


Este exemplar corresponde à redação final da
Tese/Dissertação devidamente corrigida e defendida
por: Rogério Albertoni Miranda
e aprovada pela Banca Examinadora.
Campinas, 05 de Setembro de 2002

COORDENADOR DE PÓS-GRADUAÇÃO
CPG-IC

Criptossistemas Baseados em Curvas Elípticas

Rogério Albertoni Miranda

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Criptossistemas Baseados em Curvas Elípticas

Rogério Albertoni Miranda¹

Março 2002

Banca Examinadora:

- Prof. Dr. Ricardo Dahab
Instituto de Computação, Unicamp (Orientador)
- Prof. Dr. Marcos Vinícius S. Poggi de Aragão
Departamento de Informática, PUC-Rio
- Prof. Dr. Arnaldo Vieira Moura
Instituto de Computação, Unicamp
- Prof. Dr. Cid Carvalho de Souza
Instituto de Computação, Unicamp (Suplente)

¹O autor é Bacharel em Ciência da Computação pela Universidade Federal do Mato Grosso do Sul.

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 12 de abril de 2002, pela Banca Examinadora composta pelos Professores Doutores:



Prof. Dr. Marcus V. S. Poggi de Aragão
PUC - RJ



Prof. Dr. Arnaldo V. Moura
IC - UNICAMP



Prof. Dr. Ricardo Dahab
IC - UNICAMP

200250067

UNIDADE Be
Nº CHAMADA TI/UNICAMP
M672c
V _____ EX _____
TOMBO BCI 51312
PROC 16.837/02
C _____ DX _____
PREÇO R\$ 11,00
DATA 24/10/02
Nº CPD _____

CM00175025-7

BIBID 265177

**FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DO IMECC DA UNICAMP**

Miranda, Rogério Albertoni

M672c Criptossistemas baseados em curvas elípticas / Rogério Albertoni
Miranda -- Campinas, [S.P. :s.n.], 2002.

Orientador : Ricardo Dahab

Dissertação (mestrado) - Universidade Estadual de Campinas,
Instituto de Computação.

1. Criptografia de dados (Computação). 2. Teoria da computação.
3. Teoria dos números. I. Dahab, Ricardo. II. Universidade Estadual de
Campinas. Instituto de Computação. III. Título.

Criptossistemas Baseados em Curvas Elípticas

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Rogério Albertoni Miranda e aprovada pela Banca Examinadora.

Campinas, 11 de agosto de 2002.



Prof. Dr. Ricardo Dahab
Instituto de Computação, Unicamp
(Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Agradecimentos

Quero aqui expressar os meus mais sinceros agradecimentos a todos aqueles que, de uma forma ou de outra, contribuíram para que este trabalho tenha sido efetivado. Dedico especial agradecimento:

Ao Prof. Doutor Ricardo Dahab, não apenas pelas críticas e sugestões relevantes feitas durante todo o processo de desenvolvimento do projeto e confecção desta monografia mas principalmente pela amizade e ajuda extra-mestrado, nos diversos momentos que precisei ao longo deste período.

Ao Prof. Doutor Júlio Hernández López, por compartilhar parte de seu conhecimento e experiência no assunto tratado nesta dissertação, os quais foram indispensáveis no direcionamento de meu trabalho.

Aos colegas do Mestrado, pela excelente relação pessoal que criamos e que espero não se perca. Em especial a minha grande amiga Hana Karina Salles Rubinsztejn, pelas dicas relevantes durante a confecção desta monografia.

Aos meus colegas de república José Augusto da Costa Jales Neto e Danival Taffarel Calegari, pela amizade e companheirismo a mim dispensados e pelas ótimas horas de relaxamento que compartilhamos.

Aos meus pais Ronil Miranda dos Santos e Maria Nemizia Albertoni Miranda, pelo estímulo e apoio incondicional desde a primeira hora; pela paciência e grande amizade com que sempre me ouviram, e sensatez com que sempre me ajudaram.

Ao CNPq, pelo seu apoio financeiro, sem o qual não poderia ter me estabelecido na cidade de Campinas já que minha condição humilde não me permitia.

Em resumo, agradeço a Deus, por me prover todas estas contribuições, sem as quais este trabalho não poderia ser realizado.

Resumo

Sistemas de chave pública tem sua segurança depositada sobre um problema matemático unanimemente considerado difícil pela comunidade científica. De modo geral, a dificuldade do problema cresce exponencialmente no número de *bits* das chaves utilizada por tal sistema.

Sistemas criptográficos baseados em curvas elípticas, propostos em 1985 a partir de idéias matemáticas conhecidas desde o século XIX, são sistemas que permitem que níveis desejáveis de segurança sejam obtidos com valores muito pequenos de chave, quando comparados com outros sistemas de chave pública como RSA e ElGamal. Por isso, eles tem despertado um progressivo interesse, especialmente para aplicações com sérias restrições de recurso, tal como é o caso de *smart cards*, *handhelds*, telefones celulares, aplicações *web*, etc.

Nesta dissertação, apresentamos e discutimos cada um dos elementos que compõem um sistema criptográfico baseado em curvas elípticas, dando ênfase na descrição detalhada dos critérios de seleção de um conjunto de algoritmos eficientes para as operações aritméticas envolvidas, a partir de diversas contribuições na literatura relacionada. Também descrevemos cada um dos passos de nossa implementação de um ECC completo e analisamos os resultados finais obtidos a partir desta implementação.

Sumário

Resumo	ii
1 Introdução	1
1.1 Objetivos desta Dissertação	3
1.2 Organização do Documento	4
2 Conceitos Básicos de Álgebra e Teoria dos Números	6
2.1 Grupos	6
2.1.1 Subgrupos	7
2.1.2 Grupos Finitos	7
2.2 Corpos Finitos	9
2.2.1 Bases	10
2.2.2 Corpos Finitos de Interesse à Criptografia	12
3 Introdução à Teoria das Curvas Elípticas	20
3.1 Curvas sobre Corpos de Característica $\neq 2$ e $\neq 3$	21
3.2 Curvas sobre Corpos de Característica 2	22
3.3 Grupo dos Pontos sobre uma Curva Elíptica	23
4 Sistemas Criptográficos	28
4.1 Sistemas Simétricos	29
4.2 Sistemas Assimétricos	30
4.2.1 Diffie-Hellman	31
4.2.2 RSA	32

4.2.3	ElGamal	33
4.2.4	Assinaturas Digitais	35
4.3	Hashing	36
4.4	Sistemas Híbridos	37
4.5	Segurança dos Algoritmos Criptográficos	37
5	Implementação de ECCs	39
5.1	Por que Curvas Elípticas?	40
5.2	Escolha do Corpo Finito	42
5.2.1	Corpos Primos	43
5.2.2	Corpos Binários	44
5.2.3	Corpos de Extensão Ótima (OEF)	45
5.2.4	Nossas Escolhas	45
6	Implementando a Aritmética do Corpo Finito	51
6.1	Aritmética em $GF(2^m)$	51
6.1.1	Representação	51
6.1.2	Redução	55
6.1.3	Quadrado	57
6.1.4	Inversão	58
7	Algoritmos para Multiplicação Escalar	64
7.1	Algoritmo Binário	65
7.2	Método <i>t-ário</i>	66
7.3	Método da Janela Deslizante	70
7.4	Representação por Dígito Sinalizado	72
8	Resultados	75
8.1	Objetivo	75
8.2	Componentes de um ECC	76
8.3	Aritmética no Corpo Finito	76
8.4	Aritmética na Curva Elíptica	79

9 Conclusões	82
9.1 Principais Contribuições de Nosso Trabalho	83
Bibliografia	84

Lista de Tabelas

3.1	Fórmulas para adição e duplicação elíptica de acordo com a característica do corpo base.	27
6.1	Tabela Aritmética para Soma em $GF(2)$	52
8.1	Tempos obtidos (em microssegundos) para aritmética sobre diversos corpos binários, em máquina Pentium III 450MHz.	78
8.2	Tempos obtidos (em milissegundos) para aritmética em curvas elípticas definidas sobre diversos corpos binários, usando máquina Pentium III 450MHz.	80

Lista de Figuras

3.1	Soma elíptica de dois pontos distintos P e Q	24
3.2	Soma elíptica de pontos iguais.	24
5.1	Tipos de corpos recomendados para uso em ECCs.	43

Capítulo 1

Introdução

Frequentemente, informações são tão preciosas quanto bens materiais tais como jóias, dinheiro, ouro, etc. Por isso, não é de se surpreender que necessitem de tanta segurança quanto qualquer outro bem material de alto valor agregado. No entanto, informações não são como jóias que podem ser mantidas por muito tempo dentro de um cofre e transportada por raras vezes em um carro forte. Muitas vezes, para que seja valiosa, a informação deve trafegar o mais rápido possível da sua fonte ao seu destino. Nestes casos, rapidez é um requisito tão indispensável quanto segurança.

Nos tempos atuais, o meio mais rápido de se transportar informações é através de redes de transmissão de dados. Nestas redes, dados trafegam por pequenas ou enormes distâncias, estando expostos ao ataque de qualquer indivíduo disposto a tirar proveito de informações que possivelmente estes dados representam.

Para dificultar a ação desses intrusos, é necessário que a associação entre o dado transportado e a informação que ele representa não seja direta ou facilmente obtida por alguém alheio ao sistema. É neste contexto que surge uma ciência conhecida a séculos, desde antes das redes de comunicação, desde os tempos em que o meio mais rápido de transportar informações era através de um pombo correio ou um homem correndo a pé - a Criptografia.

Formalmente, Criptografia é a ciência que se preocupa basicamente em transformar informações de modo a ocultar seu conteúdo semântico. Adicionalmente, ela também

estuda métodos que permitam estabelecer a autenticidade destas informações, detectar possíveis adulterações, bem como associá-la à sua origem de maneira inequívoca (não-repúdio). Em resumo, Criptografia estuda meios de garantir que a informação conduzida através de um meio não confiável seja preservada e permaneça desconhecida daqueles aos quais ela não está destinada.

Para realizar seu trabalho, a Criptografia se utiliza de algoritmos que transformam *mensagens*, que podem ser vistas como a informação na sua forma direta de entendimento, em *criptogramas*, que correspondem à informação na sua forma mascarada, onde não há uma relação imediata ou facilmente obtida com a mensagem original. São os criptogramas que são conduzidos através do meio não confiável para posteriormente serem convertidos em mensagens pelo destinatário, o qual deve ser detentor de alguma informação que o distingue dos outros possíveis observadores do meio. Esta informação extra é conhecida como *chave*.

Os algoritmos criptográficos podem ser divididos em duas grandes categorias, os algoritmos simétricos, ou de chave secreta e os algoritmos assimétricos ou de chave pública. Esta última categoria permite outras funcionalidades além de sigilo, tais como autenticidade, integridade e não repúdio de dados.

O maior problema com os algoritmos de chave pública é que eles são computacionalmente mais custosos que algoritmos de chave secreta, o que pode ser um fator limitante em ambientes com restrições de recursos, tal como é o caso de sistemas *wireless*, *smart-cards*, telefones celulares, *paggers*, *handelds*, etc. O mundo viu aflorar diversas modalidades destes dispositivos, a partir da década de 80. Muitos deles possuem fortes requisições de segurança porém limitações de recurso tão sérias que tornam simplesmente inviável a utilização da maioria dos algoritmos de chave pública conhecidos.

Neste cenário, cresceu o interesse por um sistema proposto em meados da década anterior (1985), que parecia exibir uma melhor relação entre esforço computacional e segurança do que o então mais conhecido sistema de chave pública, o RSA. Este sistema novo, apresentado pela primeira vez e de forma independente por Neal Koblitz[Kob87] e Victor Miller[Mil86], é baseado em uma estrutura matemática estudada há mais de 180 anos, chamada *curva elíptica*.

De modo informal, o nome curva elíptica é dado a uma família de curvas definidas por uma equação do tipo:

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_5.$$

Pensemos, por enquanto, que a_1, a_2, a_3, a_4 e a_5 são números reais.

Ao conjunto de pontos destas curvas, associa-se uma operação, comumente chamada de soma elíptica; pontos e soma elíptica constituem um grupo algébrico.

Curvas elípticas têm se tornado muito atraentes sob o ponto de vista criptográfico porque podem ser aplicadas a diversos problemas desta área, dentre eles a fatoração de números inteiros, teste de primalidade e construção de sistemas criptográficos. Esta dissertação aborda esta última categoria.

Algoritmos de chave pública, de modo geral, têm sua segurança baseada em algum problema matemático considerado difícil de se resolver. Com sistemas criptográficos baseados em curvas elípticas, normalmente abreviados por ECC, não é diferente: sua segurança depende de um problema conhecido como *problema do logaritmo discreto no grupo de pontos de uma curva elíptica* (ECDLP). Este problema é atualmente considerado de dificuldade maior que a fatoração de números inteiros (IFP), sobre o qual o RSA está baseado.

A dificuldade maior em se resolver o problema do logaritmo discreto no grupo elíptico permite que ECCs possam operar com valores significativamente menores que outros sistemas de chave pública, tais como o RSA. Esta característica motivou e tem motivado um profundo estudo sobre sistemas desta natureza, tanto sob o ponto de vista de segurança quanto em eficiência. Esta dissertação trata o aspecto de eficiência de sistemas ECC.

1.1 Objetivos desta Dissertação

O objetivo desta dissertação é reunir e analisar um conjunto de informações relacionadas a sistemas criptográficos baseados em curvas elípticas, principalmente sob o ponto de vista de implementações eficientes. Para cumprir tal objetivo, foram realizadas e apresentadas nesta dissertação as seguintes tarefas:

- Descrição dos elementos presentes na implementação de um ECC.

- Descrição e comparação de algoritmos eficientes para aritmética nos corpos finitos do tipo $GF(2^m)$.
- Descrição e comparação de vários algoritmos para multiplicação escalar.
- Implementação de um ECC completo, apresentando todos os passos necessários para sua confecção.

1.2 Organização do Documento

Este documento está organizado da seguinte forma:

- O Capítulo 2 introduz alguns conceitos matemáticos relacionados a grupos e corpos finitos, assim como o conceito de base e sua utilização para representação dos elementos de um corpo finito, principalmente através de bases polinomiais e bases normais.
- O Capítulo 3 fornece uma introdução a curvas elípticas, classificando-as em suas duas maiores categorias: curvas elípticas definidas sobre corpos binários e curvas elípticas definidas sobre corpos primos. Também são apresentadas neste capítulo as operações de adição e duplicação de pontos no grupo elíptico.
- O Capítulo 4 aborda aspectos básicos sobre criptografia. São apresentados os conceitos de algoritmos de chave pública e algoritmos de chave secreta, bem como os serviços normalmente providos por sistemas criptográficos.
- O Capítulo 5 apresenta os elementos que compõem um sistema criptográfico baseado em curvas elípticas. O capítulo detalha todo o leque de alternativas disponíveis e justificadas as escolhas tomadas para implementação de nosso sistema criptográfico.
- O Capítulo 6 apresenta uma série de algoritmos para aritmética nos corpos finitos $GF(2^m)$, sobre os quais foram definidas as curvas utilizadas em nosso ECC.
- O Capítulo 7 descreve e compara os algoritmos de multiplicação escalar implementados.

- O Capítulo 8 detalha os tempos obtidos em nossa implementação. Este capítulo compara o desempenho das duas aritméticas escolhidas, bem como confronta os tempos obtidos pelos algoritmos de multiplicação escalar implementados.
- O Capítulo 9 apresenta as conclusões finais e descreve as principais contribuições desta dissertação.

Capítulo 2

Conceitos Básicos de Álgebra e Teoria dos Números

2.1 Grupos

Definição 2.1.1 *Um grupo é uma estrutura algébrica constituída de um par (G, \diamond) , onde G é um conjunto de elementos e \diamond é uma operação binária que satisfaz as seguintes propriedades.*

1. *Fechamento: Para todo $a, b \in G$, $a \diamond b \in G$.*
2. *Associatividade: Para $a, b, c \in G$, $(a \diamond b) \diamond c = a \diamond (b \diamond c)$.*
3. *Existência de elemento neutro (identidade): Existe $e \in G$ tal que para todo $a \in G$, $a \diamond e = a$.*
4. *Existência de inverso: Para qualquer $a \in G$, existe um elemento $a^{-1} \in G$ tal que $a \diamond a^{-1} = e$.*

Além destas quatro propriedades, alguns grupos podem possuir uma quinta:

5. *Comutatividade: Para todo $a, b \in G$, $a \diamond b = b \diamond a$.*

Grupos com esta propriedade são chamados de *abelianos*.

Há inúmeros exemplos de grupos. Os mais familiares são aqueles vindos da aritmética elementar, como o dos números inteiros (\mathbb{Z}), cuja operação básica é a soma (daí chamado de grupo aditivo), onde 0 é a identidade e o inverso de um elemento é o seu negativo. Os números racionais não nulos (\mathbb{Q}^*) também formam um grupo, quando consideramos a operação de multiplicação (daí chamado de grupo multiplicativo). Este grupo tem 1 como elemento neutro e o inverso de cada $x \in \mathbb{Q}^*$ é o seu recíproco $1/x$.

2.1.1 Subgrupos

Considerando o grupo multiplicativo dos números racionais (\mathbb{Q}, \cdot) , notamos que apenas os racionais positivos (\mathbb{Q}^+, \cdot) já são por si só um grupo; o mesmo não acontece com o conjunto dos racionais negativos onde não são observadas as propriedades de fechamento (o produto de racionais negativos resulta em um racional positivo) e existência de elemento neutro.

O grupo (\mathbb{Q}^+, \cdot) é um exemplo de subgrupo. Um *subgrupo* é um subconjunto de um grupo, que obedece às quatro primeiras propriedades descritas acima, sendo portanto, também um grupo. De fato, se S é um subconjunto de G , é suficiente demonstrar que (S, \diamond) apresenta as propriedades de fechamento e existência de inverso para que seja um subgrupo de (G, \diamond) . Isso porque, associatividade é garantida do fato de que os elementos de S herdam esta propriedade de G . Além disso, se a^{-1} é o inverso de a , pela propriedade de fechamento $a \diamond a^{-1} = e \in S$, o que garante a existência do elemento neutro em S .

2.1.2 Grupos Finitos

Chamamos de *ordem* de um grupo (G, \diamond) o número de elementos de G ; denotamos a ordem de G por $|G|$.

Os grupos de ordem finita, chamados de *grupos finitos*, são os mais interessantes no estudo da criptografia. Um exemplo de grupo finito é o grupo aditivo $(\mathbb{Z}_n, +)$, onde \mathbb{Z}_n é o conjunto dos números inteiros tomados módulo n e $+$ é a operação de soma seguida de uma redução módulo n , isto é o cálculo do resto da divisão por n .

Outro grupo finito importante é (\mathbb{Z}_n^*, \cdot) onde \mathbb{Z}_n^* é o conjunto dos inteiros módulo n com

a propriedade adicional de possuírem inverso multiplicativo; a operação \cdot é a multiplicação inteira clássica, seguida de uma redução módulo n . Equivalentemente,

$$\mathbb{Z}_n^* = \{a \in \mathbb{Z}_n \mid \exists b \in \mathbb{Z}_n, a \cdot b \bmod n = 1\}.$$

O número de elementos do corpo \mathbb{Z}_n^* é portanto o número de elementos entre 1 e n que tem inverso multiplicativo módulo n . É conhecido que $a \in \mathbb{Z}$ possui inverso multiplicativo módulo n se e somente se $\text{mdc}(a, n) = 1$, onde $\text{mdc}(a, n)$ é o máximo divisor comum entre a e n . Assim, a ordem de \mathbb{Z}_n^* é dada pela seguinte função

$$\phi(n) = |\{x \mid \text{mdc}(x, n) = 1\}|.$$

Observe que se n é primo, $\phi(n) = n - 1$. Logo, o corpo \mathbb{Z}_p^* , para p primo, possui $p - 1$ elementos, ou seja, todos os inteiros entre 1 e $p - 1$ tem inverso multiplicativo módulo p .

