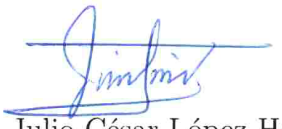
Implementação em Software de Criptografia Baseada em Emparelhamentos para Redes de Sensores Usando o Microcontrolador MSP430

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Conrado Porto Lopes Gouvêa e aprovada pela Banca Examinadora.

Campinas, 12 de maio de 2010.



Prof. Dr. Julio César López Hernández

Instituto de Computação, Universidade Estadual de Campinas (Orientador)

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

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Fabiana Bezerra Müller – CRB8 / 6162

Gouvêa, Conrado Porto Lopes
G745i Implementação em software de criptografia baseada em emparelhamentos para redes de sensores usando o microcontrolador MSP430/Conrado Porto Lopes Gouvêa-- Campinas, [S.P. : s.n.], 2010.

Orientador : Julio César López Hernández.
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1.Criptografia. 2.Redes de computadores - Medidas de segurança. 3.Sistemas de comunicação sem fio. 4.Segurança. I. López Hernández, Julio César. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Software implementation of pairing based cryptography for sensor networks using the MSP430 microcontroller

Palavras-chave em inglês (Keywords): 1.Cryptography. 2.Computer networks – Security measures. 3.Wireless communication systems. 4.Security.

Área de concentração: Teoria da Computação

Titulação: Mestre em Ciência da Computação

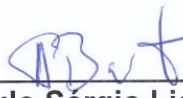
Banca examinadora: Prof. Dr. Julio César López Hernández (IC-UNICAMP)
Prof. Dr. Ricardo Dahab (IC - UNICAMP)
Prof. Dr. Paulo Sérgio Licciardi Messeder Barreto (POLI – USP)

Data da defesa: 12/05/2010

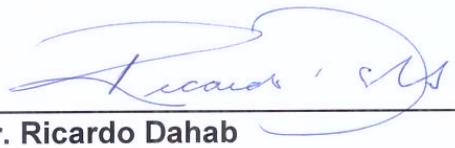
Programa de Pós-Graduação: Mestrado em Ciência da Computação

TERMO DE APROVAÇÃO

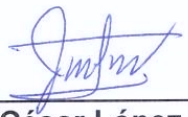
Dissertação Defendida e Aprovada em 12 de maio de 2010, pela Banca examinadora composta pelos Professores Doutores:



Prof. Dr. Paulo Sérgio Licciardi Messeder Barrreto
Departamento Engenharia da Computação / USP



Prof. Dr. Ricardo Dahab
IC / UNICAMP



Prof. Dr. Julio César López Hernández
IC / UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Implementação em Software de Criptografia Baseada em Emparelhamentos para Redes de Sensores Usando o Microcontrolador MSP430

Conrado Porto Lopes Gouvêa[1]

Agosto de 2010

Banca Examinadora:

- Prof. Dr. Julio César López Hernández
  Instituto de Computação, Universidade Estadual de Campinas (Orientador)
- Prof. Dr. Ricardo Dahab
  Instituto de Computação, Universidade Estadual de Campinas
- Prof. Dr. Paulo Sérgio Licciardi Messeder Barreto
  Departamento de Engenharia de Computação e Sistemas Digitais, Universidade de São Paulo
- Prof. Dr. Flávio Keidi Miyazawa (Suplente)
  Instituto de Computação, Universidade Estadual de Campinas
- Prof. Dr. Leonardo Barbosa e Oliveira (Suplente)
  Faculdade de Tecnologia, Universidade Estadual de Campinas

[1]Suporte financeiro: bolsa do CNPq (131996/2008-8) 2008–2010.

Resumo

Redes de sensores sem fio têm se tornado populares recentemente e possuem inúmeras aplicações. Contudo, elas apresentam o desafio de como proteger suas comunicações utilizando esquemas criptográficos, visto que são compostas por dispositivos de capacidade extremamente limitada. Neste trabalho é descrita uma implementação eficiente em software, para redes de sensores sem fio, de duas tecnologias de criptografia pública: a Criptografia Baseada em Emparelhamentos (CBE) e a Criptografia de Curvas Elípticas (CCE). Nossa implementação foca a família de microcontroladores MSP430 de 16 bits, utilizada em sensores como o Tmote Sky e TelosB. Em particular, para a CBE, foram implementados algoritmos para o cálculo de emparelhamentos nas curvas MNT e BN sobre corpos primos; para a CCE, foi implementado o esquema de assinatura ECDSA sobre corpos primos para os níveis de segurança de 80 e 128 bits. As principais contribuições deste trabalho são um estudo aprofundado dos algoritmos de emparelhamentos bilineares e novas otimizações na aritmética de corpos primos para a MSP430, que consequentemente melhoram o desempenho dos criptossistemas de CBE e CCE em tal plataforma.

Abstract

Wireless sensor networks have become popular recently and provide many applications. However, the deployment of cryptography in sensor networks is a challenging task, given their limited computational power and resource-constrained nature. This work presents an efficient software implementation, for wireless sensor networks, of two public-key systems: Pairing-Based Cryptography (PBC) and Elliptic Curve Cryptography (ECC). Our implementation targets the MSP430 microcontroller, which is used in some sensors including the Tmote Sky and TelosB. For the PBC, we have implemented algorithms for pairing computation on MNT and BN curves over prime fields; for the ECC, the signature scheme ECDSA over prime fields for the 80-bit and 128-bit security levels. The main contributions of this work are an in-depth study of bilinear pairings algorithms and new optimizations for the prime field arithmetic in the MSP430, which improves the running times of the PBC and ECC cryptosystems on the platform.

Agradecimentos

À minha família: Ubirajara, Solange (in memoriam) e Anahy, por todo o apoio, carinho e incentivo.

À Daniela, pelo amor, e paciência quando eu achava que "ia dar tudo errado".

Ao Florivaldo e Shirley, por me acolherem tão bem em Campinas.

Ao meu orientador e professor, Julio López, por me guiar com seu conhecimento durante o mestrado.

Ao professor Ricardo Dahab, Diego Aranha e demais integrantes do Laboratório de Criptografia Aplicada pelo seu apoio.

Aos professores Orlando Soares e Carlos Maziero, da PUCPR, por me incentivarem a seguir carreira acadêmica.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pelo apoio financeiro.

Sumário

Resumo **vii**

Abstract **ix**

Agradecimentos **xi**

1 Introdução **1**
1.1 Objetivo . . . 3
1.2 Organização do Documento . . . 3

2 Fundamentos **5**
2.1 Grupos . . . 5
2.2 Corpos Finitos . . . 7
2.3 Curvas Elípticas . . . 8
2.3.1 Operação de Grupo . . . 9
2.3.2 Coordenadas Projetivas e Jacobianas . . . 11
2.3.3 Traço de Frobenius . . . 13
2.4 Emparelhamentos Bilineares . . . 14
2.4.1 Pontos de r-torção . . . 15
2.4.2 Raízes da Unidade . . . 17
2.4.3 Funções Racionais . . . 17
2.4.4 Divisores . . . 19
2.4.5 Funções, Equações e Divisores de Retas . . . 23
2.4.6 Emparelhamento de Tate . . . 24
2.4.7 Algoritmo de Miller . . . 26
2.5 Famílias de Curvas . . . 28
2.5.1 Curvas MNT . . . 29
2.5.2 Curvas BN . . . 30
2.6 Protocolos . . . 30
2.6.1 Elliptic Curve Digital Signature Algorithm . . . 31

2.6.2 Non-Interactive Key Distribution Protocol 33

3 Algoritmos **37**
3.1 Introdução à Aritmética de Corpos Finitos Primos 38
3.2 Multiplicação de Precisão Múltipla 39
3.2.1 Multiplicação *Schoolbook* 39
3.2.2 Multiplicação Comba 40
3.2.3 Multiplicação Híbrida 41
3.2.4 Multiplicação Karatsuba-Ofman 44
3.2.5 Outros Métodos de Multiplicação 44
3.3 Redução Modular 44
3.3.1 Redução Montgomery 44
3.4 Multiplicação Modular Híbrida para Curvas BN 46
3.5 Aritmética de Corpos de Extensão 48
3.5.1 Extensão Quadrática 49
3.5.2 Extensão Cúbica 51
3.5.3 *Lazy Reduction* 53
3.5.4 Mapa de Frobenius 55
3.6 Multiplicação de Ponto 55
3.6.1 Multiplicação Binária 56
3.6.2 wNAF 56
3.6.3 Comb 57
3.6.4 *Interleaving* 58
3.7 Otimizações de Emparelhamentos Bilineares 60
3.7.1 *Twists* 60
3.7.2 Eliminação de Denominador 61
3.7.3 Emparelhamento Ate 63
3.7.4 Emparelhamentos Ótimos 66
3.7.5 Exponenciação Final 67
3.7.6 Duplicação e Soma Encapsuladas de Pontos 69
3.8 *Hashing* em $\mathbb{G}_1$ e $\mathbb{G}_2$ 74

4 Implementação **77**
4.1 A Família MSP430 77
4.2 Ferramentas 79
4.3 Parâmetros 79
4.3.1 CCE 80
4.3.2 CBE 80
4.4 Aritmética de Corpos Finitos 81

4.4.1 Multiplicação 81
4.4.2 Redução Modular 85

5 Resultados **87**
5.1 Aritmética de Corpos Finitos 87
5.2 Criptografia de Curvas Elípticas 88
5.3 Criptografia Baseada em Emparelhamentos 91

6 Conclusões e Trabalhos Futuros **95**

A Instruções da MSP430 **97**

Bibliografia **99**

Lista de Acrônimos

BDH *Bilinear Diffie-Hellman*

BN Barreto-Naehrig

CBE Criptografia Baseada em Emparelhamentos

CBI Criptografia Baseada em Identidades

CCE Criptografia de Curvas Elípticas

CGC Centro de Geração de Chaves

CH-SQR *Chung-Hasan Squaring*

CPI *Cycles per Instruction*

CPU *Central Processing Unit*

DSA *Digital Signature Algorithm*

ECDSA *Elliptic Curve Digital Signature Algorithm*

HMMB *Hybrid Modular Multiplication Algorithm for BN Curves*

MNT Miyaji-Nakabayashi-Takano

MOV Menezes-Okamoto-Vanstone

NAF *Non-Adjacent Form*

NIKDP *Non-Interactive Key Distribution Protocol*

NIST *National Institute of Standards and Technology*

PC *Program Counter*

RAM *Random Access Memory*

RISC *Reduced Instruction Set Computing*

ROM *Read-Only Memory*

RSA Rivest-Shamir-Adleman

SECG *Standards for Efficient Cryptography Group*

Lista de Tabelas

2.1 Tabela de multiplicação de $\mathbb{F}_2$ 8
2.2 Tabela de multiplicação de $\mathbb{F}_{2^2}$ representado como $\mathbb{F}_2[u]/(u^2+u+1)$ . . . 8
2.3 Tamanhos recomendados para r e q^k 35

3.1 Comparação das multiplicações *schoolbook*, Comba e Híbrida 43
3.2 Custos para operações em extensões quadráticas 51
3.3 Custos para operações em extensões cúbicas 54
3.4 Custos da duplicação e soma encapsuladas de pontos 74

4.1 Instruções principais da família MSP430 78
4.2 Comparação de números de instruções para a multiplicação Comba MAC e Híbrida para números de 160 bits 84

5.1 Tempos da multiplicação e quadrado 88
5.2 Tempos da redução modular 89
5.3 Tempos da multiplicação em corpo finito (usando redução Montgomery) . . 89
5.4 Tempos das operações no corpo e multiplicação de ponto 90
5.5 Tempos do ECDSA 91
5.6 Tamanho de ROM e máximo de RAM utilizada em programas usando curvas elípticas 91
5.7 Tempos do cálculo de emparelhamento em curvas MNT 92
5.8 Tempos do cálculo de emparelhamento em curvas BN 92
5.9 Tamanho de ROM e máximo de RAM utilizada em programas usando emparelhamentos 93

A.1 Instruções de *jump* presentes na família MSP430 97
A.2 Instruções de até um operando presentes na família MSP430 98
A.3 Instruções de dois operandos presentes na família MSP430 98

Lista de Figuras

2.1 Curva elíptica sobre reais dada por $y^2 = x^3 - 10x + 5$ ilustrando soma de pontos $P + Q = R$ 10
2.2 Curva elíptica sobre reais dada por $y^2 = x^3 - 10x + 5$ ilustrando duplicação de ponto $P + P = R$ 11
2.3 O grupo da curva elíptica $E/\mathbb{F}_{17} : y^2 = x^3 + 7x + 5$ 12
2.4 Função polinomial $f(x) = x^2 - x + 6$ e seus zeros 19
2.5 Função $f(x) = 3x + 2y + 1$ na curva elíptica $E\colon y^2 = x^3 - x + 1$ e seus zeros 20
2.6 Zeros da função $f(x) = 3x + 2y + 1$, curva elíptica $E\colon y^2 = x^3 - x + 1$ e zeros da função na curva 21

3.1 Estrutura de uma implementação de Criptografia Baseada em Emparelhamentos 37
3.2 Multiplicação *Schoolbook* e Comba 40
3.3 Multiplicação Híbrida 43

4.1 Passo na multiplicação Comba, computando $a_0 b_1$ 82
4.2 Passo na multiplicação Comba usando MAC, computando $a_0 b_1$ 83

5.1 Tempos dos diversos algoritmos implementados 94

Lista de Algoritmos

2.1 Algoritmo de Miller 28
2.2 Emparelhamento de Tate 28
2.3 ECDSA: Inicialização 31
2.4 ECDSA: Geração de chaves 31
2.5 ECDSA: Assinatura 32
2.6 ECDSA: Verificação 32
2.7 NIKDP: Inicialização 33
2.8 NIKDP: Geração da chave mestra 34
2.9 NIKDP: Distribuição de chave privada 34
2.10 NIKDP: Distribuição de chaves 34
3.1 Algoritmo de multiplicação *schoolbook* 39
3.2 Algoritmo de multiplicação Comba 41
3.3 Algoritmo de multiplicação Híbrida 42
3.4 Algoritmo de redução Montgomery baseado em *schoolbook* 46
3.5 Algoritmo de redução Montgomery baseado em Comba 47
3.6 Algoritmo binário de multiplicação de ponto 56
3.7 Algoritmo wNAF de multiplicação de ponto 57
3.8 Algoritmo Comb de multiplicação de ponto 58
3.9 Algoritmo Interleaving de wNAFs para multiplicação simultânea de pontos 59
3.10 Algoritmo de Miller com eliminação de denominador 62
3.11 Algoritmo de Miller com eliminação de denominador para variantes do Ate 65

Capítulo 1

Introdução

O surgimento da criptografia assimétrica, concebida por Diffie e Hellman em 1976 [18], foi uma grande revolução na área da criptografia. Ela resolveu, em parte, o problema da troca de chaves — antes, se Alice quisesse mandar uma mensagem cifrada para Beto ou vice versa, ambos deveriam combinar uma chave com antecedência e de modo seguro para então poderem se comunicar de forma confidencial. Na criptografia assimétrica são utilizadas duas chaves, uma pública e uma privada. A chave pública, como o nome diz, pode ser divulgada livremente e permite a qualquer pessoa enviar uma mensagem cifrada para o dono da chave ou verificar a autenticidade de uma mensagem assinada digitalmente pelo dono da chave. Contudo, surge um novo problema: como se pode certificar que uma chave pública, obtida por algum meio arbitrário, realmente pertence à pessoa com quem se deseja comunicar? A incapacidade de se realizar essa verificação abre espaço para uma classe de ataques que pode comprometer a segurança da comunicação.

