiplo de a.

Para todo inteiro b temos que $1 \mid b$ e $b \mid b$. Dizemos que b é primo quando possuir exatamente dois divisores positivos, a saber, 1 e ele mesmo.

Dados $a, b \in \mathbb{Z}$ com $a \neq 0$ ou $b \neq 0$, o máximo divisor comum de a e b é o maior inteiro que divide simultaneamente a e b. Notação: $mdc(a, b)$. Se $mdc(a, b) = 1$, dizemos que a e b são relativamente primos.

Teorema 3.2.1 *(Algoritmo da Divisão): Dados inteiros a e b, $b > 0$, existe um único par de inteiros q e r tais que $a = qb + r$, com $0 \leq r < b$. Neste caso, q é chamado de quociente e r de resto da divisão de a por b.*

Lema 3.2.2 *Se c e d são inteiros e $c = dq + r$ onde q e r são inteiros, então $mdc(c, d) = mdc(d, r)$.*

Teorema 3.2.3 *(Algoritmo de Euclides): Sejam $r_0 = a$ e $r_1 = b$ inteiros tais que $a \geq b > 0$. Se o algoritmo da divisão for aplicado sucessivamente para se obter*

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

para $j = 0, 1, ..., n-1$ e $r_{n+1} = 0$, então $mdc(a, b) = r_n$, o último resto não nulo.

Exemplo. Para encontrar $mdc(252, 198)$, usamos o algoritmo da divisão sucessivamente para obter $252 = 198 \cdot 1 + 54$, $198 = 54 \cdot 3 + 36$, $54 = 36 \cdot 1 + 18$, e $36 = 18 \cdot 2 + 0$. O último resto não nulo é 18. Portanto $mdc(252, 198) = 18$.

Prova. Sejam $r_0 = a$ e $r_1 = b$ inteiros, tais que $a \geq b > 0$. Aplicando sucessivamente o algoritmo da divisão (Teorema 3.2.1), encontramos que

$$\begin{aligned}
r_0 &= q_1 r_1 + r_2 \text{ e } 0 \leq r_2 < r_1, \\
r_1 &= q_2 r_2 + r_3 \text{ e } 0 \leq r_3 < r_2, \\
&\vdots \\
r_j &= q_{j+1} r_{j+1} + r_{j+2} \text{ e } 0 \leq r_{j+2} < r_{j+1}, \\
&\vdots \\
r_{n-2} &= q_{n-1} r_{n-1} + r_n \text{ e } 0 \leq r_n < r_{n-1}, \\
r_{n-1} &= q_n r_n + 0.
\end{aligned}$$

Visto que $r_1 > r_2 > r_3 > ...$, e todos estes restos são inteiros não negativos, finalmente um resto 0 pode ser obtido.

Pelo lema 3.2.2 temos que $mdc(a, b) = mdc(r_0, r_1) = mdc(r_1, r_2) = ... = mdc(r_{n-1}, r_n) = mdc(r_n, 0) = r_n$. Então $mdc(a, b) = r_n$. ∎

Das equações acima resulta o seguinte resultado importante:

Teorema 3.2.4 *(Algoritmo de Euclides Estendido) Sejam a e b inteiros positivos e $d = mdc(a, b)$. Então existem inteiros λ e μ tais que $\lambda a + \mu b = d$.*

O Algoritmo de Euclides Estendido é o cálculo do Algoritmo de Euclides feito de trás para frente.

Exemplo. Dado $mdc(252, 198) = 18$, fazendo o cálculo de trás para frente, teremos

$$\begin{aligned} 18 &= 54 - 36 \cdot 1 = 54 - (198 - 54 \cdot 3) \cdot 1 = 54 \cdot 4 - 198 \cdot 1 \\ 18 &= (252 - 198 \cdot 1) \cdot 4 - 198 \cdot 1 = 252 \cdot 4 - 198 \cdot 5, \end{aligned}$$

donde $\lambda = 4$ e $\mu = -5$.

O próximo teorema refere-se às equações diofantinas, também usadas na determinação dos logaritmos discretos. Chamam-se de equações diofantinas às equações polinomiais com coeficientes inteiros. Nos ocuparemos de um tipo especial, da forma $ax + by = n$, com a, b, n inteiros.

Teorema 3.2.5 *(Equações Diofantinas) Sejam a e b inteiros, com $a \neq 0$ e $b \neq 0$ e $d = mdc(a, b)$. A equação $ax + by = n$ admite solução se, e somente se, $d \mid n$. Se (x_0, y_0) é uma solução particular da equação $ax + by = n$, então (x, y) é uma solução da equação se, e somente se,*

$$\begin{aligned} x &= x_0 + t\frac{b}{d} \\ y &= y_0 - t\frac{a}{d}, \end{aligned}$$

para algum $t \in \mathbb{Z}$.

A prova é facilmente encontrada na literatura (ver, por exemplo, [14], pg. 113).

3.3 Congruências

As congruências são o instrumento adequado para dar ênfase ao resto na divisão euclidiana.

Definição 3.3.1 *Sejam a e b números inteiros e m um inteiro positivo. Dizemos que a é congruente a b módulo m, e escrevemos $a \equiv b \pmod{m}$, se $m \mid (a - b)$, isto é, se $a = b + km$, para algum inteiro k. Se $a \equiv b \pmod{m}$, dizemos que b é um representante da classe residual de a módulo m.*

Exemplo. $22 \equiv 4 \pmod 9$, pois $9 \mid (22 - 4)$.

Dados a, b, c, d e m inteiros e $m > 0$, as congruências módulo m satisfazem as seguintes propriedades:

1. $a \equiv a \pmod m$ (reflexiva);
2. Se $a \equiv b \pmod m$, então $b \equiv a \pmod m$ (simétrica);
3. Se $a \equiv b \pmod m$ e $b \equiv c \pmod m$, então $a \equiv c \pmod m$ (transitiva).

   Portanto a relação de conjuntos módulo m é uma relação de equivalência.

   Então, se $a \equiv b \pmod m$ e $c \equiv d \pmod m$, é fácil ver que:

4. $a + c \equiv b + d \pmod m$;
5. $ac \equiv bd \pmod m$.

Dados inteiros a, b e m, chamamos de congruência linear em uma variável a uma congruência da forma $ax \equiv b \pmod m$ onde x é uma incógnita.

Teorema 3.3.2 *Sejam a, b e m inteiros, com $m > 0$ e $mdc(a, m) = d$. Se d não divide b, então $ax \equiv b \pmod m$ não admite solução. Se d divide b, então $ax \equiv b \pmod m$ possui exatamente d soluções incongruentes módulo m.*

Definição 3.3.3 *Sejam a e m inteiros, com $mdc(a, m) = 1$, então a solução de $ax \equiv 1 \pmod m$ é dita o inverso de a módulo m.*

Para determinar o inverso de um número módulo m utilizamos o Algoritmo de Euclides Estendido. Como $mdc(a, m) = 1$, então de acordo com o Teorema 3.2.4 existem inteiros x e k tais que $ax + km = 1$, portanto $m \mid ax - 1$, logo $ax \equiv 1 \pmod m$.

Exemplo. Cálculo do inverso de 5 módulo 14.

Como $mdc(5,14) = 1$, pois $14 = 5 \cdot 2 + 4$, $5 = 4 \cdot 1 + 1$, e $4 = 2 \cdot 2 + 0$.

Fazendo o cálculo de trás para frente, teremos

$$1 = 5 - 4 \cdot 1 = 5 - (14 - 5 \cdot 2) \cdot 1 = 5 \cdot 3 - 1 \cdot 14,$$

portanto, $5 \cdot 3 \equiv 1 \pmod{14}$, isto é, o inverso de 5 módulo 14 é 3.

Convém observar que nem sempre existe o inverso de a módulo m (ver Teorema 3.2.5).

A seguir apresentaremos o Teorema Chinês dos Restos que também é amplamente usado para determinar logaritmos discretos.

Teorema 3.3.4 *(Teorema Chinês dos Restos) Sejam $m_1, m_2, ..., m_r$ inteiros positivos relativamente primos dois a dois. Então o sistema de congruências*

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

tem uma única solução módulo $M = m_1 m_2 ... m_r$.

Para cada $k = 1, 2, ..., r$, seja $M_k = \frac{M}{m_k}$. Temos $mdc(M_k, m_k) = 1$, portanto pelo Teorema 3.3.2 existe um inteiro y_k tal que $M_k y_k \equiv 1 \pmod{m_k}$. Seja $x = a_1 M_1 y_1 + a_2 M_2 y_2 + ... + a_r M_r y_r$. Como $m_k \mid M_j$ se $j \neq k$ então temos que $x \equiv a_k M_k y_k \pmod{m_k}$, portanto $x \equiv a_k \cdot 1 \equiv a_k \pmod{m_k}$

Unicidade da solução

Sejam x_0 e x_1 ambas soluções simultâneas do sistema de r congruências acima. Então para cada k, $x_0 \equiv x_1 \pmod{m_k}$, logo $m_k \mid (x_0 - x_1)$. Como temos que $mdc(m_i, m_j) = 1$ para $i \neq j$, então $M \mid (x_0 - x_1)$ e $x_0 \equiv x_1 \pmod{M}$. Isto mostra que a solução é única. ∎

Vamos ilustrar o teorema acima através do seguinte exemplo.

Exemplo. Seja o sistema de congruências

$$x \equiv 1 \pmod 3$$

$$x \equiv 2 \pmod 5$$

$$x \equiv 3 \pmod 7.$$

Calculamos $M = 3 \cdot 5 \cdot 7 = 105$ e $M_1 = \frac{105}{3} = 35$, $M_2 = \frac{105}{5} = 21$, $M_3 = \frac{105}{7} = 15$. Usando Algoritmo de Euclides Estendido, calculamos y_1, y_2 e y_3, resolvendo $35y_1 \equiv 1 \pmod 3$, $21y_2 \equiv 1 \pmod 5$ e $15y_3 \equiv 1 \pmod 7$, respectivamente. Encontramos $y_1 \equiv 2 \pmod 3$, $y_2 \equiv 1 \pmod 5$ e $y_3 \equiv 1 \pmod 7$, formando assim a soma

$$x = 1 \cdot 35 \cdot 2 + 2 \cdot 21 \cdot 1 + 3 \cdot 15 \cdot 1 = 70 + 42 + 45 = 157 \equiv 52 \pmod{105},$$

que é a solução simultânea para o nosso sistema de congruências.

3.4 Caracterização de Corpos Finitos

Corpos com um número finito e grande de elementos têm um papel importante em vários ramos da Matemática: Teoria dos Números, Teoria de Grupos, Geometria Projetiva, etc. Os exemplos mais familiares de tais corpos são os $\mathbb{Z}_p$ para p primo, mas estes não são todos. É possível dar uma classificação completa de todos os corpos finitos. Mostra-se que um corpo finito é univocamente determinado, a menos de isomorfismo, pelo número de elementos que contém; que este número é necessariamente uma potência de um primo; e que para cada primo p e para cada inteiro $n > 0$ existe um corpo com p^n elementos.

Aqui faremos uma breve caracterização dos corpos finitos, suficiente para a compreensão do que segue.

Teorema 3.4.1 *Se F é um corpo finito, então F possui característica $p > 0$, e o número de elementos de F é p^n, onde n é o grau de F sobre seu corpo primo.*

Lembramos que o corpo primo $\pi(k)$ de um corpo k é a intersecção de todos os subcorpos de k. É fácil ver que $\pi(k)$ é um subcorpo de k. O grau de uma extensão de corpos M sobre k é a dimensão de M como espaço vetorial sobre k.

Portanto não existem corpos com 6, 10, 12, 14, 18, 20,... elementos. Note o contraste com a Teoria dos Grupos, onde existem grupos com qualquer ordem dada. Entretanto, existem grupos não-isomorfos de mesma ordem. Isto não pode acontecer com corpos finitos, como veremos a seguir.

Teorema 3.4.2 *Seja p um número primo arbitrário e seja $q = p^n$ onde n é um inteiro positivo qualquer. Um corpo F tem q elementos se, e somente se, F é o corpo de decomposição de $f(t) = t^q - t$ sobre o subcorpo primo $P \simeq \mathbb{Z}_p$ de F.*

Lembramos que o subcorpo primo de um corpo F é a interseção de todos os subcorpos de F.

Visto que os corpos de decomposição existem e são únicos a menos de isomorfismo, podemos deduzir o seguinte

Teorema 3.4.3 *Todo corpo finito F tem $q = p^n$ elementos onde p é a característica do corpo e n é um inteiro positivo. Para cada q deste tipo existe, a menos de isomorfismo, precisamente um corpo com q elementos, que é o corpo de decomposição para $t^q - t$ sobre $\mathbb{Z}_p$.*

O corpo com q elementos será denotado por $GF(q)$. A notação origina-se de *Galois Field*, dada em homenagem a Évariste Galois.

A classificação de corpos finitos dada acima, embora seja um resultado útil em si mesmo, não nos dá informações mais detalhadas sobre a estrutura mais profunda. Há várias questões que podemos formular - quais são os subcorpos?

Quantos eles são? Quais são os grupos de Galois? Nós nos contentamos em provar um importante teorema, que dá a estrutura do grupo multiplicativo $F \setminus \{0\}$ de um corpo finito F arbitrário. Para tal, necessitamos conhecer um pouco mais sobre grupos abelianos.

Definição 3.4.4 *O* expoente $e(G)$ *de um grupo finito* G *é o mínimo múltiplo comum das ordens dos elementos de* G.

Como a ordem de cada elemento de G é um divisor da ordem de G, temos que $e(G)$ divide a ordem de G. Em geral G não precisa ter um elemento de ordem $e(G)$; por exemplo se $G = S_3$, onde S_3 indica o grupo das permutações de 3 elementos, então $e(G) = 6$. Mas G não possui elementos de ordem 6. No entanto, grupos abelianos comportam-se melhor com respeito a isto:

Lema 3.4.5 *Todo grupo abeliano finito* G *contém um elemento de ordem* $e(G)$.

Corolário 3.4.6 *Se* G *é um grupo abeliano tal que* $e(G) = |G|$, *então* G *é cíclico.*

Podemos aplicar o Corolário 3.4.6 imediatamente.

Teorema 3.4.7 *Se* G *é um subgrupo finito do grupo multiplicativo* $F \setminus \{0\}$ *de um corpo* F, *então* G *é cíclico.*

Corolário 3.4.8 *O grupo multiplicativo de um corpo finito é cíclico.*

Por esta razão para cada corpo finito F existe ao menos um elemento x tal que todo elemento não nulo de F é um potência de x.

Parece não haver nenhum outro procedimento conhecido para achar um gerador além do de tentativa e erro. Felizmente, a existência de um gerador é usualmente uma informação suficiente para o estudo dos algoritmos apresentados no nosso trabalho.

Definição 3.4.9 *Sejam K um corpo e $p(x)$ um polinômio em $K[x]$ tal que $p(x) \neq 0$. Sejam $a(x)$ e $b(x)$ elementos arbitrários de $K[x]$. Então escrevemos $a(x) \equiv b(x)$ (mod$p(x)$), se $p(x) \mid (a(x) - b(x))$.*

Seja n o grau de $p(x)$, cada polinômio $d(x)$ em $K[x]$ satisfaz a equivalência $d(x) \equiv r(x) \pmod{p(x)}$ para um único polinômio $r(x)$, tal que $r(x) = 0$ ou $r(x) \neq 0$ e grau de $r(x)$ menor que n, e portanto podemos formar o conjunto com todos possíveis restos depois da divisão por $p(x)$, isto é, o conjunto formado pelo polinômio nulo e por todos os polinômios não nulos cujo grau é menor que n.

A adição de elementos deste conjunto é a adição usual de $K[x]$. A multiplicação é feita via congruência módulo $p(x)$, isto é, se $r_1(x)$ e $r_2(x)$ são elemntos deste conjunto, $r_1(x) \cdot r_2(x)$ será o único elemento $r(x)$ do conjunto tal que $r_1(x) \cdot r_2(x) \equiv r(x) \pmod{p(x)}$. É fácil mostrar que com estas operações o conjunto é um anel que será denotado $K[x]/p(x)$.