A função $\phi(n)$ é conhecida como função ϕ -Euler. Ela possui algumas propriedades muito interessantes para teoria dos números. Dentre elas, temos

Fato 2.1.1 Se $\text{mdc}(a, b) = 1$,

$$\phi(a \cdot b) = \phi(a) \cdot \phi(b).$$

Grupos Cíclicos

Se em um grupo G existe um elemento g tal que para todo $a \in G$

$$a = \underbrace{g \diamond g \diamond \dots \diamond g}_{k \text{ vezes}}$$

para algum inteiro k , dizemos que g gera o grupo G . Neste caso, g é chamado de *gerador* e G é dito ser um *grupo cíclico*. Por definição, se $k = 0$, $a = e$.

A operação $a \diamond a \diamond \dots \diamond a$, onde $a \in G$ ocorre k vezes é comumente chamada *exponenciação* e denotada por a^k . O valor a^k é frequentemente chamado *potência* de a .

Não é difícil demonstrar que as potências de um elemento a formam um subgrupo de G , com o mesmo elemento neutro e de G . Neste caso dizemos que este grupo é *gerado* por a e o denotamos por $\langle a \rangle$. Segue daí que para todo elemento $a \in G$, existe um inteiro

i tal que $a^i = e$. Claramente, i é a ordem do subgrupo $\langle a \rangle$; i também é chamado *ordem* do elemento a .

Um dos teoremas fundamentais da teoria dos grupos, o teorema de Lagrange, diz que a ordem de qualquer subgrupo H de um grupo G divide a ordem de G .

Teorema 2.1.1 *Se H é um subgrupo de G , então $|H|$ divide $|G|$.*

Desse teorema e da discussão acima segue que

Corolário 2.1.1.1 *Se G é um grupo cíclico de ordem n , então a ordem de qualquer elemento $a \in G$ divide n .*

2.2 Corpos Finitos

Neste capítulo fazemos uma breve introdução à teoria dos corpos finitos. Para mais informações, consulte [Her64] ou [Kob94].

Definição 2.2.1 *Um corpo finito consiste de um conjunto finito de elementos F , juntamente com duas operações, chamadas adição e multiplicação, que satisfazem as seguintes propriedades.*

- $(F, +)$ é um grupo abeliano;
- $(F \setminus \{e\}, \cdot)$ é um grupo, onde e é o elemento neutro da operação $+$;
- Para todo $a \in F$,

$$(a + b) \cdot c = (a \cdot c) + (b \cdot c)$$

$$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$$

A *ordem* de um corpo finito é o número de elementos no corpo. Existe um corpo finito de ordem q se e somente se q é uma potência de um primo. Além disso, se q é potência de um primo, então essencialmente há apenas um corpo de ordem q ; este corpo é denotado por F_q ou $GF(q)$. Todos os corpos de mesma ordem são isomorfos.

Se $q = p^m$, onde p é primo e m é um inteiro positivo, então p é chamado a *característica* de F_q e m é chamado seu *grau de extensão*. Uma importante propriedade dos corpos

finitos, relacionada a sua característica, é a distributividade da exponenciação com relação a soma quando a potência é igual a característica do corpo. Assim, sendo F_{p^m} um corpo de característica p , para quaisquer elementos $a_0, a_1, \dots, a_{n-1} \in F_{p^m}$,

$$(a_0 + a_1 + \dots + a_{n-1})^p = a_0^p + a_1^p + \dots + a_{n-1}^p.$$

2.2.1 Bases

O corpo $GF(q)$, para p primo e $q = p^m$, pode ser visto como um espaço vetorial de dimensão m sobre $GF(p)$. Deste modo, cada elemento $a \in GF(q)$ é um vetor cujas coordenadas pertencem a $GF(p)$.

Sendo $GF(q)$ um espaço vetorial sobre $GF(p)$, existem elementos

$$\alpha_0, \alpha_1, \dots, \alpha_{m-1},$$

tais que qualquer elemento $a \in GF(q)$ pode ser escrito como combinação linear destes elementos. Logo, para todo $a \in GF(q)$,

$$a = \sum_{i=0}^{m-1} a_i \alpha_i,$$

onde $a_i \in GF(p)$ para $\{0, \dots, m-1\}$.

O conjunto $\{\alpha_0, \alpha_1, \dots, \alpha_{m-1}\}$ é chamado de *base* de $GF(q)$. A definição de uma base nos permite representar os elementos de um corpo $GF(p^m)$ como uma seqüência ordenada de elementos de $GF(p)$. Assim, é comum se representar o elemento a como o vetor $(a_{m-1}, a_{m-2}, \dots, a_1, a_0)$, uma vez definida a base.

É um fato conhecido da Teoria dos Números que para qualquer corpo finito sempre existe pelo menos uma base. De modo geral, muitas bases estão disponíveis para um mesmo corpo $GF(q)$. Diferentes bases produzem diferentes representações para um mesmo elemento de um corpo $GF(q)$.

Em termos de implementação da aritmética de um corpo, a escolha da base tem grande importância sobre complexidade de desempenhar cada uma das operações envolvidas. Dentre as diversas possibilidades, duas formas de base são as mais comumente encontradas na literatura, para usos relacionados à Criptografia. São elas as bases polinomiais e as bases normais.

Bases Polinomiais

Dado um polinômio

$$f(x) = f_m x^m + f_{m-1} x^{m-1} + \dots + f_0$$

onde $f_i \in GF(p)$. Se $f(x)$ é *irredutível*, ou seja, não pode ser escrito como o produto de dois polinômios de grau menor do que m , então para qualquer raiz $\alpha \in GF(p^m)$ de $f(x)$ (isto é, $f(\alpha) = 0$), o conjunto

$$\{\alpha^{m-1}, \alpha^{m-2}, \dots, \alpha, 1\}$$

forma uma base sobre $GF(p^m)$. Esta tipo de base é conhecida como base polinômial.

Quando uma base do tipo polinomial é utilizada, os elementos de $GF(p^m)$ podem ser visto como polinômios de coeficientes em $GF(p)$. A soma é desempenhada da mesma forma como é definida para polinômios. A multiplicação, no entanto, deve ser acrescida de um passo extra que consiste em dividir o polinômio produto pelo polinômio irredutível $f(x)$. A representação polinomial do elemento produto será o polinômio resto desta divisão.

Bases Normais

Uma base normal sobre $GF(p^m)$ é um conjunto da forma

$$\{\alpha^{p^{m-1}}, \alpha^{p^{m-2}}, \dots, \alpha^p, \alpha\}.$$

Uma das características mais importantes de bases normais é a baixa complexidade da exponenciação por potências de p . Isso porque, se $a \in GF(p^m)$, então

$$a = \sum_{i=0}^{m-1} a_i \alpha^{p^i},$$

para algum conjunto de $a_i \in GF(p)$. Logo,

$$\alpha^p = \left(\sum_{i=0}^{m-1} a_i \alpha^{p^i} \right)^p = \sum_{i=0}^{m-1} a_i^p \alpha^{p^{i+1}}.$$

Como $a_i^p = a_i$, já que $a_i \in GF(p)$ e $p - 1$ é a ordem do grupo multiplicativo de $GF(p)$, e ainda $\alpha^{p^m} = \alpha$, uma vez que $p^m - 1$ é a ordem do grupo multiplicativo do corpo $GF(p^m)$, temos que

$$\alpha^p = a_{m-1}\alpha + \sum_{i=0}^{m-2} a_i \alpha^{p^{i+1}}.$$

Assim, uma exponenciação por p consiste em um simples rearranjo dos coeficientes a_i da base.

2.2.2 Corpos Finitos de Interesse à Criptografia

Muitos padrões que especificam técnicas criptográficas baseadas em curvas elípticas se restringem a $GF(p)$ ou $GF(2^m)$. Os aspectos mais importantes destes corpos para a criptografia baseada em curvas elípticas são apresentados nas próximas seções.

Corpos Primos

Seja p um primo. Qualquer corpo $GF(p)$ pode ser representado pelo conjunto de inteiros

$$\{0, 1, \dots, p - 1\}$$

com as seguintes operações aritméticas:

- Adição módulo p : Se $a, b \in GF(p)$, então $r = (a + b) \bmod p$, isto é o resto da divisão de $(a + b)$ por p . Assim $0 \leq r < p$.
- Multiplicação módulo p : Se $a, b \in GF(p)$, então $r = a \cdot b \bmod p$.
- Inversão módulo p : Se a é não nulo em $GF(p)$, o inverso de a , denotado por a^{-1} é o único inteiro c tal que $0 \leq c < p$ e $a \cdot c \equiv 1 \pmod{p}$.

Apesar de $GF(2)$ satisfazer a definição acima, freqüentemente $GF(p)$ é chamado de *corpo primo* quando $p > 2$. Em especial, para aplicações criptográficas, p é geralmente um número grande de pelo menos 160 *bits*.

Corpos Binários

A aritmética para bases polinomiais e para bases normais, em corpos $GF(2^m)$ são descritas a seguir.

Bases Polinomiais

Considere $f(x) = x^m + f_{m-1}x^{m-1} + \dots + f_1x + 1$ (onde $f_i \in \{0, 1\}$ para $i \in \{1, \dots, m-1\}$) um *polinômio irredutível* de grau m . Desta forma, podemos definir

$$GF(2^m) = \{a_{m-1}x^{m-1} + \dots + a_1x + a_0 \mid a_i \in \{0, 1\}\}.$$

O elemento $a_{m-1}x^{m-1} + \dots + a_1x + a_0$ é usualmente denotado por uma cadeia de *bits* $(a_{m-1}a_{m-2} \dots a_1a_0)$. Então, os elementos de $GF(2^m)$ podem ser representados pelo conjunto de todas as cadeias binárias de comprimento m . A identidade multiplicativa é a cadeia $1 = (00 \dots 01)$ e identidade aditiva é a cadeia de $0 = (00 \dots 00)$.

Seja $f(x)$ um polinômio irredutível de grau m com coeficientes em $GF(2)$. As operações aritméticas definidas sobre $GF(2^m)$ usando bases polinomiais são:

- **Adição módulo $f(x)$:** Se $a = (a_{m-1} \dots a_1a_0)$ e $b = (b_{m-1} \dots b_1b_0)$ são elementos de $GF(2^m)$, então $a + b = c = (c_{m-1}c_{m-2} \dots c_1c_0)$, onde $c_i = a_i + b_i \pmod{2}$ (Computacionalmente isto equivale a um ou-exclusivo dos *bits* a_i e b_i).
- **Multiplicação módulo $f(x)$:** Se $a = (a_{m-1} \dots a_1a_0)$ e $b = (b_{m-1} \dots b_1b_0)$ são elementos de $GF(2^m)$, então $a \cdot b = r = (r_{m-1}r_{m-2} \dots r_1r_0)$, onde o polinômio $r = r_{m-1}x^{m-1} + \dots + r_1x + r_0$ é o resto obtido quando dividimos o polinômio $a \cdot b$ por $f(x)$.
- **Inversão módulo $f(x)$:** Se a é um elemento não nulo de $GF(2^m)$, o inverso de a , denotado por a^{-1} , é o único elemento $c \in GF(2^m)$ para o qual $a \cdot c = 1$.

Trataremos em maiores detalhes da aritmética em corpos binários no Capítulo 6.

Bases Normais

Uma base normal de $GF(2^m)$ sobre $GF(2)$ tem a forma $\{\alpha, \alpha^{2^1}, \dots, \alpha^{2^{m-1}}\}$, onde $\alpha \in GF(2^m)$. Tal base sempre existe e qualquer elemento $a \in GF(2^m)$ pode ser escrito

como:

$$a = \sum_{i=0}^{m-1} a_i \alpha^{2^i},$$

onde $a_i \in \{0, 1\}$.

A representação por bases normais tem a vantagem computacional de que o quadrado de um elemento pode ser calculado de forma bastante eficiente, como veremos a seguir. Entretanto, a multiplicação de dois elementos distintos pode ser muito custosa de modo geral. Por esta razão, o padrão ANSI X9.62 aconselha o uso de bases normais com propriedades especiais, chamadas *Gaussianas*.

Nas próximas seções apresentaremos os algoritmos básicos para as operações em corpos $GF(2^m)$, usando bases normais.

Quadrado

Seja x um elemento de $GF(2^m)$ escrito como

$$x = \sum_{i=0}^{m-1} a_i \alpha^{2^i}, \quad (2.1)$$

onde $\{\alpha, \alpha^2, \dots, \alpha^{2^{m-1}}\}$ é uma base normal e $a_i \in \{0, 1\}$, então

$$x^2 = \left(\sum_{i=0}^{m-1} a_i \alpha^{2^i} \right)^2 = \sum_{i=0}^{m-1} a_i (\alpha^{2^i})^2 = \sum_{i=0}^{m-1} a_i \alpha^{2^{i+1}}. \quad (2.2)$$

Mas $\alpha^{2^m} = \alpha$, uma vez que $2^m - 1$ é a ordem do grupo multiplicativo do corpo $GF(2^m)$. Assim, podemos reescrever a equação 2.2 como

$$x^2 = a_{m-1} \alpha + \sum_{i=0}^{m-2} a_i \alpha^{2^{i+1}}$$

Para simplificar a notação, freqüentemente costuma-se omitir os termos α^{2^i} da representação de cada elemento, simbolizando-o como uma cadeia de *bits*, cujos valores são os termos a_i do somatório da equação 2.1. Assim, x pode ser escrito de forma simplificada como

$$x = (a_{m-1} a_{m-2} \dots a_1 a_0)$$

e seu quadrado como

$$x^2 = (a_{m-2} \dots a_2 a_0 a_{m-1}).$$

Portanto, a representação de x^2 equivale a um *shift* cíclico para a esquerda da representação de x .

De modo similar ao quadrado, a raiz quadrada de um elemento x , em bases normais, pode ser calculada como um *shift* cíclico para a direita.

Multiplicação

Sejam $x, y \in GF(2^m)$, onde

$$x = \sum_{i=0}^{m-1} x_i \alpha^{2^i}$$

e

$$y = \sum_{j=0}^{m-1} y_j \alpha^{2^j}$$

Então,

$$xy = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} x_i y_j \alpha^{2^i} \alpha^{2^j} \quad (2.3)$$

Uma dificuldade na equação 2.3 é calcular $\alpha^{2^i} \alpha^{2^j}$, para $0 \leq i, j \leq m-1$. Evidentemente, como a $\alpha^{2^i} \alpha^{2^j} \in GF(2^m)$, podemos escrever

$$\alpha^{2^i} \alpha^{2^j} = \sum_{k=0}^{m-1} \lambda_{i,j}^{(k)} \alpha^{2^k}, \quad (2.4)$$

para determinados valores de $\lambda_{i,j}^{(k)} \in \{0, 1\}$. Assim, para cada par (i, j) temos um conjunto de m valores $\lambda_{i,j}^{(k)}$ que são a representação de $\alpha^{2^i} \alpha^{2^j}$ na base $\{\alpha, \alpha^2, \dots, \alpha^{2^{m-1}}\}$. Armazenar todos estes valores requer espaço $O(m^3)$. A seguir, descrevemos uma maneira de reduzir o número de valores $\lambda_{i,j}^{(k)}$ necessários para o cálculo do produto 2.3, que resulta em espaço de armazenamento da ordem de m^2 .

Defina uma função $B : GF(2^m) \times GF(2^m) \rightarrow \mathbb{Z}_2$ que simplesmente retorna o *bit* que ocupa a posição 0 do produto $xy = z = (z_{m-1} z_{m-2} \dots z_1 z_0)$, para $x, y \in GF(2^m)$, então

$$B(x, y) = z_0.$$

Se usarmos B para extrair o *bit* 0 da raiz quadrada de xy , veremos que

$$B(x^{1/2}, y^{1/2}) = z_1,$$

já que cada *bit* de xy está uma posição mais à direita em $(xy)^{1/2}$ (com exceção do *bit* z_0 que se encontra na posição $m - 1$). Assim, z_1 esta na posição 0 de $(xy)^{1/2}$.

O método acima pode ser repetido m vezes, de modo a se obter todos os *bits* da representação de xy , já que

$$B(x^{2^{-i}}, y^{2^{-i}}) = z_i \quad 0 \leq i \leq m - 1.$$

Combinando as equações 2.3 e 2.4, temos que

$$xy = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} x_i y_j \sum_{k=0}^{m-1} \lambda_{i,j}^{(k)} \alpha^{2^k}.$$

Portanto,

$$B(x, y) = z_0 = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} x_i y_j \lambda_{i,j}^{(0)},$$

e então é somente necessário armazenar m^2 valores $\lambda_{i,j}^{(0)}$ para o cálculo de xy .

Os valores λ podem ser armazenados em uma matriz, chamada *matriz de multiplicação* para a base $\{\alpha, \alpha^2, \alpha^{2^2} \dots, \alpha^{2^{m-1}}\}$, cuja entrada i, j contém o *bit* correspondente a $\lambda_{i,j}^{(0)}$. É claro que o número de entradas diferentes de 0 nesta matriz determina a complexidade do cálculo da função B . O padrão ANSI X9.62 aconselha o uso de bases normais Gaussianas, abreviadas GNB. Estas bases normais possuem um baixo número de 1's em sua matriz de multiplicação. A complexidade da multiplicação de uma base normal Gaussiana é expressa pelo seu tipo T , um inteiro positivo. Para um dado valor de m , quanto menor for valor de T , mais eficiente será a multiplicação da GNB daquele tipo. Apresentamos a seguir as condições para que uma GNB do tipo T exista.

Teorema 2.2.1 *Seja m um inteiro positivo não divisível por 8 e seja T ser um inteiro positivo. Então uma GNB do tipo T existe se e somente se $p = Tm + 1$ é um primo e $\text{mdc}(Tm/k, m) = 1$, onde k é a ordem multiplicativa de 2 módulo p .*

O padrão P1363 IEEE [iee98] apresenta uma fórmula para se calcular a função $B(x, y)$, ou seja, o *bit* 0 do produto xy .

Seja $p = Tm + 1$ e seja $u \in GF(p)$ um elemento de ordem T . Define-se a sequencia $F(1), F(2), \dots, F(p-1)$ por

$$F(2^i u^j \bmod p) = i,$$

para $0 \leq i \leq m-1$ e $0 \leq j \leq T-1$.