Atualmente, este problema é parcialmente resolvido com uma infra-estrutura de chaves públicas, onde certas autoridades certificadoras são responsáveis por emitir certificados (assinados digitalmente pela autoridade) garantindo a autenticidade das chaves públicas. Esse mecanismo transfere o problema de se verificar a autenticidade de uma chave pública para o problema de se verificar a assinatura digital de uma determinada autoridade certificadora; porém, como o número de autoridades é muito menor que o número total de possíveis usuários, a complexidade do problema é bastante reduzida. Navegadores de internet, por exemplo, já vêm com as chaves públicas das autoridades certificadoras mais importantes, que por sua vez são utilizadas para se verificar a autenticidade das chaves públicas dos sites sendo acessados.

Nesse contexto, ganhou destaque a Criptografia Baseada em Identidades (CBI) introduzida conceitualmente em [67] mas construída de fato muitos anos depois, principalmente com os trabalhos [6, 12, 60]. Ela consiste em um esquema de criptografia assimétrica onde a chave pública pode ser uma *string* arbitrária, ao contrário de uma *string* de bits de-

pendente da chave privada como era no caso assimétrico tradicional. Tal *string* deve representar a identidade do destinatário e geralmente é exemplificada como sendo um endereço de e-mail, mas pode ser qualquer *string* como um número de matrícula, de identidade, entre outros.

Na CBI, se Alice quiser enviar uma mensagem cifrada para Beto, ela usa como chave pública o e-mail de Beto — a *string* "`beto@cryptoland.net`" (por exemplo). A mensagem cifrada resultante só poderá ser lida por Beto se ele obtiver a chave privada correspondente a essa chave pública. Aqui se encontra um dos pontos centrais da CBI: para Beto realizar isto, ele deve comprovar sua identidade a um centro de geração de chaves (CGC) em quem ele deve confiar implicitamente, e então o centro lhe enviará sua chave privada (obtida combinando-se a chave pública com uma chave mestra conhecida somente pelo centro) e Beto poderá decifrar a mensagem.

As implicações deste sistema são as seguintes. Primeiramente, Alice pode enviar uma mensagem cifrada para uma pessoa qualquer somente tendo a identidade do destinatário. De fato, o destinatário pode nunca ter usado o sistema até então. A autenticidade da chave pública (a identidade) é implícita e não é necessária uma infra-estrutura de chaves públicas para verificá-la. Portanto, o envio de mensagens cifradas (ou verificação de assinaturas) se torna muito mais prático que no cenário assimétrico clássico. Por outro lado, percebe-se que o centro de geração de chaves conhece todas as chaves privadas de seus usuários (diz-se que a *custódia de chaves* é inerente) e assim pode se fazer passar por qualquer um deles. Contudo, em alguns cenários, a existência de tal centro é aceitável.

Um desses cenários é o de redes de sensores sem fio. Tais redes são compostas por inúmeros nós de pequeno poder computacional e têm muitas aplicações em monitoração e coleta de dados para fins civis, comerciais e militares. Os nós são colocados na área de interesse e então passam a monitorá-la, trocando mensagens entre si e uma estação base colocada em lugar próximo mas seguro. Contudo, a comunicação entre os sensores — e os próprios sensores — podem ser comprometidos por atacantes. Por esse motivo, é importante garantir a confidencialidade e autenticidade de tais comunicações utilizando-se primitivas criptográficas. Muitas abordagens envolvendo criptografia simétrica e assimétrica foram propostas, mas a criptografia baseada em identidade aparenta ser a ideal, pelos motivos citados anteriormente. Além disso, neste cenário a custódia de chaves inerente não é um problema, visto que os sensores confiam implicitamente na estação base (que possui o papel de centro de geração de chaves). De fato, aplicações onde os usuários são máquinas — e não pessoas — parecem ser a aplicação ideal da CBI.

Os sistemas mais eficientes para se implementar a CBI são aqueles que utilizam emparelhamentos bilineares. Eles são uma construção matemática baseada em curvas elípticas e foram aplicados na criptografia primeiramente em um ataque à própria Criptografia de Curvas Elípticas (CCE). Esquemas que utilizam emparelhamentos pertencem à cha-

mada Criptografia Baseada em Emparelhamentos (CBE). Com isso, o único problema restante é saber se o cálculo de emparelhamentos é viável em redes de sensores sem fio. Múltiplos trabalhos, como [58], mostraram tal viabilidade, porém deixam claro que ainda são necessárias pesquisas que visem melhorar o desempenho de tal computação.

1.1 Objetivo

Baseada nesta motivação, o objetivo do trabalho abordado nesta dissertação é uma implementação eficiente em software de criptografia baseada em identidade utilizando emparelhamentos bilineares. Também foram implementados algoritmos e um protocolo de Criptografia de Curvas Elípticas, por esta ser mais eficiente (ignorando o problema da autenticação de chaves públicas) e para efeitos de comparação.

A plataforma alvo é o microcontrolador de 16 bits MSP430 da Texas Instruments utilizada em sensores como Tmote Sky, TelosB e TinyNode. Tal plataforma foi escolhida por possuir ambiente de simulação de fácil acesso que permite a implementação mesmo sem o hardware subjacente; ser uma plataforma com grandes limitações de espaço e memória e portanto servindo como exemplo de cenário restrito; e por se acreditar que plataformas de 16 bits possam se tornar mais populares com o avanço da tecnologia de sensores.

1.2 Organização do Documento

Este documento é estruturado como se segue. No Capítulo 2, são introduzidos os fundamentos necessários para se compreender o cálculo de emparelhamentos bilineares, como também são descritos os protocolos implementados. No Capítulo 3, são descritos os algoritmos envolvidos em tais protocolos. A implementação e novas otimizações propostas estão presentes no Capítulo 4 e resultados obtidos no Capítulo 5. A dissertação é concluída no Capítulo 6.

Teoremas normalmente serão apresentados sem provas, mas com uma referência de onde a demonstração pode ser encontrada.

Capítulo 2

Fundamentos

2.1 Grupos

Nesta seção será descrito o conceito de grupo, uma construção matemática abstrata com aplicações imediatas nos dois tipos de criptografia utilizados nesta dissertação. Não se pretende uma abordagem extensa; somente serão citados resultados importantes para a compreensão das seções seguintes. Para uma referência mais completa, recomenda-se o livro [68].

Definição 2.1. *[68, Definição 6.1] Um* grupo abeliano *consiste no par* $(\mathbb{G}, \oplus)$*, onde* $\mathbb{G}$ *é um conjunto e* $\oplus$ *é uma operação binária* $\mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}$*, tal que*

1. *para todos* $x, y, z \in \mathbb{G}$ *tem-se que* $x \oplus (y \oplus z) = (x \oplus y) \oplus z$ *(isto é,* $\oplus$ *é associativa);*
2. *existe* $e \in \mathbb{G}$ *(denominado* elemento neutro*) tal que para todos* $x \in \mathbb{G}$ *tem-se que* $x \oplus e = x = e \oplus x$*;*
3. *para todo* $x \in \mathbb{G}$ *existe* $y \in \mathbb{G}$ *(denominado* inverso de x*) tal que* $x \oplus y = e$*;*
4. *para todos* $x, y \in \mathbb{G}$ *tem-se que* $x \oplus y = y \oplus x$ *(isto é,* $\oplus$ *é comutativa).*

Geralmente se refere por $\mathbb{G}$ tanto o grupo propriamente dito quanto o conjunto que compõe o grupo, se a operação não for relevante ou for implícita no contexto. Por esse motivo, por exemplo, para um grupo $\mathbb{G}$ define-se o número de elementos de $\mathbb{G}$ como o número de elementos do conjunto subjacente; um subconjunto de $\mathbb{G}$ é um subconjunto do conjunto subjacente; e assim por diante.

Apesar de existir a definição de grupo (não abeliano), sem a propriedade comutativa, ela não é usada neste documento. Portanto, por simplificação, se utilizará "grupo" para se referir a "grupo abeliano". Existem muitos exemplos de grupos; um dos mais simples

é composto pelos números inteiros e a operação de adição. Os números reais, excluído o zero, mais a operação de multiplicação também formam um grupo.

Um grupo é denominado *aditivo* se sua operação $\oplus$ é escrita com o sinal de adição usual, por exemplo, $x + y$; mas não necessariamente tal operação será a adição propriamente dita. Analogamente, um grupo é denominado *multiplicativo* se sua operação é escrita com o sinal de multiplicação, por exemplo, $x \cdot y$ ou simplesmente xy. É importante ressaltar que um grupo qualquer não é, por essência, aditivo ou multiplicativo; essa é somente uma questão de notação. Em particular, qualquer teorema ou definição que mencione um grupo aditivo também vale para grupos multiplicativos. Normalmente, o elemento neutro de um grupo aditivo é denominado 0 e o elemento neutro de um grupo multiplicativo é denominado 1; o inverso de um elemento x de um grupo aditivo é escrito $-x$ e o inverso de um elemento x de um grupo multiplicativo é escrito x^{-1} ou $1/x$.

A multiplicação de um elemento a de um grupo aditivo $\mathbb{G}$ por um inteiro k é definida por

$$ka = \underbrace{a + a + \ldots + a}_{k \text{ vezes}} .$$

Se $k = 0$, define-se $ka = 0$. Se $k < 0$, define-se $ka = (-k)(-a)$.

A k-ésima potência de um elemento a de um grupo multiplicativo $\mathbb{G}$ é definida por

$$a^k = \underbrace{a \cdot a \cdots a}_{k \text{ vezes}} .$$

Se $k = 0$, define-se $a^k = 1$. Se $k < 0$, define-se $a^k = (1/a)^{-k}$.

A *ordem de um grupo* $\mathbb{G}$ é definida como o número de elementos de $\mathbb{G}$, se este for finito; senão, diz-se que a ordem é infinita. A *ordem de um elemento* $a \in \mathbb{G}$ escrito aditivamente é o menor $k \in \mathbb{Z}$ tal que $ka = 0$ ($a^k = 1$ para grupo multiplicativo). Um grupo $\mathbb{H}$ é denominado *subgrupo* de $\mathbb{G}$ se $\mathbb{H}$ possuir a mesma operação de $\mathbb{G}$ e for um subconjunto não vazio de $\mathbb{G}$ tal que a operação de grupo seja fechada em $\mathbb{H}$.

Os seguintes teoremas serão necessários neste trabalho.

Teorema 2.2 (Lagrange)**.** *[68, Teorema 6.15] Se $\mathbb{G}$ é um grupo abeliano de ordem finita, e $\mathbb{H}$ é um subgrupo de $\mathbb{G}$, então a ordem de $\mathbb{H}$ divide a ordem de $\mathbb{G}$.*

Teorema 2.3. *[68, Teorema 6.30] Seja $\mathbb{G}$ um grupo abeliano aditivo de ordem finita e seja $a \in \mathbb{G}$ e n a ordem de $\mathbb{G}$. Então $na = 0$ e a ordem de a divide n.*

Teorema 2.4. *[68, Teorema 6.41] Seja $\mathbb{G}$ um grupo abeliano finito de ordem n. Se p é um primo que divide n, então $\mathbb{G}$ contém um elemento de ordem p.*

2.2 Corpos Finitos

Nesta seção serão descritos corpos finitos, uma outra construção matemática abstrata com aplicações imediatas neste trabalho. Novamente, refere-se a [68] para uma abordagem mais completa.

Definição 2.5. *Um* corpo *consiste na tripla* $(\mathbb{F}, \oplus, \otimes)$ *onde* $\mathbb{F}$ *é um conjunto,* $\oplus$ *e* $\otimes$ *são operações binárias* $\mathbb{F} \times \mathbb{F} \mapsto \mathbb{F}$*, tal que*

1. $(\mathbb{F}, \oplus)$ *é um grupo abeliano;*
2. $(\mathbb{F} \setminus \{0\}, \otimes)$ *é um grupo abeliano, onde* 0 *é o elemento neutro de* $(\mathbb{F}, \oplus)$*;*
3. *para todos* $x, y, z \in \mathbb{F}$ *tem-se que* $x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$ *(isto é, a operação* $\otimes$ *é distributiva sobre* $\oplus$*).*

Por convenção escreve-se o grupo $(\mathbb{F}, \oplus)$ aditivamente, com identidade 0, e o grupo $(\mathbb{F} \setminus \{0\}, \otimes)$ multiplicativamente, com identidade 1. Exemplos de corpos são os números racionais e os reais com as operações comuns de adição e multiplicação.

Em criptografia geralmente são utilizados corpos finitos, isto é, com $\mathbb{F}$ contendo um número finito de elementos. Tal número de elementos é denominado *ordem* do corpo. Curiosamente, corpos finitos somente podem possuir ordens na forma p^m, onde p é um primo e m é um inteiro positivo [68, Teorema 7.7]. O primo p é denominado *característica* do corpo. Por convenção, denomina-se $\mathbb{F}_q$ um corpo finito com $q = p^m$ elementos. Se o número de elementos for primo (isto é, $m = 1$), normalmente escreve-se $\mathbb{F}_p$ para deixar claro tal fato. Tal notação não causa perda de generalidade, pois todos os corpos finitos de mesma ordem são isomórficos [68, Teorema 19.15].