Definição 3.4.10 *Sejam K um corpo, $p(x) \in K(x)$ um polinômio não nulo. O anel $(K[x]/p(x), +, \cdot)$ é dito ser o anel quociente de $K[x]$ módulo $p(x)$.*

Teorema 3.4.11 *Sejam K um corpo e $p(x)$ um polinômio em $K[x]$. Então $K[x]/p(x)$ é um corpo se e somente se $p(x)$ é irredutível em $K[x]$.*

Considerando que um corpo H satisfaz o Teorema 3.4.11, este será chamado extensão do corpo K.

Um corpo finito é um corpo com número finito de elementos. Se H for um corpo finito, pelo Teorema 3.4.3, temos que H tem p^n elementos , onde p é a característica de H e n é o grau de H sobre $\mathbb{Z}_p$. Pelo Teorema 3.4.3, temos que, a menos de isomorfismo, existe um só corpo com p^n elementos. Se no Teorema 3.4.11 tivermos $K = Z_p$ e grau de $p(x)$ igual a n, então o corpo $\mathbb{Z}_p[x]/p(x)$ é a menos de isomorfismo o único corpo com p^n elementos. Os elementos de $\mathbb{Z}_p[x]/p(x)$ são unicamente representados módulo $p(x)$, pelos polinômios de grau menor que n de $\mathbb{Z}_p$.

Exemplo. Seja $p(x) = x^3 + x^2 + 1$ que é um polinômio irredutível em $\mathbb{Z}_2[x]$ $GF(2^3)$ e $H = \mathbb{Z}_2[x]/(x^3 + x^2 + 1)$. Os elementos de H consistem de todos polinômios de grau menor que 3 com coeficientes em $\mathbb{Z}_2$. Visto que tem somente duas escolhas, 0 e 1, para cada coeficiente, os polinômios são os que seguem (os polinômios serão representados pelos seus coeficientes segundo as potências decrescentes de x): $0 = 000$, $1 = 001$, $x = 010$, $x + 1 = 011$, $x^2 = 100$, $x^2 + 1 = 101$, $x^2 + x = 110$, $x^2 + x + 1 = 111$.

+	000	001	010	011	100	101	110	111
000	000	001	010	011	100	101	110	111
001	001	000	011	010	101	100	111	110
010	010	011	000	001	110	111	100	101
011	011	010	001	000	111	110	101	100
100	100	101	110	111	000	001	010	011
101	101	100	111	110	001	000	011	010
110	110	111	100	101	010	011	000	001
111	111	110	101	100	011	010	001	000

·	000	001	010	011	100	101	110	111
000	000	000	000	000	000	000	000	000
001	000	001	010	011	100	101	110	111
010	000	010	100	110	101	111	001	011
011	000	011	110	101	001	010	111	100
100	000	100	101	001	111	011	010	110
101	000	101	111	010	011	110	100	001
110	000	110	001	111	010	100	011	101
111	000	111	011	100	110	001	101	010

3.5 Complexidade Computacional

Se queremos discutir a segurança de criptosistemas contra ataques computacionais, precisamos discutir sua dificuldade computacional. O campo da Matemática que trata disto é a Teoria da Complexidade Computacional.

Antes de abordar esta teoria queremos definir algoritmo.

Um algoritmo é simplesmente uma seqüência de regras para realizar um cálculo, manualmente, ou mais usualmente, por máquinas. Nosso estudo está envolvido com algoritmos para uso em computadores. Os métodos usados para somar, multiplicar ou dividir números são exemplos de algoritmos. O mais conhecido algoritmo é o Algoritmo de Euclides (Teorema 3.2.3) para calcular o máximo divisor comum de dois inteiros.

Complexidade não tem o significado de complicado. Complexidade de um algoritmo é a quantidade de trabalho feito por ele, que em muitos casos é o número de perações. A quantidade de trabalho feito não pode ser representada por um único número porque o número de passos realizados não é o mesmo para todas as entradas de dados. A quantidade de trabalho feito usualmente depende do tamanho desta entrada de dados.

Diante da necessidade da solução de um problema e considerando que, freqüentemente, mais de um algoritmo pode ser utilizado, deve-se escolher o mais adequado. Dependendo das nossas prioridades e do limite do equipamento disponível, podemos escolher o algoritmo que leva menor tempo de execução, ou que usa menos armazenagem de dados, ou que é mais fácil para implementar, e assim por diante.

Dentre os parâmetros utilizados para avaliar o desempenho de um algoritmo, destacam-se o tempo de execução e a memória utilizada. O tempo de execução é sem sombra de dúvida o parâmetro mais usual na avaliação da eficiência de algoritmos. Freqüentemente, o tempo de execução dependerá do tamanho da instância. O tamanho ou comprimento de uma instância corresponde formalmente ao número de *bits* necessários para representá-la no computador.

A função que associa o tempo de execução de um algoritmo ao comprimento ou tamanho dos dados de entrada denomina-se complexidade em tempo do algoritmo. Em outros casos, quando o interesse é examinar a quantidade de memória

necessária para acomodar os dados de entrada e executar o processo, define-se a complexidade em espaço do algoritmo.

Para se obter a complexidade em tempo e espaço, deve ser especificado o tempo necessário para executar cada instância e o espaço utilizado por cada registro. Para tanto, sabemos que o número de *bits* $\omega(\varphi)$ necessários para representar um inteiro $\varphi \in \mathbb{Z}$ em um registro é dado por

$$\omega(\varphi) = \lfloor \log |\varphi| \rfloor + 1.$$

O tempo exigido por um algoritmo, ou a armazenagem usada, podem variar consideravelmente entre duas instâncias diferentes de mesmo tamanho. Em função destas grandes variações que podem ocorrer, às vezes fica complicado para decidir o tempo que um algoritmo leva somente em termos do tamanho da instância a ser resolvida. Para isto, usualmente são considerados dois casos: pior caso e caso médio de um algoritmo.

No pior caso de um algoritmo, para cada tamanho de instância somente se consideram aqueles onde o algoritmo requer o maior tempo. A análise no pior caso é apropriada para um algoritmo onde o tempo de resposta (reação) é crítico. Seja D_n o conjunto de *inputs* de tamanho n para um problema, e seja I um elemento de D_n. Seja $t(I)$ o número de operações básicas realizadas pelo algoritmo no *input* I. Definimos a função W por

$$W(n) = max\{t(I) \mid I \in D_n\}.$$

$W(n)$ é o número máximo de operações básicas realizadas pelo algoritmo em qualquer *input* de tamanho n. A análise no pior caso fornece um limite superior do trabalho realizado por um algoritmo.

Se um algoritmo precisa ser usado muitas vezes em muitas instância diferentes, pode ser mais importante conhecer o tempo de execução médio em instâncias de mesmo tamanho, isto é, calcular o número de operações realizadas em cada *input* de tamanho n e então fazer a média. Na prática alguns *inputs* po-

dem ocorrer muito mais freqüentemente que outros, assim um peso médio é muito significante. Seja $p(I)$ a probabilidade de ocorrência do *input* I. Então o comportamento médio de um algoritmo é definido como

$$A(n) = \sum_{I \in D_n} p(I)t(I).$$

Determinamos $t(I)$ analisando o algoritmo, mas $p(I)$ não pode ser calculado analiticamente. A função p é determinada com experiência e/ou informação especial sobre a aplicação para a qual o algoritmo será usado, ou fazendo algumas simples suposições. Por exemplo, supondo que todos *inputs* de tamanho n têm a mesma probabilidade de ocorrerem. Se p é complicado, o cálculo de comportamento médio é difícil.

É usualmente mais difícil analisar o comportamento médio de um algoritmo do que analisar seu comportamento no pior caso. Além disso, uma análise de comportamento médio pode ser enganosa se as instâncias a serem resolvidas não são realmente escolhidas aleatoriamente quando o algoritmo é usado na prática.

Uma análise útil do comportamento médio de um algoritmo requer, portanto, algum conhecimento na distribuição das instâncias a serem resolvidas. Isto é normalmente um requerimento não realístico. Especialmente quando um algoritmo é usado como um procedimento interno em algum algoritmo mais complexo, isto pode não ser prático para estimar em qual instância é mais provável para acontecer, e em qual ocorre raramente.

A conclusão de uma análise no caso médio pode depender crucialmente da hipótese de uma distribuição de probabilidade inicial nos momentos que o algoritmo pode ser solicitado para resolver. Esta análise pode ser enganosa se realmente nossa hipótese produzida não corresponder á realidade de aplicação que usa o algoritmo. Na maioria das vezes, as análises no caso médio são interpretadas segundo uma hipótese mais ou menos realística tal que todas as instâncias de qualquer tamanho dado são igualmente prováveis.

3.5.1 Notação Assintótica

Esta notação permite simplificar substancialmente a expressão do tempo ou espaço exigido por um algoritmo. É chamada assintótica porque trata o comportamento da função no seu limite.

Considerando a função $t : \mathbb{N} \rightarrow \mathbb{R}^+$ e n o tamanho das instâncias exigidas por um dado algoritmo, então $t(n)$ representa a quantidade de determinado recurso gasto naquela instância com uma implementação particular deste algoritmo.

Sejam f e g: $\mathbb{N} \rightarrow \mathbb{R}^+$ duas funções. Diz-se que f domina g assintoticamente quando existem constantes $C \in \mathbb{R}^+$ e $n_0 \in \mathbb{N}^*$ tais que $g(n) \leq Cf(n)$, $\forall n \geq n_0$.

As funções assintóticas apresentam razão de crescimento diferente, isto é, f domina g assintoticamente se, e somente se

$$\lim_{n \rightarrow +\infty} \frac{g(n)}{f(n)} = 0, \text{ caso o limite exista.}$$

A idéia geral é que, se $f(n)$ e $g(n)$ representarem complexidades de tempo para dois algoritmos, então para problemas de tamanho n_0 ou mais, a execução do algoritmo de tempo $g(n)$ nunca será maior que C vezes o tempo de execução do algoritmo de complexidade $f(n)$ para a mesma instância do problema. Portanto, $Cf(n)$ pode ser utilizado como limite assintótico para $g(n)$.

Para exprimir a complexidade assintótica de um algoritmo, utiliza-se a notação $O(.)$, introduzida por P. Bachmann (1894).

Definição 3.5.1 *Seja $f : \mathbb{N} \rightarrow \mathbb{R}^*$. Dizemos que $O(f)$ é o conjunto de funções $g : \mathbb{N} \rightarrow \mathbb{R}^*$ tais que para algum $c \in \mathbb{R}^+$ e algum $n_0 \in \mathbb{N}$, $g(n) \leq c.f(n)$, para todo $n \geq n_0$.*

O conjunto $O(f)$ é usualmente chamado "Oh de f".

Existe uma técnica alternativa para mostrar que g pertence à $O(f)$:

$$g \in O(f) \text{ se } \lim_{n \to +\infty} \frac{g(n)}{f(n)} = c, \text{ para algum } c \in \mathbb{R}^* \text{ (caso o limite exista).}$$

Isto é, se o limite da razão de g para f existe e não é infinito, então g não cresce mais rápido que f.

Analogamente pode-se especificar um limite inferior para $g(n)$. Utiliza-se neste caso a notação $\Omega(.)$.

Definição 3.5.2 *Seja $f : \mathbb{N} \to \mathbb{R}^*$. Dizemos que $\Omega(f)$ é o conjunto de funções $g : \mathbb{N} \to \mathbb{R}^*$ tal que para algum $c \in \mathbb{R}^+$ e algum $n_0 \in \mathbb{N}$, $g(n) \geq c.f(n)$, para todo $n \geq n_0$.*

A técnica alternativa para mostrar que g pertence à $\Omega(f)$ é:

$$g \in \Omega(f) \text{ se } \lim_{n \to +\infty} \frac{g(n)}{f(n)} = \infty, \text{ ou se } \lim_{n \to +\infty} \frac{g(n)}{f(n)} = c > 0 \text{ (caso o limite exista).}$$

Também podemos limitar o tempo de execução simultaneamente acima e abaixo. Para isto utilizamos a notação $\Theta(.)$.

Definição 3.5.3 *Seja $f : \mathbb{N} \to \mathbb{R}^*$. Então definimos $\Theta(f) = O(f) \cap \Omega(f)$.*

Também temos:

$g \in \Theta(f)$ se $\lim_{n \to +\infty} \frac{g(n)}{f(n)} = c$, para algum $c \in \mathbb{R}^+$. Isto é, $c \neq 0$ e $c \neq \infty$.

Exemplo 6: Sejam $f, g : \mathbb{N} \to \mathbb{R}^*$ dadas por $f(n) = \frac{n^3}{2}$ e $g(n) = 37n^2 + 120n + 17$, $\forall n \in \mathbb{N}$. Então $g \in O(f)$, mas $f \notin O(g)$.

4 ALGORITMOS SHANKS E POLLARD

Neste capítulo descreveremos dois algoritmos para a determinação dos logaritmos discretos em grupos arbitrários. São algoritmos que funcionam na ausência de qualquer informação extra sobre o grupo.

Sejam G um grupo e $b \in G$ a base para a qual iremos calcular o logaritmo discreto. Suponhamos que a ordem de b em G seja n. O algoritmo mais óbvio é simplesmente construir uma tabela contendo todas as n potências de b e olhar os elementos do grupo na tabela para encontrar seu logaritmo discreto. Isto evidentemente exige pelo menos n operações para calcular a tabela e espaço $O(n)$ para armazená-la. A meta de qualquer algoritmo é melhorar este limite.

4.1 Algoritmo Shanks

Em 1973, D. Shanks (ver, por exemplo, [17]) desenvolveu um método mais eficiente para calcular logaritmos discretos. Para este algoritmo necessitamos uma enumeração dos elementos de G.

No que segue, seja G um grupo e $b \in G$ um elemento de ordem finita. Dado $a \in \langle b \rangle$, existe um único natural x, $0 \leq x \leq |\langle b \rangle| - 1$ tal que $a = b^x$, portanto, o logaritmo discreto de a na base b é bem definido.

Seja $n \in \mathbb{N}$, $n \geq |\langle b \rangle|$ e $m = \lceil \sqrt{n} \rceil$.

Dado $a \in G$, vamos calcular o logaritmo discreto

$$x = log_b a.$$

Apresentamos a seguir um esboço do algoritmo Shanks.

Algoritmo Shanks. Sob as hipóteses acima, seguimos as seguintes etapas:

(i) Calcular ab^{-i}, com $0 \leq i < m$.

(ii) Construir o conjunto S formado pelos pares ordenados (i, ab^{-i}).

(iii) Calcular b^{mj}, com $0 \leq j \leq m$.

(iv) Construir o conjunto T formado pelos pares ordenados (j, b^{mj}).

(v) Encontrar um par $(i, y) \in S$ e um par $(j, y) \in T$ (as segundas coordenadas são iguais).

(vi) Calcular $mj + i$, que é a solução de $log_b a$ módulo $|\langle b \rangle|$.

Faremos a seguir uma descrição mais detalhada do algoritmo.

Primeiramente mostraremos a existência de um par ordenado em cada conjunto S e T, tal que segundas coordenadas são iguais.

Pelo Algoritmo da Divisão (Teorema 3.2.1)existem inteiros q e r tal que $x = mq + r$, com $0 \leq r < m$. Como x e m são positivos, teremos $q \geq 0$, caso contrário, teríamos $r > m$. Por outro lado $q \leq m$, pois se não fosse, teríamos $qm > m^2 > n \geq |\langle b \rangle| > x$, portanto $r = x - qm$ seria negativo. Logo $x = mq + r$, com $0 \leq r < m$ e $0 \leq q \leq m$. Portanto, $a = b^x = b^{mq+r} = b^{mq}b^r$, logo $ab^{-r} = b^{mq}$ com $0 \leq r < m$ e $0 \leq q \leq m$. Fazendo $j = q$ e $i = r$ temos $(i, ab^{-i}) \in S$, $(j, b^{mj}) \in T$ e $ab^{-i} = b^{mj}$.

A seguir, mostraremos que para i e j conforme acima, obteremos $x \equiv mj + i \pmod{|\langle b \rangle|}$. De fato, como $ab^{-i} = b^{mj}$ então $a = b^x = b^{mq+r} = b^{mj+i}$, logo $\log_b a \equiv mj + i \pmod{|\langle b \rangle|}$

A seguir apresentaremos alguns exemplos.

Exemplo 1. Queremos determinar $x \equiv log_2 12 \pmod{52}$. Considere $G = \mathbb{Z}_{53}^*$ que é um grupo cíclico gerado por 2. Tome $n = 53 \geq |\langle b \rangle|$, $m = \lceil \sqrt{53} \rceil = 8$.