Se $x = (x_{m-1} \dots x_1 x_0)$ e $y = (y_{m-1} \dots y_1 y_0)$ são elementos de $GF(2^m)$, representados usando base normal, então se $xy = z = (z_{m-1} z_{m-2} \dots z_1 z_0)$,

$$B(x, y) = z_0 = \begin{cases} \sum_{k=1}^{p-2} x_{F(k+1)} y_{F(p-k)} & \text{se } T \text{ é par} \\ \sum_{k=1}^{m/2} (x_{k-1} y_{m/2+k-1} + x_{m/2+k-1} y_{k-1}) + \\ \sum_{k=1}^{p-2} x_{F(k+1)} y_{F(p-k)} & \text{se } T \text{ é ímpar} \end{cases}$$

Inversão

Um dos algoritmos mais simples para se efetuar inversão em $GF(2^m)$, não necessariamente usando representação por bases normais, usa a relação

$$\tau^{-1} = \tau^{2^m-2}, \quad \tau \in GF(2^m). \quad (2.5)$$

Itoh, Techai e Tsujii[IT88] propõem um método eficiente para realizar a exponenciação da equação 2.5, que consiste em reescrever o termo $2^m - 2$ como $2(2^{m-1} - 1)$ e em seguida aplicar uma das seguintes regras

$$2^{m-1} - 1 = \begin{cases} \left(2^{\frac{m-1}{2}} + 1\right) \left(2^{\frac{m-1}{2}} - 1\right), & \text{se } m \text{ ímpar} \\ 2 \left(2^{\frac{m-2}{2}} + 1\right) \left(2^{\frac{m-2}{2}} - 1\right) + 1, & \text{se } m \text{ par.} \end{cases}$$

O mesmo processo pode ser aplicado recursivamente para $\left(2^{\frac{m-1}{2}} - 1\right)$ ou $\left(2^{\frac{m-2}{2}} - 1\right)$, e assim sucessivamente até que não reste nenhum termo da forma $(2^r - 1)$, $2 \leq r \leq m-1$.

Observe que, se definirmos a função:

$$e(x, n) = x^{2^n - 1}$$

para algum $x \in GF(2^m)$ e algum $n < m$, teremos

$$e(\tau, n) = \begin{cases} [e(\tau, \frac{n}{2})]^{2^{\frac{n}{2}} + 1}, & \text{se } n \text{ par} \\ \tau \left[(e(\tau, \frac{n-1}{2}))^{2^{\frac{n-1}{2}} + 1} \right]^2 & \text{se } n \text{ ímpar.} \end{cases}$$

Apresentamos a seguir um exemplo de aplicação do método descrito acima para $GF(2^{53})$

Exemplo: Usando as decomposições sugeridas por Itoh *et al*, o expoente $2^{53} - 2$ pode ser reescrito usando as seguintes recorrências

$$2^{53} - 2 = 2(2^{52} - 1)$$

$$2^{52} - 1 = (2^{26} - 1)(2^{26} + 1)$$

$$2^{26} - 1 = (2^{13} - 1)(2^{13} + 1)$$

$$2^{13} - 1 = 2(2^6 - 1)(2^6 + 1) + 1$$

$$2^6 - 1 = (2^3 - 1)(2^3 + 1)$$

$$2^3 - 1 = 2(2 + 1) + 1$$

Então, a inversão de um elemento τ do corpo $GF(2^{53})$ pode ser implementada calcu-

lando-se as recorrências na seguinte ordem:

$$\begin{aligned}
 e(\tau, 3) &= (\tau^2 \tau)^2 \tau \\
 e(\tau, 6) &= e(\tau, 3)^{2^3} e(\tau, 3) \\
 e(\tau, 13) &= \tau \left(e(\tau, 6)^{2^6} e(\tau, 6) \right)^2 \\
 e(\tau, 26) &= e(\tau, 13)^{2^{13}} e(\tau, 13) \\
 e(\tau, 52) &= e(\tau, 26)^{2^{26}} e(\tau, 26) \\
 \tau^{-1} &= \tau^{2^{53}-2} = e(\tau, 52)^2.
 \end{aligned}$$

Assim, uma possível implementação para calcular τ^{-1} seria:

$$\begin{aligned}
 x &\leftarrow \tau * (\tau^2 * \tau)^2 \\
 x &\leftarrow x^{2^3} * x \\
 x &\leftarrow \tau * \left(x^{2^6} * x \right)^2 \\
 x &\leftarrow x^{2^{13}} * x \\
 x &\leftarrow x^{2^{26}} * x \\
 \tau^{-1} &\leftarrow x^2
 \end{aligned}$$

onde x é uma variável que é capaz de armazenar elementos de $GF(2^{53})$.

Como em bases normais a operação de elevar ao quadrado possui um baixo custo computacional, é vantajoso se utilizar o método de Itoh *et al* pois o número de multiplicações é reduzido em detrimento do número de quadrados. No caso de $GF(2^{53})$, por exemplo, foram necessárias 7 multiplicações para se calcular o inverso e 52 quadrados. De modo geral, o número de multiplicações do método pode ser calculado pela fórmula

$$M(m) = \lfloor \log_2(m-1) \rfloor + \omega(m-1) - 1$$

onde $\omega(m-1)$ denota o número de *bits* 1 na codificação binária de $m-1$.

Capítulo 3

Introdução à Teoria das Curvas Elípticas

Seja K um corpo. Dizemos que K é *algebricamente fechado* se todo polinômio de grau n com coeficientes em K possui n raízes em K . Um exemplo de corpo algebricamente fechado é o dos números complexos.

É um fato conhecido que todo corpo tem um supercorpo que é algebricamente fechado, chamado seu *fecho algébrico*. Se F_q é um corpo finito com q elementos, F_{q^r} , para $r > 1$, é uma extensão de F_q . Se considerarmos agora que $K = F_q$, então o fecho algébrico de K , geralmente denotado por \overline{K} , é a união de todas as extensões de F_q . Ou seja, $F_q := \bigcup_{r \geq 1} F_{q^r}$.

O *plano projetivo* $P^2(\overline{K})$ sobre \overline{K} é um conjunto de classes de equivalência em $\overline{K}^3 \setminus \{(0, 0, 0)\}$ onde $(x, y, z) \sim (u, v, w)$ se e somente se existir $\tau \in \overline{K}^*$ tal que $x = \tau u$, $y = \tau v$ e $z = \tau w$. Uma *equação de Weierstrass* é uma equação da forma

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_5Z^3, \quad (3.1)$$

onde $a_1, a_2, a_3, a_4, a_5 \in \overline{K}$. Uma equação de Weierstrass é *não-singular* se para todo ponto projetivo $P = (X, Y, Z) \in P^2(\overline{K})$, satisfazendo

$$F(X, Y, Z) = Y^2Z + a_1XYZ + a_3YZ^2 - X^3 - a_2X^2Z - a_4XZ^2 - a_5Z^3 = 0,$$

pelo menos uma das três derivadas parciais $\frac{\partial F}{\partial X}$, $\frac{\partial F}{\partial Y}$, $\frac{\partial F}{\partial Z}$ é diferente de 0 em P .

Uma *curva elíptica* é o conjunto de todas as soluções em $P^2(\overline{K})$ de uma equação de Weierstrass não-singular. Ela contém exatamente um ponto onde a coordenada Z é igual a 0. Este ponto é chamado *ponto no infinito* e é simbolizado por \mathcal{O} .

Algumas vezes é preferível escrever a equação de uma curva elíptica em coordenadas afim. Para isto, simplesmente usamos as relações $x = X/Z, y = Y/Z$ e a equação 3.1, obtendo

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_5. \quad (3.2)$$

Neste caso, uma curva elíptica é o conjunto de soluções da equação 3.2 no plano $\overline{K} \times \overline{K}$, juntamente com um ponto extra no infinito \mathcal{O} . Se $a_1, a_2, a_3, a_4, a_5 \in K$, então dizemos que E é definida sobre K e denotamos a curva por E/K . Além disto, o conjunto de todos os pontos sobre E cujas coordenadas pertencem a K é denotado $E(K)$.

Dois curvas elípticas podem ser isomorfas. O próximo teorema descreve quando este isomorfismo ocorre.

Teorema 3.0.2 *Dois curvas elípticas E_1/K e E_2/K dadas pelas equações*

$$E_1 : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_5$$

$$E_2 : y^2 + \overline{a}_1xy + \overline{a}_3y = x^3 + \overline{a}_2x^2 + \overline{a}_4x + \overline{a}_5$$

são isomorfas sobre K , se e somente se existem $u, r, s, t \in K$ e $u \neq 0$, tal que a mudança de variáveis.

$$(x, y) \longrightarrow (u^2x + r, u^3y + u^2sx + t) \quad (3.3)$$

transforma a equação de E_1 na equação de E_2 .

Usando o conceito de isomorfismo entre curvas elípticas, a equação 3.2 pode ser significativamente simplificada de acordo com a característica do corpo K sobre o qual ela está definida. A seguir descrevemos estas simplificações.

3.1 Curvas sobre Corpos de Característica $\neq 2$ e $\neq 3$

Seja E/K

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_5,$$

tal que a característica de K é diferente de 2. Uma transformação admissível de variável é

$$(x, y) \longrightarrow \left(x, y - \frac{a_1}{2}x - \frac{a_3}{2} \right),$$

a qual transforma E/K em

$$E_1/K : y^2 = x^3 + b_2x^2 + b_4x + b_5,$$

onde

$$\begin{aligned} b_2 &= a_2 + \frac{a_1^2}{4} \\ b_4 &= a_4 + \frac{a_1a_3}{2} \\ b_5 &= a_5 + \frac{a_3^2}{4} \end{aligned}$$

Nos casos em que a característica de K também é diferente de 3, a mudança de variável

$$(x, y) \longrightarrow \left(\frac{x - 3b_2}{36}, \frac{y}{216} \right)$$

pode ser aplicada sobre E_1/K para transformá-la em

$$E_2/K : y^2 = x^3 + ax + b.$$

Como E/K é isomorfa a E_1/K e E_1/K é isomorfa a E_2/K , podemos considerar que toda curva elíptica, definida sobre um corpo cuja característica é diferente de 2 e diferente de 3, pode ser escrita como

$$E : y^2 = x^3 + ax + b,$$

onde $a, b \in K$.

3.2 Curvas sobre Corpos de Característica 2

Seja E/K

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_5,$$

uma curva elíptica sobre K , de característica 2. Se $a_1 \neq 0$, então a mudança de variáveis

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

transforma E/K na curva

$$E_1/K : y^2 + xy = x^3 + b_2x^2 + b_5.$$

Quando $a_1 = 0$, uma mudança admissível de variáveis é

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

que transforma E/K na curva

$$E_2/K : y^2 + c_3y = x^3 + c_4x + c_5,$$

com

$$c_3 = a_3$$

$$c_4 = a_4 + a_2^2$$

$$c_5 = a_5 + a_4a_2$$

3.3 Grupo dos Pontos sobre uma Curva Elíptica

Pode-se definir uma operação, aplicável a quaisquer dois pontos sobre uma curva elíptica. Esta operação, comumente denominada *soma elíptica* e simbolizada por $+$, pode ser descrita geometricamente da seguinte forma.

Seja s a reta que passa pelos pontos P e Q . Uma vez que a equação de E possui grau 3, deve existir, salvo em certas situações especiais que discutiremos em breve, um terceiro ponto T , tal que $T \neq P$, $T \neq Q$ e $T \in (s \cap E)$. Dados dois pontos $P, Q \in E$, sua soma $P + Q = -T$, onde $-T$ é o ponto obtido pela reflexão de T em relação ao eixo x (Veja Figura 3.1). Existem dois casos especiais a serem considerados aqui. O primeiro deles é quando s é perpendicular ao eixo x : neste caso, não existe uma terceira intersecção da reta s com a curva E e convencionou-se que $P + Q = \mathcal{O}$, onde \mathcal{O} é chamado ponto no infinito. O outro caso especial é quando $P = Q$: neste caso, s é a reta tangente a E no ponto P e o restante da definição segue como anteriormente (Veja Figura 3.2).

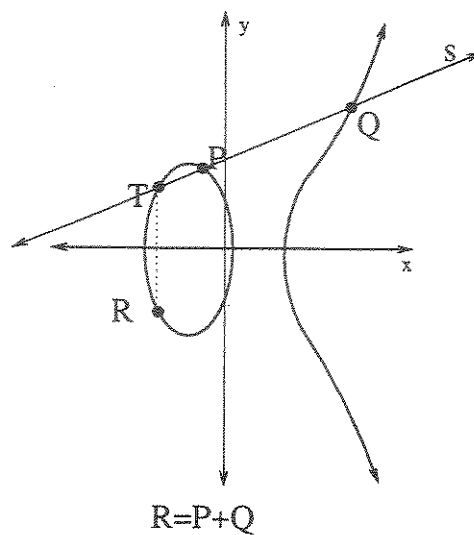


Figura 3.1: Soma elíptica de dois pontos distintos P e Q .

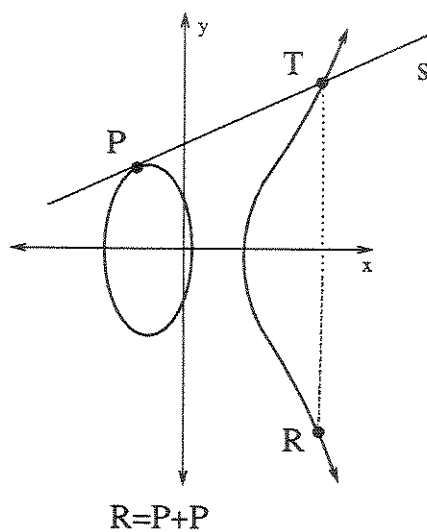


Figura 3.2: Soma elíptica de pontos iguais.

A soma elíptica possui as seguintes propriedades.

1. Associatividade - Para $P, Q, S \in E$, $(P + Q) + S = P + (Q + S)$;
2. Fechamento - Se $P, Q \in E$, então $(P + Q) \in E$;

3. Existência de inverso - Para todo $P \in E$ existe um $Q \in E$ tal que $P + Q = \mathcal{O}$. O ponto Q pode ser simbolizado por $-P$;
4. Existência de elemento neutro - Para todo $P \in E$, $P + \mathcal{O} = P$.
5. Comutatividade - Para $P, Q \in E$, $P + Q = Q + P$.

Assim, pelas propriedades 1-4, a soma elíptica e o conjunto de pontos sobre E formam um grupo. Além disto, pela propriedade 5, este grupo é abeliano.

As quatro últimas propriedades do grupo elíptico podem ser provadas sem grande esforço. Por outro lado, a associatividade exige um conhecimento matemático mais profundo.

Usando a definição geométrica da soma elíptica, $P + Q$ é o ponto no infinito \mathcal{O} se $P = (x_p, y_p)$ e $Q = (x_q, y_q)$ possuem a mesma coordenada x . Como \mathcal{O} é o elemento neutro do grupo (Propriedade 4 da soma elíptica), então $-P = Q$. Q pode ser facilmente encontrado, se fizermos a interseção da reta s , de equação $x = x_p$, com a curva E , descrita pela equação 3.2. O valor encontrado será

$$-P = Q = (x_p, -y_p - a_1x_p - a_3). \quad (3.4)$$

Podemos, a partir da definição geométrica descrita anteriormente, escrever fórmulas que expressem as coordenadas dos operandos P e Q em função das coordenadas de P e Q . Usamos para isso a equação geral de uma curva elíptica no espaço afim (Veja equação 3.2).

Seja s a reta que passa pelos pontos $P = (x_p, y_p)$ e $Q = (x_q, y_q)$, com $P \neq \pm Q$. A equação de s em função das coordenadas de Q e T é

$$y = \lambda(x - x_p) + y_p, \quad (3.5)$$

onde

$$\lambda = \frac{y_q - y_p}{x_q - x_p}.$$

Portanto, substituindo y na equação 3.2, pela expressão acima, obtemos

$$x^3 + (a_2 - \lambda^2 - a_1\lambda)x^2 + (a_4 + a_3\lambda_3 - a_1c - 2kc)x + a_5 - c^2 - a_3c, \quad (3.6)$$

onde $c = y_p - \lambda x_p$.

Como a equação 3.6 tem grau 3, o inverso aditivo da soma de suas raízes é igual ao quociente entre os coeficientes de x^2 e x^3 . Assim, sendo $T = (x_t, y_t)$ o terceiro ponto de interseção de s com a curva E ,

$$-(x_p + x_q + x_t) = a_2 - \lambda^2 - a_1\lambda.$$

Neste caso, o valor de x_t pode ser calculado pela seguinte fórmula

$$x_t = \lambda^2 + a_1\lambda - x_p - x_q - a_2.$$

Como P, Q e T estão sobre a mesma reta s , a coordenada y_t é obtida a partir da equação 3.5

$$y_t = \lambda(x_t - x_p) + y_p.$$

Finalmente, seja $R = (x_r, y_r)$ o inverso aditivo de T . Pela equação 3.4,

$$(x_r, y_r) = -T = (x_t, -y_t - a_1x_t - a_3).$$

Resta-nos agora tratar o caso em que $P = Q$. Como dissemos anteriormente, s será a tangente à curva E no ponto P . Nestas condições, a inclinação λ da reta s é dada pelo valor da derivada de y em relação a x da equação 3.2, ou seja,

$$y'(x_p) = \lambda = \frac{3x_p^2 + 2a_2 - a_1y_p + a_4}{2y_p + a_1x_p + a_3}.$$

O restante do cálculo de $P + P$ segue como descrito anteriormente.

Pelas equações acima, podemos resumir a soma e a duplicação elíptica nas seguintes equações, de acordo com a característica do corpo.

F_p	$E(F_p)$	$P_1 + P_2 = P_3$	$2P_1 = P_3$
$p = 2$	$y^2 + xy = x^3 + ax^2 + b,$ $b \neq 0$	$\lambda = \frac{y_2 + y_1}{x_2 + x_1}$ $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$ $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$	$\lambda = x_1 + \frac{y_1}{x_1}$ $x_3 = \lambda^2 + \lambda + a$ $y_3 = x_1^2 + (\lambda + 1)x_3$
$p > 3$	$y^2 = x^3 + ax + b,$ $4a^3 + 27b^2 \neq 0$	$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$ $x_3 = \lambda^2 - x_1 - x_2$ $y_3 = \lambda(x_1 - x_3) - y_1$	$\lambda = \frac{3x_1^2 + a}{2y_1}$ $x_3 = \lambda^2 - 2x_1$ $y_3 = \lambda(x_1 - x_3) - y_1$

Tabela 3.1: Fórmulas para adição e duplicação elíptica de acordo com a característica do corpo base.

De forma analoga à multiplicação inteira, a soma elíptica $P + P$ pode ser escrita como $2P$. De modo geral, a soma de k pontos P pode ser simbolizada por kP . Esta operação é chamada de *multiplicação escalar*.

A multiplicação escalar kP pode ser calculada em tempo polinomial. Entretanto, não se conhece nenhum algoritmo determinístico eficiente para, dados kP e P , calcular o valor de k . Este problema, formalmente definido abaixo, é chamado *problema do logaritmo discreto sobre curvas elípticas* (ECDLP). Ele é a base dos sistemas criptográficos baseados em curvas elípticas. A seguinte definição segue de [Kob94].

Definição 3.3.1 *Seja E uma curva elíptica sobre um corpo K e B e P pontos de E . O problema do logaritmo discreto sobre E (para a base B) é o de encontrar um inteiro $k \in \mathbb{Z}$, se existir, tal que $kB = P$.*

Capítulo 4

Sistemas Criptográficos

É desejo antigo do homem proteger suas mensagens do alcance daqueles a quem elas não estejam diretamente endereçadas. Hoje, além de objetivos militares, ter privacidade em suas comunicações particulares tem se tornado cada vez mais importante para empresas e também para cidadãos comuns, interessados em manter o conteúdo do que enviam longe dos olhos de concorrentes ou curiosos, em uma rede de extensão global.

A *criptografia* é a ciência que se preocupa basicamente em transformar a grafia de textos, de modo a ocultar seu conteúdo semântico. Adicionalmente, compreende métodos que permitam estabelecer a autenticidade de informações, detectar possíveis adulterações, bem como associa-las à sua origem de maneira inequívoca.