Para instanciar concretamente o conceito abstrato de corpo finito, primeiramente pode-se verificar que o conjunto $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$, com p primo, define um corpo finito de ordem p ($\mathbb{F}_p$) junto das operações $x \oplus y \mapsto x + y \bmod p$ e $x \otimes y \mapsto xy \bmod p$. Por esse motivo, geralmente (e de agora em diante) denota-se $\oplus$ como $+$ e $\otimes$ como $\cdot$ apesar de não serem a soma e multiplicação usuais.

Resta a instanciação de $\mathbb{F}_{p^m}$, com $m > 1$. Tais corpos são denominados de *corpos de extensão*. Pode-se construí-los da seguinte forma. Primeiramente, escolha um polinômio irredutível $f(u)$ de grau m e com coeficientes em $\mathbb{F}_p$. Assim, representa-se $\mathbb{F}_{q^m}$ como todos os polinômios de grau $m-1$ com coeficientes em $\mathbb{F}_p$ junto das operações $x \oplus y \mapsto x + y \bmod f(u)$ e $x \otimes y \mapsto xy \bmod f(u)$. Pode-se verificar que tal representação possui p^m elementos e atende a todas as propriedades de um corpo finito. Tal construção é denominada $\mathbb{F}_p[u]/f(u)$.

Exemplo 2.6. *O corpo finito* $\mathbb{F}_2$ *consiste apenas nos elementos* 0 *e* 1*, com adição e multiplicação módulo* 2*. A Tabela 2.1 apresenta sua tabela de multiplicação, onde cada*

elemento é o produto dos elementos da respectiva linha e coluna. Já a extensão $\mathbb{F}_{2^2}$*, construída como* $\mathbb{F}_2[u]/(u^2+u+1)$*, está ilustrada na Tabela 2.2.*

Exemplo 2.7. *Considere o corpo finito* $\mathbb{F}_{7^2}$*. Pode-se construí-lo como* $\mathbb{F}_7[u]/(u^2+1)$*; assim, elementos de* $\mathbb{F}_{7^2}$ *são representados como polinômios em u de grau 2 e coeficientes em* $\mathbb{F}_7$*. Para se multiplicar, por exemplo,* $3u+4$ *por* $2u+1$*, multiplicam-se os dois operandos obtendo-se* $6u^2+11u+4 = 6u^2+4u+4$ *e calcula-se o resto da divisão de tal resultado por* u^2+1*, obtendo-se* $4u+5$*.*

	0	1
0	0	0
1	0	1

Tabela 2.1: Tabela de multiplicação de $\mathbb{F}_2$

	0	1	u	$u+1$
0	0	0	0	0
u	0	$u+1$	1	u
$u+1$	0	1	u	$u+1$
1	0	u	$u+1$	1

Tabela 2.2: Tabela de multiplicação de $\mathbb{F}_{2^2}$ representado como $\mathbb{F}_2[u]/(u^2+u+1)$

No contexto de corpos finitos, um elemento $a \in \mathbb{F}_q$ é denominado um *resíduo n-ésimo*, n inteiro maior que 1, se existe $x \in \mathbb{F}_q$ tal que $x^n = a$. Em particular, se $n = 2$ denomina-se *resíduo quadrático*, se $k = 3$ *resíduo cúbico*, e assim por diante. Se não existe tal x, então a é denominado um *não-resíduo n-ésimo*.

2.3 Curvas Elípticas

Curvas elípticas são uma construção matemática nascida do estudo de integrais elípticas. Seu primeiro uso na criptografia ocorreu por Hendrik W. Lenstra [40] em um método para se fatorar inteiros, e posteriormente na construção de criptossistemas por Victor S. Miller [50] e Neal Koblitz [36], independentemente. Desde então, elas serviram de base para inúmeros protocolos criptográficos e fornecem o fundamento de emparelhamentos bilineares.

Definição 2.8. *Uma* curva elíptica *E sobre um corpo K, denotada E/K, é definida pela equação*

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{2.1}$$

onde $a_1, a_2, a_3, a_4, a_6 \in K$.

A equação 2.1 é conhecida como *equação generalizada de Weierstrass*. Pode-se simplificá-la com uma certa transformação [30, seção 3.1.1], dependendo do corpo K. Se a característica de K não for 2 nem 3, então pode-se transformar E em

$$y^2 = x^3 + ax + b \tag{2.2}$$

com $a, b \in K$. A equação 2.2 é denominada *equação simplificada de Weierstrass*. Curvas sobre corpos de característica 2 também são denominadas *curvas binárias* e não são estudadas neste documento, assim como curvas sobre corpos de característica 3. Curvas sobre corpos de característica grande são denominadas *curvas primas*.

2.3.1 Operação de Grupo

A característica mais interessante das curvas elípticas é que os pares (x, y) que satisfazem a equação, denominados *pontos na curva*, junto de um ponto no infinito e uma certa operação formam um grupo.

Tal operação de grupo possui uma definição geométrica. A soma de dois pontos $P, Q \in E(K)$, ilustrada na Figura 2.1, é calculada da seguinte forma. Primeiramente, traça-se a reta que passa pelos pontos P e Q. Tal reta obrigatoriamente interceptará a curva em um terceiro ponto T. Traça-se uma reta vertical por T. Tal reta interceptará a curva em um segundo ponto R. Define-se então que $P + Q = R$. Para calcular a soma de um ponto P com si mesmo (operação denominada duplicação de ponto), traça-se a tangente a esse ponto para obter o ponto T, conforme ilustrado na Figura 2.2. Caso a reta que passe por P e Q (na soma) ou a tangente de P (na duplicação) seja vertical, então o resultado é o ponto no infinito. Assim, pode-se observar que no exemplo da adição tem-se que $T + R = \infty$, portanto $T = \infty - R = -R$.

Teorema 2.9. *[78, Teorema 2.1] Seja E/K uma curva elíptica sobre o corpo K. O conjunto*

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

e a operação $+$ definida anteriormente formam um grupo, onde o ponto no infinito denotado por ∞ é o elemento neutro. Tal grupo é denominado $E(K)$.

Apesar de parecer artificial, o ponto no infinito surge naturalmente ao se considerar a versão projetiva da curva elíptica, o que será abordado mais adiante.

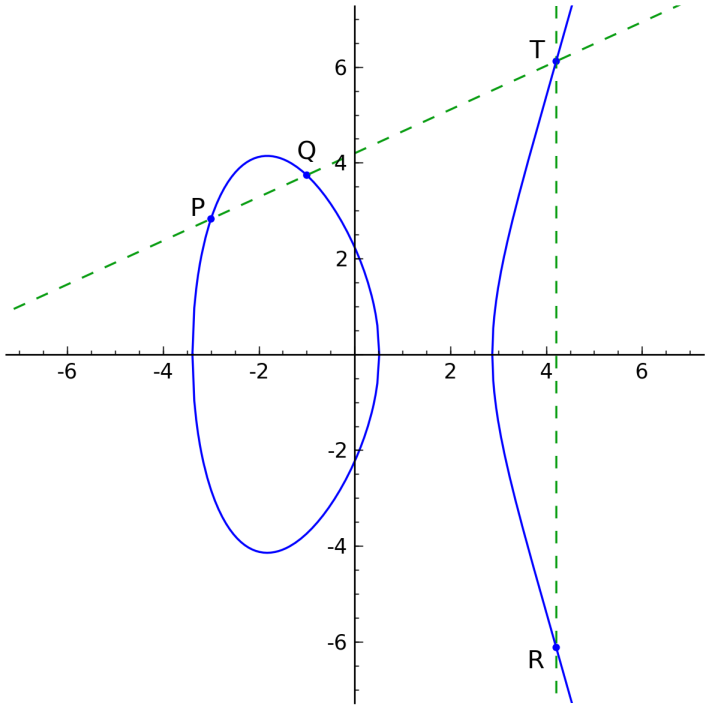


Figura 2.1: Curva elíptica sobre reais dada por $y^2 = x^3 - 10x + 5$ (linha sólida) ilustrando soma de pontos $P + Q = R$

Para se computar a operação de grupo, parte-se da interpretação geométrica para se derivar fórmulas que fornecem as coordenadas do ponto resultante em função das coordenadas dos operandos [78, seção 2.2]. Considere a curva elíptica $E/K : y^2 = x^3 + ax + b$. A soma de um ponto $P = (x_P, y_P)$ com um ponto $Q = (x_Q, y_Q)$ é dada por $P + Q = S = (x_S, y_S)$, com

$$x_S = \left(\frac{y_Q - y_P}{x_Q - x_P}\right)^2 - x_P - x_Q$$
$$y_S = \left(\frac{y_Q - y_P}{x_Q - x_P}\right)(x_P - x_S) - y_P\,.$$

A duplicação de um ponto $P = (x_P, y_P)$ é dada por $2P = T = (x_T, y_T)$, com

$$x_T = \left(\frac{3x_P^2 + a}{2y_P}\right) - 2x_P$$
$$y_T = \left(\frac{3x_P^2 + a}{2y_P}\right)(x_P - x_T) - y_P\,.$$

O inverso de um ponto $P = (x_P, y_P)$ é dado por $-P = (x_P, -y_P)$.

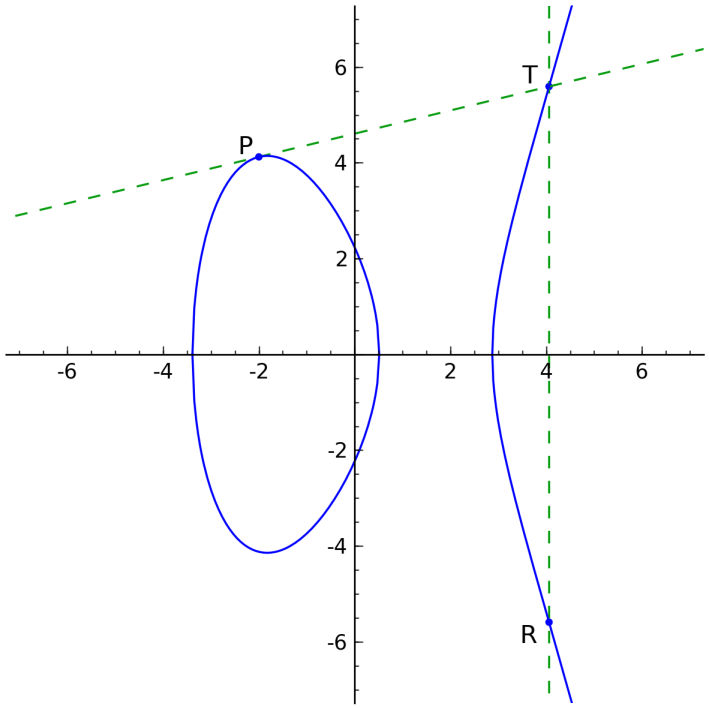


Figura 2.2: Curva elíptica sobre reais dada por $y^2 = x^3 - 10x + 5$ (linha sólida) ilustrando duplicação de ponto $P + P = R$

Curiosamente, tais fórmulas funcionam para qualquer corpo K de característica diferente de 2 e 3, apesar de serem derivadas da interpretação geométrica da operação do grupo, que só é válida para números reais.

Exemplo 2.10. *Considere a curva elíptica* $E/\mathbb{F}_{17} : y^2 = x^3+7x+5$*, que possui um número finito de pontos, ilustrados na Figura 2.3. Nela estão marcados os pontos* $P = (13, 7)$*,* $Q = (3, 6)$*,* $R = (9, 7)$ *e* $T = (9, 10)$ *tal que* $P+Q = R$ *e* $T = -R$*. Pode-se verificar que a interpretação geométrica da operação de grupo é perdida ao se deixar de usar o corpo dos números reais. Ainda assim, as fórmulas mencionadas continuam válidas; por exemplo:* $x_{P+Q} = \left(\frac{6-7}{3-13}\right)^2 - 13 - 3 = (16 \cdot 7^{-1})^2 + 1 = (16 \cdot 5)^2 + 1 = 9 = x_R$.

2.3.2 Coordenadas Projetivas e Jacobianas

As fórmulas de adição e duplicação de pontos envolvem divisões que em curvas sobre um corpo finito são implementadas através de inversões no corpo, que por sua vez são computacionalmente caras. Por esse motivo, é utilizada uma representação alternativa das coordenadas que resulta em fórmulas sem divisões para adição e duplicação. Pontos em uma curva elíptica são representados por duas coordenadas x, y; o ponto no infinito

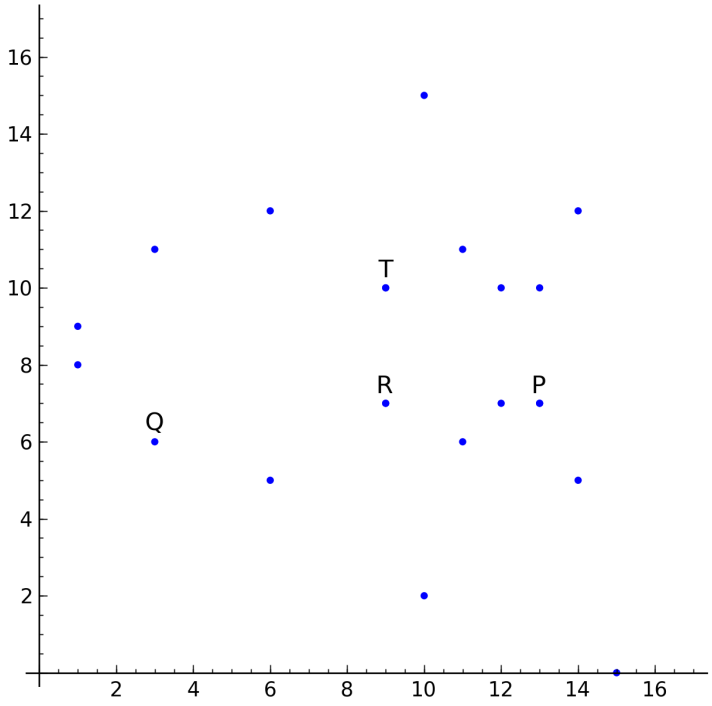


Figura 2.3: O grupo da curva elíptica $E/\mathbb{F}_{17} : y^2 = x^3 + 7x + 5$

sendo uma exceção. Tais coordenadas são chamadas de *afins*. Contudo, pode-se adicionar uma ou mais coordenadas em cada ponto, criando as *coordenadas projetivas*. Na definição a seguir, são introduzidas as coordenadas projetivas em sua forma clássica.