Por (i) calcula-se ab^{-i} módulo 53, para $0 \leq i \leq 7$. Obtemos assim o conjunto S (conforme (ii))

$$S = \{(0,12),(1,6),(2,3),(3,28),(4,14),(5,7),(6,30),(7,15)\}.$$

Por (iii) calcula-se b^{mj} módulo 53, para $0 \leq j \leq 8$. Obtemos o conjunto T (conforme (iv))

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

Analisando os conjuntos S e T encontramos os pares $(3,28) \in S$, $(2,28) \in T$, cujas segundas coordenadas são iguais. Segue que $i = 3$ e $j = 2$. Podemos então calcular $8 \cdot 2 + 3 = 19$. Logo $log_2 12 \equiv 19 \pmod{52}$.

Exemplo 2. Conforme exemplo anterior, supondo $p = 53$. Então $m = \lceil \sqrt{53} \rceil = 8$. Para $a = 45$ e $b = 2$, determinemos $x \equiv log_2 45 \pmod{53}$.

Por (i) calcula-se ab^{-i} módulo 53, para $0 \leq i \leq 7$. Obtemos assim o conjunto S, (conforme (ii))

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

Por (iii) calcula-se b^{mj} módulo 53, para $0 \leq j \leq 8$. Obtemos o conjunto T (conforme (iv))

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

Analisando os conjuntos S e T encontramos os pares $(5,13) \in S$, $(3,13) \in T$, cujas segundas coordenadas são iguais. Assim $i = 5$ e $j = 3$. Então podemos calcular $8 \cdot 3 - 5 = 29$. Portanto, $log_2 45 \equiv 29 \pmod{52}$.

O algoritmo Shanks é determinístico. O seu tempo de execução é $O(\sqrt{n} \log n)$. De fato, o tempo de execução é dominado pela aritmética requerida para calcular as duas listas S e T e o tempo para resolvê-las, onde $\log n$ é o número

de operações de grupo necessárias para calcular as potências ([2], pg. 243-246) e $\sqrt{n}$ é um limitante superior para o número de potências ab^{-i} e b^{mj} das listas. O espaço requerido é $O(\sqrt{n})$ elementos do grupo, que é o espaço necessário para guardar as duas listas S e T.

Uma questão importante é que o algoritmo Shanks pode funcionar até se n é somente um limite superior da ordem do grupo $\langle b \rangle$, e não necessariamente a ordem do grupo. Podemos até retirar a necessidade do limite superior, visto que podemos simplesmente escolher qualquer inteiro n para usar e, se o limite resultar muito pequeno, então simplesmente o dobramos e repetimos os passos até que a resposta seja encontrada. Portanto, o algoritmo Shanks pode também ser usado para calcular a ordem n de b em tempo $O(\sqrt{n} \log n)$.

O algoritmo Shanks pode ser considerado de interesse principalmente teórico, visto que existe outro muito mais prático devido a J.M. Pollard (será apresentado a seguir) que parece utilizar essencialmente o mesmo tempo de execução e requer muito menos espaço. Em contrapartida, este precisa da ordem do grupo.

4.2 Algoritmo Pollard

O algoritmo apresentado a seguir é também conhecido como o método Rho de Pollard. Este método foi desenvolvido em 1974 por J.M. Pollard para ser um método de fatoração baseado em congruências.

O método é chamado Rho porque analisando o comportamento periódico de uma seqüência $(x_i)_{i \in \mathbb{N}}$, verificamos que uma parte desta não é periódica. Ela ocorre antes da periodicidade e é a cauda do Rho (ρ) (letra do alfabeto grego); a parte periódica forma o laço.

Antes de apresentar o algoritmo Pollard faremos uma descrição do método de fatoração, chamado método Monte Carlo para fatoração ([32]).

4.2.1 Método Rho de Pollard para Fatoração

Sejam n um inteiro composto grande e p seu menor divisor primo. O objetivo é encontrar inteiros $x_0, x_1, ..., x_s$, tais que os resíduos não negativos são distintos módulo n, mas os resíduos não negativos módulo p não são todos distintos.

Depois de encontrados os inteiros x_i e x_j onde $0 \leq i < j \leq s$ tais que $x_i \equiv x_j \pmod{p}$ mas $x_i \not\equiv x_j \pmod{n}$, portanto $p \mid (x_i - x_j)$, logo $mdc(x_i - x_j, n)$ é um divisor não trivial de n. Assim, o objetivo é gerar, aleatoriamente, um número apropriado de inteiros distintos $x_0, x_1, ..., x_s$ com $0 \leq x_i < n$ e então verificar todos os pares (x_i, x_j), com $i \neq j$, para obter $mdc(x_i - x_j, n) > 1$. Podemos encontrar $mdc(x_i - x_j, n)$ usando o algoritmo de Euclides. Para encontrar $mdc(x_i - x_j, n)$ para todo par (i, j) com $0 \leq i < j \leq s$, requer que encontremos $O(s^2)$ máximos divisores comuns, pois são $s + 1$ combinações de dois elementos $\frac{(s+1)s}{2} \approx s^2$.

A seguir, vamos mostrar como este número de máximos divisores comuns pode ser reduzido.

Para encontrar os inteiros x_i e x_j, iniciamos selecionando um valor x_0, o qual é escolhido aleatoriamente, e uma função polinomial $f(x)$ com coeficientes inteiros e grau maior que um. Calculamos os termos x_k, $k = 1, 2, ...$ usando a definição recursiva

$$x_{k+1} \equiv f(x_k) \pmod{n}, \text{ para } 0 \leq x_{k+1} < n.$$

A função polinomial $f(x)$ deve ser tal que a seqüência $(x_0, x_1, ..., x_k, ...)$ seja muito próxima de uma seqüência aleatória. Ela não é uma seqüência aleatória de fato, visto que damos uma regra para calcular os termos. No entanto, esta seqüência pode ter termos com algumas propriedades de uma seqüência aleatória.

O seguinte exemplo ilustra como esta seqüência é gerada.

Exemplo 2: Sejam $n = 8051$, $x_0 = 2$ e $f(x) = x^2 + 1$. Então é fácil ver que $x_1 = 5$, $x_2 = 26$, $x_3 = 677$,

$$f(x_3) = x_4 \equiv 677^2 + 1 \equiv 7474 \pmod{8051},$$

e continuando, obteremos $x_5 = 2839$, $x_6 = 871$, $x_7 = 1848$, $x_8 = 1481$, $x_9 = 3490$, $x_{10} = 6989$, $x_{11} = 705$, $x_{12} = 5915$, $x_{13} = 5631$, $x_{14} = 3324$, $x_{15} = 3005$, $x_{16} = 4855$, $x_{17} = 5749$, $x_{18} = 1647$, $x_{19} = 7474$, $x_{20} = 2839$, $x_{21} = 871$, $x_{22} = 1848$, e assim por diante.

Pela a definição recursiva de x_k, com $0 \leq x_k < n$, e visto que $x^2 + 1$ é um polinômio, segue que se $x_i \equiv x_j \pmod{n}$ então,

$$x_{i+1} \equiv f(x_i) \equiv f(x_j) \equiv x_{j+1} \pmod{n}.$$

Assim se $x_i \equiv x_j \pmod{n}$, então a seqüência torna-se periódica módulo n com período $j - i$. Isto é, $x_q \equiv x_r \pmod{n}$ quando $q \equiv r \pmod{j-i}$, e $q \geq i$ e $r \geq i$. Portanto se k é o menor múltiplo de $j - i$ (período), que é pelo menos i, então $2k$ é também um múltiplo de $j - i$ e, conseqüentemente, $x_k \equiv x_{2k} \pmod{n}$. Logo, ao invés de considerar todos pares x_i, x_j com $0 \leq i, j \leq s$, basta comparar x_i com x_{2i}.

A seguir faremos um argumento heurístico de que k é da ordem de $\sqrt{p}$, onde p é o menor fator primo de n.

Consideremos a seqüência $(x_0, x_1, ..., x_s)$ para a qual os x_i são todos distintos módulo p e $x_s \equiv x_{2s} \pmod{p}$. A probabilidade de que esta seqüência ocorra é $(\frac{p-1}{p})(\frac{p-2}{p})...(\frac{p-s}{p}) = P(s)$, de modo que o comprimento da "cauda" é s, que é a quantidade de vezes que necessitamos calcular x_i e x_{2i} e mais a comparação $mdc(x_{2i} - x_i, n)$. A expressão $\sum_{s=1}^{p} sP(s)$ é a estimativa que procuramos para o tempo médio de execução do algoritmo. É possível mostrar que $\sum_{s=1}^{p} sP(s) \in O(\sqrt{p})$ ([15]). Assim, o número médio de cálculos de máximos divisores comuns efetuados é $O(\sqrt{p})$.

Para encontrar um fator de n, basta encontrarmos $mdc(x_{2k} - x_k, n)$ para $k = 1, 2, ...$ Teremos encontrado um fator de n quando encontrarmos $k \in \mathbb{N}$ para o qual $1 < x_{2k} - x_k < n$ e $mdc(x_{2k} - x_k, n) \neq 1$.

Na prática, quando o método Rho de Pollard é usado, muitas vezes a função polinomial $f(x) = x^2 + 1$ é escolhida para gerar a seqüência de inteiros $(x_0, x_1, ..., x_k, ...)$, pois esta função quase sempre se comporta como se fosse aleatória. Além disso, muitas vezes pré-selecionamos $x_0 = 2$. A escolha desta função polinomial e do valor x_0 produzem uma seqüência que se comporta bem (como uma seqüência aleatória) para o objetivo deste método de fatoração.

Exemplo 3: Sejam a função polinomial geradora $f(x) = x^2 + 1$ e $x_0 = 2$. Queremos encontrar um fator primo de 8051, usando o método Rho de Pollard.

Note que a seqüência $(x_i)_{i \in \mathbb{N}}$ já foi construída no exemplo anterior.

Em seguida calculamos $mdc(x_{2k} - x_k, 8051)$ para $k = 1, 2, 3....$ Teremos encontrado um fator não trivial de 8051 quando encontrarmos um valor $k \in \mathbb{N}$ para o qual $1 < x_{2k} - x_k < n$.

$mdc(x_2 - x_1, n) = mdc(21, 8051) = 1$ fator trivial

$mdc(x_4 - x_2, n) = mdc(7448, 8051) = 1$ fator trivial

$mdc(x_6 - x_3, n) = mdc(194, 8051) = 97$ fator primo não trivial.

4.2.2 Algoritmo Pollard

J.M. Pollard usou a mesma teoria da fatoração descrita acima para calcular logaritmos discretos. A seguir, apresentamos um esboço do algoritmo Pollard.

Seja G um grupo cíclico de ordem n, e $b \in G$ um gerador de G. Dado $a \in G$, vamos calcular o logaritmo discreto

$$x = log_b a.$$

Sob as hipóteses acima, seguimos as seguintes etapas:

(i) Repartir G em três conjuntos S_1, S_2 e S_3 de tamanho aproximadamente igual.

(ii) Calcular a seqüência $(x_j)_{j\in\mathbb{N}}$ definida abaixo, até obter $i \in \mathbb{N}$ tal que $x_i = x_{2i}$.

(iii) Calcular as seqüências $(a_j)_{j\in\mathbb{N}}$ e $(b_j)_{j\in\mathbb{N}}$, tais que $x_i = a^{a_i}b^{b_i}$.

(iv) Calcular inteiros s e t tais que $a^s = b^t$, onde $s \equiv a_i - a_{2i} \pmod{n}$ e $t \equiv b_i - b_{2i} \pmod{n}$.

(v) Se $mdc(s,n) = 1$, calcular u, inverso de s módulo n. Caso contrário, passar para o item (vii).

(vi) Calcular $t \cdot u$ módulo n, que é a solução procurada.

(vii) Se $mdc(s,n) = d > 1$, usamos o algoritmo de Euclides estendido para construir inteiros u e v tais que $d = us + vn$.

(viii) Calcular $k \equiv (\frac{ut}{d}) \pmod{n}$.

(ix) Calcular $k + (i\frac{n}{d})$ para $1 \leq i \leq d$ até obter $a = b^{k+(i\frac{n}{d})}$, obtendo assim a solução procurada.

Faremos agora um esboço da prova da correção do algoritmo Pollard.

No primeiro estágio do algoritmo, calculamos inteiros s e t tais que

$$a^s = b^t$$

O procedimento desta atividade envolve a construção de uma seqüência de elementos $(x_0, x_1, ...)$. Começamos partindo G em três conjuntos S_1, S_2 e S_3 de tamanho aproximadamente igual. Definimos $x_0 = 1$ e, para $i \geq 0$, seja

$$x_{i+1} = \left\{ \begin{array}{ll} ax_i, & \text{para } x_i \in S_1 \\ {x_i}^2, & \text{para } x_i \in S_2 \\ bx_i, & \text{para } x_i \in S_3 \end{array} \right\} \qquad (4.1)$$

A idéia desta definição é que as três possibilidades são escolhidas de uma maneira aleatória, e a seqüência resultante é suficientemente complicada a ponto de poder ser considerada como um mapeamento aleatório. Além disso, todos x_i são facilmente representáveis em termos de a_i e b_i.

Se $x_i \in S_1$, isto é se $x_{i+1} = ax_i$, então $a^{a_{i+1}}b^{b_{i+1}} = aa^{a_i}b^{b_i} = a^{a_i+1}b^{b_i}$. Podemos escolher $a_{i+1} \equiv a_i + 1 \pmod{n}$ e $b_{i+1} \equiv b_i \pmod{n}$.

Se $x_i \in S_2$, isto é se$x_{i+1} = {x_i}^2$, então $a^{a_{i+1}}b^{b_{i+1}} = (a^{a_i}b^{b_i})^2 = a^{2a_i}b^{2b_i}$. Podemos escolher $a_{i+1} \equiv 2a_i \pmod{n}$ e $b_{i+1} \equiv 2b_i \pmod{n}$.

E, se $x_i \in S_3$, isto é se$x_{i+1} = bx_i$, então $a^{a_{i+1}}b^{b_{i+1}} = ba^{a_i}b^{b_i} = a^{a_i}b^{b_i+1}$. Podemos escolher $a_{i+1} \equiv a_i \pmod{n}$ e $b_{i+1} \equiv b_i + 1 \pmod{n}$.

Daí, fixando $a_0 \equiv 0 \pmod{n}$ e $b_0 \equiv 0 \pmod{n}$, obtemos

$$a_{i+1} \equiv \left\{ \begin{array}{rl} a_i + 1, & \text{para } x_i \in S_1 \\ 2a_i, & \text{para } x_i \in S_2 \\ a_i, & \text{para } x_i \in S_3 \end{array} \right\} \pmod{n}, \tag{4.2}$$

e similarmente

$$b_{i+1} \equiv \left\{ \begin{array}{rl} b_i, & \text{para } x_i \in S_1 \\ 2b_i, & \text{para } x_i \in S_2 \\ b_i + 1, & \text{para } x_i \in S_3 \end{array} \right\} \pmod{n}. \tag{4.3}$$

A seqüência $(x_i)_{i\geq 0}$ gerada comporta-se como se fosse uma seqüência aleatória. Ela não é aleatória, pois foi obtida seguindo regras, mas seus termos têm algumas propriedades que uma seqüência aleatória possui. Se a seqüência $(x_i)_{i\geq 0}$ for uma seqüência aleatória de elementos de G, então podemos esperar que exista um inteiro $i \leq 3\sqrt{n}$ tal que $x_i = x_{2i}$. Tal inteiro pode ser encontrado calculando as sequências $(x_i, a_i, b_i, x_{2i}, a_{2i}, b_{2i})$ recursivamente usando (4.1), (4.2), (4.3).

Encontrado $x_i = x_{2i}$, então obtemos $a^{a_i}b^{b_i} = a^{a_{2i}}b^{b_{2i}}$, ou equivalentemente, $a^{a_i - a_{2i}} = b^{b_{2i} - b_i}$.

Segue que $a^s = b^t$ se s e t forem tais que $s \equiv a_i - a_{2i} \pmod{n}$ e $t \equiv b_{2i} - b_i \pmod{n}$.

Se $mdc(s, n) = 1$, então simplesmente calculamos um inteiro u tal que $us \equiv 1 \pmod{n}$, onde u é o inverso de s módulo n. Este inverso sempre vai existir, pois satisfaz o teorema 3.2.5.