A transformação de textos envolve dois processos: *ciframento* e *deciframento*. Ciframento corresponde a conversão de mensagens, ou *textos em claro* em criptogramas ou *textos cifrados*. Deciframento, por sua vez, é o processo que recupera a informação original a partir do criptograma.

Ciframento e deciframento baseiam-se em dois componentes: um *algoritmo criptográfico* e uma *chave*. O primeiro corresponde a uma transformação matemática que converte um texto claro em um texto cifrado e vice-versa. Um bom algoritmo criptográfico não deve depositar sua segurança na hipótese de que seu adversário o desconheça. Todos os algoritmos considerados seguros atualmente dependem da chave utilizada.

Uma chave é uma seqüência de caracteres ou dígitos numéricos que é usada para

personalizar o processo de ciframento ou deciframento, tornando-o de controle exclusivo do proprietário da chave.

O conjunto de todos os possíveis textos, criptogramas e chaves, para um determinado algoritmo criptográfico, bem como cada uma de suas possíveis regras de ciframento e deciframento, é chamado de *sistema criptográfico*. A definição formal de sistema criptográfico, extraída de [Sti95], é apresentada a seguir.

Definição 4.0.2 *Um sistema criptográfico é uma quintupla $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, onde as seguintes condições são satisfeitas:*

1. \mathcal{P} é um conjunto finito de todas as mensagens possíveis, designado por espaço de mensagens;
2. \mathcal{C} é um conjunto finito de todos os criptogramas possíveis, designado por espaço de criptogramas;
3. \mathcal{K} , designado por espaço de chaves, é o conjunto finito de todas as chaves possíveis;
4. Para cada $K \in \mathcal{K}$, há uma regra de ciframento $e_K \in \mathcal{E}$ e uma regra de deciframento $d_K \in \mathcal{D}$. Cada $e_K : \mathcal{P} \rightarrow \mathcal{C}$ e $d_K : \mathcal{C} \rightarrow \mathcal{P}$ são funções tais que $d_K(e_K(x)) = x$ para cada $x \in \mathcal{P}$.

Existem dois tipos de sistemas criptográficos: os *sistemas de chave secreta*, também chamados de *sistemas simétricos* e os *sistemas de chave pública*, chamados de *sistemas assimétricos*. No que segue, A e B denotam duas partes comunicantes.

4.1 Sistemas Simétricos

Em um sistema simétrico ou de chave secreta, ambos A e B conhecem as chaves de ciframento e deciframento. Na maioria dos casos as chaves são iguais. Assim, é impossível para alguém externo ao par A e B , determinar a autoria de um criptograma. Por outro lado, tanto A como B tem certeza da autoria desse criptograma, desde que a chave não seja revelada a mais ninguém.

Exemplos modernos de algoritmos simétricos são apresentados a seguir.

- DES - Data Encryption Standard - cifra textos de até 64 *bits* com chaves de 56 *bits*. Está caindo em desuso pois a chave é pequena o suficiente para permitir o ataque por força bruta.
- Triple DES - DES reforçado - cifra textos de 64 *bits* com chave de 128 bits.
- IDEA - International Data Encryption Algorithm - cifra textos de 128 *bits* com chave de 128 bits. Método desenvolvido na Europa. Após vários anos de utilização e divulgação pública, não se conhecem ataques com êxito.
- AES - Advanced Encryption Standard - Novo padrão internacional, resultado de um concurso público, ganho pelo algoritmo Rijndael, dos belgas Joan Daemen, da Proton World e Vincent Rijmen, da Computer Security and Industrial Cryptography(COSIC). Admite chaves de 128, 192 ou 256 bits.

Sistemas simétricos freqüentemente são mais rápidos que sistemas assimétricos. Entretanto a hipótese de que ambas as entidades devam ter cópias da chave secreta nem sempre é prática. Isso porque esta hipótese exige que pelo menos uma das entidades deva obter a chave através de um canal seguro (o que nem sempre é possível) ou que exista previamente uma chave para cada par de entidades. Isso obriga que cada entidade possua tantas chaves quantos sejam os elementos da rede, o que é inviável em muitas situações. Felizmente, ainda existe uma terceira possibilidade: utilizar um *protocolo de distribuição de chaves*, como o protocolo Diffie-Hellman, que discutiremos na seção 4.2.1.

4.2 Sistemas Assimétricos

A maneira de resolver os problemas de distribuição de chaves da criptografia simétrica é a utilização da *criptografia assimétrica* ou de *chave pública*. Na criptografia assimétrica, cada entidade *A* possui um par de chaves, uma pública e outra privada. A chave pública de *A* destina-se ao envio de mensagens cifradas para *A*, por outras entidades. *A* decifra essas mensagens usando sua chave privada. Como o próprio nome diz, a chave pública de *A* é amplamente disponível ao público, enquanto que a chave privada é de conhecimento

exclusivo de A . É claro que este sistema é efetivo somente se a chave privada não puder ser deduzida facilmente da chave pública.

A função de deciframento dos sistemas de chave pública é uma função unidirecional com porta de escape (*trapdoor one way function*), ou seja, uma função de ciframento que é fácil de calcular, difícil de inverter (decifrar), mas que com o conhecimento de alguma informação adicional, chamada de *trapdoor*, pode ser facilmente invertida. Este conhecimento adicional deve ser mantido em segredo pelo receptor [Sti95]. Assim, a chave secreta é o conhecimento adicional necessário para que o receptor seja capaz de decodificar mensagens cifradas com sua chave pública.

A grande vantagem de sistema assimétricos é permitir que qualquer um possa enviar uma mensagem secreta, apenas utilizando a chave pública de quem irá recebê-la. Como a chave pública está amplamente disponível, não há necessidade do envio de chaves como é feito no modelo simétrico. A confidencialidade da mensagem é garantida enquanto a chave privada estiver segura. Caso contrário, quem possuir acesso à chave privada terá acesso às mensagens.

Apresentamos a seguir uma breve descrição de alguns sistemas de chave pública.

4.2.1 Diffie-Hellman

O sistema de troca de chaves de Diffie e Hellman, foi proposto em [DH76] e marca o início da criptografia de chave pública. Este sistema não é um sistema de ciframento, mas sim um sistema público de estabelecimento de chaves secretas entre duas entidades, utilizando um meio de comunicação não seguro. A sua segurança é baseada na dificuldade aparente do cálculo de logaritmos discretos.

No sistema Diffie-Helman, para um dado grupo cíclico G e um gerador $g \in G$, cada entidade i gera aleatoriamente um número $a_i \in \{1, \dots, |G| - 1\}$. Em seguida, a entidade i calcula o valor g^{a_i} , que é disponibilizado imediatamente ou enviado após uma solicitação de estabelecimento de sessão por alguma outra entidade. Assim, se duas entidades i e j desejam se comunicar, a chave comum de sessão $k_{i,j}$ é calculada independentemente por i e por j da seguinte forma:

$$k_{i,j} = (g^{a_i})^{a_j} = (g^{a_j})^{a_i}.$$

4.2.2 RSA

Um dos sistemas criptográficos de chave pública mais populares é o RSA, sigla constituída pelas iniciais dos nomes de seus criadores Ronald L. Rivest, Adi Shamir e Leonard Adleman. Seu funcionamento é muito simples e pode ser dividido em três fases:

1. geração de chaves

- (a) Encontre dois primos grandes p e q (por exemplo, de 1024 *bits*) e calcule o produto $n = pq$;
- (b) Escolha um inteiro e menor do que pq , tal que $\text{mdc}(e, (p-1)(q-1)) = 1$;
- (c) Compute d tal que $de \equiv 1 \pmod{(p-1)(q-1)}$. Uma chave pública RSA consiste no par (n, e) , sendo sua chave privada o número d .

2. ciframento

Mensagens em texto claro são convertidas em criptograma com a função de ciframento:

$$f_e(m) = m^e \pmod{n},$$

onde m é a mensagem, convertida para um inteiro positivo, $0 < m < n$.

3. deciframento

Nesta fase, criptogramas são convertidos em texto claro, usando a função de deciframento:

$$f_d(m) = (m')^d \pmod{n},$$

onde m' é o texto cifrado (criptograma) computado pela função f_e .

A força do RSA vem do fato de que obter p e q , a partir do valor de n é uma tarefa muito difícil. De outro modo, obter a chave secreta a partir da chave pública seria simples, uma vez que d é apenas o inverso multiplicativo de e módulo $(p-1)(q-1)$. No entanto, nada foi provado, de modo genérico, que encontrar d dependa de (ou pelo menos seja tão difícil quanto) encontrar os valores de p e q . Por isso, prefere-se desvincular o problema

de encontrar p e q a partir de n , conhecido como Problema da Fatoração de Números Inteiros (IFP), do problema de encontrar d a partir de e e n , conhecido como Problema RSA (RSAP).

Um estudo sobre a relação entre RSAP e IFP é apresentado por Boneh e Venkatesan, em [BV98], no qual os autores mostram que para valores pequenos de e , RSAP pode ser mais fácil que IFP. No entanto, com exceção de tais valores especiais, o único método conhecido para se atacar o RSA é através da fatoração de n .

Se fatorar n for o único meio de solucionar RSAP, a análise histórica do IFP fornece sérios indícios de que este problema é realmente muito difícil. O IFP é um problema quase tão antigo quanto a teoria dos números e no entanto, nenhum algoritmo prático foi desenvolvido até o dia de hoje que pudesse fatorar números maiores que 160 dígitos decimais, com tempo e processamento condizentes com a tecnologia disponível.

4.2.3 ElGamal

Outro algoritmo assimétrico bastante popular é o algoritmo de El Gamal, o qual está baseado na dificuldade de se resolver o logaritmo discreto para alguns grupos cíclicos.

O esquema de ciframento de ElGamal é geralmente descrito sobre o grupo multiplicativo \mathbb{Z}_p^* , mas ele pode ser facilmente generalizado para qualquer grupo cíclico finito G . Para isso, G deve satisfazer duas condições:

1. Eficiência- As operações em G devem ser fáceis de aplicar.
2. Segurança- O problema do logaritmo discreto deve ser intratável no grupo G .

Alguns grupos que aparentemente obedecem as condições acima são: o grupo multiplicativo \mathbb{Z}_p^* de inteiros módulo p , o grupo multiplicativo $\mathbb{F}_{2^m}^*$ do corpo finito $GF(2^m)$, o grupo multiplicativo \mathbb{F}_q^* do corpo finito $GF(q)$, onde $q = p^m$, p primo, e, como veremos no capítulo 3, o grupo de pontos de uma curva elíptica sobre um corpo finito.

Apresentamos a seguir a descrição do algoritmo genérico de ElGamal. Assim como no RSA, El Gamal pode ser dividido em três fases: geração de chaves, ciframento e deciframento.

1. geração de chaves

Cada entidade cria uma chave pública e uma chave privada correspondente. Para isto cada entidade A deve executar os seguintes passos:

- (a) Selecionar um grupo cíclico apropriado G de ordem n , com um gerador α .
- (b) Selecionar um inteiro aleatório a , $1 \leq a \leq n - 1$, e calcular α^a .
- (c) A chave pública de A é (α, α^a) em conjunto com a descrição de como multiplicar elementos em G , e a chave privada é a .

2. ciframento

Para cifrar uma mensagem m e enviá-la a uma entidade A , uma entidade B deve executar os seguintes passos:

- (a) Obter a chave pública (α, α^a) de A .
- (b) Representar a mensagem como um elemento m do grupo G .
- (c) Selecionar um inteiro aleatório k , $1 \leq k \leq n - 1$.
- (d) Calcular $\gamma = \alpha^k$ e $\delta = m \cdot (\alpha^a)^k$.
- (e) Enviar o texto cifrado $y = (\gamma, \delta)$ para A .

3. deciframento

Para recuperar o texto m , enviado por B , a partir de y , A deve:

- (a) Usar a chave privada a para calcular γ^a e então calcular γ^{-a} .
- (b) Recuperar m através do cálculo $(\gamma^{-a})\delta$.

O sistema criptográfico de ElGamal é não determinístico, já que o texto cifrado depende tanto da mensagem clara como do valor aleatório k escolhido pelo emissor da mensagem. Assim, uma mesma mensagem pode ser cifrada de várias formas diferentes.

4.2.4 Assinaturas Digitais

Assinatura digital é um mecanismo criptográfico que permite associar a uma mensagem m , de forma inequívoca, a identidade do autor de m . Assim, garante-se não só a autenticidade e integridade de m como também a impossibilidade de futura repudição da sua autoria.

De forma geral, para gerar uma assinatura digital s de uma mensagem m , uma entidade A deve calcular uma função

$$s = f(r_A, m),$$

onde r_A é a chave privada de A e f é tal que existe uma função booleana g com a propriedade de que $g(p_A, m, s)$, onde p_A é a chave pública de A , devolve verdadeiro somente se, com altíssima probabilidade, A é a única entidade que poderia ter gerado s a partir de r_A e m . Com isso, fica garantida a autoria de A sobre s (e também a autenticidade e integridade de m). Como exemplo, no sistema RSA,

$$s = f(r_A, m) = m^{r_A} \bmod n,$$

enquanto

$$g(p_A, m, s) = \begin{cases} \text{verdadeiro} & \text{se } s^{p_A} \bmod n = m, \\ \text{falso} & \text{se } s^{p_A} \bmod n \neq m. \end{cases}$$

Note que assinaturas digitais não garantem o sigilo da mensagem. Este requisito deve ser obtido com a utilização adicional de um algoritmo de ciframento. Assim, A deve assinar sua mensagem usando sua chave secreta e em seguida cifrar o par (mensagem, assinatura) usando a chave pública de B .

Alguns algoritmos de chave pública podem ser usados também para assinaturas digitais. Como exemplo, temos o RSA, que tem a propriedade de ser comutativo: $(m^e)^d = (m^d)^e = m$. No entanto, alguns algoritmos foram especificamente projetados para serem utilizados como assinaturas digitais. Este é o caso do DSA, proposto pelo NIST[NIS00] em agosto de 1991, para utilização como padrão de assinatura digital do governo norte-americano.

4.3 Hashing

Todos os algoritmos de assinatura digital são muito lentos e usá-los para assinar grandes mensagens, degradaria em muito o desempenho de um sistema criptográfico.

A solução deste problema é a utilização de funções de *Hashing* criptográfico. Também denominada *Message Digest*, *One-Way Hash Function*, Função de Condensação ou Função de Espalhamento Unidirecional, uma função *Hashing* gera, a partir de uma entrada de tamanho variável, um valor fixo pequeno: o *digest* ou valor *hash*.

Obviamente, não há uma relação de um-para-um entre os valores de entrada e os valores de saída de uma função de *hashing* h . É perfeitamente possível que várias mensagens tenham o mesmo valor de *hash*. No entanto, é extremamente difícil, a partir de um valor de *hash* $h(m)$, encontrar uma ou mais mensagens m_1, \dots, m_j tais que $h(m_1) = \dots = h(m_j)$. Assim, o valor de *hash* pode funcionar como o dígito verificador de uma conta corrente ou o *check sum* de um pacote de dados. Assinaturas podem então ser calculadas sobre $h(m)$ sem perda as propriedades de autenticidade e não-repúdio.

A seguir são descritas algumas funções *hashing* criptográfico bem conhecidas.

- MD2, MD4, MD5 - As funções de *hash* MD (*Message Digest*) foram propostas por Rivest e são utilizadas por diversas aplicações, entre elas pelo SNMP (*Secure Network Management Protocol*), pelo PGP e por muitos dos esquemas de assinaturas digitais utilizados. A função proposta inicialmente foi a MD2, havendo então uma evolução contínua até a MD5, a versão mais recente. O MD4 é o mais rápido e o MD5 é uma forma mais segura do MD4, porém mais lenta. Todos os três algoritmos recebem como entrada mensagens de tamanho arbitrário e produzem valores *digest* de 128 *bits*. Apesar dos MDs possuírem estruturas similares, o projeto do MD2 é muito diferente daquele do MD4 e do MD5. MD2 foi otimizado para máquinas de 8 *bits*, ao passo que MD4 e MD5 foram feitos para máquinas de 32 *bits*. A descrição do código fonte para os três algoritmos é encontrada nas RFCs 1319-1321 [Jr92, Riv92a, Riv92b]
- SHA, SHA-1 - O *Secure Hash Algorithm* (SHA) foi desenvolvido pelo NIST e é especificado no *Secure Hash Standard* (SHS, FIPS 180). SHA-1 é uma revisão para

esta versão e foi publicado em 1994, em uma modificação técnica do padrão SHS (SHS, FIPS 180-1). Ele é também descrito no padrão ANSI X9.30 (part 2). SHA-1 produz um valor de *hash* com 160 *bits* (20 byte). Embora seja mais lento do que o MD5, é amplamente utilizado devido ao fato de ser menos vulnerável a ataques do tipo força bruta, uma vez que sua chave possui um número maior de *bits*.

4.4 Sistemas Híbridos

Sistemas simétricos e assimétricos possuem características complementares. Assim, eles podem ser utilizados em conjunto de modo que cada um supra as desvantagens do outro. Sistemas que utilizam algoritmos simétricos e assimétricos em conjunto são chamados de *sistemas híbridos*.

Em um sistema híbrido, um algoritmo de chave pública é utilizado para realizar a troca de chaves secretas entre duas partes comunicantes. Em seguida, um algoritmo simétrico rápido pode ser utilizado para a efetiva troca de mensagens entre as duas entidades. Paralelamente, cada mensagem pode ser assinada pelo remetente usando um algoritmo de assinatura digital, cuja eficiência será garantida por uma função de *hashing*, que reduz o número de *bits* que necessitam ser assinados.

4.5 Segurança dos Algoritmos Criptográficos

É desejável que sistemas criptográficos sejam projetados de tal modo que o custo para quebrá-lo seja superior ao valor da informação por eles ocultada. No entanto, construir sistemas deste tipo é uma tarefa que exige do projetista algum esforço intelectual, cuidado e perícia. Além disso, todo e qualquer procedimento utilizado para reforçar a segurança de um algoritmo deve ser explicitamente documentado e deve estar disponível aos usuários deste sistema. Deve-se ser muito cuidadoso a respeito de algoritmos não publicados. Muito vezes, o projetista destes algoritmos não tem certeza quanto a sua segurança ou esta segurança depende do ocultamento dos procedimentos internos do algoritmo. Neste último caso, deve-se levar em consideração que, especialmente em *software*, é possível

utilizar técnicas de análise do código executável e de engenharia reversa para se obter o algoritmo. Além disso, a história tem mostrado que a maioria dos algoritmos secretos, ao serem tornados de conhecimento público, se mostraram na realidade fracos.

Na teoria, qualquer método criptográfico com uma chave pode ser quebrado tentando-se todas as possíveis chaves sequencialmente. Isto é chamado de *ataque por força bruta*. Se usar força bruta é a única opção, o poder computacional exigido cresce exponencialmente com o comprimento da chave. Uma chave de 32 *bits* exige 2^{32} (cerca de 10^9) passos. Isto é algo que qualquer um pode fazer em seu computador pessoal. No entanto, chaves com 56 *bits* (tal como um DES) já exigem um pouco mais de poder computacional, ao qual se pode acrescentar esforço distribuído ou mesmo a ajuda de *hardware* especializado. Apesar do custo para quebrar esta última chave ser um pouco maior, ele está dentro do alcance das organizações criminais, grandes companhias e governos, que possam ter algum interesse no conteúdo das informações ocultadas pelo algoritmo.