Definição 2.11. *Um* ponto projetivo *em uma curva elíptica E/K é definido como a tripla (x, y, z) tal que $(x/z, y/z)$ é um ponto de coordenadas afins em E/K se $z \neq 0$. Se $z = 0$, o ponto (x, y, z) é um ponto no infinito em E/K.*

Considere a equação simplificada de uma curva elíptica, $y^2 = x^3 + ax + b$. Efetuando a troca de variáveis $x \mapsto x/z$, $y \mapsto y/z$ e eliminando denominadores, obtém-se a equação

$$y^2 z = x^3 + axz^2 + bz^3 \,,$$

denominada *equação projetiva* da curva. Por construção, qualquer ponto projetivo satisfaz tal equação. Dois pontos projetivos (x_1, y_1, z_1) e (x_2, y_2, z_2) são equivalentes (mapeados para um mesmo ponto afim) se existir algum $\lambda \in K$, $\lambda \neq 0$, tal que $x_1 = \lambda x_2$, $y_1 = \lambda y_2$ e $z_1 = \lambda z_2$. Por esse motivo, denota-se por $(x : y : z)$ a classe de equivalência à qual o ponto (x, y, z) pertence; para simplificar também pode-se denotar a classe de equivalência por "ponto". É interessante observar que o ponto no infinito agora deixa de ser uma exceção

e passa a ser (a classe de equivalência de) um ponto que satisfaz a equação na curva; mais especificamente, o ponto $(0 : 1 : 0)$.

Para compreender porque as fórmulas de adição e duplicação em tais coordenadas não requerem divisões, considere a fórmula para o cálculo de x_{2P}. Efetuando a troca de variáveis $x_P \mapsto x_P/z_P$, $y_P \mapsto y_P/z_P$ e simplificando, obtém-se

$$\frac{x_{(2P)}}{z_{(2P)}} = \frac{3x_P^2 + az_P^2 - 2x_P y_P}{2y_P z_P}\,.$$

Pode-se eliminar o denominador definindo $z_{(2P)} = 2y_P z_P$, obtendo-se $x_{(2P)} = 3x_P^2 + az_P^2 - 2x_P y_P$. O mesmo pode ser feito para $y_{(2P)}$, com a única restrição de se usar esse mesmo valor de $z_{(2P)}$. Assim, é possível duplicar um ponto sem realizar nenhuma divisão.

Na prática, a adição de dois pontos projetivos raramente é utilizada; é suficiente o uso de adição misturada (*mixed addition*) que consiste na adição de um ponto projetivo com um ponto afim.

Adicionalmente, pode-se verificar [30, Tabela 3.3] que realizar a troca de variáveis $x \mapsto x/z^2$, $y \mapsto y/z^3$ resulta em uma duplicação mais rápida do que com coordenadas projetivas comuns, em troca de uma adição ligeiramente menos eficiente. Como tanto em CCE e CBE são realizadas mais duplicações do que somas, tais coordenadas — denominadas *coordenadas Jacobianas* — apresentam vantagens computacionais. A soma de um ponto $P = (x_P, y_P, z_P)$ em coordenadas Jacobianas com um ponto $Q = (x_P, y_Q)$ em coordenadas afins é dada por $P + Q = S = (x_S, y_S, z_S)$ também em coordenadas Jacobianas, com

$$\begin{aligned}
x_S &= (y_Q z_P^3 - y_P)^2 - (x_Q z_P^2 - x_P)^2(x_P + x_Q z_P^2) \\
y_S &= (y_Q z_P^3 - y_P)(x_P(x_Q z_P^2 - x_P)^2 - x_S) - y_P(x_Q z_P^2 - x_P)^3 \\
z_S &= (x_Q z_P^2 - x_P) z_P\,.
\end{aligned}$$

A duplicação de um ponto $P = (x_P, y_P, z_P)$ em coordenadas Jacobianas é dada por $2P = T = (x_T, y_T, z_T)$, também em coordenadas Jacobianas, com

$$\begin{aligned}
x_T &= (3x_P^2 + az_P^4)^2 - 8x_P y_P^2 \\
y_T &= (3x_P^2 + az_P^4)(4x_P y_P^2 - x_T) - 8y_P^4 \\
z_T &= 2y_P z_P\,.
\end{aligned}$$

O inverso de um ponto $P = (x_P, y_P, z_P)$ em coordenadas Jacobianas é dado por $-P = (x_P, -y_P, z_P)$.

2.3.3 Traço de Frobenius

O traço de Frobenius é uma propriedade de curvas elípticas definido da seguinte forma.

Definição 2.12. *Seja $E/\mathbb{F}_q$ uma curva elíptica e n a ordem do grupo $E(\mathbb{F}_q)$. O* traço de Frobenius *de $E/\mathbb{F}_q$ é definido como $t = q + 1 - n$.*

O seguinte teorema mostra que o valor do traço de Frobenius é restrito a um intervalo.

Teorema 2.13 (Hasse)**.** *[78, Teorema 4.2] Seja $E/\mathbb{F}_q$ uma curva elíptica, n a ordem do grupo $E(\mathbb{F}_q)$ e t seu traço de Frobenius. Tem-se que $|t| \leq 2\sqrt{q}$.*

Como o traço de Frobenius tem aproximadamente metade dos bits de q, o teorema de Hasse implica que n e q possuem aproximadamente a mesma ordem de grandeza (pois $q - n = t - 1$). Assim, ao se escolher uma curva elíptica, a escolha de q influenciará o tamanho de n.

Se p, a característica de $\mathbb{F}_q$, divide t, então a curva é denominada *supersingular*. Curvas supersingulares têm propriedades especiais, em particular, seu grau de mergulho (conforme será definido adiante) é sempre menor ou igual a 6. Contudo, elas não serão abordadas neste trabalho.

2.4 Emparelhamentos Bilineares

Emparelhamentos bilineares (*bilinear pairings*) são uma construção matemática usada, por exemplo, na prova de um caso especial da hipótese de Riemann por André Weil [79] em 1941. Em 1986, Victor S. Miller descreveu um algoritmo [49] para calcular o emparelhamento de Weil em tempo polinomial. Nesse artigo, Miller já mencionava possíveis aplicações na área de Criptografia de Curvas Elípticas. Em 1993, o primeiro uso de destaque de emparelhamentos na criptografia surgiu com o ataque MOV [48] usado para se reduzir o problema do logaritmo discreto no grupo de pontos em uma curva elíptica para o problema do logaritmo no subgrupo multiplicativo de uma extensão do corpo finito usado na curva. Em 2000, o primeiro uso criptográfico construtivo de emparelhamentos surgiu com o protocolo Diffie-Hellman tripartite de Joux [34] que permite três pessoas combinarem uma chave única através de um canal inseguro com somente uma rodada de comunicações. Finalmente, em 2001, a CBI baseada em emparelhamentos surgiu com o trabalho independente de Sakai, Ohgishi e Kasahara [60] e de Boneh e Franklin [6]. Esquemas criptográficos que utilizam emparelhamentos fazem parte da Criptografia Baseada em Emparelhamentos (CBE).

Primeiramente, será descrita a definição matemática abstrata de emparelhamento bilinear, que posteriormente será instanciada concretamente.

Definição 2.14. *Dados três grupos $\mathbb{G}_1$, $\mathbb{G}_2$ e $\mathbb{G}_T$ de ordem prima r, com $\mathbb{G}_1$ e $\mathbb{G}_2$ aditivos e $\mathbb{G}_T$ multiplicativo, um* emparelhamento bilinear *é definido como um mapa*

$$e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

que possui as seguintes propriedades:

1. *(bilinearidade) para todo* $R,\ S \in \mathbb{G}_1$ *e* $T \in \mathbb{G}_2$ *tem-se que*

$$e(R+S,T) = e(R,T)e(S,T)$$

   e para todo $R \in \mathbb{G}_1$ *e* $S,\ T \in \mathbb{G}_2$ *tem-se que*

$$e(R,S+T) = e(R,S)e(R,T)\,.$$

   Uma conseqüência particularmente importante é que para todo $a, b \in \mathbb{Z}$, $R \in \mathbb{G}_1$ *e* $S \in \mathbb{G}_2$ *tem-se que* $e(aR,bS) = e(R,S)^{ab}$*;*

2. *(não-degeneração) tem-se que* $e(R,S) = 1$ *para todo* $S \in \mathbb{G}_2$ *se e somente se* $R = 0$*, e similarmente* $e(R,S) = 1$ *para todo* $R \in \mathbb{G}_1$ *se e somente se* $S = 0$*;*

3. *(computabilidade) para todo* $R \in \mathbb{G}_1$ *e* $S \in \mathbb{G}_2$ *tem-se que* $e(R,S)$ *é eficientemente computável.*

Essa definição é conhecida como emparelhamento do Tipo 3, como definido em [25]. Emparelhamentos do Tipo 1, também denominados *simétricos*, utilizam curvas supersingulares e não são abordados nesta dissertação. Emparelhamentos do Tipo 2 aparentam ser apenas versões ineficientes do Tipo 3 [9].

No restante desta seção serão explorados os fundamentos e uma construção de emparelhamentos usando curvas elípticas, que é a única construção conhecida de emparelhamentos com aplicações para criptografia. Quando não definidos, seja $K = \mathbb{F}_q = \mathbb{F}_{p^m}$ um corpo finito de característica p, seja E/K uma curva elíptica qualquer sobre o corpo K, seja $E(K)$ o grupo de pontos em E/K, e seja n a ordem de tal grupo.

2.4.1 Pontos de r-torção

Pontos de r-torção serão utilizados na definição dos grupos $\mathbb{G}_1$ e $\mathbb{G}_2$.

Definição 2.15. *Os* pontos de r-torção *de* $E(K)$*, onde* r *é um inteiro positivo, são o conjunto* $\{P \in E(K) \mid rP = \infty\}$*, denotado por* $E(K)[r]$*.*

Em outras palavras, pode-se considerar os pontos de r-torção como todos os pontos cuja ordem divide r.

Exemplo 2.16. *Considere a curva elíptica* $E/\mathbb{F}_5\colon y^2 = x^3 + x + 2$. *O grupo* $E(\mathbb{F}_5)$ *possui os pontos* ∞ *com ordem 1,* $(4,0)$ *com ordem 2,* $(1,2)$ *e* $(1,3)$ *com ordem 4. Portanto, todos os pontos são de 4-torção e* $\{\infty,(4,0)\}$ *são de 2-torção.*

O tamanho do conjunto de pontos de r-torção é importante em emparelhamentos. Ao se contar o tamanho de tal conjunto sobre uma curva sobre extensões do corpo original, verifica-se que esse tamanho pode chegar até a r^2. Após isso, posteriores extensões não irão aumentar o tamanho do conjunto.

Definição 2.17. *O* grau de mergulho de $E(\mathbb{F}_q)$ em relação a r *é o menor inteiro positivo* k *tal que o tamanho do conjunto* $E(\mathbb{F}_{q^k})[r]$ *de pontos de* r*-torção em* $E(\mathbb{F}_{q^k})$ *seja* r^2. *Tal conjunto é denominado* conjunto completo de pontos de r-torção de $E(\mathbb{F}_q)$ *(tal denominação não é usual, mas facilita o entendimento).*

Teorema 2.18. *[68, Teorema 6.8] O conjunto* $E(\mathbb{F}_{q^k})[r]$ *é um subgrupo de* $E(\mathbb{F}_{q^k})$. *Por esse motivo, também é denominado* grupo de r-torção de $E(\mathbb{F}_q)$.

Exemplo 2.19. *Considere a curva elíptica* $E/\mathbb{F}_5\colon y^2 = x^3 + x + 1$. *O grupo* $E(\mathbb{F}_5)$ *possui 9 pontos; seus pontos de 3-torção são* $\{\infty,(2,1),(2,4)\}$. *Tal conjunto não é o conjunto completo de pontos de* 3*-torção pois não possui todos os* $3^2 = 9$ *possíveis pontos de 3-torção.*

Fazendo-se uma extensão de grau 2 no corpo base, tem-se que o grupo $E(\mathbb{F}_{5^2})$ *possui 27 pontos; 9 deles de 3-torção. Portanto, tal conjunto é o completo, e como* 2 *é o menor grau em que isso acontece, o grau de mergulho de* $E(\mathbb{F}_5)$ *em relação a 3 é 2. Caso se realize extensões de grau maior que 2, o conjunto de pontos de 3-torção nunca terá tamanho maior que 9.*

Para certos valores de r, o grau de mergulho pode ser facilmente calculado pelo seguinte teorema.

Teorema 2.20. *[2, Teorema 1] Seja* $E/\mathbb{F}_q$ *uma curva elíptica de ordem* n. *Se* r *for primo com* $r \mid n$ *e* $r \nmid q-1$, *então o grau de mergulho de* $E(\mathbb{F}_q)$ *em relação a um inteiro positivo* r *é o menor inteiro positivo* k *tal que* $r \mid q^k - 1$.

É mais fácil escolher r com as características especificadas do que considerar o caso em que ele não atende tais características; portanto sempre se assume que r é primo, $r \mid n$ e $r \nmid q^k - 1$.

Exemplo 2.21. *Considere a curva elíptica* $E/\mathbb{F}_5\colon y^2 = x^3 + x + 1$ *e seja* k *o grau de mergulho de* $E(\mathbb{F}_5)$ *em relação a* 3. *Tem-se que* $3 \nmid 5^1 - 1 = 4$, *mas tem-se que* $3 \mid 5^2 - 1 = 24$. *Portanto,* $k = 2$, *como determinado no Exemplo 2.19.*

Exemplo 2.22. *Considere agora o grau de mergulho k de $E(\mathbb{F}_{19})\colon y^3 = x^2 + x + 6$ em relação a 3. Note que $r = 3$ não permite o cálculo simples do grau de mergulho através do Teorema 2.20 pois $r \mid q - 1$ ($3 \mid 19 - 1$). De fato, pode-se verificar (com um computador, por exemplo) que $k = 3$, e não 1 como sugeriria o cálculo simples.*

2.4.2 Raízes da Unidade

Curiosamente, assim como o grau de mergulho está relacionado ao conjunto de pontos de r-torção, ele também está relacionado com um conjunto denominado raízes r-ésimas da unidade. Tal conjunto também será utilizado na definição do grupo $\mathbb{G}_T$.

Definição 2.23. *Em um corpo K, as* raízes r-ésimas *da unidade são o conjunto $\{x \in K \mid x^r = 1\}$, onde r é um inteiro positivo.*

Em outras palavras, são as raízes r-ésimas do elemento neutro da multiplicação de K. Note a semelhança com a definição de pontos de r-torção.