Assim, $(a^s)^u = (b^t)^u$, ou equivalentemente, $a = b^{tu}$. Donde concluímos que $log_b a = tu$. Portanto $x \equiv tu \pmod{n}$.

Por outro lado, se $mdc(s, n) = d > 1$, então usamos o algoritmo de Euclides Estendido para construir inteiros u e v tais que $d = us + vn$. Como temos $d \equiv us \pmod{n}$, podemos escolher u tal que $0 < u < n$.

De $a^s = b^t$, segue $a^{us} = b^{ut}$. Como $d = us + vn$, temos que

$$\begin{aligned} a^d &= a^{vn+us} \Rightarrow \\ a^d &= a^{vn}a^{us} \Rightarrow \\ a^d &= (a^n)^v a^{us} \Rightarrow \\ a^d &= 1a^{us} = a^{us} = b^{ut}. \end{aligned}$$

Portanto $a^d = b^{ut}$.

Como $a = b^x$ com $x < n$, então de $a^d = b^{ut}$, segue que $b^{xd} = b^{ut}$, o que implica que $xd \equiv ut \pmod{n}$. Como $d \mid n$ temos que d divide ut, com u, t, d e x todos menores que n. Podemos então escrever $xd = ut + in$ ou $x = \frac{ut}{d} + i\frac{n}{d}$ para algum inteiro i com $0 \leq i \leq d$ e $k \equiv (\frac{ut}{d}) \pmod{n}$. Temos assim $\log_b a = x = k + i\frac{n}{d}$ para algum i.

Se $i > d$, então $in > xd$, pois $n \geq x$. Como u, t são positivos então $in + ut > xd$. Logo $i \leq d$. Se $i = 0$, então $xd = ut$ e $x = \frac{ut}{d}$.

Para calcular o logaritmo discreto precisamos testar a equação acima com os valores de i até que o valor correto seja encontrado. Note que se s for um resíduo aleatório módulo n, então podemos esperar que valores grandes de d sejam

raros. Assim o fator dominante do tempo de execução usualmente origina-se do tempo para encontrar i.

A seguir, apresentaremos alguns exemplos.

Exemplo 4. Vamos determinar $log_2 12$ em $(\mathbb{Z}_{53}^*, \cdot)$ que será um natural único módulo 52. Pois $a \in \mathbb{Z}_{53}^*$ e $a^x = a^y$, então $a^{x-y} = 1$ pois $ordem(a) \mid x - y$. Se $\mathbb{Z}_{53}^* = \langle a \rangle$, então $52 \mid x - y$, isto é $x \equiv y \pmod{52}$.

Inicialmente vamos dividir $G = \mathbb{Z}_{53}^*$ em três grupos S_1, S_2 e S_3

$$S_1 = \{1, 2, ..., 17\},\ S_2 = \{18, ..., 35\},\ S_3 = \{36, ..., 52\}.$$

Em seguida calculamos $x_0, x_1, ...$ até encontrar $x_i = x_{2i}$. Para evitar a armazenagem de dados, podemos considerar que com a seqüência $x_{i+1} = f(x_i)$ e $y_{i+1} = f(f(y_i))$, com $x_0 = y_0$, daí, segue que $y_k = x_{2k}$ e, portanto, não precisa armazenar nada. Começamos com

$$x_1 = 12, x_2 = 38$$

a partir daí usaremos a idéia acima:

$$y_0 = 1, x_0 = 1$$

$$y_1 = 38, x_2 = 38$$

$$y_2 = 52, x_4 = 52$$

$$y_3 = 49, x_6 = 49$$

$$y_4 = 37, x_8 = 37$$

$$y_5 = 17, x_{10} = 17$$

$$y_6 = 37, x_{12} = 37$$

$$y_7 = 17, x_{14} = 17$$

$$y_8 = 37, x_{16} = 37$$

Encontramos $x_8 = x_{16}$. Precisamos calcular $a_0, a_1, ..., a_{16}$ e $b_0, b_1, ..., b_{16}$

$$a_0 = 0, a_1 = 1, a_2 = 2, a_3 = 2, a_4 = 4, a_5 = 4, a_6 = 4, a_7 = 4, a_8 = 4,$$

$$a_9 = 4, a_{10} = 8, a_{11} = 9, a_{12} = 9, a_{13} = 9, a_{14} = 18, a_{15} = 19, a_{16} = 19$$

$$b_0 = 0, b_1 = 0, b_2 = 0, b_3 = 1, b_4 = 2, b_5 = 3, b_6 = 4, b_7 = 5, b_8 = 6,$$

$$b_9 = 7, b_{10} = 14, b_{11} = 14, b_{12} = 15, b_{13} = 16, b_{14} = 32, b_{15} = 32, b_{16} = 33$$

Calculamos s e t,

$$s = a_8 - a_{16} = 4 - 19 \equiv 37 \pmod{52} \text{ e}$$

$$t = b_{16} - b_8 = 33 - 6 \equiv 27 \pmod{52}.$$

Como $mdc(37, 52) = 1$, então resolvemos $37u \equiv 1 \pmod{52}$. Donde $u \equiv 45 \pmod{52}$. Então $x = 45 \cdot 27 \equiv 19 \pmod{52}$. Logo $log_2 12 \equiv 19 \pmod{52}$.

Exemplo 5. Supondo $p = 53$, $a = 45$ e $b = 2$. Vamos determinar $x \equiv log_2 45 \pmod{52}$.

Inicialmente dividimos $G = \mathbb{Z}_{53}^*$ em três grupos S_1, S_2 e S_3

$$S_1 = \{1, 2, ..., 17\},\ S_2 = \{18, ..., 35\},\ S_3 = \{36, ..., 52\}.$$

Em seguida, calculamos $x_0, x_1, ...$ até encontrar $x_i = x_{2i}$. Começamos com $x_1 = 45, x_2 = 37$ e seguimos com:

$$y_0 = 1, x_0 = 1$$

$$y_1 = 37, x_2 = 37$$

$$y_2 = 17, x_4 = 17$$

$$y_3 = 52, x_6 = 52$$

$$y_4 = 49, x_8 = 49$$

$$y_5 = 37, x_{10} = 37$$

$$y_6 = 17, x_{12} = 17$$

$$y_7 = 52, x_{14} = 52$$

$$y_8 = 49, x_{16} = 49$$

Encontramos $x_8 = x_{16}$. Precisamos calcular $a_0, a_1, ..., a_{16}$ e $b_0, b_1, ..., b_{16}$

$$a_0 = 0, a_1 = 1, a_2 = 1, a_3 = 1, a_4 = 2, a_5 = 3, a_6 = 6, a_7 = 6, a_8 = 6,$$

$$a_9 = 6, a_{10} = 6, a_{11} = 6, a_{12} = 12, a_{13} = 13, a_{14} = 26, a_{15} = 26, a_{16} = 26$$

$$b_0 = 0, b_1 = 0, b_2 = 1, b_3 = 2, b_4 = 4, b_5 = 4, b_6 = 8, b_7 = 9, b_8 = 10,$$

$$b_9 = 11, b_{10} = 12, b_{11} = 13, b_{12} = 26, b_{13} = 26, b_{14} = 0, b_{15} = 1, b_{16} = 2$$

Calculamos s e t,

$$s = a_8 - a_{16} = 6 - 26 \equiv 32 \pmod{52} \text{ e}$$

$$t = b_{16} - b_8 = 2 - 10 \equiv 44 \pmod{52}.$$

Como $mdc(32, 52) = 4 > 1$ resolvemos $4 = 32u + 52v$, encontrando $u = 5$. Calculamos $k = \frac{5 \cdot 44}{4} \equiv 3 \pmod{52}$. Logo $x = 3 + \frac{52i}{4} = 3 + 13i$. Testando os valores de $i = 0, 1, 2, 3$ concluímos que $i = 2$ satisfaz $45 \equiv 2^{3+13i} \pmod{53}$. Portanto $x = 3 + 13 \cdot 2 = 29$. E $log_2 45 \equiv 29 \pmod{52}$.

O algoritmo Pollard tem o mesmo tempo de execução do algoritmo Shanks, que é da ordem de $O(\sqrt{n} \log n)$, e requer muito menos espaço, praticamente nenhum. De fato, o tempo de execução é de $O(\sqrt{p})$, onde p é o menor fator primo de n, este tempo é exigido para calcular os elementos da seqüência x_i e fazer a comparação para encontrar $x_i = x_{2i}$. Para o espaço, temos que com a seqüência $x_{i+1} = f(x_i)$ e $y_{i+1} = f(f(y_i))$, com $x_0 = y_0$, daí, segue que $y_k = x_{2k}$ e, portanto, não precisa armazenar nada.

Para o tempo de execução dos algoritmos Shanks e Pollard não parece terem sido obtidos melhoramentos consideráveis até o momento. Somente melhoras com fatores constantes aparecem na literatura, ([34]). Por outro lado, tem havido progresso na obtenção de versões paralelas rápidas ([34]), onde o tempo necessário para o cálculo reduz-se a um fator linear em relação ao número de processadores

usados. Mas o processo básico para qualquer um destes algoritmos ainda requer um total de $\sqrt{p}$ passos.

Um ponto importante é que se podemos resolver o problema do logaritmo discreto usando uma base b, então podemos também resolver o problema do logaritmo discreto para uma base h, contanto que h pertença ao subgrupo cíclico gerado por b. Isto pode ser facilmente visto do fato que se $a \in \langle h \rangle$, então

$$\log_b a \equiv \log_b h \cdot \log_h a \pmod{n}$$

pois $a = h^x$, donde $\log a = \log h^x$. Portanto $\log_h a = x$ e $h = b^y$, donde $\log h = \log b^y$. Assim $\log_b h = y$. Então $a = b^{yx}$ e, aplicando o logaritmo, teremos $\log a = \log b^{xy}$. Donde $\log a = xy \log b$ e $\log_b a = xy$. Substituindo x e y obtemos a relação acima, onde n é a ordem de b.

5 ALGORITMO SILVER, POHLIG E HELLMAN

Neste capítulo estudaremos outro algoritmo para a determinação de logaritmos discretos. Este algoritmo é para grupos com ordem suave. Um inteiro positivo é chamado suave se seus fatores primos não são grandes. O algoritmo que será abordado foi deduzido em 1978, por S. C. Pohlig e M. E. Hellman ([37]) e independentemente por R. Silver. Por brevidade de notação vamos nos referir ao algoritmo Silver-Pohlig-Hellman como SPH.

5.1 Algoritmo SPH

No que segue, seja G um grupo cíclico multiplicativo b um gerador de G.

Dado $a \in G$, vamos calcular o logaritmo discreto $x = log_b a$. Note que o logaritmo discreto é bem definido. Como b é gerador de G temos que existe um único inteiro x, $0 \leq x < n$, onde n é a ordem de b, tal que $a = b^x$. Assim, para n, b, a como acima sempre existe o logaritmo discreto.

Sob as hipóteses acima, seguimos as etapas seguintes:

(i) Decompor n em fatores primos irredutíveis, isto é, $n = \prod_{i=1}^{t} {p_i}^{\alpha_i}$, onde p_i são números primos distintos, $\forall\ 1 \leq i \leq t$;

(ii) Calcular $y_i \equiv x \pmod{{p_i}^{\alpha_i}}$ para cada i, $1 \leq i \leq t$.

(iii) Calcular $x \equiv log_b a$ (mod n), usando o teorema Chinês dos Restos.

A seguir faremos um detalhamento dos passos do algoritmo SPH.

Seja p um número primo, $p \neq 2$, $n = p - 1$, $p_1, ..., p_t$ primos distintos tal que $n = \prod_{i=1}^{t} {p_i}^{\alpha_i}$. Seja b um gerador de ${GF_p}^*$ e $a \in {GF_p}^*$. Primeiramente, para cada p_i calculamos $r_{p_i,j} = b^{j\frac{n}{p_i}}$ com $0 \leq j \leq p_i - 1$.

Para cada i, $1 \leq i \leq t$, seja $R_i = \{r_{p_i,j} \,|0 \leq j \leq p_i - 1\}$. Seja $y_i \equiv x \pmod{{p_i}^{\alpha_i}}$, sabemos que existem e são únicos $x_0, x_1, ..., x_{\alpha_i - 1} \in \{0, 1, ..., p_i^{\alpha_i - 1}\}$ tal que $y_i \equiv x_0 + x_1 p_i + ... + x_{\alpha_i - 1} p_i^{\alpha_i - 1} \pmod{{p_i}^{\alpha_i}}$. Também podemos escrever $y_i \equiv \sum_{l=0}^{\alpha_i - 1} x_l {p_i}^l \pmod{{p_i}^{\alpha_i}}$, onde $0 \leq x_l \leq p_i - 1$. Vamos mostrar agora como determinar $x_0, x_1, ..., x_{\alpha_i - 1}$ e portanto y_i.

Inicialmente, determinemos x_0.

Como $y_i \equiv x \pmod{{p_i}^{\alpha_i}}$ então $y_i - x = k{p_i}^{\alpha_i}$ para algum $k \in \mathbb{Z}$, logo $\frac{n}{p_i} y_i - \frac{n}{p_i} x = \frac{n}{p_i} k{p_i}^{\alpha_i} = \frac{n}{p_i {p_j}^{\alpha_j}} k{p_i}^{\alpha_i} {p_j}^{\alpha_j}$. Observe que se $j \neq i$, $\frac{n}{p_i {p_j}^{\alpha_j}} \in \mathbb{N}$, logo $\frac{n}{p_i} y_i \equiv \frac{n}{p_i} x \pmod{{p_i}^{\alpha_i}}$, para $\forall j \in \{1, ..., t\}$. Como $n = {p_1}^{\alpha_1} ... {p_t}^{\alpha_t}$, então temos $\frac{n}{p_i} y_i \equiv \frac{n}{p_i} x \pmod{n}$, logo $b^{\frac{n}{p_i} y_i} = b^{\frac{n}{p_i} x}$. Como $a = b^x$, então $a^{\frac{n}{p_i}} = b^{x\frac{n}{p_i}}$. Portanto

$$
\begin{aligned}
a^{\frac{n}{p_i}} &= b^{y_i \frac{n}{p_i}} \Longrightarrow \\
a^{\frac{n}{p_i}} &= b^{(x_0 + x_1 p_i + ... + x_{\alpha_i - 1} {p_i}^{\alpha_i - 1})\frac{n}{p_i}} \Longrightarrow \\
a^{\frac{n}{p_i}} &= b^{x_0 \frac{n}{p_i}} b^{(x_1 p_i \frac{n}{p_i} + x_2 {p_i}^2 \frac{n}{p_i} + ... + x_{\alpha_i - 1} {p_i}^{\alpha_i - 1} \frac{n}{p_i})} \Longrightarrow \\
a^{\frac{n}{p_i}} &= b^{x_0 \frac{n}{p_i}} b^{(x_1 n + x_2 p_i n + ... + x_{\alpha_i - 1} {p_i}^{\alpha_i - 2} n)}.
\end{aligned}
$$

Como $b \in {GF_p}^*$, temos $b^n = 1$. Portanto $b^{(x_1 n + x_2 p_i n + ... + x_{\alpha_i - 1} {p_i}^{\alpha_i - 2} n)} = 1$, pois o expoente é múltiplo de n. Logo $a^{\frac{n}{p_i}} = b^{x_0 \frac{n}{p_i}}$.

Comparando $a^{\frac{n}{p_i}}$ com os elementos de R_i, encontramos x_0 tal que $x_0 = j$ satisfaz $a^{\frac{n}{p_i}} = b^{j\frac{n}{p_i}}$, isto é x_0 é o único elemento j, $0 \leq j \leq p_i - 1$, tal que $a^{\frac{n}{p_i}} = b^{j\frac{n}{p_i}}$.

Se $\alpha_i = 1$, o processo termina e podemos resolver o sistema (5.1) apresentado a seguir, usando o Teorema Chinês dos Restos 3.3.4, para encontrar a solução procurada.

Se $\alpha_i > 1$ precisamos determinar x_1.

Seja $a_1 = ab^{-x_0}$. Como $a = b^x$, portanto segue que $a_1 = b^{x-x_0}$. Então a_1 tem logaritmo discreto

$$x - x_0 \equiv x_1 p_i + ... + x_{\alpha_i - 1} {p_i}^{\alpha_i - 1} \pmod{{p_i}^{\alpha_i}}.$$

Logo $x - x_0 \equiv \sum_{l=1}^{\alpha_i - 1} x_l {p_i}^l \pmod{{p_i}^{\alpha_i}}$.