Em geral, elevar o tamanho das chaves provê acréscimo na segurança do algoritmo a ataques do tipo força bruta. Todavia, muitos algoritmos podem ser quebrados sem que tentemos todas as possíveis chaves. Entretanto, é muito difícil projetar cifradores que não possam ser quebrados mais eficientemente usando outros métodos. Por isso, projetar seus próprios cifradores não é recomendado para aplicações reais, a menos que o usuário seja um especialista e saiba o que está fazendo.

Capítulo 5

Implementação de ECCs

Implementar um sistema criptográfico baseado em curvas elípticas é uma tarefa que exige uma série de escolhas. Tais escolhas envolvem:

1. Tipo de corpo base ($GF(p)$ ou $GF(2^m)$);
2. Representação (polinomial, normal, etc);
3. Tipo de curva (randômicas ou de Koblitz);
4. Representação dos pontos da curva (afim ou projetiva);

Há muitos fatores que podem influenciar a decisão a ser tomada. Todos eles devem ser considerados simultaneamente de modo a se alcançar a melhor solução para uma aplicação particular. Dentre estes fatores se inclui:

- Considerações de segurança;
- Disponibilidade de métodos para otimizar a aritmética no corpo (adição, multiplicação, quadrado e inversão);
- Disponibilidade de métodos para otimizar a aritmética na curva elíptica (adição, duplicação e multiplicação escalar);
- Plataforma da aplicação (software, hardware, etc);

- Restrições de um particular ambiente de computação (velocidade do processador, armazenamento, consumo de potência, etc);
- Restrições de um particular ambiente de comunicação (largura de banda, por exemplo)

Nosso objetivo aqui é dar um exemplo de construção de um ECC que proverá os serviços de ciframento e assinatura digital. Pretendemos detalhar cada uma das etapas, de modo que o leitor seja capaz de identificar todas as escolhas e tomar decisões acertadas na implementação de seu próprio sistema criptográfico.

5.1 Por que Curvas Elípticas?

A segurança dos sistemas criptográficos baseados em curvas elípticas (ECC) depende da dificuldade de determinar o logaritmo discreto no grupo de pontos de uma curva elíptica (ECDLP). Este problema é atualmente considerado de dificuldade maior que a fatoração de números inteiros (IFP) ou a computação de logaritmos discretos em um corpo finito (DLP). Na verdade, desde que em 1985, V. Miller[Mil86] e N. Koblitz[Kob87] sugeriram, independentemente, o uso de curvas elípticas em sistemas criptográficos de chave pública, nenhum progresso substancial em atacar o ECDLP ocorreu, com exceção de alguns casos específicos, os quais podem ser facilmente identificados e evitados. Isto permite que ECCs possam usar parâmetros muito menores que sistemas RSA ou sistemas baseados no clássico problema do logaritmo discreto sobre corpos finitos. Este *overhead* reduzido torna ECCs ideais para aplicações com poucos recursos, tais como *smartcards*, telefones celulares, transações via *web*, etc. O interesse por curvas elípticas está profundamente relacionado ao surgimento e crescimento de tais aplicações.

Outra vantagem importante de curvas elípticas é o fato de a geração de chaves ser um processo bastante simples e rápido. Como vimos no Capítulo 4.2.2, a geração de um par de chaves RSA exige algoritmos para (i) obtenção de números aleatórios para candidatos a primos; (ii) testar a primalidade destes candidatos e (iii) solucionar a equação modular para determinar o expoente d . Isto é um processo relativamente complexo, sendo que todo

ele deve ser feito de um modo seguro e sem erros, exclusivamente pelo futuro proprietário do par de chaves.

De forma diferente, em ECCs a geração de chaves possui duas partes: (i) geração de um conjunto válido de parâmetros de domínio público, isto é, valores que podem ser compartilhados por mais de uma entidade; (ii) geração de um par de chaves associada com aquele conjunto, isto é, criação de chaves exclusivas a uma entidade. Para gerar um conjunto de parâmetros de domínio público, um corpo finito deve ser selecionado, uma curva elíptica precisa ser gerada sobre aquele corpo, a ordem desta curva deve então ser determinada, via contagem de pontos e um ponto gerador P , com uma ordem prima grande deve ser escolhido. Isto também é um processo complexo mas não há nada que exija que todo este cálculo deva ser feito pelo proprietário da chave.

Dado um conjunto válido de parâmetros de domínio público, a geração do par de chaves ECC consiste em gerar um número aleatório d , de certo tamanho (menor que ordem do gerador P), que será a chave secreta. A chave pública do usuário será então um ponto $Q = dP$. Isto significa que a geração de um par de chaves ECC exige apenas a capacidade de obter números aleatórios e calcular multiplicações escalares, o que é uma operação normal em ECCs. Assim, se considerarmos que a parte (i) da geração já tenha sido realizada previamente, gerar unicamente o par de chaves ECC é um processo muito mais simples e rápido que o equivalente em sistemas RSA.

Outro aspecto interessante dos ECCs é a divisão das operações aritméticas em duas camadas. Isto permite uma certa independência entre corpo, curva elíptica e chaves do sistema, possibilitando um melhor ajuste desses elementos de acordo com nossos requisitos de segurança, eficiência e interoperabilidade. Assim, em ECCs é possível compartilhar uma biblioteca otimizada de aritmética em um corpo específico. Isto garante, além de eficiência, um maior grau de interoperabilidade.

No entanto, mesmo com as vantagens apresentadas anteriormente e mais de 16 anos de existência, freqüentemente ECCs são considerados sistemas imaturos, não suficientemente expostos ao escrutínio da comunidade científica. Isto acontece porque, na ausência de provas que realmente atestem a segurança (ou insegurança) de seus principais algoritmos, a criptografia de chave pública é obrigada a conviver com incertezas. Neste contexto,

só o tempo e muita pesquisa pode garantir um certo grau de confiança a um algoritmo. Tal é o caso do RSA, sujeito a mais de vinte anos de estudo específico e outros muitos séculos de conhecimento do problema subjacente da fatoração de inteiros. Assim, não é de se surpreender que curvas elípticas tenham tido papel secundário por diversos anos e despertem até hoje receios quanto à sua utilização em aplicações de grande escala. No entanto, na década de 90, o surgimento de aplicações com sérias restrições de recursos, como *smartcards* e comunicações sem fio, despertaram forte interesse por curvas elípticas e motivaram, paralelamente, intensa pesquisa sobre o ECDLP. É verdade que ECCs ainda são relativamente recentes e que o ECDLP não tenha sido alvo de tantos ataques como é o caso do IFP e do DLP. Entretanto, é muito provável que esta redescoberta dos ECCs aumente a confiança em curvas elípticas e que em um futuro próximo, elas ocupem um papel de destaque no seletto mundo dos algoritmos de chave pública.

5.2 Escolha do Corpo Finito

Quando trabalhamos com curvas elípticas, devemos ter em mente que existem dois tipos de aritmética envolvidas. A primeira delas, às vezes chamada de aritmética de baixo nível, envolve o corpo finito sobre o qual a curva está definida. É este corpo que determina o tamanho dos blocos de ciframento e está intimamente relacionado ao tamanho das chaves do sistema criptográfico. A chamada aritmética de alto nível, por sua vez, consiste das operações no grupo de pontos da curva elíptica. A operação básica deste grupo é a soma elíptica $+$, da qual deriva a multiplicação escalar.

A soma elíptica, e portanto a multiplicação escalar, consiste de um conjunto de operações de baixo nível. Portanto, uma das escolhas mais importantes a serem feitas é a do corpo finito \mathbb{F} que será utilizado. A eficiência de cada operação de alto nível depende da eficiência das operações elementares sobre \mathbb{F} .

Apesar de não ser um requerimento obrigatório, a literatura recomenda basicamente três tipos de corpos finitos para uso em ECCs. O primeiro deles corresponde aos corpos $GF(p)$, onde p é um primo grande; o segundo compreende os corpos de Galois de característica 2 (corpos binários); o terceiro compõem-se dos Corpos de Extensão Ótima

(OEF) (Veja Figura 5.1).

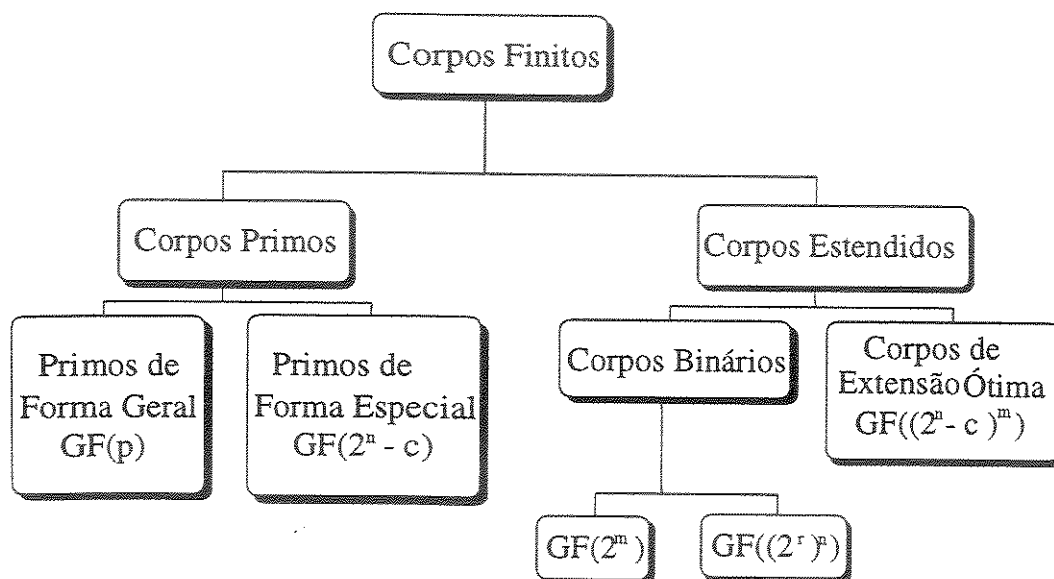


Figura 5.1: Tipos de corpos recomendados para uso em ECCs.

5.2.1 Corpos Primos

Corpos $GF(p)$ de característica p , para p primo grande, são muito populares, uma vez que podem ser eficientemente implementados usando técnicas herdadas de outros sistemas criptográficos baseados em corpos finitos, tais como RSA e DSA. Os elementos em $GF(p)$ podem ser representados de forma simples pelos números inteiros no intervalo $[0, p - 1]$, com a recomendação de que p tenha pelo menos 160 *bits*. Neste caso, as operações sobre o corpo correspondem a operações modulares (módulo p) de precisão múltipla. Algoritmos para tais operações podem ser encontrados em [MvOV97].

Certos primos possuem características que permitem que se utilize técnicas de redução modular muito eficientes. Exemplos destes primos são os chamados *Primos de Mersenne*, os quais tem a forma geral $p = 2^k - 1$. Entretanto, o número de primos Mersenne de tamanho apropriado para uso criptográfico é limitado. Isto tem levado inúmeros autores a proporem generalizações deste tipo de primos.

Crandall, em [Cra92], propõe o uso de primos da forma $p = 2^k \pm c$, onde c é um inteiro pequeno (em comparação com 2^k), com tamanho próximo ao da palavra da arquitetura alvo. Primos desta forma freqüentemente são chamados *Pseudo Primos de Mersenne*.

Em outra direção, Solinas em [Sol99] introduz o conceito de *Primos de Mersenne Generalizados (GM-primos)*, da forma $p = f(2^k)$, onde f é um polinômio de grau pequeno com poucos coeficientes não nulos. O valor de k é um múltiplo do tamanho da palavra da arquitetura alvo.

O uso de GM-primos tem se tornado popular nos últimos anos, devido à adoção destes primos nas curvas recomendadas pelos padrões ANSI [ANS99] e NIST [NIS00].

5.2.2 Corpos Binários

Existem inúmeras representações para os elementos de $GF(2^m)$, cada uma com suas vantagens específicas. A mais popular é a representação por *base polinomial*, também chamada de *base padrão* (Veja Seção 2.2.2). Nela, elementos são representados como polinômios de grau $m - 1$ com coeficientes em $\{0, 1\}$ e as operações sobre polinômios são feitas módulo um polinômio irredutível de grau m . Implementações usando bases padrão podem ser tornadas mais eficientes através da escolha de um polinômio irredutível com reduzido número de coeficientes não nulos, tais como trinômios ou pentanômios. É um fato conhecido que pelo menos um destes pode ser encontrado para qualquer valor de m .

Os elementos de $GF(2^m)$ também podem ser representados por combinações lineares de uma base $\{\beta, \beta^2, \beta^{2^2}, \dots, \beta^{2^{m-1}}\}$, onde $\beta \in GF(2^m)$. Este tipo de base, chamada de *base normal* (Veja Seção 2.2.2), possui características que as tornam ideais para implementações em *hardware*; porém, a experiência tem mostrado que tais propriedades não são transferidas para implementações em *software*, onde bases polinomiais costumam ser mais eficientes.

Uma terceira forma de representação utiliza um conjunto especial de corpos de característica 2, do tipo $GF((2^r)^n)$, onde r é geralmente um valor pequeno. Estes corpos são chamados de corpos compostos e seus elementos podem ser representados como polinômios sobre o corpo $GF(2^r)$. Entretanto, o fato destes corpos possuírem uma estrutura mais rica, pode tornar o ECDLP mais fácil, reduzindo a segurança do sistema resultante.

Gaudry *et al.* em [GHS00], alertam que curvas elípticas baseadas sobre corpos deste tipo possuem algumas vulnerabilidades.

5.2.3 Corpos de Extensão Ótima (OEF)

Um outro tipo de corpo, introduzido em [BP98], corresponde a corpos da forma $GF(p^m)$, onde p pode ser representado na forma $2^n \pm c$, para n, c números inteiros. O tamanho de p deve ser menor (preferencialmente próximo) que o tamanho da palavra da arquitetura alvo, de modo que as instruções de multiplicação nativas do processador possam ser usadas. Além disso, m é escolhido tal que o polinômio irredutível $P(x) = x^m - w$ exista, sendo w um elemento qualquer do corpo $GF(p)$.

É vantajoso escolher valores pequenos para c e w (normalmente 1,2 ou 3), de tal modo que algumas multiplicações possam ser substituídas por somas. Um fator que contribui para tornar a aritmética OEF eficiente é o fato de reduções modulares poderem ser feitas rapidamente através do uso da equivalência $2^n \equiv c \pmod{p}$. Esta mesma idéia pode ser usada para agilizar as operações módulo o polinômio irredutível $P(x) = x^m - w$, como pode ser visto em [BP98, BP01].

5.2.4 Nossas Escolhas

Como motivação para descrevermos passo a passo o desenvolvimento de um sistema criptográfico baseado em curvas elípticas, implementaremos um ECC em software, usando a linguagem de programação C++. Este sistema terá a habilidade de cifrar e decifrar mensagens, bem como gerar assinaturas.

Apresentamos a seguir cada uma das escolhas que fizemos para o corpo base, as curvas elípticas empregadas e os algoritmos de ciframento/deciframento e assinatura digital utilizados.

Corpo Escolhido

Os corpos mais populares, não somente para curvas elípticas mas para criptografia de modo geral, são os chamados corpos primos $GF(p)$. Estes corpos tem sido de interesse

para criptografia desde os primeiros algoritmos de chave pública propostos.

Curvas elípticas fizeram crescer o interesse por uma nova modalidade de corpos, os corpos binário $GF(2^m)$. Tais corpos, surgiram com uma opção direcionada principalmente para implementações em *hardware*. No entanto, tem crescido, nos anos recente, o conjunto de técnicas relacionadas a implementação em *software*.

As propriedades especiais dos corpos $GF(2^m)$ parecem se adequar perfeitamente ao estilo de processamento dos computadores, permitindo que se tome vantagem de características intrínsecas destes dispositivos para agilizar os processos aritméticos envolvidos. Por isso, optamos por estes corpos para o corpo base das curvas de nosso ECC.

Dentre as opções de corpos binários, escolhemos aqueles onde o grau de extensão m seja um número primo, uma vez que, como dissemos na Seção 5.2.2, corpos onde m é composto podem enfraquecer o sistema criptográfico.

A base escolhida foi a polinomial, uma vez que tem sido consenso na literatura de que este tipo de base é a mais adequada para implementações de corpos binários em *software*.

Escolha da Curva e da Representação dos Pontos

Como nosso objetivo é apenas analisar a implementação de um sistema criptográfico baseado em curvas elípticas sob o ponto de vista de eficiência, não nos preocuparemos em desenvolver nenhum método de geração ou verificação da segurança de curvas. As curvas utilizadas serão aquelas recomendadas pelo governo federal americano, através do National Institute Standard Technology em [NIS00]. Este documento apresenta uma série de curvas, baseadas tanto sobre corpos binários como sobre corpos primos.

Para representação dos pontos elípticos, usamos o sistema de coordenadas afim.

Algoritmos Criptográficos Utilizados

O problema do logaritmo discreto no grupo de pontos de uma curva elíptica é análogo ao problema do logaritmo discreto sobre o grupo multiplicativo dos corpos finitos. Assim, é possível adaptar diversos algoritmos criptográficos originalmente propostos sobre corpos finitos ao grupo elíptico. Às vezes, pequenas adaptações devem ser feitas, como é o caso

da versão do DSA para curvas elípticas. Outras vezes, a aplicação pode ser imediata, tal como no algoritmo de ciframento/deciframento de ElGamal.

No nosso caso, para o ciframento/deciframento, optamos pelo método MQV (Menezes-Qu-Vanstone), apresentado em [MQV95]. Este método é uma melhoria do algoritmo de ElGamal, pois ele resolve o problema de mapear textos claros em pontos de uma curva elíptica. Essa é uma tarefa complicada pois o mapeamento deve, univocamente, associar um texto qualquer a um par de coordenadas (x, y) que obedecem à equação da curva elíptica [MvOV97].

O método MQV não mapeia textos claros em pontos, ele na verdade usa um ponto obtido aleatoriamente na curva para mascarar uma mensagem. O processo completo de ciframento/deciframento, usando MQV é apresentado a seguir.

Geração das Chaves

Cada entidade cria uma chave pública e uma chave privada próprias. Para isto cada entidade A deve executar os seguintes passos:

1. Seleciona uma curva elíptica apropriada E sobre um corpo finito F , obtendo um grupo cíclico de ordem n e um ponto gerador P desse grupo.
2. Seleciona um número inteiro aleatório a , $1 \leq a \leq n - 1$, e calcula aP .
3. A chave pública de A é (E, P, aP) ; a chave privada é o inteiro a .

Algoritmos de Ciframento e Deciframento

A entidade B , deseja cifrar e enviar para A a mensagem $m = (m_1, m_2)$, $m_1, m_2 \in F$. Note que m_1 e m_2 devem ser obtidos a partir da mensagem. O método de transformação não influi sobre o comportamento do algoritmo. m_1 e m_2 , por exemplo, podem ser simplesmente as duas metades da sequência de *bits* que compõe m .

(a) *Ciframento*. B deve executar os seguintes passos:

1. Obter a chave pública (E, P, aP) de A ;
2. selecionar um inteiro aleatório k , $1 \leq k \leq n - 1$;
3. calcular kP e $kaP = (\bar{x}, \bar{y})$;

4. enviar o texto cifrado $c = (kP, \bar{x}m_1, \bar{y}m_2)$ para A . Note que o primeiro elemento do texto c é um ponto da curva e os dois seguintes são elementos do corpo finito.