Seja ℓ o menor inteiro tal que o conjunto de raízes r-ésimas da unidade em $\mathbb{F}_{q^\ell}$ tenha r elementos. Tal conjunto é denotado por μ_r e denominado *conjunto completo de raízes r-ésimas da unidade de $\mathbb{F}_q$*, pois sabe-se não existem mais do que r raízes r-ésimas [78, Apêndice C] em extensões de $\mathbb{F}_q$. A relação com o grau de mergulho se dá pelo seguinte teorema.

Teorema 2.24. *[78, Corolário 3.11] Seja $\mu_r \subset \mathbb{F}_{q^\ell}$ o conjunto completo de raízes r-ésimas da unidade de $\mathbb{F}_q$ e seja k o grau de mergulho de $E(\mathbb{F}_q)$ em relação a r. Então $\mu_r \subset \mathbb{F}_{q^k}$.*

Em emparelhamentos, são necessários tanto o conjunto completo de pontos de r-torção de $E(\mathbb{F}_q)$ quanto o conjunto completo de raízes r-ésimas da unidade de $\mathbb{F}_q$. Como visto, o primeiro conjunto está contido em $E(\mathbb{F}_{q^k})$ e o segundo em $\mathbb{F}_{q^k}$, onde k é o grau de mergulho em relação a r, que pode ser calculado facilmente para certos valores de r de acordo com o Teorema 2.20.

Exemplo 2.25. *A única raiz terceira ("3-ésima") da unidade em $\mathbb{F}_5$ é 1 (que trivialmente sempre é raiz). Já em $\mathbb{F}_{5^2}$, as raízes terceiras da unidade são 1, $2u+1$ e $3u+3$, utilizando-se a representação $\mathbb{F}_5[u]/(u^2 - 2)$. Tal conjunto é o completo pois tem todas as 3 raízes.*

2.4.3 Funções Racionais

Funções racionais em curvas elípticas formam os blocos básicos do cálculo de emparelhamentos. Denote por $K[x, y]$ o conjunto de polinômios nas variáveis x e y com coeficientes no corpo finito K. Note que qualquer elemento de $K[x, y]$ possui uma função $f(x, y)$ associada a ele, de fato, pode-se referir a elementos de $K[x, y]$ como polinômios ou como funções.

Exemplo 2.26. *O polinômio $3x + 2y + 5$ é um elemento de $\mathbb{F}_5[x, y]$. A função associada a esse polinômio é $f(x, y) = 3x+2y+5$. O polinômio $(3u+1)x+6y+(2u)$ é um elemento de $\mathbb{F}_{7^2}[x, y]$.*

Definição 2.27. *Seja $E/K\colon y^2 = x^3 + ax + b$ uma curva elíptica sobre K. Uma* função em $E(K)$ *é uma função associada a um polinômio em $K[x, y]$ cujo domínio é restrito a valores de x, y tal que $P = (x, y)$ está em $E(K)$.*

Definição 2.28. *Seja $E/K\colon y^2 = x^3 + ax + b$ uma curva elíptica sobre K. Uma* função racional em $E(K)$ *é uma função correspondente à divisão de duas funções em $E(K)$ cujo domínio é restrito a valores de (x, y) tal que $P = (x, y)$ está em $E(K)$.*

Exemplo 2.29. *A função $f(x, y) = 2x + 3y + 4$ é uma função em $E(K)$ qualquer. A função $g(x, y) = (x + 2y + 5)/(4x + y)$ é uma função racional em $E(K)$ qualquer. Veja que a influência da curva na função só se dá no seu domínio e no corpo em que seus coeficientes são definidos.*

Uma função (racional ou não) $f(x, y)$ em uma curva $E(K)$ também pode ser referenciada por $f(P)$ com $P \in E(K)$. Neste caso, define-se que $f(P)$ é calculado por $f(x_P, y_P)$ e indefinido se $P = \infty$.

É importante ressaltar alguns aspectos dessas funções racionais em curvas elípticas (geralmente abreviadas para "funções racionais", embora deve-se lembrar que estão relacionadas a curvas elípticas). Como somente são considerados os (x, y) correspondentes a pontos na curva elíptica, então duas funções racionais são equivalentes se fornecem o mesmo valor para todos os pontos da curva, mesmo se fornecerem valores diferentes para pontos fora dela.

Além disso, como todos os (x, y) são de pontos da curva, então a equação $y^2 = x^3 + ax + b$ da curva é sempre válida para qualquer (x, y) que é fornecido como entrada para uma função racional. Por ilustrar a utilidade desta observação, considere a função

$$f(x, y) = \frac{x}{y}$$

na curva $y^2 = x^3 + x$ sobre um corpo qualquer. A princípio f é indefinida no ponto $(0, 0)$. Mas como a equação da curva é sempre válida nos valores de (x, y) que estamos interessados, podemos reorganizá-la de forma a achar uma nova forma para $f(x, y)$, da seguinte forma:

$$y^2 = x^3 + x \implies y^2 = x(x^2 + 1) \implies \frac{x}{y} = \frac{y}{x^2 + 1}\,.$$

Portanto, $f(x, y)$ é equivalente a $y/(x^2+1)$ e fornece o valor 0 no ponto $(0, 0)$. Pode-se mostrar que é sempre possível transformar uma função racional de forma a fornecer um valor único e diferente de $0/0$ para todos pares (x, y) [78, Seção 11.1].

Um último detalhe a se notar é que o polinômio denominador de uma função racional não pode ser um múltiplo do polinômio associado à curva elíptica (por exemplo, $y^2 - x^3 - ax - b$) pois assim o denominador sempre valeria zero.

2.4.4 Divisores

Divisores fornecem um método para se estudar funções racionais, representando os seus zeros e pólos (cuja definição será dada em breve).

É interessante traçar um paralelo entre funções em curvas elípticas e funções polinomiais de uma variável. Uma função polinomial $f(x)$ possui um conjunto de zeros, ou raízes, dados pela equação $f(x) = 0$. Sabendo-se os zeros de uma função, é possível construir a função original, salvo um fator constante. Graficamente, pode-se representar $f(x)$ como todos os pontos $(x, f(x))$; seus zeros podem ser representados por pontos x tal que $f(x) = 0$. Os pontos que formam o gráfico de $f(x)$ possuem duas coordenadas; já os pontos que formam suas raízes, apenas uma (pois sabemos que $y = 0$).

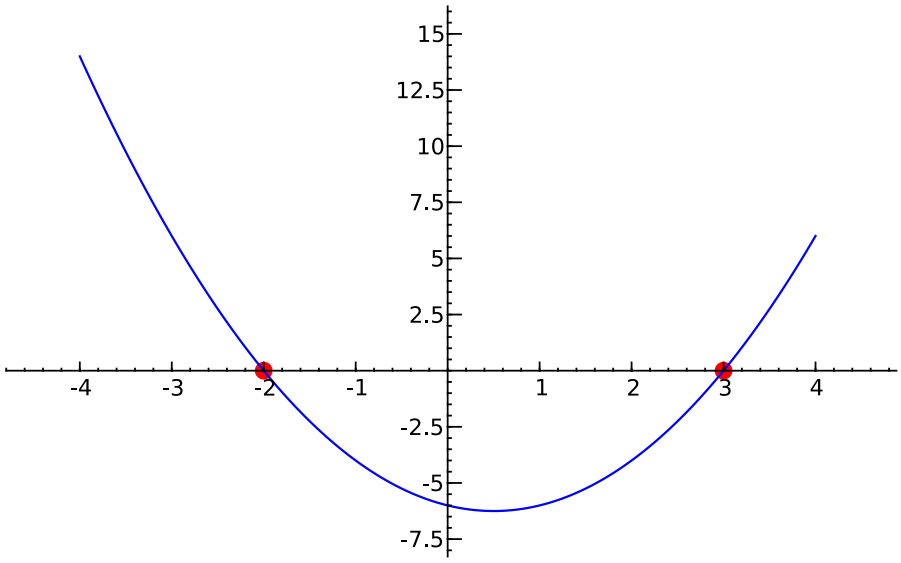


Figura 2.4: Função polinomial $f(x) = x^2 - x + 6$ e seus zeros, em vermelho

Exemplo 2.30. *Considere a função $f(x) = x^2 - x + 6$ plotada na Figura 2.4. O conjunto de pontos $(x, f(x))$ forma uma parábola. O conjunto de pontos x tal que $f(x) = 0$ é $\{-2, 3\}$; são os zeros da função. Se multiplicarmos os polinômios das funções das retas verticais que passam pelos zeros, tem-se*

$$(x - (-2))(x - 3) = x^2 - x + 6\,,$$

que é a função original.

Considere agora uma função $f(x, y)$ em uma curva elíptica. Analogamente à função polinomial, ela possui um conjunto de zeros dados pela equação $f(x, y) = 0$. Sabendo-se os zeros de uma função em uma curva, é possível construir a função original, salvo um fator constante. Graficamente, pode-se representar $f(x, y)$ como todos os pontos $(x, y, f(x, y))$; seus zeros podem ser representados por pontos (x, y) tal que $f(x, y) = 0$. Compare com o caso anterior: como agora temos uma função com duas variáveis, então $f(x, y)$ e suas raízes naturalmente possuem uma coordenada a mais.

Exemplo 2.31. *Considere a função $f(x, y) = 3x + 2y + 1$ na curva elíptica $E/\mathbb{R}$: $y^2 = x^3 - x + 1$ sobre o corpo dos números reais. Tal função está plotada na figura 2.5. O conjunto de pontos (x, y) em $E(\mathbb{R})$ tal que $f(x, y) = 0$ são $\{(-1, 1), (1/4, -7/8), (3, -5)\}$, os zeros da função. Note que tais pontos são os pontos de intersecção entre a função e a curva, assim com os zeros de uma função polinomial são os pontos de intersecção entre a função e a reta $y = 0$. Note também que o gráfico não é uma superfície, como normalmente seria para uma função com dois parâmetros, devido à restrição do domínio aos pares (x, y) da curva elíptica.*

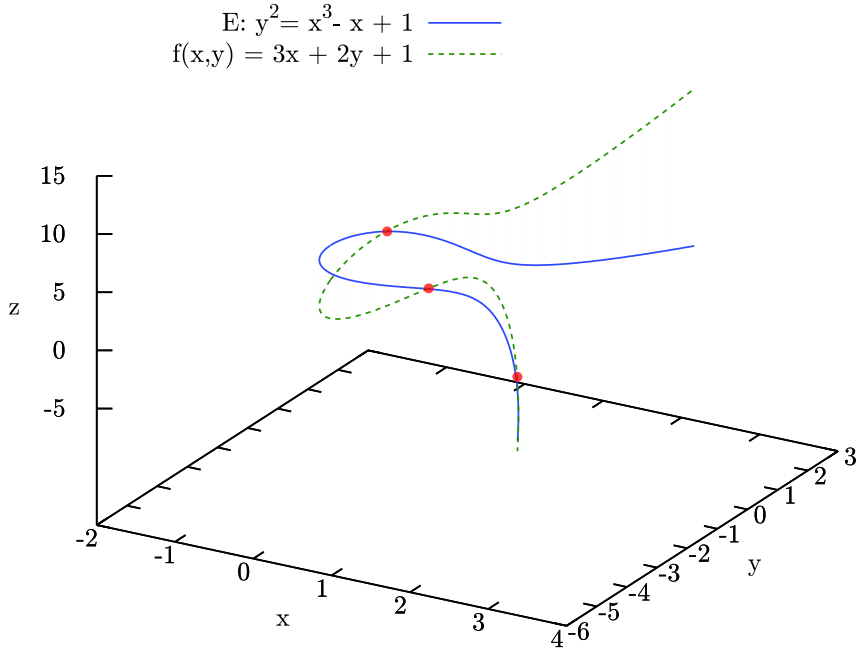


Figura 2.5: Função $f(x) = 3x + 2y + 1$ na curva elíptica E: $y^2 = x^3 - x + 1$ e seus zeros

Embora a visualização em três dimensões de funções em curvas elípticas possa facilitar o entendimento, ela não é estritamente necessária. Como estamos interessados apenas

nos zeros da função, pode-se pensar da seguinte forma. Continuando o exemplo acima, podemos achar os zeros de $f(x)$ em E primeiramente traçando o gráfico de $f(x, y) = 0$ para qualquer x, y (todos os zeros da função), para então traçar o gráfico de E e achar a intersecção de ambos. Tais pontos serão os zeros de f que também estão em E. Tal gráfico está ilustrado na figura 2.6.

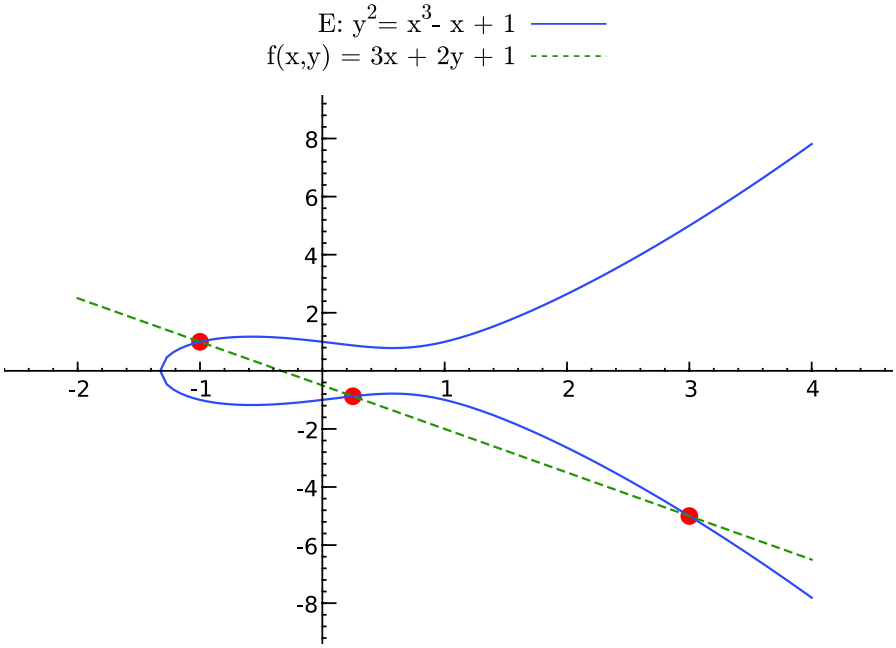


Figura 2.6: Zeros da função $f(x) = 3x + 2y + 1$, curva elíptica $E\colon y^2 = x^3 - x + 1$ e zeros da função na curva

Sabe-se que uma reta não vertical cruza uma curva elíptica em três pontos, o que confirma termos encontrado três zeros para $f(x, y)$. Porém há um detalhe adicional que será apenas citado por requerer geometria projetiva para se demonstrar. Toda reta vertical também cruza com o ponto no infinito com multiplicidade dois, e demais retas cruzam com o ponto no infinito com multiplicidade três. Tais pontos são denominados *pólos* da função. Tem-se, portanto, que no exemplo anterior f possui três pólos em ∞. Pode-se agora introduzir a definição de divisor.