Como $a_1 = b^{x-x_0}$, então ${a_1}^{\frac{n}{{p_i}^2}} = b^{(x-x_0)\frac{n}{{p_i}^2}}$. Repetindo o argumento acima temos $\frac{n}{{p_i}^2} y_i \equiv \frac{n}{{p_i}^2} x \pmod{{p_j}^{\alpha_j}}$, para todo j, logo $\frac{n}{{p_i}^2} y_i \equiv \frac{n}{{p_i}^2} x \pmod{n}$, daí

$$\begin{aligned} {a_1}^{\frac{n}{{p_i}^2}} &= b^{(x_1 p_i + ... + x_{\alpha_i - 1} {p_i}^{\alpha_i - 1})\frac{n}{{p_i}^2}} \Longrightarrow \\ {a_1}^{\frac{n}{{p_i}^2}} &= b^{x_1 \frac{n}{p_i}} b^{(x_2 n + ... + x_{\alpha_i - 1} {p_i}^{\alpha_i - 3} n}. \end{aligned}$$

Novamente, conforme acima, $b^{(x_2 n + ... + x_{\alpha_i - 1} {p_i}^{\alpha_i - 3} n)} = 1$. Logo teremos ${a_1}^{\frac{n}{{p_i}^2}} = b^{x_1 \frac{n}{p_i}}$.

Comparando ${a_1}^{\frac{n}{{p_i}^2}}$ com os elementos de R_i, encontramos x_1 tal que $x_1 = j$ satisfaz ${a_1}^{\frac{n}{{p_i}^2}} = b^{j\frac{n}{p_i}}$, isto é x_1 é o único elemento j, $0 \leq j \leq p_i - 1$, tal que $a^{\frac{n}{{p_i}^2}} = b^{j\frac{n}{{p_i}^2}}$.

Se $\alpha_i = 2$ o processo termina e podemos resolver a equação (5.1).

Se $\alpha_i > 2$ repete-se o processo $(\alpha_i - 2)$ vezes mais, obtendo $x_2, ..., x_{\alpha_i - 1}$.

Este procedimento pode ser estendido para todo $2 \leq l \leq \alpha_i - 1$.

Seja $a_l = a(b^{(x_0 + x_1 p_i + ... + x_{l-1} {p_i}^{l-1})})^{-1}$. Então temos que

$$a_l = b^{x - (x_0 + x_1 p_i + ... + x_{l-1} {p_i}^{l-1})}.$$

Segue que a_l tem logaritmo discreto e que

$$x - (x_0 + x_1 p_i + ... + x_{l-1} {p_i}^{l-1}) \equiv x_l {p_i}^l + ... + x_{\alpha_i - 1} {p_i}^{\alpha_i - 1} \pmod{{p_i}^{\alpha_i}}.$$

Como $a_l = b^{x - (x_0 + x_1 p_i + ... + x_{l-1} {p_i}^{l-1})}$, então temos que

$${a_l}^{\frac{n}{{p_i}^{l+1}}} = b^{(x - (x_0 + x_1 p_i + ... + x_{l-1} {p_i}^{l-1}))\frac{n}{{p_i}^{l+1}}}.$$

Donde, como antes

$$\begin{aligned} a_l^{\frac{n}{p_i^{l+1}}} &= b^{(x_l p_i{}^l + \ldots + x_{\alpha_i-1} p_i{}^{\alpha_i-1})\frac{n}{p_i^{l+1}}} \Longrightarrow \\ a_l^{\frac{n}{p_i^{l+1}}} &= b^{((x_l p_i{}^l)\frac{n}{p_i^{l+1}} + (x_{l+1} p_i{}^{l+1})\frac{n}{p_i^{l+1}} + \ldots + (x_{\alpha_i-1} p_i{}^{\alpha_i-1})\frac{n}{p_i^{l+1}})} \Longrightarrow \\ a_l^{\frac{n}{p_i^{l+1}}} &= b^{x_l \frac{n}{p_i}} b^{(x_{l+1} n + x_{l+2} p_i n + \ldots + x_{\alpha_i-1} p_i{}^{\alpha_i-2} n)}. \end{aligned}$$

Sendo $b^{(x_{l+1}n + x_{l+2}p_i n + \ldots + x_{\alpha_i-1}p_i{}^{\alpha_i-2}n)} = 1$, temos que $a_l^{\frac{n}{p_i^{l+1}}} = b^{(x_l)\frac{n}{p_i}}$.

Outra vez existirá um único elemento $j \in \mathbb{R}_i$, tal que $x_l = j_l$ satisfaz $a_l^{\frac{n}{p_i^{l+1}}} = b^{j\frac{n}{p_i}}$. Toma-se $x_l = j$

Segue que para todo $1 \leq i \leq t$, $x \equiv y_i \equiv \sum_{l=0}^{\alpha_i - 1} x_l p_i{}^l \pmod{p_i{}^{\alpha_i}}$.

Obtemos assim o sistema

$$x \equiv y_i \pmod{p_i{}^{\alpha_i}}, \text{ para todo } 1 \leq i \leq t. \tag{5.1}$$

Para resolvê-lo, denotemos, para cada i, $1 \leq i \leq t$, $M_i = \frac{n}{p_i{}^{\alpha_i}}$.

Como $mdc(M_i, p_i{}^{\alpha_i}) = 1$, usando o Algoritmo de Euclides Estendido (teorema 3.2.4) encontramos, para cada i, um Y_i tal que $M_i Y_i \equiv 1 \pmod{p_i{}^{\alpha_i}}$. Pelo Teorema Chinês dos Restos (teorema 3.3.4), tomemos $x \equiv \sum_{i=1}^{t} y_i M_i Y_i \pmod{n}$.

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

Por motivo de completitude, apresentamos alguns exemplos de aplicação.

Exemplo 1. Supondo $p = 53$ temos $p - 1 = 52 = 2^2 \cdot 13^1$. Para $a = 12$ e $b = 2$, determinemos $x \equiv log_2 12 \pmod{52}$.

Primeiramente calcula-se x_a módulo 2^2 e x_b módulo 13^1. Para tal, inicia-se com a construção da tabela $R_1 = \{r_{p_1,j} \,|0 \leq j \leq 1\}$ e $R_2 = \{r_{p_2,j} \,|0 \leq j \leq 12\}$.

Para $p_1 = 2$, $r_{2,j} \equiv 2^{\frac{j(53-1)}{2}} \pmod{53}$, para $0 \leq j \leq 1$. Portanto temos que $r_{2,0} = 2^0 = 1$ e $r_{2,1} = 2^{26} = 52$

Para $p_2 = 13$, $r_{13,j} \equiv 2^{\frac{j(53-1)}{13}} \pmod{53}$, $0 \leq j \leq 12$.

Donde $r_{13,0} = 2^0 = 1$, $r_{13,1} = 2^4 = 16$, $r_{13,2} = 2^8 = 44$, $r_{13,3} = 2^{12} = 15$, $r_{13,4} = 2^{16} = 28$, $r_{13,5} = 2^{20} = 24$, $r_{13,6} = 2^{24} = 13$, $r_{13,7} = 2^{28} = 49$, $r_{13,8} = 2^{32} = 42$, $r_{13,9} = 2^{36} = 36$, $r_{13,10} = 2^{40} = 46$, $r_{13,11} = 2^{44} = 47$, $r_{13,12} = 2^{48} = 10$.

Agora calculamos x_i módulo ${p_i}^{\alpha_i}$, para $i = 1, 2$.

Para $p_1 = 2$, calculamos x_a, onde $x_a \equiv x_0 + 2x_1 \pmod 4$. Usando $12^{\frac{52}{2}} \equiv 52 \pmod{53}$ e comparando com R_1, temos $x_0 = 1$. De ${y_1}^{\frac{52}{4}} \equiv 52 \pmod{53}$, com $y_1 = \frac{12}{2} = 6$, concluímos que $x_1 = 1$. Donde $x_a \equiv 3 \pmod 4$.

Para $p_2 = 13$, calculamos x_b módulo 13, onde $x_b \equiv x_0 \pmod{13}$. Usando $12^{\frac{52}{13}} \equiv 13 \pmod{53}$ e comparando com R_2, concluímos que $x_0 = 6$. Donde $x_b \equiv 6 \pmod{13}$.

Obtemos assim o sistema

$$\begin{aligned} x_a &\equiv 3 \pmod 4 \\ x_b &\equiv 6 \pmod{13} \end{aligned}$$

Sejam $M = 4 \cdot 13 = 52$, $M_1 = \frac{52}{4} = 13$ e $M_2 = \frac{52}{13} = 4$. Calculamos y_1 e y_2 tal que, $13y_1 \equiv 1 \pmod 4$ e $4y_2 \equiv 1 \pmod{13}$. Donde $y_1 = 1$ e $y_2 = 10$. Portanto $x = 3 \cdot 13 \cdot 1 + 6 \cdot 4 \cdot 10 = 39 + 240 = 279$.

Logo $x \equiv 19 \pmod{52}$. Finalmente $log_2 12 \equiv 19 \pmod{52}$.

Exemplo 2. Supondo $p = 53$ temos $p - 1 = 52 = 2^2 \cdot 13^1$. Para $a = 45$ e $b = 2$, determinemos $x \equiv log_2 45 (\mathrm{mod} 52)$.

Primeiramente calcula-se x_a módulo 2^2 e x_b módulo 13^1. As tabelas R_1 e R_2 são as mesmas do Exemplo 1. Calculamos x_i módulo ${p_i}^{\alpha_i}$, para $i = 1, 2$.

Para $p_1 = 2$, calculamos x_a, onde $x_a \equiv x_0 + 2x_1 \pmod 4$. Usando $45^{\frac{52}{2}} \equiv 52 \pmod{53}$ e comparando com R_1, temos $x_0 = 1$. De ${y_1}^{\frac{52}{4}} \equiv 1 \pmod{53}$, com $y_1 = \frac{45}{2}$, concluímos que $x_1 = 0$. Donde $x_a \equiv 1 \pmod 4$.

Para $p_2 = 13$, calculamos x_b módulo 13, onde $x_b \equiv x_0 \pmod{13}$. Usando $45^{\frac{52}{13}} \equiv 15 \pmod{53}$ e comparando com R_2, concluímos que $x_0 = 3$. Donde $x_b \equiv 3 \pmod{13}$.

Obtemos assim o sistema

$$\begin{aligned} x_a &\equiv 1 \pmod 4 \\ x_b &\equiv 3 \pmod{13}. \end{aligned}$$

Sejam $M = 4 \cdot 13 = 52$, $M_1 = \frac{52}{4} = 13$ e $M_2 = \frac{52}{13} = 4$. Calculamos y_1 e y_2 tal que $13y_1 \equiv 1 \pmod 4$ e $4y_2 \equiv 1 \pmod{13}$. Donde $y_1 = 1$ e $y_2 = 10$. Portanto $x = 1 \cdot 13 \cdot 1 + 3 \cdot 4 \cdot 10 = 13 + 120 = 133$.

Logo $x \equiv 29 \pmod{52}$. Finalmente $log_2 45 \equiv 29 \pmod{52}$.

S.C. Pohlig e M.E. Hellman ([37]) mostraram que se $n = \prod_{i=1}^{t} {p_i}^{\alpha_i}$, com p_i primos distintos e $r_1, ..., r_k \in R$ com $0 \leq r_i \leq 1$, para todo $1 \leq i \leq t$, então logaritmos sobre F_p podem ser calculados utilizando-se apenas

$$O(\sum_{i=1}^{t} \alpha_i(\log p + {p_i}^{1-r_i}(1 + \log {p_i}^{r_i})))$$

operações de grupo. Faremos a análise detalhada desta complexidade a seguir.

O algoritmo SPH exige $O(\log p \sum_{i=1}^{t}(1 + {p_i}^{r_i}))$ *bits* de memória para guardar as tabelas R_i. Sabemos que $\log p$ determina o tamanho de cada instância, ou seja, o número de *bits* de cada elemento, ${p_i}^{r_i}$ representa o número de elementos de cada tabela R_i e o somatório indica a soma das i tabelas.

Para calcular o logaritmo discreto, o algoritmo exige um pré-cálculo (construção das tabelas R_i) que requer que $O(\sum_{i=1}^{t}({p_i}^{r_i} \log {p_i}^{r_i} + \log p))$ operações no grupo sejam realizadas inicialmente. Visto que a memória tende ser mais cara que o cálculo, valores de $r < \frac{1}{2}$ são de maior interesse, e então o esforço do pré-cálculo não é significativo.

Prova da complexidade do algoritmo SPH:

Consideremos $b^{j(\frac{p-1}{p_i})} = w$ e $b^{(\frac{p-1}{p_i})} = y$, portanto temos $w = y^j$, com $0 \leq j \leq p_i - 1$. Uma multiplicação ou adição módulo p é considerada como uma única operação.

Seja $m = \lceil p_i^{r_i} \rceil$. Então existem inteiros c e d tais que $j = cm + d$, com $0 \leq c < \lceil \frac{p_i}{m} \rceil \approx p_i^{1-r_i}$ e $0 \leq d < m \approx p_i^{r_i}$.

Resolvendo $w = y^j$ para j é equivalente a encontrar c e d tais que $y^d = wy^{-cm}$, pois $w = y^j = y^{cm+d} = y^{cm}y^d$. Donde $wy^{-cm} = y^d$.

Para encontrar os elementos c e d construímos uma tabela com y^d para $d = 0, 1, ..., m-1$ em $O(p_i^{r_i})$ operações, pois são $p_i^{r_i}$ potenciações, e então calculamos os valores restantes em $O(p_i^{r_i} \log_2 p_i^{r_i})$ operações, pois são $p_i^{r_i}$ potenciações e uma potenciação exige $\log_2 p_i^{r_i}$ operações.

Em seguida calculamos $w, wy^{-m}, wy^{-2m}, ...$ e comparamos com a tabela calculada de $\{y^d\}$.

Cada valor de c testado requer uma multiplicação módulo p e $\log p_i^{r_i}$ comparações, ou $(1 + \log p_i^{r_i})$ operações ao todo. Há $O(p_i^{1-r_i})$ valores de c para serem testados, pois $0 \leq c < p_i^{1-r_i}$.

O Teorema Chinês dos Restos (teorema 3.3.4) requer $O(t)$ operações, $O(t \log_2 p)$ *bits* de memória e $O(t \log_2 p)$ operações de pré-cálculo.

Então, para $\{r_i\}_{i=1}^t$, com $0 \leq r_i \leq 1$, logaritmos sobre G podem ser calculados em $O(\sum_{i=1}^t \alpha_i(\log p + p_i{}^{1-r_i}(1 + \log p_i{}^{r_i})))$ operações de grupo com $O(\log p \sum_{i=1}^t (1 + p_i{}^{r_i}))$ bits de memória. O pré-cálculo exigido para calcular as tabelas R_i requer $O(\sum_{i=1}^t (p_i{}^{r_i} \log p_i{}^{r_i} + \log p))$ operações e é sem importância quando $r_t < \frac{1}{2}$.

O algoritmo SPH calcula logaritmos discretos em grupos G usando ordem de $\sqrt{p_i} \log p_i$ operações e uma quantidade comparável de armazenagem, onde p_i é o maior fator primo de n. Comparando com o algoritmo Shanks, notamos que

o algoritmo SPH é mais rápido, pois o algoritmo Shanks é da ordem de $(\sqrt{n}\log n)$. O algoritmo SPH é eficiente quando todos os fatores primos de n são razoavelmente pequenos. (Ele é muito eficiente em grupos nos quais p é um primo de Fermat, $p = 2^m + 1$).

O algoritmo SPH mostra que nos sistemas de criptografia devem ser evitados grupos para os quais a ordem tem todos fatores primos pequenos.

6 ALGORITMO INDEX CALCULUS

Neste capítulo estudaremos uma classe de algoritmos para a determinação de logaritmos discretos, que se aplicam muito bem quando o grupo envolvido tem uma estrutura especial. O algoritmo *Index Calculus* é constantemente melhorado. Apresentaremos aqui a versão básica e o melhoramento feito por D. Coppersmith.