(b) *Deciframento*. Para recuperar o texto $m = (m_1, m_2)$ a partir de c , A deve:

1. Usar a chave privada a para calcular $a(kP) = akP = (\bar{x}, \bar{y})$;
2. calcular os elementos inversos \bar{x}^{-1} e \bar{y}^{-1} em F ;
3. recuperar $m = (m_1, m_2)$ calculando

$$m_1 = (\bar{x}m_1)\bar{x}^{-1}$$

$$m_2 = (\bar{y}m_2)\bar{y}^{-1}.$$

Como podemos ver, MQV dobra o tamanho da mensagem, uma vez que, além de $\bar{x}m_1$ e $\bar{y}m_2$, que em geral tem o mesmo tamanho de m_1 e m_2 , também requer que o ponto “máscara” kP seja fornecido.

Como algoritmo de assinatura digital, escolhemos a versão para curvas elípticas do DSA, conhecido como ECDSA.

O DSA (Digital Signature Algorithm) foi especificado no Padrão de Processamento de Informações do Governo Federal dos Estados Unidos (FIPS), como Digital Signature Standard (DSS). Ele está baseado na intratabilidade do problema do logaritmo discreto (DLP) sobre sub-corpos de ordem prima de \mathbb{Z}_p^* , p primo.

O ECDSA é um método de assinatura digital análogo ao DSA. Ele foi proposto em 1992 por Scott Vanstone [Van92] em resposta a uma requisição do NIST (National Institute of Standards and Technology). Ele foi aceito em 1998 como um padrão ISO (International Standards Organization), em 1999 como padrão ANSI (American National Standard Institute) em ANSI X9.62 e em 2000 como padrão IEEE (Institute of Electrical and Electronics Engineers) em IEEE P1363.

Os procedimentos de geração de chave, geração de assinaturas e verificação de assinaturas no ECDSA são apresentados a seguir:

Geração de chave

Cada entidade A efetua os seguintes passos:

1. Seleciona uma curva elíptica E definida sobre \mathbb{Z}_p . O grupo elíptico sobre E deve ser divisível por um número primo n acima de 160 bits;
2. seleciona um ponto $P \in E(\mathbb{Z}_p)$ de ordem n ;
3. seleciona um inteiro d aleatório no intervalo $[1, n - 1]$;
4. computa $Q = dP$;
5. A chave pública de A é (E, P, n, Q) ; A chave secreta de A é d .

Geração de assinaturas

Para assinar uma mensagem m , A efetua os seguintes passos:

1. Seleciona um número inteiro aleatório k no intervalo $[1, n - 1]$;
2. calcula $kP = (x_1, y_1)$ e $r = x_1 \bmod n$. Se $r = 0$ então volte ao passo 1. (Isto é uma condição de segurança: se $r = 0$ então a equação $s = k^{-1}\{h(m) + dr\} \pmod{n}$ não envolve a chave privada d);
3. compute $k^{-1} \bmod n$;
4. compute $s = k^{-1}\{h(m) + dr\} \bmod n$, onde h é o Algoritmo de Hash Seguro (SHA-1);
5. se $s = 0$ então volte ao passo 1. (Se $s = 0$ então $s^{-1} \bmod n$ não existe; s^{-1} é necessário no passo 2 de verificação de assinatura);
6. A assinatura para uma mensagem m é o par de inteiros (r, s) .

Verificação de assinaturas

Para verificar a assinatura de A (r, s) sobre m , B deve:

1. Obter uma cópia autêntica da chave pública de $A (E, P, n, Q)$;
2. verificar se r e s são inteiros no intervalo $[1, n - 1]$;
3. computar $w = s^{-1} \bmod n$ e $h(m)$;
4. computar $u_1 = h(m)w \bmod n$ e $u_2 = rw \bmod n$;
5. computar $u_1P + u_2Q = (x_0, y_0)$ e $v = x_0 \bmod n$;
6. aceitar a assinatura se e somente se $v = r$;

Capítulo 6

Implementando a Aritmética do Corpo Finito

Como dissemos no capítulo 5, cada operação aritmética no grupo de pontos de uma curva elíptica resume-se a um conjunto de operações no corpo finito sobre o qual esta curva foi definida. Assim, antes de qualquer outra implementação, se faz necessário disponibilizar um conjunto de métodos que permitam realizar as principais operações no corpo finito escolhido.

Dentre as opções apresentadas na literatura, optamos pelos corpos binários do tipo $GF(2^m)$. A implementação das operações fundamentais destes corpos, utilizadas pela aritmética no grupo elíptico, são apresentadas nas próximas seções.

6.1 Aritmética em $GF(2^m)$

Nesta seção descrevemos nossa implementação da aritmética no corpo binário $GF(2^m)$.

6.1.1 Representação

A representação escolhida foi a polinomial. Neste tipo de representação, cada pode ser visto como um polinômio cujos coeficientes admitem apenas os valores 0 ou 1. Assim, cada elemento $A \in GF(2^m)$ será uma sequência binária $(a_{m-1} a_{m-2} a_1 a_0)$.

A seqüência binária correspondente a um elemento A do corpo finito pode ser convenientemente representada em *software* como um *array* de palavras da arquitetura. Este tipo de representação consiste em agrupar em palavras, seqüências de w bits, onde w é o número bits da palavra da arquitetura alvo.

Por convenção, agrupamos os elementos da seqüência binária do elemento A de tal forma que o bit de mais baixa ordem da palavra de menor índice no *array* corresponda a a_0 . De modo geral, se i é o índice da palavra no *array* e j é o bit nesta palavra, correspondente ao termo a_r , então

$$i = \lfloor r/w \rfloor$$

$$j = r \bmod w.$$

A soma de dois termos $a_v, a_r \in GF(2)$ é definida pela seguinte tabela

+	0	1
0	0	1
1	1	0

Tabela 6.1: Tabela Aritmética para Soma em $GF(2)$.

A Tabela 6.1 mostra que a aritmética $GF(2)$ tem o mesmo efeito que a operação de *ou-exclusivo*, normalmente representada pelo símbolo \oplus . Deste modo, a soma de dois elementos de $GF(2^m)$, representados pelos *arrays* A e B , pode ser efetuada calculando-se o ou-exclusivo entre cada par de palavras de A e B que possuem mesmo índice. Esta operação é apresentada no Algoritmo 6.1.

Algoritmo 6.1 Soma em $GF(2^m)$

ENTRADA: A e $B \in GF(2^m)$

SAÍDA: $C = A + B$

para $i := 1$ até $\lfloor m/w \rfloor$ faça

$C \leftarrow A_i \oplus B_i$

Multiplicação

O algoritmo padrão para multiplicação, chamado método de deslocar-e-somar é baseado na observação de que $A \cdot B = a_{m-1}x^{m-1}B + \dots + a_2x^2B + a_1xB + a_0B$. Deste modo, podemos calcular iterativamente o valor de Bx^i e somá-lo ou não a um determinado acumulador C caso a_i seja 1 ou 0, respectivamente.

Toda vez que o termo Bx^i , alcançar grau m , ele deve ser reduzido. Considerando $f(x) = x^m + 1 + \sum_{i=1}^{m-1} f_i x^i$ o polinômio irredutível do corpo, podemos usar a relação

$$x^m \equiv 1 + \sum_{i=1}^{m-1} f_i x^i \pmod{f(x)}.$$

Assim, o termo x^m de Bx^i poderá ser substituído por seu polinômio congruente de grau menor do que m . Segue o algoritmo para multiplicação em $GF(2^m)$, usando o método padrão.

Algoritmo 6.2 Método de deslocar-e-somar (direita para esquerda)

ENTRADA: Dois arrays A e B

SAÍDA: $C = A \cdot B$

se $a_0 = 1$ então $C \leftarrow B$; senão $C \leftarrow 0$

para $i := 0$ até $m - 1$ faça

$B \leftarrow B \cdot x \pmod{f(x)}$

se $a_i = 1$ então

$C \leftarrow C + B$

Note que multiplicar um elemento $B = (b_{m-1}b_{m-2} \dots b_1b_0)$ do corpo por x significa deslocar cada um dos termos b_i de sua seqüência binária uma posição para esquerda. Em *software*, isso se comporta como a operação de deslocamento de *bits* (*shift*). No entanto, deve-se tomar o cuidado de, ao deslocar as palavras do *array* que representa B , transferir o *bit* de mais alta ordem da palavra B_i para o *bit* de mais baixa ordem da palavra B_{i+1} , para cada $i = 0 \dots \lceil m/w \rceil$.

Algoritmo de López e Dahab

A grande quantidade de deslocamentos exigida pelo algoritmo 6.2, não o torna muito atraente para implementações em *software*. López e Dahab, em [LD99], apresentam um método mais rápido, que se beneficia da divisão em palavras dos elementos de $GF(2^m)$ para reduzir o número total de deslocamentos necessários na multiplicação de dois elementos a e b .

Se A_i é a i -ésima palavra do *array* A que armazena um elemento do corpo, então A pode ser escrito como

$$A = \sum_{i=0}^{s-1} A_i x^{iw} \quad (6.1)$$

onde s é o número de palavras da representação do elemento, ou seja, $s = \lceil m/w \rceil$.

· Sendo $A_i = \sum_{j=0}^{w-1} a_{iw+j} x^j$, podemos descrever a multiplicação de dois elementos a e b de $GF(2^m)$ como:

$$AB = \sum_{i=0}^{s-1} \sum_{j=0}^{w-1} a_{iw+j} x^{iw+j} B \quad (6.2)$$

Rearranjando convenientemente a equação 6.2, podemos notar que

$$AB = \sum_{j=0}^{w-1} x^j \sum_{i=0}^{s-1} a_{iw+j} (x^{iw} B).$$

A multiplicação Bx^{iw} consiste em deslocar B por um múltiplo do tamanho de cada palavra (um múltiplo de w). A velocidade do algoritmo de López e Dahab (Algoritmo 6.3) está justamente em explorar o fato de que deslocamentos desta natureza podem ser realizados sem nenhum custo computacional. Basta simplesmente mudar o alinhamento das palavras ao adicionar B ao valor acumulado pelo somatório.

Algoritmo 6.3 Método de López e Dahab

ENTRADA: : A e $B \in GF(2^m)$ SAÍDA: $C = A \cdot B$ $C \leftarrow 0$ **para** $j := 0$ até $w - 1$ **faça** **para** $i := 0$ até $s - 1$ **faça** **se** $a_{iw+j} = 1$ **então** **para** $k := 0$ até $s - 1$ **faça** $C_{i+k} := C_{i+k} \oplus B_k$ **se** $j \neq w - 1$ **então** $B \leftarrow B \cdot x$

Após a execução do algoritmo 6.3, o polinômio $C = AB$ pode possuir grau até $2m - 2$. Assim, será necessário se fazer a redução deste polinômio pelo polinômio irredutível do corpo. O algoritmo para efetuar esta operação é descrito na seção 6.1.2.

6.1.2 Redução

Reduzir um polinômio $A(x)$, onde $\deg(A(x)) \geq m$, consiste em encontrar o único polinômio $B(x)$, de grau menor do que m , cuja diferença entre $A(x)$ e $B(x)$ é um múltiplo do polinômio irredutível que define o corpo. Em outras palavras, precisamos encontrar $B(x)$ tal que

$$B(x) \equiv A(x) \pmod{f(x)}$$

onde $f(x)$ é o polinômio irredutível que define o corpo.

Em nossa implementação, a redução foi usada como passo final dos algoritmos de multiplicação (Algoritmo de López e Dahab) e quadrado. Isso exigiu uma escolha criteriosa do algoritmo para desempenhá-la, uma vez que o seu tempo de processamento poderia influenciar o desempenho de outras operações.

Apresentamos a seguir um método rápido de redução usando trinômios irredutíveis, proposto em [Her00]. Note que este método pode ser facilmente estendido para outros

tipos de polinômio irredutível.

Suponha que $x^m + x^t + 1$, onde $m > t$, é um trinômio irredutível. Assim,

$$x^m + x^t + 1 \equiv 0 \pmod{x^m + x^t + 1}$$

e então, $x^m \equiv x^t + 1 \pmod{x^m + x^t + 1}$ ou, de outra forma:

$$x^{-m}(x^t + 1) \equiv 1 \pmod{x^m + x^t + 1}.$$

Este resultado é importante porque permite que multipliquemos todos os termos de grau maior ou igual a m de um polinômio $A(x)$ por $(x^{-m+t} + x^{-m})$, sem alterar a congruência. O resultado é um polinômio com grau $\deg(A(x)) - m + t$. Ou seja, uma redução de $m - t$ no grau do polinômio $A(x)$

Se tivermos um polinômio de $2m - 2$ termos, podemos implementá-lo com v palavras de w bits, onde $v = \lceil (2m - 2)/w \rceil$. Na redução de $A(x)$, estamos unicamente interessados nas palavras que possuem bits acima de $m - 1$. Estas palavras são $A_s, A_{s+1}, \dots, A_{v-1}$, com $s = \lceil m/w \rceil$. Se m não for múltiplo de w , a palavra A_{s-1} deve ser incluída no conjunto. Entretanto, somente seus $m - sw$ bits mais significativos devem ser considerados.

O processo de redução consiste em multiplicar cada uma das palavras com bits de grau acima de $m - 1$, das mais significativas às menos significativas (ou seja, de A_{v-1} até A_s ou A_{s-1}) por $x^{-m}(x^t + 1)$. Esta ordem de redução agiliza o processo já que os bits mais significativos de $A(x)$ serão os primeiros a serem eliminados, não havendo necessidade de se voltar a uma palavra já reduzida uma vez que ela tenha se tornado 0.

Cada termo de $A(x)$ pode ser escrito como $A_i x^{iw}$, onde A_i é a i -ésima palavra da representação de $A(x)$. Somar um termo do tipo $A_i x^{pw+q}$ a um elemento $B(x) = (B_{s-1}, \dots, B_0)$, equivale a

$$\begin{aligned} B_p &\leftarrow B_p \oplus [A_i \lll q] \\ B_{p+1} &\leftarrow B_{p+1} \oplus [A_i \ggg (w - q)] \end{aligned}$$

Como

$$A_i x^{iw} (x^{t-m} + x^{-m}) = A_i x^{iw+t-m} + A_i x^{iw-m}$$

e sendo $m = rw + z$, $t = gw + h$, com $0 \leq z, h < w$, temos que

$$A_i x^{iw} (x^{t-m} + x^{-m}) = A_i x^{(i+g-r)w+(h-z)} + A_i x^{(i-r-1)w+(w-h)},$$

quando $h > z$. Por outro lado, quando $h \leq z$,

$$A_i x^{iw} (x^{t-m} + x^{-m}) = A_i x^{(i+g-r-1)w+(w+h-z)} + A_i x^{(i-r-1)w+(w-h)}.$$

Como o grau de $A(x)$ é reduzido de $m - t$ em cada iteração, a eficiência do algoritmo de redução depende da diferença entre m e t . Quanto maior for o valor de $m - t$, mais rápido o algoritmo reduzirá um elemento $A(x)$.

6.1.3 Quadrado

Em $GF(2^m)$, quadrados são distributivos com relação a soma (Veja 2.2. Isto significa que, sendo $A = \sum_{i=0}^{m-1} a_i x^i$, $A^2 = \sum_{i=0}^{m-1} a_i x^{2i}$. Portanto, o quadrado de uma seqüência qualquer de *bits* de A possui o dobro de seu tamanho original e pode ser calculado de forma independente das demais seqüências de *bits* de A .

O quadrado de um elemento A pode ser agilizado utilizando-se uma tabela com os quadrados pré-computados de todas as possíveis seqüências de v *bits*. Deste modo, podemos converter cada seqüência de v *bits* de A para sua forma expandida equivalente, de $2v$ *bits*. Uma vez feita a conversão, o passo final será reduzir A^2 pelo polinômio irredutível $f(x)$.

Algoritmo 6.4 Quadrado usando tabela pré-computada

Precompute os quadrados de todas as 2^v possíveis variações de v *bits* e insira estes valores no vetor Mat

ENTRADA: Um elemento $A = (\bar{A}_{r-1}, \dots, \bar{A}_1, \bar{A}_0)$, onde \bar{A}_i possui v *bits*.

SAÍDA: $C = (\bar{C}_{r-1}, \dots, \bar{C}_1, \bar{C}_0)$, onde \bar{C}_i é uma seqüência de $2v$ *bits* e $C = A^2$.

para $i := 0$ at $r - 1$ faça

$$\bar{C}_i \leftarrow Mat[\bar{A}_i]$$

6.1.4 Inversão

Dentre as operações básicas sobre elementos de $GF(2^m)$, a inversão é aquela com o maior preço computacional. Em geral ela é o gargalo da aritmética sobre estes corpos finitos, sendo muitas vezes preferível que as aplicações evitem-na, mesmo ao custo de multiplicações extras. Tal é o caso, por exemplo, dos ECCs que utilizam coordenadas projetivas.

Os dois algoritmos mais eficientes conhecidos para inversão em $GF(2^m)$ são o **Algoritmo Euclidiano Estendido** e o *Almost Inverse Algorithm* (AIA). Ambos possuem certas características em comum e tempos de processamento equiparáveis, com uma ligeira vantagem para o AIA. Em nossa implementação, optamos por desenvolver uma versão modificada do AIA, apresentada em [HHM00] e denominada MAIA. MAIA apresenta um tempo de processamento bastante próximo do AIA original e exige um passo a menos que este último.

Uma pequena descrição do Algoritmo Euclidiano Estendido, do AIA e do MAIA é apresentada a seguir.

Algoritmo Euclidiano Estendido

Usando a representação polinomial, o inverso de um elemento $A \in GF(2^m)$, cujo polinômio irredutível é $f(x)$, é um elemento B , tal que $A(x)B(x) \equiv 1 \pmod{f(x)}$. Isto equivale a dizer que existe um polinômio $M(x)$ tal que $A(x)B(x) + M(x)f(x) = 1$. $B = A^{-1}$ pode ser computados com uma extensão do algoritmo de Euclides para encontrar o máximo divisor comum. Uma descrição em alto nível do algoritmo é apresentada a seguir. Nela, a função $\deg(\cdot)$ representa o grau do argumento, por exemplo, $\deg(f(x))$ representa o grau do polinômio $f(x)$.

Algoritmo 6.1 Algoritmo Euclidiano Estendido

0 - inicialize os polinômios:

$$B \leftarrow 1$$

$$C \leftarrow 0$$

$$F \leftarrow A$$

$$G \leftarrow f$$

1 - se $\deg(F) = 0$ então retorne B 2 - se $\deg(F) < \deg(G)$ então

$$F \leftrightarrow G$$

$$B \leftrightarrow C$$

3 - $j \leftarrow \deg(F) - \deg(G)$

$$4 - F \leftarrow F + x^j G$$

$$B \leftarrow B + x^j C$$

5 - volte a linha 1

Este algoritmo mantém os relacionamentos $F = BA + Mf$ e $G = CA + Nf$ invariantes, onde inicialmente $F = A$, $G = f(x)$, $B = 1$ e $C = 0$. Note que para implementação do algoritmo, não há nenhuma necessidade de se armazenar os valores de M, N . Em cada iteração, o grau do polinômio de maior grau é decrementado pela adição de um múltiplo apropriado do polinômio de grau menor. A invariância dos relacionamentos é preservada pela aplicação da mesma operação, tanto para B como para C . Estas operações são repetidas até que F ou G seja 1.

Almost Inverse Algorithm

Em [SOOS95], o *Almost Inverse Algorithm* (AIA) é proposto para calcular inversos em $GF(2^m)$, usando base polinomial. Ele é dividido em duas etapas. Na primeira etapa, encontra-se um polinômio B e um inteiro k , satisfazendo $BA + Mf = x^k$. A partir deste

resultado, o inverso pode então ser computado através de um segundo passo, onde B é dividido por x^k . O pseudo-código da primeira etapa do AIA é apresentado a seguir.