Definição 2.32. *Um* divisor em uma curva elíptica $E(K)$ *consiste em uma soma formal* $a_1\langle P_1\rangle + \ldots + a_n\langle P_n\rangle$ *onde* a_i *são inteiros e* $P_i \in E(K)$.

O conceito de "soma formal" significa que o divisor é apenas um método de se listar pontos com certos inteiros associados a eles, e que não se deve calcular a soma de fato. Divisores podem ser somados e subtraídos de modo análogo a polinômios.

Definição 2.33. *O* divisor de uma função $f(x, y)$ em uma curva elíptica $E(K)$ *consiste em uma soma formal* $a_1\langle P_1\rangle + \ldots + a_n\langle P_n\rangle$ *onde* a_i *é a multiplicidade do ponto* P_i *se ele for um zero de* f *ou é o negativo da multiplicidade de* P_i *se ele for um pólo de* f*. O divisor de uma função* f *é denotado por* $\mathrm{div}(f)$*.*

Pode-se ver que o conjunto de divisores *de funções* em uma curva elíptica está contido no conjunto de divisores em uma curva elíptica. O método para se encontrar a multiplicidade de um zero está detalhado, por exemplo, em [78, Seção 11.1]; mas não é relevante para este estudo.

Exemplo 2.34. *O divisor da função* f *do Exemplo 2.31, cujos zeros eram* $(-1, 1)$*,* $(1/4, -7/8)$ *e* $(3, -5)$ *é*

$$\mathrm{div}(f) = 1\langle(-1, 1)\rangle + 1\langle(1/4, -7/8)\rangle + 1\langle(3, -5)\rangle - 3\langle\infty\rangle\,.$$

O conceito de divisor pode ser facilmente estendido para funções *racionais* em curvas elípticas.

Definição 2.35. *O* divisor de uma função racional $f(x, y) = g(x, y)/h(x, y)$ em uma curva elíptica $E(K)$*, onde* g *e* h *são funções em* $E(K)$*, é calculado por* $\mathrm{div}(f) = \mathrm{div}(g) - \mathrm{div}(h)$*.*

Em outras palavras, os zeros de uma função racional são os zeros do numerador e os pólos do denominador, e os pólos de uma função racional são os pólos do numerador e os zeros do denominador.

Exemplo 2.36. *Sejam* $g(x, y) = x - y + 2$*,* $h(x, y) = x + 1$ *e* $f(x, y) = g(x, y)/h(x, y)$ *funções em* $E(\mathbb{R})$*:* $y^2 = x^3 + 2x + 4$*. A reta* $g(x, y) = 0$ *cruza a curva elíptica nos pontos* $(0, 2)$*,* $(2, 4)$ *e* $(-1, 1)$*, portanto* $\mathrm{div}(g) = 1\langle(0, 2)\rangle + 1\langle(2, 4)\rangle + 1\langle(-1, 1)\rangle - 3\langle\infty\rangle$*. A reta (vertical)* $h(x, y) = 0$ *cruza a curva elíptica nos pontos* $(-1, 1)$ *e* $(-1, -1)$*, portanto* $\mathrm{div}(h) = 1\langle(-1, 1)\rangle + 1\langle(-1, -1)\rangle - 2\langle\infty\rangle$*. Logo,* $\mathrm{div}(f) = \mathrm{div}(g) - \mathrm{div}(h) = 1\langle(0, 2)\rangle + 1\langle(2, 4)\rangle - 1\langle(-1, -1)\rangle - 1\langle\infty\rangle$.

Por outro lado, algo similar pode ser dito para o produto de duas funções. O divisor de uma função $f(x, y) = g(x, y)h(x, y)$ é calculado por $\mathrm{div}(f) = \mathrm{div}(g) + \mathrm{div}(h)$. Isso é fácil de se observar: o produto de duas funções possui como zeros os zeros de ambas as funções; o mesmo se aplica para seus pólos.

O seguinte teorema mostra que divisores de funções racionais possuem uma forma especial.

Teorema 2.37. *[78, Proposição 11.1, Teorema 11.2] Um divisor* $a_1\langle P_1\rangle + \ldots + a_n\langle P_n\rangle$ *é o divisor* $\mathrm{div}(f)$ *de uma função racional* $f(x, y)$ *em* $E(K)$ *se e somente se* $a_1P_1 + \ldots + a_nP_n = \infty$ *e* $a_1 + \ldots + a_n = 0$*.*

2.4.5 Funções, Equações e Divisores de Retas

Como mencionado, os zeros (e por conseqüência, o divisor) de uma função $f(x, y)$ em uma curva elíptica são os pontos em que a curva $f(x, y) = 0$ intercepta a curva elíptica. Em emparelhamentos somente são utilizadas funções onde estas curvas são na verdade retas, isto é, funções da forma $ax + by + c$. Por esse motivo, nesta seção serão estudadas as equações e divisores de retas em curvas elípticas. É necessário tratar de três casos: retas verticais, tangentes e as demais.

Retas verticais

Como a equação de uma curva elíptica E/K é quadrática em y, sabe-se que uma reta vertical intercepta E/K em dois pontos, possuindo, portanto, dois zeros. Como mencionado, ela intercepta o ponto no infinito com multiplicidade dois, possuindo, portanto, dois pólos (tal fato pode também ser verificado através do Teorema 2.37: o número de pólos de uma função em uma curva elíptica deve ser igual ao número de zeros).

Definição 2.38. *Seja E/K uma curva elíptica e P um ponto em $E(K)$. A* função da reta vertical $V_P \colon K \times K \mapsto K$ *é dada por* $V_P(x, y) = x - x_P$.

A função da reta vertical possui tal forma pois $x - x_P = 0$ é a equação da reta vertical que passa por P. Pela operação de grupo na curva, sabe-se que o outro ponto de intersecção da reta com a curva elíptica é $-P$. Assim, tem-se que $\mathrm{div}(V_P) = \langle P \rangle + \langle -P \rangle - 2\langle \infty \rangle$.

Em emparelhamentos, eventualmente pode ser necessário computar $V_\infty(P)$ para algum ponto P. A princípio, pode não fazer sentido pensar na reta vertical que passa pelo infinito, mas pode-se abandonar a interpretação geométrica e pensar em divisores. Tem-se que, a partir do que acabou de ser visto, $\mathrm{div}(V_\infty) = \langle \infty \rangle + \langle -\infty \rangle - 2\langle \infty \rangle = 0$ (já que $-\infty = \infty$ e assim cancelam-se todos os termos). Uma função cujo divisor é nulo é uma função que não possui zeros e que pode ser escrita como $f(x, y) = c$ com $c \neq 0$. Escolhe-se $c = 1$ porque tal valor será irrelavante no cálculo do emparelhamento, como será visto adiante. Assim, tem-se que $V_\infty(Q) = 1$ para qualquer Q.

Retas tangentes

Como a equação de uma curva elíptica $E/K \colon y^2 = x^3 + ax + b$ é cúbica em x, sabe-se que qualquer reta não vertical intercepta a curva em três pontos, não necessariamente distintos. De fato, uma reta tangente intercepta E/K em dois pontos, um deles (onde a reta é tangente) com multiplicidade dois. Como mencionado, ela intercepta o ponto no infinito com multiplicidade três, possuindo portanto três pólos.

Definição 2.39. *Seja* $E/K\colon y^2 = x^3+ax+b$ *uma curva elíptica e* P *um ponto em* $E(K)$. *A* função da reta tangente $T_P\colon K \times K \mapsto K$ *é dada por* $T_P(x,y) = y - y_P - \lambda(x - x_P)$ *com* $\lambda = (3x_p^2 + a)/(2y_P)$.

A função da reta tangente possui tal forma pois $y - y_P - \lambda(x - x_P) = 0$ é a equação da reta tangente que passa por P, onde λ é o coeficiente angular da reta, obtido por derivação parcial da equação da curva elíptica. Pela operação de grupo na curva, sabe-se que o outro ponto de intersecção da reta com a curva elíptica é $-2P$. Tem-se que $\mathrm{div}(T_P) = 2\langle P\rangle + \langle -2P\rangle - 3\langle\infty\rangle$.

Demais retas

Sabe-se que uma reta não vertical e não tangente intercepta uma curva elíptica em três pontos e que ela intercepta o ponto no infinito com multiplicidade três, possuindo portanto três zeros e três pólos.

Definição 2.40. *Seja* E/K *uma curva elíptica e* P, Q *pontos em* $E(K)$. *A* função da reta $L_{P,Q}\colon K\times K \mapsto K$ *é dada por* $L_{P,Q}(x,y) = y-y_P-\lambda(x-x_P)$ *com* $\lambda = (y_Q-y_P)/(x_Q-x_P)$.

A função da reta possui tal forma pois $y - y_P - \lambda(x - x_P) = 0$ é a equação da reta que passa por P e Q, onde λ é o coeficiente angular da reta, obtido pela tangente. Pela operação de grupo na curva, verifica-se que o outro ponto de intersecção da reta com a curva elíptica é $-(P+Q)$. Assim, tem-se que $\mathrm{div}(L_{P,Q}) = \langle P\rangle+\langle Q\rangle+\langle -(P+Q)\rangle-3\langle\infty\rangle$.

Deve-se ressaltar que pode-se referir às três funções definidas acima, de modo genérico, por "funções das retas".

2.4.6 Emparelhamento de Tate

Agora é possível definir emparelhamentos de forma concreta. O primeiro emparelhamento utilizado para fins criptográficos foi o de Weil [79], porém logo passou a se utilizar o emparelhamento de Tate [74] por ser mais eficiente, e ele será definido a seguir. A forma moderna do emparelhamento de Tate que será descrita possui contribuições de Lichtenbaum [41] e portanto é às vezes denominada emparelhamento de Tate-Lichtenbaum; mas por motivos de clareza será usada a primeira denominação. Também é utilizada a simplificação descrita em [4, Teorema 1].

Definição 2.41. *O* emparelhamento de Tate $\tau_r\colon E(\mathbb{F}_{q^k})[r] \times E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k}) \to \mu_r$ *é definido por* $\tau_r(P,Q) = f_\tau(Q)^{(q^k-1)/r}$ *onde* f_τ *é uma função racional com* $\mathrm{div}(f_\tau) = r\langle P\rangle - r\langle\infty\rangle$.

Teorema 2.42. *[78, Teorema 11.8] O emparelhamento de Tate é bilinear e não degenerado.*

Primeiramente, será necessário o seguinte teorema para justificar qual valor de r e qual domínio do emparelhamento são utilizados na prática.

Teorema 2.43. *Seja $E/\mathbb{F}_q$ uma curva elíptica, k o grau de mergulho de $\mathbb{F}_q$ em relação a r e n a ordem de $E(\mathbb{F}_q)$. Se r é primo, $r \mid n$ e $r^2 \nmid n$, então a ordem de $E(\mathbb{F}_q)[r]$ é r.*

Demonstração. Seja x a ordem de $E(\mathbb{F}_q)[r]$. Pode-se verificar que $E(\mathbb{F}_q)[r]$ é um subgrupo de $E(\mathbb{F}_{q^k})[r]$ e de $E(\mathbb{F}_q)$. Como a ordem de $E(\mathbb{F}_{q^k})[r]$ é r^2, tem-se que (i) $x \mid r^2$ e (ii) $x \mid n$ pelo Teorema 2.2. Como r é primo, então de acordo com (i), $x \in \{1, r, r^2\}$. Se $x = 1$, então $E(\mathbb{F}_q)[r] = \{\infty\}$, mas de acordo com o Teorema 2.4, existe algum $a \in E(\mathbb{F}_q)$ de ordem r e assim $a \neq \infty$ e $a \in E(\mathbb{F}_q)[r]$, uma contradição. Se $x = r^2$, então de acordo com (ii), $r^2 \mid n$, contradizendo a premissa. Portanto, $x = r$. □

Assim, a escolha de r e do domínio de τ_r partem do seguinte raciocínio. Seja n a ordem de $E(\mathbb{F}_q)$. Segundo o Teorema 2.20, para calcular o grau de mergulho facilmente, r deve ser primo e $r \mid n$ mas $r \nmid q - 1$. De acordo com a definição de emparelhamento da seção 2.4, o grupo $\mathbb{G}_1$ deve ter ordem prima r. Contudo, $E(\mathbb{F}_{q^k})[r]$ tem ordem r^2, então deve-se usar como $\mathbb{G}_1$ um subgrupo de $E(\mathbb{F}_{q^k})[r]$ de ordem r. A escolha mais simples e eficiente é $\mathbb{G}_1 = E(\mathbb{F}_q)[r]$. De acordo com o Teorema 2.43, tal grupo possui r elementos se $r^2 \nmid n$. Finalmente, como a segurança de $\mathbb{G}_1$ depende da sua ordem, é natural escolher o maior r possível dentro das condições listadas. Idealmente, $r = n$, mas isso nem sempre é possível. Neste caso, como $r \mid n$, escreve-se $n = rh$, onde h é denominado *cofator* da curva. Portanto, condensando todos os requisitos, r é escolhido como o maior divisor primo de n tal que $r \nmid q - 1$ e $r^2 \nmid n$.

Note que quanto menor o cofator, mais eficiente é o sistema; suponha que $\rho = \log_2 n / \log_2 r$ seja próximo de 2, por exemplo. Neste caso, seria necessário usar uma curva elíptica sobre um corpo aproximadamente do dobro do tamanho necessário se ρ fosse próximo de 1 (o ideal).