O algoritmo *Index Calculus*, aparentemente, apareceu primeiro nos trabalhos de M. Kraitchik ([21], pg 119-123, [20], pg 69-70, 216-267) e Cunningham ([39]). As idéias básicas são devido a A.E. Western e J.C.P. Miller. O algoritmo foi redescoberto mais tarde independentemente por L.M. Adleman ([1]), R. Merkle ([28]) e J.M. Pollard ([33]), e sua complexidade computacional foi parcialmente analisada por L.M. Adleman.

6.1 Descrição Geral

O algoritmo *Index Calculus* tem três estágios. Os primeiros dois, são chamados de pré-cálculo, porque não dependem do elemento a do qual queremos determinar o logaritmo discreto. Estes só precisam ser realizados uma vez, e podem então ser usados para muitos cálculos de logaritmos discretos de mesma base b. O terceiro estágio consiste do cálculo do logaritmo discreto desejado. A seguir especificaremos estes três estágios.

Seja G um grupo multiplicativo. Para $b \in G$ sejam $\langle b \rangle$ um subgrupo cíclico gerado por b e n a ordem de b.

Dado $a \in \langle b \rangle$, vamos calcular o logaritmo discreto

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

Sob as hipóteses acima, sejam $p_1, ..., p_m$ elementos de $\langle b \rangle$, seguimos as etapas seguintes:

Primeiro estágio

(i) Escolher aleatoriamente um elemento $g_i \in \mathbb{N}$ e calcular $r_i = b^{g_i}$.

(ii) Tentar fatorar r_i como um produto de $(p_1, p_2, ..., p_m)$. Se r_i se fatora, então obtemos a relação $b^{g_i} = \prod_{j=1}^{m} {p_j}^{a_{ij}}$ e se não se fatora, escolhemos outro g_i.

Segundo estágio

(iii) Resolver as congruências lineares $g_i \equiv \sum_{j=1}^{m} a_{ij} \log_b p_j \pmod{n}$.

Terceiro estágio

(iv) Escolher aleatoriamente um elemento $e_t \in \mathbb{N}$ e calcular $l_t = ab^{e_t}$.

(v) Tentar fatorar l_t como um produto de $(p_1, p_2, ..., p_m)$. Se l_t fatora, obtemos $l_t = \prod_{j=1}^{m} {p_j}^{u_j}$.

(vi) Calcular $\log_b a \equiv \sum_{j=1}^{m} u_j \log_b p_j - e_t \pmod{n}$, que é a solução procurada.

O algoritmo *Index Calculus* depende de uma múltipla divisão de elementos (inteiros ou polinômios) em elementos tirados de um conjunto menor, tipicamente consistindo de valores que são "pequenos". Elementos que partem deste caminho são chamados suaves e um problema fundamental na análise do algoritmo é estimar a probabilidade com que algum processo pode produzir elementos suaves.

O método *Index Calculus* não é um método geral porque não é óbvio como gerar, de maneira eficiente, a relação do item (ii) para um grupo genérico. O *Index Calculus* é conhecido para alguns corpos finitos e classes de grupos de corpos

de números quadráticos imaginários. Queremos descrever os métodos básicos quando aplicados para corpos finitos $GF(p)$ para p um primo e $GF(2^k)$ um anel (quociente) de polinômios com coeficientes em $\mathbb{Z}_2$ e módulo um polinômio irredutível de grau k. Para a descrição do algoritmo em classes de grupos de corpos de números quadráticos imaginários ver ([25]); e para uma discussão de algoritmos para aplicar em outras classes de corpos finitos ver ([9]) e ([12]).

6.2 Algoritmo para Grupos Finitos $GF(p)^*$

O algoritmo *Index Calculus* aplica-se bem à $GF(p)^*$, pois o problema de fatoração, que é muito usada nesta técnica, é considerado fácil para muitos inteiros.

Sejam p um número primo, $GF_p{}^*$ o grupo multiplicativo do corpo finito F_p^*, b um gerador de $GF_p{}^*$ e $n = p - 1$ a ordem de b.

Dado $b \in GF_p{}^*$, vamos calcular o logaritmo discreto $x \equiv log_b a \pmod{n}$.

No método *Index Calculus* para $GF_p{}^*$, os elementos $p_1, ..., p_m$ são m primos pequenos.

Primeiro escolhemos aleatoriamente um inteiro $g_i \in [1, n]$, $1 \leq i \leq n$ e calculamos o menor inteiro positivo r_i com $r_i \equiv b^{g_i} \pmod{p}$. Então tentamos fatorar r_i como um produto dos m primos pequenos $(p_1, p_2, ..., p_m)$, usando simples divisão com estes primos. Se r_i fatora deste modo (na maioria dos casos r_i não fatora e g_i será descartado), então obtemos a relação

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

Aplicando log de ambos os lados, obtemos $\log_b b^{g_i} = \log_b \prod_{j=1}^{m} p_j{}^{a_{ij}}$ e, de acordo com as propriedades dos logaritmos, segue que

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

Este conjunto de identidades pode ser interpretado como um conjunto de congruências lineares

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

Na segunda fase resolvemos o sistema para $\log_b p_j$. Depois destes dois estágios iniciais, calculamos, no terceiro estágio, o logaritmo individual $\log_b a$. Para calcular $\log_b a$, novamente escolhemos aleatoriamente um inteiro e_t, calculamos

$$l_t \equiv ab^{e_t} \pmod{p},$$

e como no primeiro estágio do algoritmo, testamos se l_t fatora como o produto dos m primeiros primos. Se não, escolhemos aleatoriamente outro e_t, e assim por diante, até obter o inteiro e_t tal que $l_t = \prod_{j=1}^{m} p_j{}^{u_j}$. Isto implica $ab^{e_t} = \prod_{j=1}^{m} p_j{}^{u_j}$ ou $a = \prod_{j=1}^{m} p_j{}^{u_j} \cdot (b^{e_t})^{-1}$.

Aplicando log em ambos os membros, obtemos

$$\log_b a = \log_b \prod_{j=1}^{m} p_j{}^{u_j} \cdot (b^{e_t})^{-1}.$$

Portanto, temos

$$\log_b a = \log_b \prod_{j=1}^{m} p_j{}^{u_j} - \log_b b^{e_t}$$

Donde segue que

$$\begin{aligned}
\log_b a &= \sum_{j=1}^{m} \log_b p_j{}^{u_j} - \log_b b^{e_t} \Rightarrow \\
\log_b a &= \sum_{j=1}^{m} u_j \log_b p_j - e_t \log_b b \Rightarrow \\
\log_b a &= \sum_{j=1}^{m} u_j \log_b p_j - e_t.
\end{aligned}$$

Donde

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

A primeira fase gera as equações, e é sempre a que consome mais tempo (entretanto, pode usualmente ser feita em paralelo, visto que exige pequena necessidade de comunicação; assim uma enorme potência de computação pode reduzir o tempo neste problema). A segunda fase é a solução destas equações e leva muito menos tempo que a primeira, mas isto precisa ser em um único processador, e assim causa algumas dificuldades. Finalmente, depois que as equações são resolvidas, a terceira fase calcula logaritmos individuais usando os dados das duas primeiras fases. Esta fase é sempre consideravelmente mais rápida que as outras; entretanto nos mais recentes e mais rápidos algoritmos, a terceira fase tem se tornado cada vez mais complicada.

A seguir, apresentaremos alguns exemplos.

Exemplo 1. Supondo $p = 53$ temos $n = 52$. Para $a = 12$ e $b = 2$, determinemos $x \equiv log_2 12 \pmod{52}$.

Consideremos os primos $(2, 3, 5, 7, 11, 13, 17, 19)$.

Inicialmente escolhemos inteiros $g_i \in [1, 52]$, $1 \leq i \leq 52$. Calculamos r_i e o fatoramos.

g_i	$r_i \equiv 2^{g_i}$ (mod 53)	fatorado
$g_1 = 20$	$r_1 = 24$	$2^3 \times 3$
$g_2 = 5$	$r_2 = 32$	2^5
$g_3 = 8$	$r_3 = 44$	$2^2 \times 11$
$g_4 = 15$	$r_4 = 14$	2×7
$g_5 = 10$	$r_5 = 17$	17
$g_6 = 12$	$r_6 = 15$	3×5
$g_7 = 9$	$r_7 = 35$	7×5

Aplicando log , obtemos $g_i \equiv \sum_{j=1}^{8} a_{ij} \log_2 p_j \pmod{52}$. Donde

$$
\begin{aligned}
20 &\equiv \log_2 2^3 \times 3 \equiv \log_2 2^3 + \log_2 3 \equiv 3 + \log_2 3 \pmod{52} \\
5 &\equiv \log_2 2^5 \equiv 5 \times \log_2 2 \equiv 5 \pmod{52} \\
8 &\equiv \log_2 2^2 \times 11 \equiv \log_2 2^2 + \log_2 11 \equiv 2 + \log_2 11 \pmod{52} \\
15 &\equiv \log_2 2 \times 7 \equiv \log_2 2 + \log_2 7 \equiv 1 + \log_2 7 \pmod{52} \\
10 &\equiv \log_2 17 \pmod{52} \\
12 &\equiv \log_2 3 \times 5 \equiv \log_2 3 + \log_2 5 \pmod{52} \\
9 &\equiv \log_2 7 \times 5 \equiv \log_2 7 + \log_2 5 \pmod{52}
\end{aligned}
$$

Resolvendo estas equações lineares, obtemos

$$\log_2 2 \equiv 1, \quad \log_2 3 \equiv 17$$

$$\log_2 5 \equiv 47, \quad \log_2 7 \equiv 14$$

$$\log_2 11 \equiv 6, \quad \log_2 17 \equiv 10$$

Passando para o terceiro estágio, escolhemos inteiros e_t e calculamos

$$l_t \equiv 12 \times 2^{e_t} \pmod{53},$$

até obter o inteiro e_t tal que $l_t = \prod_{j=1}^{m} p_j{}^{e_j}$.

e_t	$l_t \equiv 12 \times 2^{e_t}$ (mod 53)	fatorado
$e_1 = 20$	$l_1 = 23$	23
$e_2 = 5$	$l_2 = 13$	13
$e_3 = 8$	$l_3 = 51$	51
$e_4 = 15$	$l_4 = 9$	3^2
$e_5 = 10$	$l_5 = 45$	$3^2 \times 5$
$e_6 = 12$	$l_6 = 21$	3×7
$e_7 = 9$	$l_7 = 49$	7^2

Escolhendo o e_4, que satisfaz as condições acima, temos $12 \times 2^{15} = 3^2$ ou $12 \equiv (\frac{3^2}{2^{15}})$ (mod 52).

Aplicando o logaritmo em ambos os membros da equivalência, obtemos $\log_2 12 \equiv 2\log_2 3 - 15$ (mod 52). Donde $\log_2 12 \equiv 19$ (mod 52).

Exemplo 2. Supondo $p = 53$ temos $n = 52$. Para $a = 45$ e $b = 2$, determinemos $x \equiv log_2 45$ (mod 52).

Consideremos os primos $(2, 3, 5, 7, 11, 13, 17, 19)$.

Podemos ir direto ao terceiro estágio, pois os outros dois estágios são os mesmos do exemplo 1. Escolhemos inteiros e_t e calculamos

$$l_t \equiv 45 \times 2^{e_t} \text{ (mod 53)},$$

até obter o inteiro e_t tal que $l_t = \prod_{j=1}^{m} p_j^{e_t}$.

e_t	$l_t \equiv 45 \times 2^{e_t}$ (mod 53)	fatorado
$e_1 = 20$	$l_1 = 20$	$2^2 \times 5$
$e_2 = 5$	$l_2 = 9$	3^2
$e_3 = 8$	$l_3 = 19$	19
$e_4 = 15$	$l_4 = 47$	47
$e_5 = 10$	$l_5 = 23$	23
$e_6 = 12$	$l_6 = 39$	3×13
$e_7 = 9$	$l_7 = 38$	2×19

Escolhendo o e_2, que satisfaz as condições acima, temos $45 \times 2^5 = 3^2$ ou $45 \equiv (\frac{3^2}{2^5})$ (mod 52). Aplicando o logaritmo em ambos os membros, obtemos $\log_2 45 \equiv 2\log_2 3 - 5$ (mod 52). Donde $\log_2 45 \equiv 29$ (mod 52).

Para analisar o tempo de execução do algoritmo queremos considerar alguns argumentos importantes.

É evidente que quanto maior escolhermos m, maior a chance de que r_i e l_t fatorem como o produto dos m primeiros fatores primos. Por outro lado, quando m aumenta, o trabalho para resolver o sistema de equações no segundo estágio aumenta e o trabalho para fatorar resíduos no primeiro estágio aumenta. Para otimizar o tempo de rodagem do algoritmo, precisamos equilibrar estas limitações.

Se g_i é escolhido de uma distribuição uniforme, então a probabilidade de que r_i e l_t fatorem como o produto dos m primeiros primos é $\psi(p, p_m)/p$, onde $\psi(x, y)$ é definido como o número de inteiros positivos menores ou iguais a x que não possuem nenhum fator primo excedendo y. O comportamento assintótico da função ψ tem sido extensivamente estudado ([4]). Em particular, é conhecido que

$$\psi(x, y) = x \exp\left((-1 + o(1)) \log u\right),$$

onde $u = \log x / \log y$, para u tendendo ao infinito, $y \geq \log x^2$ e $o(1)$ é menor que qualquer constante à medida que u tende ao infinito. Se escolhemos $p_m \approx L(p)^c$,

onde c é uma constante e

$$L(p) = \exp\left(\log p \log\log p\right)^{1/2},$$

então a probabilidade que r_i e l_t tenham todos fatores primos entre os primeiros m primos é $L(p)^{-1/(2c)+o(1)}$. Daí podemos esperar gerar uma relação

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

depois de tentarmos $L(p)^{1/(2c)+o(1)}$ valores de g_i, e gerando $2m$ tais relações podemos ganhar em torno de

$$2mL(p)^{1/(2c)+o(1)} = L(p)^{c+1/(2c)+o(1)}$$

valores de g_i. Se usamos o processo da divisão para fatorar, então para cada g_i faria pelo menos $m + \log p$ divisões para decidir se nos dá uma relação. Assim, exige um tempo total de execução para gerar a relação no primeiro estágio de $L(p)^{2c+1/(2c)+o(1)}$.

Depois de geradas $2m$ relações (ou qualquer quantidade um pouco maior que m), é razoável esperar que o sistema correspondente de $2m$ equações em m incógnitas possa ser resolvido para $\log_b p_j$.

O sistema de equações lineares exige $O(m^3)$ operações módulo $p-1$, e $O(m^2)$ aplicações do algoritmo de Euclides estendido (teorema 3.2.4). Então o tempo total esperado para os estágios um e dois para resolver o $\log_b p_j$ é

$$L(p)^{2c+1/(2c)+o(1)} + L(p)^{3c+o(1)}$$

operações em inteiros de tamanho p. Visto que cada tal operação pode ser feita em $L(p)^{o(1)}$ operações binárias, com escolha de $c = 1/2$, temos um tempo de rodagem para os dois primeiros estágios de $L(p)^{2+o(1)}$ operações binárias.

O terceiro estágio pode ser analisado da mesma maneira, dando um tempo de execução esperado de $L(p)^{c+1/(2c)+o(1)}$, ou $L(p)^{3/2+o(1)}$.

6.3 Algoritmo para Grupos Finitos $GF(2^k)$

Nesta seção apresentamos o método *Index Calculus* para grupos $GF(2^k)$, visto que eles são de grande interesse na criptografia.

Uma extensiva análise assintótica do tempo de execução do algoritmo e casos relatados $GF(p^k)$ com p fixo e $k \to \infty$ foram dadas por M.E. Hellman e J.M. Reyneri ([12]). Algumas melhoras no método do *Index Calculus* como suas aplicações para corpos $GF(2^k)$ foram feitas por I. Blake, R. Fuji-Hara. R. Mullin, e S. Vanstone, que tornou ele muito mais eficiente, embora estas melhoras não afetaram assintoticamente o tempo de execução. D. Coppersmith propôs uma dramática melhora na versão de algoritmos $GF(2^k)$ (e mais geralmente na versão $GF(p^k)$ com p fixo e $k \to \infty$) que é muito mais rápido e até tem diferente comportamento assintótico. Uma série inteira de melhoras no algoritmo básico foram descobertas por D. Coppersmith e A. M. Odlyzko ([30]).