Algoritmo 6.2 Almost Inverse Algorithm - AIA

ENTRADA: Polinômio A que se deseja inverter

SAPolinômio $B = A^{-1}$

0 - inicialize os polinômios:

$$B \leftarrow 1$$

$$C \leftarrow 0$$

$$F \leftarrow A$$

$$G \leftarrow f$$

1 - enquanto F não contiver termo constante faça

$$F \leftarrow F/x$$

$$C \leftarrow Cx$$

$$k \leftarrow k + 1$$

2 - se $F = 1$ então

retorne B

3 - se $\deg(F) < \deg(G)$ então

$$F \leftrightarrow G$$

$$B \leftrightarrow C$$

4 - $F \leftarrow F + G$

$$B \leftarrow B + C$$

5 - volte a linha 1

AIA mantém os relacionamentos $x^k F = BA + Mf$ e $x^k G = CA + Nf$ invariantes, onde os valores iniciais são $F = A$, $G = f(x)$, $B = 1$ e $C = 0$.

Note que o primeiro passo do AIA é muito parecido com o algoritmo Euclidiano Estendido. Entretanto, ele tem algumas diferenças significativas. A principal delas é a ordem de cancelamento das potências de x . Enquanto o Algoritmo Euclidiano cancela

das potências mais altas para as mais baixas, AIA age no sentido inverso. Entretanto, quando $\deg(F)=\deg(G)$, também ocorrem cancelamentos nos *bits* de mais alta ordem. Conseqüentemente, espera-se que AIA requiera menos iterações que o Algoritmo Euclidiano Estendido.

O passo de redução do AIA pode ser desempenhado da seguinte forma. Sendo $f(x) = x^m + \sum_{i=0}^{m-1} f_i x^i$ o polinômio irredutível que define o corpo e $s = \min\{i \geq 1 \mid f_i = 1\}$, suponha B' o polinômio formado pelos s *bits* de mais baixa ordem de B . Então $B \equiv B'f + B \pmod{f(x)}$ e além disso, $B'f + B$ é divisível por x^s . Se $B'' = (B'f + B)/x^s$ então $B'' = Bx^{-s} \pmod{f(x)}$. Este processo pode ser repetido até que $Bx^{-k} \pmod{f(x)}$ seja obtido. É claro que quanto maior for o valor de s , menor será o número de iterações do algoritmo. Em especial, se $s \geq w$, onde w é o tamanho em *bits* da palavra destino da implementação, o algoritmo tem seu melhor desempenho.

Modified Almost Inverse Algorithm

Utilizamos em nossa implementação uma versão modificada do AIA. Este algoritmo é chamado de MAIA (*Modified Almost Inverse Algorithm*). A diferença significativa entre AIA e MAIA é a ausencia neste último do passo de divisão por x^k . Isto permite uma ligeira melhoria no tempo final de processamento e reduz a quantidade de código necessária para sua implementação.

MAIA usa as relações invariantes $F = AB + Mf$ e $G = AC + Nf$, com os mesmos valores iniciais do AIA. O grau de F ou G é decrementado da direita para a esquerda através de divisões por x^{-1} e somas $F + G$, do mesmo modo que no AIA original. Entretanto, a invariância é preservada mantendo-se a igualdade de cada uma das equações, ou seja, toda vez que F for multiplicado por x^{-1} , $AB + Mf$ também deverá ser multiplicado. Na prática isto significa que toda vez que F for multiplicado por x^{-1} , a mesma operação deve ser aplicada sobre B .

Apresentamos no Algoritmo 6.1.4 uma versão em pseudo-código do MAIA.

Algoritmo 6.3 Modified Almost Inverse Algorithm - MAIAENTRADA: Polinômio A que se deseja inverterSA Polinômio $B = A^{-1}$

0 - inicialize os polinômios:

$$B \leftarrow 1$$

$$C \leftarrow 0$$

$$F \leftarrow A$$

$$G \leftarrow f$$

1 - enquanto F não contiver termo constante faça

$$F \leftarrow F/x$$

$$B \leftarrow B/x$$

2 - se $F = 1$ entãoretorne B 3 - se $\deg(F) < \deg(G)$ então

$$F \leftrightarrow G$$

$$B \leftrightarrow C$$

4 - $F \leftarrow F + G$

$$B \leftarrow B + C$$

5 - volte a linha 1

A divisão de B por x , no passo 1, reduzirá todos os termos em x de um grau. Isto é equivalente ao deslocamento dos *bits* de B uma posição para a direita. Porém, se B contém um termo constante, a divisão por x irá resultar em um termo x^{-1} . Tendo o polinômio irredutível $f(x)$ a forma

$$x^m + 1 + \sum_{i=1}^{m-1} f_i x^i,$$

este problema pode ser resolvido usando a seguinte congruência

$$x^{-1} \equiv x^{m-1} + \sum_{i=1}^{m-1} f_i x^{i-1} \pmod{f(x)}.$$

Assim, quando B contiver um termo constante, deveremos somar seu polinômio congruente com grau menor do que m , após o deslocamento para a direita.

Capítulo 7

Algoritmos para Multiplicação Escalar

Os protocolos de chave pública sobre curvas elípticas estão baseados na multiplicação escalar, que consiste na repetição da soma elíptica de um ponto P , $k - 1$ vezes, ou seja,

$$\underbrace{P + P + \dots + P}_{k \text{ vezes } P}$$

cujo resultado é denotado por kP .

Para propósitos criptográficos, k deve ser um valor grande o suficiente para que sua obtenção não seja possível por uma simples verificação de todos os seus possíveis valores. Consequentemente, a computação trivial de kP , pelo cálculo de $k - 1$ somas, também é um processo inviável. Felizmente, é possível implementar métodos eficientes para calcular kP . A literatura oferece uma grande variedade destes algoritmos.

Os métodos que apresentaremos nas próximas seções, são algoritmos genéricos de multiplicação escalar, podendo ser empregados para qualquer tipo de curva elíptica. Todos eles aproveitam-se da representação do multiplicador $k \in \mathbb{Z}$ em uma determinada base b , de modo a reduzir a complexidade do cálculo de kP para algo próximo a $\log k$.

Doravante, usaremos a notação $[k_{l-1} \dots k_1 k_0]_b$ para representar k na base b , sendo

$k_{l-1} \neq 0$. Essa representação é única e significa que

$$k = \sum_{i=0}^{l-1} k_i b^i, \quad k_i \in \{0, 1, b-1\}.$$

7.1 Algoritmo Binário

O método binário, também conhecido como método de duplicar e somar, é uma versão adaptada de um algoritmo clássico, usado para efetuar exponenciação inteira.

O método abaixo varre os *bits* de k da esquerda para a direita:

Algoritmo 7.1 Método Binário Esquerda-Direita

ENTRADA: P e $k = [k_{l-1} \dots k_1 k_0]_2$

SAÍDA: kP

se $k = 0$ então

 retorne O

$R \leftarrow P$

para $i := l - 2$ decrescendo até 0 faça

$R \leftarrow 2R$

 se $k_i = 1$ então

$R \leftarrow R + P$

retorne R

O algoritmo 7.1 pode ser modificado para processar os *bits* de k na ordem reversa. Esta modificação exige uma variável a mais que a primeira para armazenar os valores de $2^i P$, como pode ser visto a seguir.

Algoritmo 7.2 Método Binário Direita-EsquerdaENTRADA: P e $k = [k_{l-1} \dots k_1 k_0]_2$ SAÍDA: kP

```

se  $k = 0$  então
  retorne  $\mathcal{O}$ 
 $R \leftarrow \mathcal{O}$ 
 $S \leftarrow P$ 
para  $i := 0$  até  $l - 2$  faça
  se  $k_i = 1$  então
     $R \leftarrow R + S$ 
     $S \leftarrow 2S$ 
retorne  $R + S$ 

```

O algoritmo binário requer no pior caso $\lceil \log k \rceil$ somas elípticas e $\lceil \log k \rceil - 1$ duplicações. No caso médio são necessárias $\lceil \frac{\log k}{2} \rceil$ somas (em média, espera-se que metade dos *bits* de k sejam 1) e $\lceil \log k \rceil - 1$ duplicações.

7.2 Método *t*-ário

O método binário possui uma generalização simples que consiste em usar a representação de k em uma base maior do que 2. O seguinte algoritmo calcula kP , usando o chamado método *t*-ário.

Algoritmo 7.3 Método *t*-árioENTRADA: P e $k = [k_{l-1} \dots k_1 k_0]_t$ SAÍDA: kP se $k = 0$ então retorne \mathcal{O} $R \leftarrow k_{l-1}P$ para $i := l - 2$ até 0 faça $R \leftarrow mR$ se $k_i \neq 0$ então $R \leftarrow R + k_i P$ retorne R

O algoritmo acima torna-se mais eficiente quando $t = 2^r$, uma vez que $2^r R$ pode ser obtido por apenas r duplicações.

A grande vantagem do *t*-ário sobre o método Binário está na possibilidade de se reduzir o número de somas elípticas necessárias, através de pré-computação de $k_i P$ para todas as $t - 1$ possibilidades não nulas de k_i . Portanto, o número máximo de operações executadas pelo algoritmo 7.3, com $m = 2^r$, seria $\lceil \log k \rceil - 1$ duplicações e $\lceil \frac{\log k}{r} \rceil$ somas na fase principal, mais $t - 2$ adições, na fase de pré-computação.

No método *t*-ário, usando pré-computação, a escolha de t deve levar em consideração o compromisso entre velocidade e espaço de armazenamento. Isto porque, a medida que t cresce, cresce também o número de valores a serem pré-processados, bem como o espaço necessário para armazenar estes valores. No entanto, quanto mais valores tiverem sido pré-processados, menor será a quantidade de somas necessárias durante a fase de computação. Assim, considerando $t = 2^r$ e sendo o número total de operações do método dado pela função

$$T_{op}(r) = (\log k - 1) + \left(\frac{\log k}{r} \right) + (2^r - 2),$$

$T_{op}(r)$ tem um mínimo em

$$2^r \ln 2 - \frac{\log k}{r^2} = 0$$

Para valores típicos de k , onde $\log k$ está em torno de 170 (*bits*), o valor ótimo para r está próximo de 4, o que significa que 16 pontos devem ser pré-processados e armazenados. Esta quantidade pode ser muito alta para algumas aplicações com recursos limitados. Por isto, algumas modificações deste algoritmo tem sido propostas na literatura com o objetivo de reduzir tal necessidade de espaço. Uma delas é pré-computar apenas os valores $k_i P$ em que k_i for um número ímpar e usar a relação $k_i = 2^v \tau_i$, com $0 < v < r$ e τ_i ímpar, se k_i for par. Este método reduz pela metade o número de pontos armazenados. O pseudo-código para esta modificação do *t*-ário é apresentado no Algoritmo 7.4.

Algoritmo 7.4 Método t -ário ModificadoENTRADA: P e $k = [k_{l-1} \dots k_1 k_0]_t$, $t = 2^r$ SAÍDA: kP se $k = 0$ então retorne \mathcal{O}

{Pré-computação}

 $P_1 \leftarrow P$ $P_2 \leftarrow 2P$ para $i := 1$ até $2^{r-1} - 1$ faça $P_{2i+1} \leftarrow P_{2i-1} + P_2$ $R \leftarrow \mathcal{O}$

{Laço principal}

para $i := l - 1$ até 0 faça se k_i é ímpar então $R \leftarrow 2^i R + P_{k_i}$

senão

 encontre o máximo v tal que $k_i = 2^v \tau_i$ e $0 < v < r$ $R \leftarrow 2^{r-v} R + P_{\tau_i}$ $R \leftarrow 2^v R$ retorne R

Podemos ver facilmente que o algoritmo 7.4 requer 1 duplicação e $2^{r-1} - 1$ somas elípticas na fase de pré-computação e no máximo $\lceil \log k \rceil - 1$ duplicações e $\lceil \frac{\log k}{r} \rceil$ somas, durante a execução.

7.3 Método da Janela Deslizante

Um outro método que, a exemplo do t -ário, processa um bloco de *bits* de cada vez é o Método da Janela Deslizante. Neste algoritmo, o bloco ou janela de *bits* possui tamanho variável. Estas janelas podem não ser adjacentes e qualquer seqüência de zeros após uma janela é totalmente consumida antes que uma nova janela comece. Deste modo, uma janela é uma seqüência de *bits* $(k_i, k_{i-1}, \dots, k_{i-j+1})$, tal que $k_i = k_{i-j+1} = 1$ e $0 < j \leq r$, onde r é o tamanho máximo da janela de *bits*.

Algoritmo 7.5 Método da Janela DeslizanteENTRADA: P e $k = [k_{l-1} \dots k_1 k_0]_2$ SAÍDA: kP se $k = 0$ então retorne \mathcal{O}

{Pré-computação}

 $P_1 \leftarrow P$ $P_2 \leftarrow 2P$ para $i := 1$ até $2^{r-1} - 1$ faça $P_{2i+1} \leftarrow P_{2i-1} + P_2$ $R \leftarrow \mathcal{O}$ $i \leftarrow l - 1$

{Laço principal}

enquanto $i \geq 0$ faça se $k_i = 0$ então $R \leftarrow 2R$ $i \leftarrow i - 1$

senão

 encontre o maior $s_i = (k_i, k_{i-1}, \dots, k_{i-j+1})$, tal que

$$k_i = k_{i-j+1} = 1 \text{ e } j < r$$

 $R \leftarrow 2^j R + P_{s_i}$ $i \leftarrow i - j$ retorne R

É fácil ver que, sem considerar a pré-computação, o número de duplicações é sempre $\lceil \log k \rceil - 1$. Entretanto, o número de somas pode variar de acordo com o valor de k . K.-

Y. Lam e L.C.K.Hui em [LH94] estimaram o número total de somas em $\lceil \log k \rceil / (r + 1)$. O argumento deles é que, se $W_{r,n}$ é o número total de somas realizadas durante o laço principal do algoritmo 7.5, onde r é o tamanho da janela e n o número de *bits* da cadeia a ser processada, então o valor médio de $W_{r,n}$ pode ser calculado pela seguinte fórmula recursiva.

$$W_{r,n} = \begin{cases} 1 - \frac{1}{2^r} & \text{se } 0 \leq n \leq r, \\ \frac{1}{2} + \frac{1}{2}W_{r,n-1} + \frac{1}{2}W_{r,n-r} & \text{se } r < n. \end{cases}$$

Isto é justificado pelo fato de que, se $n \leq r$, nenhuma soma elíptica é necessária quando todos os *bits* de k são zero, o que ocorre com probabilidade $\frac{1}{2^r}$. Nos demais casos, apenas uma soma deve ser efetuada. Assim, o número esperado de somas, se $0 \leq n \leq r$, será

$$W_{r,n} = \frac{1}{2^r} \times 0 + \left(1 - \frac{1}{2^r}\right) \times 1.$$

Para $r < n$, podemos avaliar o primeiro *bit* da representação. Se ele for 0, o que ocorre com probabilidade $1/2$, o número total de somas será $W_{r,n-1}$. Por outro lado, caso este *bit* seja 1, uma multiplicação será necessária para processar o maior prefixo ímpar dentro da janela, duplicações removerão a cadeia de zeros restantes e $W_{r,n-r}$ somas completarão a execução do algoritmo. Assim, quando $r < n$, o número médio de somas elípticas é

$$W_{r,n} = \frac{1}{2}W_{r,n-1} + \frac{1}{2}(1 + W_{r,n-r}).$$

K.-Y. Lam e L. C. K. Hui provam que

$$\lim_{n \rightarrow \infty} \frac{W_{r,n}}{n} = \frac{1}{r+1},$$

o que significa que o número total de somas elípticas, para valores grandes de n , está em torno de $\frac{\lceil \log k \rceil}{r+1}$. Assim, incluindo a pré-computação, o algoritmo 7.5 realiza, em média, $\frac{\lceil \log k \rceil}{r+1} + 2^{r-1} - 1$ somas e $\lceil \log k \rceil - 1$ duplicações.

7.4 Representação por Dígito Sinalizado

Soma e subtração elíptica têm o mesmo preço computacional. Esta característica permite a utilização de um método baseado em cadeias de soma e subtração, o qual pode reduzir o número total de operações elípticas necessárias em uma multiplicação escalar.

Considere a representação

$$k = \sum_{i=0}^{l-1} k_i 2^i,$$

onde $k_i \in \{-1, 0, 1\}$. Chamamos esta representação de representação por *dígito sinalizado*. Claramente esta definição inclui a representação binária como um caso particular de representação sinalizada.

Todos os inteiros k , $0 \leq k \leq 2^l - 1$, bem como seus correspondentes negativos, podem ser representados usando uma representação sinalizada de l bits. Entretanto, 3^l configurações de bits são possíveis, o que mostra que esta representação possui redundância. Por exemplo, o inteiro 3 pode ser representado como $(011)_2$ ou $(10\bar{1})$, onde $\bar{1} = -1$.

A representação por dígito sinalizado é dita *esparsa* se ela não possui dois dígitos não nulos consecutivos, ou seja, $k_i \times k_{i-1} = 0$ para todo $i > 0$. Esta notação também pode ser chamada de *Forma Não Adjacente* (NAF). O Lema 7.4.1 descreve algumas propriedades importantes da NAF.

Lema 7.4.1 *Todo inteiro k tem uma única NAF. Além disto, sua NAF tem o menor número de dígitos não-zero dentre todas as representações por dígito sinalizado deste inteiro k e é, no máximo, um dígito mais longa do que a mais curta representação por dígito sinalizado de k .*

NAFs em geral têm menos dígitos não-zero que a representação binária. Morain e Olivos em [MO90] mostraram que em uma NAF de comprimento l , espera-se que apenas $l/3$ dígitos sejam não-zeros. Esta característica é bastante interessante no contexto da multiplicação escalar, onde reduzir o número de bits não nulos significa reduzir o número total de somas (ou subtrações) elípticas. Reitwiesner, em [Rei60], descreve um algoritmo para computar a forma não adjacente de um inteiro k ; este algoritmo foi combinado com a multiplicação escalar, a fim de produzir o Método Binário Sinalizado, cujo pseudo-código é exibido no Algoritmo 7.6.

Algoritmo 7.6 Método Binário Sinalizado

ENTRADA: P e $k = [k_{l+1}k_l k_{l-1} \dots k_1 k_0]_2$, $k_{l+1} = k_l = 0$ SAÍDA: kP SAÍDA kP se $k = 0$ então retorne \mathcal{O} $c \leftarrow 0$ $R \leftarrow \mathcal{O}$ $S \leftarrow P$ para $i := 0$ até l faça $b \leftarrow k_i + c$ se $(b + k_{i+1}) > 1$ então $c \leftarrow 1$

senão

 $c \leftarrow 0$ $b \leftarrow b - 2c$ se $b \neq 0$ então $R \leftarrow R + bS$ $S \leftarrow 2S$ retorne R

Capítulo 8

Resultados

Como resultado final de nosso trabalho de pesquisa sobre curvas elípticas, procuramos desenvolver uma biblioteca de rotinas para aritmética no corpo finito, bem como rotinas para soma, duplicação e multiplicação escalar. Além disso, para que estas rotinas fossem testadas tanto em corretude como em desempenho, implementamos um algoritmo de ciframento/deciframento e um algoritmo para assinatura digital.

Algumas das experiências adquiridas nesta implementação, bem como os tempos obtidos para cada um dos algoritmos implementados são descritos neste capítulo.