O fato do segundo grupo ser $E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$ (as classes laterais de $rE(\mathbb{F}_{q^k})$ em $E(\mathbb{F}_{q^k})$, ver [68, Seção 6.3]) implica, na prática, que qualquer elemento de $E(\mathbb{F}_{q^k})$ pode servir como segundo parâmetro pois todo elemento é um representativo de uma determinada classe lateral. Naturalmente, todos os elementos de uma mesma classe lateral resultarão no mesmo valor quando emparelhados com algum determinado ponto qualquer do primeiro grupo. Contudo, é importante ressaltar que o ponto Q não pode ser linearmente dependente de P, pois nesse caso pode-se escrever $P = kG$ e $Q = \ell G$ para algum ponto G e inteiros k, ℓ e portanto $\tau_r(P, Q) = \tau_r(G, G)^{k\ell}$; mas o emparelhamento de Tate é indefinido em $e(G, G)$ para G qualquer. Assim, toma-se $\mathbb{G}_2 = E(\mathbb{F}_{q^k}) \setminus E(\mathbb{F}_q)$. Posteriormente, será estudada uma outra opção para $\mathbb{G}_2$ com exatamente r elementos.

Sem a exponenciação por $(q^k - 1)/r$ (denominada *exponenciação final*), ainda se teria um emparelhamento válido, mas o resultado seria um representante de uma classe lateral

em $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^r$. A exponenciação faz com que se obtenha um elemento canônico da classe lateral.

2.4.7 Algoritmo de Miller

Falta apenas saber como calcular $f_\tau(Q)$ onde f_τ é uma função racional com $\mathrm{div}(f_\tau) = r\langle P\rangle - r\langle\infty\rangle$. Lembrando-se da discussão sobre divisores, a estratégia natural seria descobrir a equação da função f_τ (já que, por definição, temos seu divisor e logo sabemos seus zeros e pólos) e então calcular $f_\tau(x_Q, y_Q)$. É possível descobrir a função dado seu divisor, porém com um número de passos proporcional a r, o que é inviável (a segurança do emparelhamento depende de r, portanto ele é grande).

A idéia central do algoritmo de Miller [49] é a seguinte. Seja $f_{i,P}(Q)$ uma função racional cujo divisor é $\mathrm{div}(f_{i,P}) = i\langle P\rangle - \langle iP\rangle - \langle(i-1)\infty\rangle$. Tal função é denominada *função de Miller*. Primeiramente, note que $f_{r,P}(Q) = f_\tau(Q)$, pois $rP = \infty$. Segundo, note que $f_{1,P}(Q) = 1$, pois $\mathrm{div}(f_{1,P}) = \langle P\rangle - \langle P\rangle = 0$. Uma função sem zeros pode ser dada por uma constante não nula; novamente escolhemos 1 para simplificar os cálculos. A vantagem das funções $f_{i,P}$ é que se pode calculá-las a partir de funções parametrizadas por valores menores que i, como é demonstrado no seguinte teorema. A seguir, por brevidade, se escreverá f_i para denotar a função $f_{i,P}$.

Teorema 2.44.

$$f_{i+j}(Q) = f_i(Q)f_j(Q)\frac{L_{iP,jP}(Q)}{V_{(i+j)P}(Q)}\,.$$

Demonstração. Como visto, para mostrar que duas funções são iguais (salvo um fator constante), pode-se mostrar que elas têm o mesmo divisor. Assim, tem-se que

$$\begin{aligned}
\mathrm{div}(f_i f_j \frac{L_{iP,jP}}{V_{(i+j)P}}) =& \mathrm{div}(f_i) + \mathrm{div}(f_j) + \mathrm{div}(L_{iP,jP}) - \mathrm{div}(V_{(i+j)P}) \\
=& \{i\langle P\rangle - \langle iP\rangle - (i-1)\langle\infty\rangle\} \\
&+ \{j\langle P\rangle - \langle jP\rangle - (j-1)\langle\infty\rangle\} \\
&+ \{\langle iP\rangle + \langle jP\rangle + \langle -(i+j)P\rangle - 3\langle\infty\rangle\} \\
&- \{\langle(i+j)P\rangle + \langle -(i+j)P\rangle - 2\langle\infty\rangle\} \\
=& (i+j)\langle P\rangle - \langle(i+j)P\rangle - (i+j-1)\langle\infty\rangle \\
=& \mathrm{div}(f_{i+j})
\end{aligned}$$

□

A princípio, não fica claro se duas funções iguais *salvo um fator constante* são o suficiente. Como será visto adiante, a exponenciação final torna irrelevante qualquer fator em $\mathbb{F}_q$, portanto tal condição basta.

Versão "ingênua"

Uma primeira tentativa para o cálculo de $f_r(Q)$ seria pensar indutivamente. Tendo-se $f_1(Q)$ e um jeito de calcular $f_k(Q)$ com base em $f_{k-1}(Q)$, consegue-se facilmente calcular $f_r(Q)$. Verifica-se que

$$f_k(Q) = f_{k-1}(Q)f_1(Q)\frac{L_{(k-1)P,P}(Q)}{V_{kP}(Q)},$$

e portanto, como $f_1(Q) = 1$,

$$f_r(Q) = \frac{L_{1P,P}(Q)}{V_{2P}(Q)} \cdot \frac{L_{2P,P}(Q)}{V_{3P}(Q)} \cdots \frac{L_{(r-1)P,P}(Q)}{V_{rP}(Q)}$$

Apesar de correto este método continua sendo inviável, pois é necessário calcular o produto de $r-1$ funções racionais. Contudo, note que se pode calcular não apenas $f_{i+1}(Q)$ baseado em $f_i(Q)$, mas também f_{i+j} baseado em $f_i(Q)$ e $f_j(Q)$, como descrito originalmente. Pode-se então aperfeiçoar o cálculo de $f_r(Q)$ utilizando-se o mesmo método de *square-and-multiply* (elevar ao quadrado e multiplicar) ou *double-and-add* (dobrar e somar) utilizado, por exemplo, em exponenciações modulares e multiplicação de pontos de curvas elípticas por inteiros.

Versão aperfeiçoada

Pode-se verificar que a partir da fórmula de $f_{i+j}(Q)$ chega-se em:

$$f_k(Q) = f_{k/2}(Q)^2\frac{T_{(k/2)P}(Q)}{V_{kP}(Q)}$$
$$f_k(Q) = f_{k-1}(Q)\frac{L_{(k-1)P,P}(Q)}{V_{kP}(Q)}.$$

Com elas é possível escrever o algoritmo aperfeiçoado. Primeiramente, descreve-se a fórmula para calcular f_r recursivamente, por ser de fácil entendimento:

$$f_r(Q) = \begin{cases} 1 & \text{se } r = 1; \\ f_{r/2}(Q)^2\frac{T_{(r/2)P}(Q)}{V_{rP}(Q)} & \text{se } r \text{ par}; \\ f_{r-1}(Q)\frac{L_{(r-1)P,P}(Q)}{V_{rP}(Q)} & \text{se } r \text{ ímpar}. \end{cases}$$

Como um algoritmo iterativo é mais eficiente para implementação, chega-se ao Algoritmo 2.1, que é classificado como *left-to-right* por processar os bits de r da esquerda para a direita.

Algoritmo 2.1 Algoritmo de Miller

Entrada: $P \in E(\mathbb{F}_{q^k})[r]$, $Q \in E(\mathbb{F}_{q^k})$, $r \in \mathbb{N}$
Saída: $f_{r,P}(Q) \in \mathbb{F}_{q^k}$

1: $(r_n, r_{n-1}, \ldots, r_0)_2 \leftarrow r$
2: $x \leftarrow 1$
3: $Z \leftarrow P$
4: **for** $i = n - 1$ **to** 0 **do**
5: $\quad x \leftarrow x^2 \cdot T_Z(Q)/V_{2Z}(Q)$
6: $\quad Z \leftarrow 2Z$
7: $\quad$ **if** $r_i = 1$ **then**
8: $\quad\quad x \leftarrow x \cdot L_{Z,P}(Q)/V_{Z+P}(Q)$
9: $\quad\quad Z \leftarrow Z + P$
10: $\quad$ **end if**
11: **end for**
12: **return** x

Note que o bit mais significativo de r não é processado. Isso acontece porque algoritmos no estilo dobrar e somar calculam a partir do índice 0, que no nosso caso seria f_0; mas pode-se começar logo de f_1.

Com tal algoritmo, tem-se finalmente o Algoritmo 2.2 para o cálculo do emparelhamento de Tate.

Algoritmo 2.2 Emparelhamento de Tate

Entrada: $P \in E(\mathbb{F}_{q^k})[r]$, $Q \in E(\mathbb{F}_{q^k})$, $r \in \mathbb{Z}$
Saída: $\tau_r(P, Q) \in \mu_r$

$x \leftarrow f_{r,P}(Q)$
$x \leftarrow x^{(q^k-1)/r}$
return x

2.5 Famílias de Curvas

Gerar curvas elípticas para uso em emparelhamentos é um trabalho razoavelmente complexo. Primeiramente, deve-se escolher $E(\mathbb{F}_q)$ de modo que sua ordem seja prima ou que tenha um fator primo grande, para garantir a segurança desejada (o impacto do nível de segurança em tais parâmetros será melhor especificado adiante). Pode-se medir a "eficiência" da curva neste quesito usando-se o valor $\rho = \log_2 q / \log_2 r$. Se a ordem da curva for prima, então $\rho = 1$. Existem muitas curvas com $\rho = 2$, neste caso, é necessário um corpo base com o dobro do tamanho necessário, gerando um certo desperdício computacional.

Segundo, a curva deve possuir um grau de mergulho compatível com o nível de segurança alvo. Por último, é desejável a existência de *twists* quárticos ou sêxticos, como será explicado posteriormente.

Por esse motivo, há muita pesquisa ativa no sentido de se criar famílias de curvas elípticas que atendam a certos requisitos específicos. Freeman et al. fizeram um apanhado de tais famílias e as classificaram em certas categorias [23]. Nesta dissertação serão utilizadas duas destas famílias: curvas BN e curvas MNT, que serão descritas a seguir.

2.5.1 Curvas MNT

Em 2001, Miyaji et al. [51] propuseram famílias de curvas ordinárias de ordem prima com grau de mergulho $k = 3, 4, 6$. Tais famílias são classificadas como esparsas, já que contêm um número pequeno de curvas; e também como paramétricas, porque o valores n (ordem da curva), p (ordem do corpo finito subjacente) e t (traço de Frobenius) são definidos como polinômios sobre um parâmetro x. Posteriormente, Scott e Barreto [63] e Galbraith et al. [24] estenderam tais famílias para permitir pequenos cofatores, de modo a aumentar o número de curvas disponíveis.

Para cada k, existem duas maneiras de se construir curvas MNT clássicas (de ordem prima). Assim, cada família receberá um nome, por clareza, e estão listadas a seguir:

MNT3 $k = 3$, $p(x) = 12x^2 - 1$, $n(x) = 12x^2 - 6x + 1$, $t(x) = 6x - 1$

MNT4A $k = 4$, $p(x) = x^2 + x + 1$, $n(x) = x^2 + 2x + 2$, $t(x) = -x$

MNT4B $k = 4$, $p(x) = x^2 + x + 1$, $n(x) = x^2 + 1$, $t(x) = x + 1$

MNT6 $k = 6$, $p(x) = 4x^2 + 1$, $n(x) = 4x^2 - 2x + 1$, $t(x) = 2x + 1$

Já para curvas MNT com cofatores pequenos as construções são inúmeras e refere-se a [24] para a lista completa. Todas curvas MNT possuem $\rho \approx 1$ e apenas *twists* quadráticos.

Exemplo 2.45. *O valor* $x = -4274916562956307883771 83$*, na família MNT6, permite gerar a curva elíptica* $E/\mathbb{F}_p$: $y^2 = x^3 - 3x + b$ *com* n *pontos e traço* t*, onde*

```
p = 0x800B03E122FBD10A433C8536298305719D9CBD05
n = 0x800B03E122FBD10A433D3A42E680CF76AA8531C3
t = -0xB50CBCFDCA050CE874BF
b = 0x2D851351189BA68859F3A34893D0512B2F025BF6.
```

Como será esclarecido adiante, o grupo $E(\mathbb{F}_p)$ *fornece 80 bits de segurança já que* n *é primo e tem 160 bits. Contudo, o subgrupo multiplicativo de* $\mathbb{F}_{p^6}$ *fornece aproximadamente 70 bits de segurança pois sua ordem,* $p^6 - 1$*, possui cerca de* $160 \cdot 6 = 960$ *bits.*

É importante notar que poucos valores de x geram curvas de interesse — é preciso testar se o $p(x)$ é primo e se $n(x)$ possui um fator primo grande.

2.5.2 Curvas BN

Em 2006, Barreto e Naehrig [5] propuseram uma família paramétrica densa de curvas de ordem prima com grau de mergulho $k = 12$, listada a seguir.

- $p(x) = 36x^4 + 36x^3 + 24x^2 + 6x + 1$
- $n(x) = 36x^4 + 36x^3 + 18x^2 + 6x + 1$
- $t(x) = 6x^2 + 1$

Tal família possui $\rho \approx 1$ e *twists* sêxticos, e é bastante numerosa. Isso permite escolher valores de x com peso de Hamming pequeno, o que é favorável para o desempenho do algoritmo de Miller, como será visto posteriormente. É importante notar que em alguns artigos [17, 57] são utilizadas funções considerando o valor oposto de x ($-x$); por exemplo, $p(x) = 36x^4 - 36x^3 + 24x^2 - 6x + 1$. Naturalmente, isso não influencia o resultado de qualquer computação, apenas a interpretação de valores de x sugeridos. Em particular, as fórmulas do emparelhamento Xate descrito em [57] ficam um pouco mais simples usando essa representação alternativa. Contudo, neste trabalho, será utilizada a definição original das funções das curvas BN.

Exemplo 2.46. *O valor* $x = 6922032695987994625$*, na família BN, permite gerar a curva elíptica* $E/\mathbb{F}_p$: $y^2 = x^3 + 3$ *com* n *pontos e traço* t*, onde*

```
p = 0xB6B99EDD1CF5DAD049717415E1CE764FA5E7C61F7FC6A9D6FF54147C01320067
n = 0xB6B99EDD1CF5DAD049717415E1CE764ECD9FBFD773C223127DD413B601260061
t = 0xD84806480C0486C4818000C6000C0007.
```

Como será esclarecido adiante, o grupo $E(\mathbb{F}_p)$ *fornece 128 bits de segurança já que* n *é primo e tem 256 bits. O subgrupo multiplicativo de* $\mathbb{F}_{p^{12}}$ *também fornece aproximadamente 128 bits de segurança pois sua ordem,* $p^6 - 1$*, possui cerca de* $256 \cdot 12 = 3072$ *bits.*

2.6 Protocolos

Nesta seção, serão estudados alguns protocolos de Criptografia de Curvas Elípticas (CCE) e de Criptografia Baseada em Emparelhamentos (CBE). Tais protocolos são as aplicações finais de todo o conteúdo discutido nesta dissertação, portanto é de suma importância a sua compreensão.