O algoritmo *Index Calculus*, pelo menos na forma apresentado até aqui, é um método probabilístico, no qual a análise do seu tempo de execução depende da hipótese sobre aleatoriedade e independência de vários polinômios, os quais parecem razoáveis, mas no momento não podem ser provados.

Antes de apresentar o algoritmo, é necessário especificar a notação que será usada. Usualmente será considerado o grupo $GF(2^k)$ como o anel(quociente) de polinômios sobre $GF(2)$, de grau menor que k, módulo algum polinômio irredutível $f(x)$ de grau k. No caso de $GF(p)^*$, os elementos do grupo $p_1, p_2, ..., p_m$ eram os m primeiros primos pequenos. No caso de $GF(2^k)$ tomamos $p_1, p_2, ..., p_m$ para ser considerados como polinômios irredutíveis $g(x)$ sobre o conjunto S de elementos escolhidos de $GF(2^k)$. O conjunto S usualmente consiste de todos ou quase todos polinômios (mônicos) irredutíveis sobre $GF(2)$ de grau não maior que algum valor t, onde t é apropriadamente escolhido. A relação entre t e m é facilmente deduzida do fato que o número de polinômios irredutíveis sobre $GF(2)$ de grau menor ou igual

a t é exatamente

$$\sum_{d \leq t} d^{-1} \sum_{f|d} \mu(f) 2^{d/f},$$

que é aproximadamente $2^{t+1}/t$, $(ver[16], ex.4.6.2.4)$, onde $\mu(f)$ é a função Möbius definida pelas regras:

(i) $\mu(1) = 1$

(ii) $\mu(p_1 p_2 ... p_m) = (-1)^m$ se $p_1, p_2, ..., p_m$ são primos distintos

(iii) $\mu(f) = 0$ se f é divisível pela raiz de um primo

Sejam $b = b(x)$, um polinômio de grau $< k$ sobre $GF(2)$, um elemento primitivo de $GF(2^k)$, $a = a(x) \in GF(2^k)$ e $f = f(x)$, um polinômio irredutível de grau k. Queremos encontrar $h \in Z^+$ tal que $a \equiv b^h \ (mod\ f)$, equivalentemente

$$h \equiv \log_b a \ (\mathrm{mod} f).$$

No primeiro estágio do algoritmo escolhemos aleatoriamente inteiros g_i, $1 \leq g_i \leq 2^k - 1$, e calculamos polinômios r_i

$$r_i \equiv b^{g_i} \ (mod\ f),\ grau\ r_i < k, \tag{6.1}$$

e verificamos se r_i se decompõe em fatores irredutíveis de S. Isto pode ser relativamente rápido usando um método rápido de exponenciação e um método rápido para fatoração polinomial. O problema de fatoração polinomial sobre corpos finitos é relativamente fácil; existem algoritmos probabilísticos que fatoram um polinômio sobre $F(2)$ de grau k em um tempo esperado de $O(k^c)$ para alguma constante c, (ver [16],§4.6.2).

Se r_i se decompor em fatores irredutíveis de S, teremos

$$r_i \equiv b^{g_i} \equiv \prod_{j=1}^{m} {p_j}^{a_{ij}} \ (\mathrm{mod}\ f), \tag{6.2}$$

e obteremos a congruência

$$g_i \equiv \sum_{j=1}^{m} a_{ij} \log_b p_j \pmod{2^k - 1}.$$

Depois de obtermos mais de $\mid S \mid$ tais congruências, esperamos que elas determinem o $\log_b p_j$, com $p_j \in S$, módulo $2^k - 1$. Geralmente $2^k - 1$ não é um primo, assim para resolver o sistema acima, requer trabalho separadamente módulo as diferentes potências primas divisores de $2^k - 1$ e usamos o teorema Chinês dos Restos (teorema 3.3.4) para reconstruir os valores de $\log_b p_j$.

Depois que os estágios de preprocessamento são completados, logaritmos podem ser calculados relativamente rápidos.

Então

$$\log_b a \equiv \sum_{j=1}^{m} e_j \log_b p_j - e \pmod{2^k - 1}. \tag{6.3}$$

A seguir, apresentaremos um exemplo.

Exemplo 1. Sejam $b(x) = x^2 + x + 1$, um polinômio de grau 2 sobre $GF(2)$, um elemento primitivo de $GF(2^4)$, $a(x) = x^3 + 1 \in GF(2^4)$ e $f(x) = x^4 + x^3 + 1$ um polinômio irredutível de grau 4. Queremos encontrar $h \in \mathbb{Z}^+$ tal que $h \equiv log_{(x^2+x+1)}(x^3 + 1) \pmod{(x^4 + x^3 + 1)}$, isto é, $(x^2 + x + 1)^h \equiv (x^3 + 1) \pmod{f(x)}$

Consideremos $S = x, x + 1, x^2 + x + 1$.

Inicialmente escolhemos inteiros $g_i \in [1, 15]$. Calculamos polinômios r_i e os fatoramos.

g_i	$r_i \equiv (x^2 + x + 1)^{g_i}$ $\mod(x^4 + x^3 + 1)$	fatorado
$g_1 = 1$	$r_1 = x^2 + x + 1$	$x^2 + x + 1$
$g_2 = 2$	$r_2 = x^3 + x^2$	$x^2 \cdot (x + 1)$
$g_3 = 4$	$r_3 = x^2 + x$	$x \cdot (x + 1)$

Aplicando log, obtemos $g_i \equiv \sum_{j=1}^{3} a_{ij} \log_{(x^2+x+1)} p_j \pmod{15}$, onde p_j são os polinômios irredutíveis do conjunto S. Donde

$$\begin{aligned} 1 &\equiv \log_{(x^2+x+1)} (x^2+x+1) \pmod{15} \\ 2 &\equiv \log_{(x^2+x+1)} x^2(x+1) \equiv 2 \times \log_{(x^2+x+1)} x \equiv \log_{(x^2+x+1)} (x+1) \pmod{15} \\ 4 &\equiv \log_{(x^2+x+1)} x(x+1) \equiv \log_{(x^2+x+1)} x + \log_{(x^2+x+1)} (x+1) \pmod{15} \end{aligned}$$

Resolvendo estas equações lineares, obtemos

$$\begin{aligned} \log_{(x^2+x+1)} (x^2+x+1) &\equiv 1 \\ \log_{(x^2+x+1)} (x) &\equiv 13 \\ \log_{(x^2+x+1)} (x+1) &\equiv 6 \end{aligned}$$

Passando para o terceiro estágio, escolhemos inteiros e_t e calculamos

$$l_t \equiv (x^3+1) \times (x^2+x+1)^{e_t} \pmod{(x^4+x^3+1)}$$

até obter o inteiro e_t tal que $l_t = \prod_{j=1}^{3} {p_j}^{e_j}$.

e_t	$l_t \equiv (x^3+1) \times (x^2+x+1)^{e_t}$ $\mod(x^4+x^3+1)$	fatorado
$e_1 = 1$	$l_1 = x^2+x+1$	x^2+x+1
$e_2 = 2$	$l_2 = x^3$	$x \cdot x \cdot x$
$e_3 = 4$	$l_3 = x^2$	$x \cdot x$

Escolhendo o e_3, que satisfaz as condições impostas pelo algoritmo, temos $(x^3+1) \times (x^2+x+1)^4 = x^2$ ou $(x^3+1) \equiv x^2(x^2+x+1)^{-4} \pmod{15}$.

Aplicando o logaritmo em ambos os membros da equivalência, obtemos $\log_{(x^2+x+1)} (x^3+1) \equiv 2 \log_{(x^2+x+1)} x - 4 \pmod{15}$. Donde

$$\log_{(x^2+x+1)} (x^3+1) = 2 \cdot 13 - 4 = 22 \equiv 7 \pmod{15},$$

portanto $\log_{(x^2+x+1)} (x^3+1) \equiv 7 \pmod{15}$.

Para estimar o tempo de execução do primeiro estágio, precisamos conhecer quantos polinômios sobre $GF(2)$ tem todos os seus fatores irredutíveis de

grau no máximo t. O polinômio r acima comporta-se como polinômio aleatório sobre $GF(2)$ de grau $< k$. Seja $p(n,t)$ a probabilidade que o polinômio sobre $GF(2)$ de grau exatamente n que tem todos seus fatores irredutíveis de grau $\leq t$, isto é, se $N(n,t)$ é o número de polinômios $w(x) \in GF(2)$ tal que grau $w(x) = n$ e

$$w(x) = \prod_i u_i(x)^{c_i},$$

com grau $u_i(x) \leq t$, então

$$p(n,t) = \frac{N(n,t)}{N(n,n)} = \frac{N(n,t)}{2^n}. \tag{6.4}$$

Se S consiste de polinômios irredutíveis de grau $\leq t$, o polinômio reduzido r_i em 6.1 fatorará como em 6.2 com probabilidade aproximadamente $p(k,t)$, e aproximadamente $p(k,t)^{-1}$ destes polinômios podem ser gerados antes do terceiro estágio.

A função $p(k,t)$ pode ser facilmente bem avaliada numericamente e assintoticamente. O apêndice A do Odlyzko ([30]) apresenta a recorrência básica satisfeita por $N(k,t)$ (para o qual $p(k,t)$ segue imediatamente de 6.4, e mostra que quando $k \to \infty$ e $k^{1/100} \leq t \leq k^{99/100}$, que é a série de maior interesse no algoritmo index calculus),

$$p(k,t) = \exp\left((1+o(1))\frac{k}{t}\log_e \frac{t}{k}\right). \tag{6.5}$$

Então, para gerar m relações da forma 6.1, esperamos que o estágio 1 requer investigações de

$$m\exp((1+o(1))\frac{k}{t}\log_e \frac{k}{t})$$

valores de g_i. Visto que cada g_i exige tempo $O(k^c)$ operações para uma constante c, o tempo de execução do estágio 1 é da mesma forma.

O segundo estágio do algoritmo exige $m^{2+0(1)}$ operações módulo $2^k - 1$ se usarmos o método de matrizes esparsas de Wiedemann ([40]), dando um tempo de rodagem total para os dois primeiros estágios de

$$m\exp((1+o(1))\frac{k}{t}\log_e \frac{k}{t}) + O(m^{2+o(1)}k^c).$$

Escolhendo $t \approx (\frac{k \log_e k}{2 \log_e 2})^{1/2}$, temos um tempo de rodagem total de $\exp((c_1 + o(1))(k \log_e k)^(1/2))$, onde $c_1 = (2 \log_e 2)^{1/2}$.

O tempo de rodagem do terceiro estágio pode ser analisado de maneira similar, dando um tempo de rodagem de $\exp((\frac{1}{2c_1} + o(1))(k \log_e k)^{(1/2)})$. Esta é, de fato, a melhor prova rigorosa do tempo de rodagem de um algoritmo em $GF(2^k)$(ver [35]).

Está claro que o tempo de execução do terceiro estágio pode ser diminuído com o aumento de t. Atividade que, entretanto, aumenta a necessidade de armazenagem e o tempo de execução dos estágios de pré-processamento do algoritmo.

Até aqui, a descrição do método index calculus como é aplicado para $GF(2^k)^*$ tem exatamente correspondente representação para $GF(p)^*$. No caso de $GF(2^k)^*$ há, entretanto, alguns melhoramentos muito significativos que podem ser feitos tanto na performance de implementação como no tempo de execução estimado assintótico heurístico.

6.4 Melhoramento do index calculus feito por Coppersmith

Uma variação na versão básica do método index calculus foi feita por D. Coppersmith ([5]). O algoritmo de Coppersmith conta com fatoração de polinômios de grau aproximadamente $k^{2/3}$, e é interessante por razões práticas e teóricas. Os argumentos apresentados para o tempo de execução assintótico são baseadas em hipóteses heurísticas e ainda devem estar algumas questões abertas para provar que a análise é correta. O argumento heurístico sugere que teremos um tempo de rodagem de $\exp(ck^{1/3} \log^{2/3} k)$ para alguma constante c.

Ao contrário da versão básica, no entanto, a variação de Coppersmith não se aplica para grupos $GF(p)^*$ com p primo. O algoritmo Coppersmith conta com várias suposições não provadas. Mesmo que estas hipóteses sejam baseadas em

razões heurísticas e evidências empíricas, parece que não há nenhuma razão para duvidar da validade do algoritmo.

No algoritmo de Coppersmith, assumimos que os polinômios f usados para definir o grupo tem a forma $x^k + f_1(x)$, onde o grau de f_1 é de tamanho $\log k$. Isto não é uma restrição severa, visto que é relativamente fácil transferir um logaritmo discreto de uma representação de um grupo para outro. Além disso, tem um argumento heurístico para sugerir que tal polinômio irredutível pode existir (embora isto permanece não provado).

Para descrever o primeiro estágio do método Coppersmith, necessitamos alguma notação. Seja r um inteiro, e definimos $h = \lfloor k2^{-r} \rfloor + 1$. Para gerar uma relação, primeiro escolhemos, aleatoriamente, polinômios relativamente primos $A(x)$ e $B(x)$ de grau $\leq t$. Então fixamos $\omega_1(x) = A(x)x^h + B(x)$ e

$$\omega_2(x) = \omega_1(x)^{2^r} \bmod f(x).$$

Seguindo com nossa escolha especial de $f(x)$ que podemos fazer

$$\omega_2(x) = A(x^{2^r})x^{h2^r-k} f_1(x) + B(x^{2^r}),$$

para que $grau(\omega_2) \leq 2^r t + h2^r - k + grau(f_1)$. Se escolhemos t e 2^r serem da ordem de $k^{1/3}$, então os graus de ω_1 e ω_2 serão de ordem $k^{2/3}$. Se eles se comportam como polinômios randônicos daquele grau (como esperamos), então tem-se uma boa chance que todos seus fatores irredutíveis terão grau não excedendo a t. Se assim for, então de $\omega_2(x) = \omega_1(x)^{2^r} \bmod f(x)$ obtemos uma equação linear envolvendo os logaritmos de polinômios de grau $\leq t$.

Uma análise detalhada do método mostra que os parâmetros podem ser escolhidos para dar um tempo de execução do primeiro estágio de

$$\exp((c_2 + o(1))k^{1/3} \log^{2/3} k),$$

onde $c_2 < 1.351$.

O segundo estágio do método Coppersmith é como o do método index calculus básico, requerendo a solução de um sistema de congruências lineares. O

terceiro estágio é um tanto mais complicado que no método index calculus básico. Omitimos os detalhes de como ele trabalha, mas a idéia básica é para calcular o logaritmo de um polinômio individual calculando os logaritmos de uma seqüência de polinômios com grau diminuído. O tempo de rodagem do terceiro estágio é da forma

$$\exp((c_3 + o(1))k^{1/3}\log^{2/3} k),$$

onde $c_3 < 1.098$, assim exige menos tempo que os dois primeiros estágios.

Coppersmith implementou seu algoritmo para testar o caso $GF(2^{127})$, e tem sido bem sucedido em construir uma base de dados de logaritmos que lhe tornará possível, para até certo ponto, facilmente calcular logaritmos individuais. Odlyzko em [30] realizou uma análise extensiva que sugere que o algoritmo Coppersmith é viável para calcular logaritmos discretos em $GF(2^k)$ para $k < 520$ com um super computador, e talvez para $k < 1280$ usando hardware com objetivo específico.

7 CONSIDERAÇÕES FINAIS

Os algoritmos para o cálculo de logaritmos discretos surgiram no final da década de setenta. A partir daí tiveram vários melhoramentos, alcançando com isso possibilidades práticas de cálculo efetivo de logaritmos discretos.

Estes cálculos se devem tanto ao desenvolvimento do conhecimento teórico, como também à evolução da tecnologia dos computadores. O aumento na quantidade e velocidade dos computadores que estão sendo construídos é uma ameaça para os sistemas de criptografia, pois permitem uma explosão na investigação sobre a fatoração de inteiros e logaritmos discretos, aproveitando o potencial fornecido pelo tempo ocioso de computadores na *internet*, que facilmente pode ser aproveitada.

A tecnologia está evoluindo muito rápido e com isso surgem os desafios visando quebrar as questões tidas de solução muito difícil. Os computadores evoluem muito rapidamente, as operações que há pouco tempo eram consideradas impossíveis, são realizadas hoje, cada vez mais rapidamente e com mais precisão.