8.1 Objetivo

O objetivo de implementar os algoritmos para criptografia em ECCs foi adquirir conhecimento prático sobre possíveis dificuldades de implementação que uma análise puramente teórica pudesse omitir. Além disso, disponibilizar uma biblioteca com rotinas básicas para operações aritméticas tanto no corpo como na curva elíptica pode ser importante como ponto de partida para futuros aprimoramentos e expansões.

A linguagem escolhida foi C++, pela nossa experiência anterior e por esta linguagem ser muito popular e estar presente em diversas plataformas.

8.2 Componentes de um ECC

Um sistema criptográfico baseado em curvas elípticas pode ser dividido em três camadas de operações. Cada camada utiliza um conjunto de sub-operações, disponibilizadas pela camada imediatamente anterior. Assim, a camada de aritmética na curva elíptica utiliza-se das operações de soma, quadrado, multiplicação e inversão da camada de aritmética no corpo finito, localizada abaixo desta, para realizar duplicações, somas e multiplicações escalares, indispensáveis pelos algoritmos de ciframento/deciframento e assinatura digital da camada de algoritmos criptográficos, localizada imediatamente acima desta.

Usando esta divisão, podemos separar as aritméticas envolvidas, permitindo flexibilidade do sistema a mudanças no corpo finito, no grupo elíptico ou nos algoritmos criptográficos a serem utilizados.

A seguir descrevemos nossa implementação para cada uma das camadas.

8.3 Aritmética no Corpo Finito

Mesmo restringindo nossa implementação aos corpos binários, nosso escopo ainda inclui uma ampla variedade de corpos finitos dentro deste conjunto, cuja escolha mais adequada irá depender de aspectos de segurança e eficiência. Estes aspectos em geral são relativos ao sistema a ser implementado, podendo variar de acordo com condições de *hardware*, natureza das informações, etc.

Independente do que seja eficiente para o sistema em questão, a escolha antecipada de um corpo finito permite que se explore diversas características que lhe são peculiares, o que pode melhorar bastante o desempenho do código aritmético implementado. Por outro lado, os constantes avanços matemáticos e tecnológicos exigem que as fronteiras de segurança dos sistemas criptográficos tenham de ser constantemente refeitas. Portanto, sistemas de média e longa duração devem ser razoavelmente flexíveis, a fim de permanecerem funcionais. Como parte importante de um sistema criptográfico, a aritmética no corpo finito deve refletir esta flexibilidade e dispor de algum mecanismo que permita mudar o corpo finito de forma simples e transparente para o resto do sistema.

O mecanismo que adotamos foi desenvolver um gerador de código para a aritmética

em $GF(2^m)$. Este programa é capaz de gerar, a partir da especificação do corpo finito, toda a aritmética de tal corpo, podendo se aproveitar do prévio conhecimento do corpo para fazer algumas otimizações no código gerado.

Um bom exemplo da vantagem de se gerar código para a aritmética $GF(2^m)$ é o algoritmo de redução apresentado na Seção 6.1.2. Este algoritmo é composto de uma série de deslocamentos que dependem unicamente do polinômio irredutível que determina o corpo. Assim, uma vez conhecido tal polinômio, estes valores podem ser previamente computados e tornados constantes no código gerado.

Outra vantagem que observamos com a utilização do gerador foi facilitar o gerenciamento do código quando certos recursos de programação, tais como laços, vetores e chamadas a sub-rotinas são omitidos em detrimento de eficiência. Em nossa implementação, muitos dos laços foram omitidos quando se conhecia previamente o número total de iterações. Além disso, os *bits* de cada elemento do corpo foram armazenados em variáveis separadas ao invés de um vetor, evitando *overheads* com cálculo de índices. A eliminação destes recursos no código aritmético pode tornar sua implementação manual bastante trabalhosa e propensa a erros. Felizmente, o processo de geração deste código pode ser facilmente automatizado.

Cabe aqui, no entanto, fazer uma ressalva sobre a política que adotamos de eliminação de laços e chamadas a sub-rotinas, em busca de uma aritmética mais rápida. Este processo não pode ser feito de forma indiscriminada, uma vez que pode acarretar sérios efeitos colaterais. O primeiro é o aumento do tamanho do código, que normalmente é indesejável, principalmente falando de curvas elípticas cujo principal atrativo é justamente a menor requisição de espaço de armazenamento. Outro efeito é uma inversão do efeito esperado, na qual eliminar laço incorre em perda de desempenho. Isto foi observado no algoritmo de multiplicação (Algoritmo 6.2). Este algoritmo, na sua forma padrão, consiste de dois laços aninhados. O laço externo percorre cada uma das palavras que representam o elemento e o interno percorre os *bits* de cada uma destas palavras. A eliminação total de laço neste algoritmo duplicou o seu tempo de execução em comparação com a versão onde apenas o laço externo foi expandido. Embora não tenhamos esclarecido totalmente o porquê deste fenômeno, acreditamos que este aumento drástico no tempo de execução seja reflexo de

Polinômio irreduzível	Multiplicação de López	Multiplicação Standard	Quadrado	Inversão
$x^{191} + x^9 + 1$	5.5	13	0.4	48.1
$x^{191} + x^{182} + 1$	6.0	14.5	0.7	45.1
$x^{167} + x^6 + 1$	5.5	11	0.35	43
$x^{167} + x^{161} + 1$	5.7	12.3	0.6	41.1
$x^{163} + x^7 + x^6 + x^3 + 1$	5.5	12.7	0.3	42.0

Tabela 8.1: Tempos obtidos (em microssegundos) para aritmética sobre diversos corpos binários, em máquina Pentium III 450MHz.

um aumento na taxa de *misses* do *cache* de instrução da máquina, decorrente da perda de localidade de código que seria proporcionada pelo laço. De qualquer forma, este fenômeno nos advertiu quanto a possíveis perdas em desempenho quando o código é excessivamente grande.

Os valores obtidos a partir do código gerado para alguns corpos da aritmética $GF(2^m)$ são apresentados na Tabela 8.1.

Como podemos observar, o cálculo do quadrado é uma operação bastante barata nos corpos $GF(2^m)$. A grande diferenciação nos tempos desta operação entre corpos com o mesmo número de *bits*, como também ocorre no algoritmo de multiplicação de López, deve-se ao método de redução adotado. Como dissemos no Capítulo 6, a forma de redução utilizada é mais lenta em corpos cujo *bit* correspondente ao termo de maior grau do polinômio irreduzível do corpo é armazenado na mesma palavra que um ou mais dos *bits* correspondentes aos termos de menor grau do polinômio. Isto acontece, por exemplo, nos corpos definidos pelos polinômios $x^{191} + x^{182} + 1$ e $x^{167} + x^{161} + 1$, quando implementados em máquinas de 32 *bits*, como foi o caso.

Na multiplicação de López, podemos ver que mesmo nos casos onde a redução é mais lenta, o algoritmo ainda é cerca de 2,5 vezes mais rápido que o algoritmo padrão.

Para a inversão, o algoritmo implementado foi o MAIA, apresentado na Seção 6.1.4. Para agilizar o processo, utilizamos a técnica proposta por Schroepel em [SOOS95], onde a troca entre os valores dos polinômios: $F \leftrightarrow G$ e $B \leftrightarrow C$ é substituída por um desvio para uma cópia do código na qual os nomes das variáveis estão trocados. Deste modo, um sim-

ples *goto* substitui uma série de trocas de palavras. Mesmo com essa otimização, em todos os casos a inversão é a operação mais custosa, tendo se mostrado ser aproximadamente três vezes mais lenta que a multiplicação padrão.

8.4 Aritmética na Curva Elíptica

Como apresentamos no Capítulo 3, tanto a soma como a duplicação requerem uma operação de “divisão” no corpo base, a qual normalmente é desempenhada por uma inversão e em seguida uma multiplicação. Por exemplo, a duplicação de um ponto $P = (x, y)$, sobre uma curva não-supersingular é feita da seguinte maneira:

$$\begin{aligned}\lambda &= x + \frac{y}{x} \\ x_3 &= \lambda^2 + \lambda + a \\ y_3 &= x^2 + (\lambda + 1)x_3.\end{aligned}$$

A divisão y/x , será implementada como yx^{-1} .

Uma outra forma de se realizar y/x é adaptando o próprio algoritmo de inversão para realizar diretamente a divisão de y por x . Isso será discutido a seguir.

Generalizando o processo de computação utilizado pelo MAIA, podemos dizer que ele age sobre variáveis cujos valores se relacionam do seguinte modo

$$\begin{aligned}RB + Mf &= F \\ RC + Nf &= G\end{aligned}$$

onde F é inicialmente o polinômio que se deseja inverter e G é inicializado com o polinômio irredutível. Além disso, R, M, N não são armazenados e nem participam de nenhuma computação dentro do algoritmo. No final, o método devolve um valor W tal que $RW + Zf = 1$, ou seja, $RW \equiv 1 \pmod{f}$. No MAIA, o valor suposto para R será o mesmo de F , o que obriga $B = 1$ para que as relações sejam inicialmente verdadeiras. Assim, se F for inicializado com o elemento x , $xW \equiv 1 \pmod{f}$, que resulta em $W = x^{-1}$. No entanto, para o processo generalizado, qualquer outro par de valores para R e B que tornem as relações verdadeiras é permitido. Assim, suponha $R = xy^{-1}$, então $B = y$ para $F = x$. No final do algoritmo, W se relacionará com os demais valores por

$$xy^{-1}W \equiv 1 \pmod{f} \Rightarrow W \equiv yx^{-1} \pmod{f}.$$

Polinômio irreduzível	Binário (Esq-Dir)	Binário (Dir-Esq)	t -ário	Janela Deslizante	NAF
$x^{191} + x^9 + 1$	13.2	13.3	11	10	10.4
$x^{191} + x^{182} + 1$	14	14.2	12	10.8	11
$x^{167} + x^6 + 1$	12.4	12.4	10.5	10	10.4
$x^{167} + x^{161} + 1$	11.8	11.9	9.4	8.9	9.3
$x^{163} + x^7 + x^6 + x^3 + 1$	11.6	11.6	9.3	8.8	9.1

Tabela 8.2: Tempos obtidos (em milissegundos) para aritmética em curvas elípticas definidas sobre diversos corpos binários, usando máquina Pentium III 450MHz.

Deste modo, tudo o que precisamos fazer para adaptar MAIA para “dividir” y por x é inicializar F com x e B com y . Isso permite que y/x tenha o preço de uma inversão, uma vez que F e G , que efetivamente determinam o tempo do algoritmo, são os mesmos que no MAIA convencional.

Apesar de termos utilizado apenas curvas definidas sobre corpos de característica 2, implementamos aritmética também para curvas definidas sobre corpos de característica prima.

Os tempos para as operações de adição, duplicação e multiplicação escalar são apresentadas na Tabela 8.2, para curvas definidas sobre corpos binários.

Como o objetivo foi apenas medir o desempenho das operações, escolhemos curvas e pontos aleatórios, sem levar em consideração aspectos de segurança. Para cada corpo, uma curva e dois pontos foram aleatoriamente determinados. Estes pontos foram utilizados na operação de soma e o primeiro deles nas operações de duplicação e multiplicação escalar. No caso da multiplicação escalar, 7000 multiplicadores, aleatoriamente determinados, também foram usados. O tempo final desta operação foi a média das 7000 multiplicações escalares realizadas. Além disso, 10 medições completas foram realizadas com os tempos de todas as operações elípticas, cada uma com uma curva e dois pontos diferentes. Os valores da Tabelas 8.2 são a média destas 10 medições.

Pelos tempos obtidos, as versões para o algoritmo de multiplicação escalar Binário esquerda-direita e direita-esquerda, têm tempos aproximadamente equivalentes e servem de base de comparação para os algoritmos que usam técnicas mais sofisticadas.

Em nossa implementação, a multiplicação escalar usando *bit* sinalizado (NAF) levou vantagem em desempenho, comparada ao algoritmo *t*-ário. Entretanto, os tempos obtidos foram um pouco inferiores ao algoritmo da Janela Deslizante. No entanto, essa pequena diferença não desfavorece o NAF, pois este algoritmo não exige nenhuma pré-computação, ao contrário do algoritmo da Janela Deslizante e do *t*-ário, o que é uma característica bastante atraente para sistemas com quantidade de memória limitada.

Tanto no algoritmo da Janela Deslizante, quanto no *t*-ário, usamos o tamanho de janela 4. Segundo as fórmulas para o número de operações elípticas, apresentadas no capítulo 7, este é o tamanho de janela ótimo para multiplicadores cujo número de *bits* pertence ao intervalo entre 96 e 132, no caso do *t*-ário, e 80 e 241, considerando o método da Janela Deslizante, como foi o caso medido.

Capítulo 9

Conclusões

O sistema criptográfico de chave pública mais conhecido e mais largamente utilizado, o RSA, é capaz de prover quase todos os serviços atualmente requeridos de um sistema criptográfico. No entanto, por que curvas elípticas têm ganho um destaque progressivamente maior nos anos recentes da Criptografia? A razão está intimamente relacionada à expansão dos sistemas embutidos e das comunicações sem fio, onde lentos e limitados dispositivos têm remetido os desenvolvedores a algo pior do que os primórdios da era dos computadores, uma vez que no atual contexto, a antiga escassez de recursos é acompanhada de requisitos fortes de segurança. Em tais dispositivos, fica inviável a utilização de sistemas que exijam alto esforço computacional ou grande espaço de armazenamento, características comuns a algoritmos de chave pública.

Neste cenário, curvas elípticas surgem como o melhor compromisso entre exigência de recursos e segurança. Devido a seu problema subjacente ser considerado de dificuldade superior ao da fatoração de números inteiros (IFP) ou a computação de logaritmos discretos no grupo multiplicativo de um corpo finito (DLP), as chaves utilizadas por um ECC podem ser menores que chaves utilizadas por sistemas tais como RSA ou ElGamal, para níveis similares de segurança. Além disso, o uso de chaves menores também se reflete em um esforço computacional menor.

Uma outra característica interessante de sistemas baseados em curvas elípticas é a forma diferente na qual a aritmética de ECCs está organizada. Ao invés de uma, como

é comum em outros sistemas criptográficos de chave pública, ECCs apresentam duas aritméticas, uma em um corpo finito e outra no grupo de pontos sobre uma curva elíptica. Ambas as aritméticas são independentes o suficiente para garantir uma variação muito grande de possibilidades, o que torna estes sistemas mais flexíveis, uma vez que são maiores as alternativas para se adequar aritméticas de corpo e grupo às características do ambiente alvo.

9.1 Principais Contribuições de Nosso Trabalho

Nosso trabalho teve como resultado a reunião de informações dispersas na literatura sobre sistemas criptográficos baseados em curvas elípticas, principalmente sobre o ponto de vista de algoritmos eficientes para suas operações aritméticas. Buscamos expor e analisar neste documento grande parte do leque de alternativas para implementação, tanto da aritmética no corpo quanto da aritmética no grupo elíptico.

Além disso, disponibilizamos uma implementação completa de um sistema ECC que pode ser utilizado como ponto de partida para implementações de sistemas maiores. Para a aritmética no corpo finito, construímos um programa gerador de código para a aritmética de qualquer corpo finito binário do tipo $GF(2^m)$, usando bases polinomiais. Também são implementados diversos algoritmos para multiplicação escalar, os quais podem ser utilizados por qualquer tipo de curva elíptica. Os algoritmos de ciframento/deciframento MQV e assinatura digital ECDSA completam o sistema com suas funcionalidades fins.

Referências Bibliográficas

- [ANS99] ANSI X9.62: Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA). *American National Standards Institute*, 1999.
- [BP98] Daniel V. Bailey and Christof Paar. Optimal extension fields for fast arithmetic in public-key algorithms. In *Advances in Cryptology - CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.
- [BP01] Daniel V. Bailey and Christof Paar. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. *Journal of Cryptology*, 14(3):153–176, 2001.
- [BV98] Dan Boneh and Ramarathnam Venkatesan. Breaking RSA may not be equivalent to factoring. In *Theory and Application of Cryptographic Techniques*, pages 59–71, 1998.
- [Cra92] R. Crandall. Method and apparatus for public key exchange in a cryptographic system. U.S. Patent # 5,159,632, October 1992.
- [DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.
- [GHS00] P. Gaudry, F. Hess, and N. Smart. Constructive and Destructive Facets of Weil Descent on Elliptic Curves. Technical Report HPL 2000-10, 2000.
- [Her64] I. N. Herstein. *Topics In Algebra*. Wiley Interscience Publishers, New York, 1964.

- [Her00] Júlio César López Hernández. *Implementação Eficiente em Software de Criptosistemas de Curvas Elípticas*. PhD thesis, Universidade Estadual de Campinas - Unicamp, 2000.
- [HHM00] Darrel Hankerson, López Hernández, and Alfred Menezes. Software implementation of elliptic curve cryptography over binary fields. In *Cryptographic Hardware and Embedded Systems*, pages 1–24, 2000.
- [iee98] IEEE. *IEEE P1363 - Standard Specifications for Public Key Cryptography*, February 1998.
- [IT88] T. Itoh and S. Tsujii. “A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ using Normal Bases”. In *Information and Computation*, volume 78, pages 171–177. 1988.
- [Jr92] B.S. Kaliski Jr. RFC 1319: The MD2 Message-Digest Algorithm. *RSA Laboratories*, April 1992.
- [Kob87] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.
- [Kob94] Neal Koblitz. *A Course in Number Theory and Cryptography*. Springer, 1994.
- [LD99] J. Lopez and R. Dahab. “Fast Multiplication on Elliptic Curves over $GF(2^m)$ ”. In *CHES'99*, number 1717 in LCNS. Springer-Verlag, 1999.
- [LH94] K.-Y. Lam and L.C.K. Hui. Efficiency of $ss(1)$ square-and-multiply exponentiation algorithms. In *Electronics Letters*, volume 30, pages 2115–2116, 1994.
- [Mil86] Victor Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology: Proceedings of Crypto '85*, volume 218, pages 417–426, Berlin, 1986. Springer-Verlag.
- [MO90] F. Morain and J. Olivos. Speeding up the computations on an elliptic curve using addition-subtraction chains. *Theoretical Informatics and Applications*, 24(6):531–543, 1990.

- [MQV95] A. Menezes, M. Qu, and S. Vanstone. Key Agreement and the Need for Authentication, November 1995.
- [MvOV97] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.
- [NIS00] NIST FIPS PUB 186-2: Digital Signature Standard (DSS). *National Institute for Standards and Technology*, 2000.
- [Rei60] G. Reitwiesner. Binary arithmetic. In *Advances in Computers*, pages 231–308. 1960.
- [Riv92a] R.L. Rivest. RFC 1320: The MD4 Message-Digest Algorithm. *Network Working Group*, 1992.
- [Riv92b] R.L. Rivest. RFC 1321: The MD5 Message-Digest Algorithm. *Internet Activities Board*, 1992.
- [Sol99] J. Solinas. Generalized Mersenne Numbers. Technical Report CORR-39, 1999.
- [SOOS95] Richard Schroepel, Hilarie Orman, Sean O'Malley, and Oliver Spatscheck. Fast Key Exchange with Elliptic Curve Systems. In *Advances in Cryptology, Proc. Crypto'95*, number 963 in LNCS, pages 43–56. Springer-Verlag, 1995.
- [Sti95] Douglas R. Stinson. *Cryptography - Theory and Practice*. CRC Press, 1995.
- [Van92] S. Vanstone. Responses to nist's proposal, July 1992.