O primeiro sistema de criptografia de chave pública divulgado é ainda amplamente usado. Trata-se do algoritmo de troca de chave Diffie-Hellman (o qual abordamos no capítulo 2), baseado na hipótese de que o problema do logaritmo discreto é difícil. De fato, por basear-se na exponenciação discreta, um algoritmo rápido para resolver logaritmos discretos pode destruir definitivamente a sua utilidade. Isto tem estimulado uma explosão de pesquisa na complexidade de logaritmos discretos.

Um fato infeliz para os sistemas de criptografia é que logaritmos discretos e fatoração de inteiros são tão próximos que muitos algoritmos desenvolvidos para um problema podem ser modificados para usar no outro, que é o caso do método de fatoração Rho de Pollard e o algoritmo para resolver logaritmos discretos de Pollard, ambos apresentados no capítulo 4. Para segurança dos sistemas de

criptografia, seria melhor ter muito mais diversidade. Em muitos casos, entretanto, onde algoritmos de mesma funcionalidade existem, falar num corpo finito de inteiros módulo um primo p, e outro usando um inteiro composto n de mesmo tamanho, calcular o logaritmo módulo p parece ser um pouco mais difícil que fatorar um inteiro n. Por isso, sistemas de criptografia baseados em logaritmos discretos são preferíveis à sistemas baseados em fatoração como RSA, por exemplo.

Será que o problema de logaritmos discretos é realmente difícil? Para corpos primos $GF(p)^*$, onde p não tem nenhuma propriedade especial, Weber ([38]), em 1998, realizou um cálculo com um primo de 85 dígitos decimais. No mesmo ano A. Joux e R. Lercier, realizaram um cálculo para um primo de 90 dígitos decimais com o método de inteiros gaussianos. Recentemente, em janeiro de 2001, R. Lercier ([24]) anunciou um novo recorde para o problema do logaritmo discreto geral. Ele foi capaz de calculá-lo módulo um primo de 110 dígitos. Isto foi feito em 3 semanas num único computador $525MHz$ Digital Alpha Server 8400 com 4 processadores. Este recorde foi obtido usando um corpo algébrico descrito do ponto de vista teórico de Schirokauer ([36]). Logo após, em abril de 2001 R. Lercier ([24]), conseguiu calcular o problema de logaritmo discreto geral módulo um primo de 120 dígitos. Isto foi feito em 10 semanas num único computador $525MHz$ Digital Alpha Server 8400 com 4 processadores.

Para resolver logaritmos discretos em $GF(2^k)$, o último recorde foi anunciado em setembro de 2001, por R. Lercier. Ele foi capaz de calcular logaritmos discretos em $GF(2^{521})$. Isto foi feito em um mês num único computador 525MHz Digital Alpha Server 8400 com 4 processadores. Os recordes anteriores a este eram em $GF(2^{401})$ no ano de 1992 ([11]), usando um algoritmo devido à D. Coppersmith. Uma tentativa para $GF(2^{503})$ é também descrita em ([11]), mas parece que os passos da álgebra linear foram difíceis para serem completados. Outra tentativa parece ser um cálculo em $GF(2^{607})$ com o algoritmo de D. Coppersmith, mas não temos maiores detalhes sobre isto.

Os métodos utilizados para obter os últimos recordes citados acima na resolução do problema do logaritmo discreto em corpos finitos são todos descendentes do método Index Calculus. Os dois mais novos membros da família do Index Calculus são chamados crivo de um corpo algébrico racional e crivo de corpo da função. O primeiro é uma adaptação do algoritmo de fatoração chamado crivo de um corpo algébrico para o problema de calcular logaritmos discretos em um corpo primo. O cálculo neste método, como na fatoração, realiza-se em uma extensão finita de Q. O segundo é mais apropriado para calcular logaritmos de pequena característica. Neste caso, a estrutura do corpo algébrico é transportada para uma extensão finita de $F_p(X)$.

Mesmo com os substanciais avanços em algoritmos para logaritmos discretos nas últimas décadas, em geral os logaritmos discretos ainda parecem ser difíceis, especialmente para alguns grupos, como o grupo de pontos racionais em curvas elípticas. A atração das curvas elípticas é que em geral nenhum ataque mais eficiente que o de Pollard e Shanks (ambos abordados no capítulo 4) é conhecido, visto que o algoritmo index calculus (abordado no capítulo 6) não funciona para curvas elípticas. Assim o tamanho das chaves pode ser muito menor que para o RSA ou para sistemas de logaritmos discretos em grupos finitos para um mesmo nível de segurança. A falta de ataques subexponenciais em criptosistemas de pontos racionais em curvas elípticas oferece potencial redução no tempo de execução, na armazenagem, no tamanho de mensagens e na potência elétrica.

Parece não haver registros na literatura que comprovem a existência de algoritmos para curvas elípticas que sejam mais rápidos que os de Shanks e de Pollard. Por outro lado, também não encontramos registros que afirmem a impossibilidade de algoritmos mais rápidos que estes.

A falta de progresso nesta área tem fornecido, de certo modo, uma sensação de conforto para sistemas de criptografia o que os leva a escolher um parâmetro de segurança próximo do limite do que é viável. Mas não é somente o progresso do método de Shanks e Pollard que pode ser uma ameaça. A segurança

do DSA (Algoritmo oficial U.S. Digital Signature) é baseado na hipótese que os únicos ataques são os que atuam num subgrupo multiplicativo de ordem q sem explorar qualquer propriedade especial deste grupo, ou o index calculus que atua em grupos completos módulo p. Também parece não haver registro de que uma relação algébrica não possa ser explorada para encontrar um algoritmo melhor.

A maior preocupação que ameaça os sistemas de criptografia de logaritmos discretos a longo prazo é o desenvolvimento dos computadores quânticos. Shor [30] mostrou que se tais máquinas forem construídas, os logaritmos discretos poderão ser calculados em tempo de ordem polinomial. Este resultado estimulou uma explosão na pesquisa em computadores quânticos. Enquanto debate-se a viabilidade destes computadores, até o momento, nenhum obstáculo fundamental para sua construção foi encontrado. Um fator confortante, é que todos especialistas admitem que mesmo se os computadores quânticos forem construídos, isto vai demorar muitos anos e assim poderia ser antecipado um sinal sobre a necessidade de desenvolver e dispor sistemas alternativos.

Outro fator que oferece generosa margem de segurança é que, apesar de uma nova descoberta matemática poderia transformar-se em uma grande ameaça para os logaritmos discretos, o número de pessoas que trabalham seriamente em logaritmos discretos ainda é pequeno.

O objetivo do nosso trabalho foi conhecer e analisar o problema do logaritmo discreto, estudando algoritmos para resolvê-lo. Conseguimos fazer algumas comparações, levando em conta o tempo de execução e o espaço necessário para a armazenagem de dados, podendo assim analisar vantagens e desvantagens de cada método.

Considerando os algoritmos estudados para resolver o problema do logaritmo discreto, chegou-se às seguintes conclusões básicas:

- O algoritmo Shanks e o algoritmo Pollard não são os melhores, mas são os ataques mais eficientes conhecidos até o momento para os sistemas

de criptografia de curvas elípticas. O algoritmo Shanks tem uma vantagem muito interessante em relação aos outros algoritmos estudados, pois este algoritmo não precisa conhecer a ordem do grupo. E mais, encontramos na literatura que o próprio algoritmo Shanks pode ser usado para calcular a ordem de um grupo. A vantagem do algoritmo Pollard, que tem praticamente o mesmo tempo de de execução do Shanks, é que não precisa de espaço para armazenar dados, pois não precisa armazenar dados.

- O algoritmo Silver-Pohlig-Helmann é mais rápido que o algoritmo Shanks e o algoritmo Pollard. Mas precisamos da ordem n do grupo considerado, pois ele calcula logaritmo discreto módulo os fatores primos de n com suas devidas potências. A sua eficiência depende do tamanho destes fatores primos.

- O algoritmo *Index Calculus* é o mais usado, no entanto ele não funciona para o grupo de pontos racionais em curvas elípticas. Ele está em constante melhoramento tanto no processo para produzir elementos suaves como para resolver sistemas lineares. Entretanto, mesmo com os progressos na eficiência assintótica, não resultou em grande aumento na ordem do problema a ser resolvido. Todas as novas técnicas que surgiram ultimamente são modificações do método Index Calculus, ou algum melhoramento feito neste.

No decorrer deste trabalho notamos o quanto é amplo este assunto, quantos avanços estão surgindo, e o quanto tudo isto é delicado e complicado, tanto que muitas questões ainda estão em aberto, muitas hipóteses heurísticas são feitas, principalmente na questão da complexidade.

Não abordamos aqui, sistemas de criptografia baseados na aritmética dos grupos de pontos racionais em curvas elípticas, um tópico que tem recebido intenso interesse de pesquisadores nos últimos anos (ver, por exemplo [31]). Estes

sistemas parecem oferecer vantagens no sentido de que o tamanho das chaves utilizadas pode ser menor para a mesma segurança. Também não entramos em detalhes nas alterações feitas no método Index Calculus, nosso interesse foi estudar o método na sua versão básica.

BIBLIOGRAFIA

[1] Adleman, L.M. "*A subexponencial algoritm for the discret logaritm problem with applications to cryptography*", Proc. of the 20th Annual IEEE Sympoium on Fundadions of Computer Science, 1979, 55-60.

[2] Brassard, G. and Bratley P. "*Fundamentals of Algorithmics*", Prentice Hall, 1996.

[3] Brillhart, J., D.H.Lehmer,J.L.Selfridge,B.Tieckerman, e S.S.Wagstagg, Jr, "*Factorizations of* $b^n + 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ *up to High Powers*", Contemporary Mathematics, vol. 22, American Mathematical Society, Providence, 1988.

[4] Cafield, E. R., P. Erdös and C. Pomerance. "*On a problem of Oppenheim Concerning Factorization Numerorunn*", J. Number Theory 17, 1983, 1-28.

[5] Coppersmith, D. "*Fast Evaluation of discrete logarithms in fields of characteristic two*", IEEE Trans. Infornn. Theory 30, 1984, 587-594.

[6] Coutinho, S.C. "*Números inteiros e criptografia RSA*", Rio de Janeiro, IMPA/SBM, 1997.

[7] Diffie, W. and M. E. Hellman, "*New directions in cryptography*", IEEE Trans. Inform. Theory, IT-22, 1976, 644-654.

[8] ElGamal, T. "*A public Key cryptosystem and a signature scheme based on discrete logarithms*", IEEE Trans. Inform. Theory 31, 1985, 469-472.

[9] ElGamal, T. "*A subesponencial-time algorithm for computing discrete logarithm over* $GF(p^2)$", IEEE Trans. Inform. Theory 31, 1985, 473-481.

[10] Gordon, D.M. "*Discrete logarithms in* $GF(p)$ *using the number field sieve*", SIAMI. Discr. Math. 6, 1993, 124-138.

[11] Gordon, D.M. and K.S. McCurley, "*Massively Parallel Computation of Discrete Logarithms*". Advances in Cryptology, CRYPTO'92, Lectures Notes in Computer Science, volume 740, 1992, 312-323.

[12] Hellmann, M.E. and J.M.Reyneri, "*Fast computation of discrete logarithms in $GF(q)$*" , Adv. in Cryptology, Proc. of Crypto'82, D.Chaum, R.Rivest, and A.Sherman, eds,Plenum Press, 1983, 3-13.

[13] Joux, A. and R. Lercier, "*Improvements to the general Number Field Sieve For discrete logarithms in prime fields*", a ser publicado at Math. of Comp., 2000. Preprint available at http://www.medicis.polytechnique.fr/ lercier.

[14] Kenneth, H. Rosen, "*Elementary Number Theory and its Applications*". 3 ed., Addison-Wesley Publishing Company, New Wesley, 1993.

[15] Knuth, D.E. "*The art of computer programming*", Vol.1: "*Fundamental Algorithms*", Addison-Wesley, Reading, M.A, 1973

[16] Knuth, D.E. "*The art of computer programming*", Vol.2:" *Seminumerical algorithms*", second edition, Addison- Wesley, Reading, M.A, 1981.

[17] Knuth, D.E. "*The art of computer programming*", Vol.3:"*Sorting and searching*", Addison- Wesley, Reading, M.A, 1981.

[18] Koblitz, N. "*A course in number theory and cryptography*". 2 ed. New York, Springer-Verlag, 1994.

[19] Konheim, A.G. "*Cryptography: A Primer*", Wiley, 1981.

[20] Kraitchik, M. "*Recherches sur la Théorie des numbres*", Gauthier-Villars, Paris, 1924.

[21] Kraitchik, M. "*Théorie des nombres*" , Vol 1 Gauthier-Villars, Paris, 1992.

[22] LaMacchia, B.A. and A.M.Odlyzko, "*Computation of discrete logarithms in prime fields*", Designs, Codes, and Cryptography, 1991, 46-62. Available at http://www.research.att.com/ amo.

[23] Lenstra, A. K. and H.W.Lenstra, Jr. eds. "*The Development of the Number Field Sieve*", Lecture Notes in Mathematics 1554, Springer, 1993.

[24] Lercier, R. "*Discrete logarithms in GF(p)*" , April 17 2001. Announce on the NMBRTHRY Mailing List. Available at http://www.medicis. polytechnique.fr/ lercier.

[25] McCurley, K.S. "*Cryptographic key distribution and computation in class groups*", Number Theory and applications (Proc. of the NATO Advanced Study Institute on Number Theory and Applications, Banff, 1988), Richard A. Mollin, ed., Kluwer, Boston, 1989, 459-479.

[26] McCurley, K.S. "*The discret logarithm problem*", in Cryptology and Computational Number Theory, C.Pomerance, ed., Proc. Symp. Appl. Math. 42, Amer. Math. Soc., 1990, 49-74.

[27] Menezes, A., S. Vanstone and T. Okamoto. "*Reducing elliptice curve logarithms to logarithms in a finite field*", IEEE Transactions on Information Theory, 39, 1993, 1639-1646.

[28] Merkle, R. "*Secrecy, authentication, and public key systems*", Ph.D.dissertation, Electrical Engineering Department, Stanford University, 1979.

[29] Odlysko, A.M. "*Discrete logarithms and smooth polynomials*", in Finite Fields: Theory, Applications and Algorithms, G. L. Mullen and P. Shiue, eds., Contemporary Math. ♯168, Amer. Math. Soc., 1994, 269-278. Available at http://www.research.att.com/ amo.

[30] Odlysko, A.M. "*Discrete logarithms in finite fields and their cryptographic significance*", in Advances in Cryptology: Proceedings of Eurocrypt'84,T.Beth,N.Cot,I. Ingemarsson, eds., Lecture Notes in Computer Science 209 Springer-Verlag, 1985, 224-314. Available at http://www.research.att.com/ amo.

[31] Odlysko, A.M. "*Discrete logarithms: The past and the future*". In Designs, Codes and Cryptography, 2000, 129-145.

[32] Pollard, J.M. "*A Monte Carlo method for factorization*", Bit, V. 15, 1973, 331-335.

[33] Pollard, J.M. "*Monte Carlo mathods for index computations mod p*", Math. Comp. 32, 1978, 918-924.

[34] Pollard, J.M. "*Kangaroos, Monopoly and discrete logarithm*", J. Cryptology, 2000, 437-447.

[35] Pomerance, Carl. "*Fast rigorous factorization and discrete logarithm algorithms*", in Discrete algorithms and Complexity; Proc., of the Japan-U.S. Joint Seminar, June 4, 1986, Kyotto, Japan, Academic Press, Orlando, 1987, 119-143.

[36] Schirokauer, O. "*Discrete Logarithms and local units*". Phil. Trans. R. Soc Lond. A 345, 1993, 409-423.

[37] Stephen C. Pohlig and Martin E. Hellman," *An improved algorithm for computing logarithms over $GF(p)$ and its Cryptographic significance*", IEEE Trans. on Inform. Theory 24, 1978, 106-110.

[38] Weber, D. and T.F.Denny, "*The solution of McCurley's discrete log challenge*", in Advances in Cryptology-CRYPTO '98, H. Krawezyk, ed.,Lecture Notes in Computer Science 1462, Springer, 1998, 458-471.

[39] Western, A.E. and J.C.P.Miller, "*Tables of indices e primitives roots*", Royal Society Mathematical Tables , Vol 9, Cambridge University Press, 1968.

[40] Wiedemann, Douglas H. "*Solving sparse linear equations over finite fields*", IEEE Trans. Infornn. Theory 32, 1986, 54-62.

ÍNDICE REMISSIVO