2.6.1 Elliptic Curve Digital Signature Algorithm

Embora o foco principal deste trabalho sejam protocolos baseados em identidades, é interessante estudar a Criptografia de Curvas Elípticas por ser construída sobre os mesmos fundamentos da CBE e oferecer uma implementação eficiente de criptografia assimétrica, ignorando-se o problema da autenticação de chaves públicas. Nesta seção será descrito o *Elliptic Curve Digital Signature Algorithm*, ou ECDSA [56]. Este é um protocolo de assinatura digital que é uma variante baseada em curvas elípticas do DSA, que por sua vez é um algoritmo baseado no subgrupo multiplicativo de corpos finitos proposto pelo *National Institute of Standards and Technology* (NIST).

O funcionamento do esquema é o seguinte. Primeiramente, inicializa-se o sistema de acordo com o Algoritmo 2.3, onde são especificados os parâmetros públicos conhecidos por todos os participantes. Cada participante A deve gerar seu par de chaves d_A, Q_A de acordo com o Algoritmo 2.4. Suponha agora que Alice deseja assinar um mensagem M. Para tanto, ela executa o Algoritmo 2.5 com M e sua chave privada d_A, gerando uma assinatura (r, s). Agora suponha que Beto deseja verificar se a assinatura gerada por Alice é válida. Ele deve executar o algoritmo 2.6 com (r, s) e a chave pública Q_A de Alice e assim saberá se a assinatura é válida ou não.

Algoritmo 2.3 ECDSA: Inicialização

Entrada: nível de segurança t
Saída: parâmetros globais $E(\mathbb{F}_q)$, G, n, H
1: Escolha $E(\mathbb{F}_q)$, $G \in E(\mathbb{F}_q)$ com segurança t
2: $n \leftarrow$ ordem do ponto G
3: Escolha uma função de *hash* $H\colon \{0,1\}^* \rightarrow [0, n-1]$ com segurança t
4: **return** $E(\mathbb{F}_q)$, G, n, H

Algoritmo 2.4 ECDSA: Geração de chaves

Entrada: parâmetros globais
Saída: chave privada d_A, chave pública Q_A
1: Entidade A escolhe aleatoriamente $d_A \in [1, n-1]$
2: $Q_A \leftarrow d_A G$
3: **return** d_A, Q_A

Alguns detalhes foram omitidos dos algoritmos, mas são de extrema importância; qualquer implementação real deve ter como referência padrões como [56].

A principal operação na geração de chaves é a multiplicação do ponto G. Como tal ponto é fixo para todos os participantes do esquema, é possível usar tabelas pré-computadas para acelerar o cálculo, como será descrito posteriormente. Na assinatura, a

Algoritmo 2.5 ECDSA: Assinatura

Entrada: parâmetros globais, mensagem M, chave privada d_A
Saída: assinatura (r, s)

1: Escolha aleatoriamente $k \in [1, n-1]$
2: $R = (x_R, y_R) \leftarrow kG$
3: $r \leftarrow x_R \bmod n$
4: $e \leftarrow H(M)$
5: $s \leftarrow k^{-1}(e + rd_A) \bmod n$
6: **return** (r, s)

Algoritmo 2.6 ECDSA: Verificação

Entrada: parâmetros globais, mensagem M, assinatura (r, s), chave pública Q_A
Saída: válida ou inválida

1: $e \leftarrow H(M)$
2: $u_1 \leftarrow es^{-1} \bmod n$
3: $u_2 \leftarrow rs^{-1} \bmod n$
4: $R = (x_R, y_R) \leftarrow u_1 G + u_2 Q_A$
5: $v \leftarrow x_R \bmod n$
6: **return** $v == r$

principal operação é a multiplicação do ponto G, e o mesmo raciocínio se aplica. Também são necessárias multiplicação e inversão módulo n. Finalmente, para a verificação de assinatura, é necessária a multiplicação de G e do ponto correspondente à chave pública do autor da assinatura. A multiplicação de G pode ser otimizada como mencionado, mas também é possível realizar as duas multiplicações simultaneamente de modo mais eficiente do que fazendo-as em separado. Ainda assim, a verificação é a operação mais cara das três. Também são necessárias multiplicações e inversões módulo n.

A segurança do ECDSA depende de vários fatores [7]. O principal é que o problema do logaritmo discreto em $E(\mathbb{F}_q)$ seja difícil, isto é, dados $Q_A = d_A G$ e G, seja difícil computar d_A. Normalmente, mede-se a segurança de um problema em comparação ao tempo necessário para se quebrar uma cifra simétrica por busca exaustiva. Assim, por exemplo, esquemas com nível de segurança de 80 bits fornecem a mesma segurança que um algoritmo simétrico de 80 bits. Um subgrupo gerado por um ponto cuja ordem r tem k bits fornece $k/2$ bits de segurança; tal valor está relacionado ao fato que o melhor algoritmo para resolver o problema do logaritmo discreto em uma curva elíptica possui complexidade $O(2^{k/2})$ [59]. Normalmente são utilizadas curvas de ordem n prima, portanto $r = n$ e r possui tamanho próximo ao tamanho do corpo subjacente. A Tabela 2.3, na próxima seção, também fornece um resumo dos tamanhos recomendados.

2.6.2 Non-Interactive Key Distribution Protocol

O grande problema da criptografia simétrica é a distribuição de chaves entre os participantes. A criptografia assimétrica de certo modo resolveu tal problema com protocolos de acordo de chaves que permitem dois participantes, usando um canal inseguro, combinar uma chave para posteriormente usar criptografia simétrica (que é mais eficiente) em sua comunicação pelo resto da sessão. Ainda assim, seria interessante um protocolo que não requeresse nenhuma comunicação entre os dois participantes: apenas com a identidade um do outro, esse protocolo permitiria que eles combinassem uma chave.

Evidentemente tal protocolo não interativo de distribuição de chaves (*non-interactive key distribution protocol*, NIKDP) requer a criptografia baseada em identidades. De fato, um protocolo com esse fim foi criado em um dos primeiros artigos a concretizar a CBI [60]. O NIKDP foi generalizado em [19], e tal versão será descrita a seguir.

Primeiramente, o esquema é inicializado com o Algoritmo 2.7, que consiste apenas na escolha de parâmetros públicos que serão conhecidos por todos os participantes do sistema. O centro de geração de chaves gera sua chave mestra com o Algoritmo 2.8. Cada participante obtém seu par de chaves privadas do centro de geração de chaves, que executa o Algoritmo 2.9 para tal fim. É importante ressaltar que o envio das chaves para o participante deve ser feito de forma confidencial e autenticada. Em redes de sensores sem fio, tal "envio" consiste na gravação das chaves na ROM dos sensores, que deve ser realizada em lugar seguro e longe de qualquer atacante.

Finalmente, suponha que Alice (A) e Beto (B) desejam combinar uma chave, e que suas identidades sejam ID_A = "`alice@cryptoland.net`" e ID_B = "`beto@cryptoland.net`". Para tanto, munidos apenas da identidade um do outro e de suas respectivas chaves privadas, eles executam o Algoritmo 2.10 e obtêm, cada um, um par de valores. Alice obtém o par $k_{1\mathrm{A}}$, $k_{2\mathrm{A}}$ e Beto obtém o par $k_{1\mathrm{B}}$, $k_{2\mathrm{B}}$.

Algoritmo 2.7 NIKDP: Inicialização

Entrada: nível de segurança t
Saída: parâmetros globais $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, r, H_1, H_2
1: Escolha $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ de ordem r, com segurança t
2: Escolha funções *hash* $H_1 \colon \{0,1\}^* \to \mathbb{G}_1$ e $H_2 \colon \{0,1\}^* \to \mathbb{G}_2$ com segurança t
3: **return** $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, r, H_1, H_2

Ainda assim, existe um problema não resolvido, que consiste em como derivar uma chave comum a partir destes pares de valores. Pode-se verificar que, pela bilinearidade, $k_{1\mathrm{A}} = k_{2\mathrm{B}}$ e $k_{1\mathrm{B}} = k_{2\mathrm{A}}$. Uma alternativa é cada um multiplicar seu par de valores para obter um valor final; é fácil verificar que $k_{1\mathrm{A}}k_{2\mathrm{A}} = k_{1\mathrm{B}}k_{2\mathrm{B}}$. Porém, isso exige a computação efetiva do par de valores, e portanto o cálculo de dois emparelhamentos, o que é caro. Em [72], é sugerido que, com uma pequena quantidade de comunicação, as duas

Algoritmo 2.8 NIKDP: Geração da chave mestra

Entrada: parâmetros globais
Saída: chave mestra s
1: Escolha aleatoriamente $s \in [1, r-1]$
2: **return** s

Algoritmo 2.9 NIKDP: Distribuição de chave privada

Entrada: parâmetros globais, identidade $\mathrm{ID_A} \in \{0,1\}^*$
Saída: par de chaves privadas $S_{1\mathrm{A}}$, $S_{2\mathrm{A}}$
1: $S_{1\mathrm{A}} \leftarrow sH_1(\mathrm{ID_A}) \in \mathbb{G}_1$
2: $S_{2\mathrm{A}} \leftarrow sH_2(\mathrm{ID_A}) \in \mathbb{G}_2$
3: **return** $S_{1\mathrm{A}}$, $S_{2\mathrm{A}}$

entidades podem escolher um dos valores; por exemplo, Alice computa apenas $k_{1\mathrm{A}}$ e Beto computa apenas $k_{2\mathrm{B}}$. Contudo, isso naturalmente acabaria com o aspecto não interativo do protocolo. Uma alternativa simples (mas que não se encontrou na literatura) é que cada participante P computa $k_{1\mathrm{P}}$ caso sua identidade seja menor lexicograficamente do que a identidade do outro participante, ou computa $k_{2\mathrm{P}}$ caso contrário. Por exemplo, Alice computaria $k_{1\mathrm{A}}$ e Beto computaria $k_{1\mathrm{B}}$ já que "`alice@cryptoland.net`" < "`beto@cryptoland.net`". Vale a pena ressaltar que tal problema não existe em emparelhamentos do tipo 1 (simétricos) que não são abordados neste trabalho, pois tem-se que $k_{1\mathrm{A}} = k_{2\mathrm{A}}$.

A segurança do NIKDP é equivalente ao problema Diffie-Hellman bilinear (*bilinear Diffie-Hellman problem*, BDH), como demonstrado em [19]. O BDH consiste em, dados P, Q, aP, bQ, cP, cQ computar $e(P,Q)^{abc}$, com $P \in \mathbb{G}_1, Q \in \mathbb{G}_2, a, b, c \in \mathbb{Z}$. Vale notar que, como na maioria dos protocolos baseados em emparelhamentos, quebrando-se o problema do logaritmo discreto em $\mathbb{G}_1$, $\mathbb{G}_2$ ou $\mathbb{G}_T$ também quebra-se o protocolo. Como $\mathbb{G}_1$ é normalmente um subgrupo de ordem r de uma curva elíptica $E(\mathbb{F}_q)$, sua segurança depende do valor de r, fornecendo $r/2$ bits de segurança como mencionado para o ECDSA. O grupo $\mathbb{G}_2$ segue uma abordagem análoga. Já o grupo $\mathbb{G}_T = \mathbb{F}^*_{q^k}$ é estudado de forma diferente. Enquanto que para curvas elípticas só existem ataques exponenciais ao problema

Algoritmo 2.10 NIKDP: Distribuição de chaves

Entrada: parâmetros globais, identidade $\mathrm{ID_B}$ da outra entidade B,
par de chaves privadas $S_{1\mathrm{A}}$, $S_{2\mathrm{A}}$
Saída: chave combinada $k_{1\mathrm{A}}$, $k_{2\mathrm{A}}$
1: $k_{1\mathrm{A}} \leftarrow e(S_{1\mathrm{A}}, H_2(\mathrm{ID_B}))$
2: $k_{2\mathrm{A}} \leftarrow e(H_1(\mathrm{ID_B}), S_{2\mathrm{A}})$
3: **return** $k_{1\mathrm{A}}$, $k_{2\mathrm{A}}$

Segurança (bits)	r (bits)	q^k (bits)	k ideal, $\rho = 1$
64	128	640	5
70	125	952	7
80	160	1024	7
128	256	3072	12

Tabela 2.3: Tamanhos recomendados para r e q^k

do logaritmo discreto (sem levar em conta o ataque usando os próprios emparelhamentos, que transfere o problema para $\mathbb{G}_T$ [48], e ataques para curvas específicas), para corpos finitos existem algoritmos sub-exponenciais como o cálculo de índices [1]. Por esse motivo, o tamanho de $\mathbb{F}^*_{q^k}$ precisa ser razoavelmente maior que r — idealmente, exatamente k vezes maior. Os tamanhos recomendados de r e q^k em [38, 39, 55] estão listados na Tabela 2.3, junto com o k ideal para curvas com $\rho = 1$ (logo com $r \approx q$), calculado como $k = \lceil \log_2 q^k / \log_2 r \rceil$. Quando não é possível usar uma curva com o k ideal, tem-se três opções: usar um k menor e um r maior que o necessário, usar um k maior e portanto um q^k maior que o necessário, ou aceitar uma perda de segurança em $\mathbb{G}_1$ ou $\mathbb{G}_T$ (e portanto em todo o esquema).

Capítulo 3

Algoritmos

Neste capítulo, serão estudadas técnicas computacionais para a implementação de protocolos de CCE e CBE. Basicamente, tais técnicas podem ser divididas em vários níveis, ilustrados na Figura 3.1. Os níveis mais baixos são compostos por algoritmos para a aritmética de corpos finitos; que basicamente envolve soma, subtração, multiplicação e inversão em $\mathbb{F}_{q^m}$. Acima deles, tem-se a aritmética de curvas elípticas, que consiste basicamente na adição de pontos e na multiplicação escalar de ponto (adição repetida). Acima, tem-se o nível do cálculo de emparelhamentos, cujo algoritmo principal é o algoritmo de Miller. No topo estão os protocolos sendo utilizados.

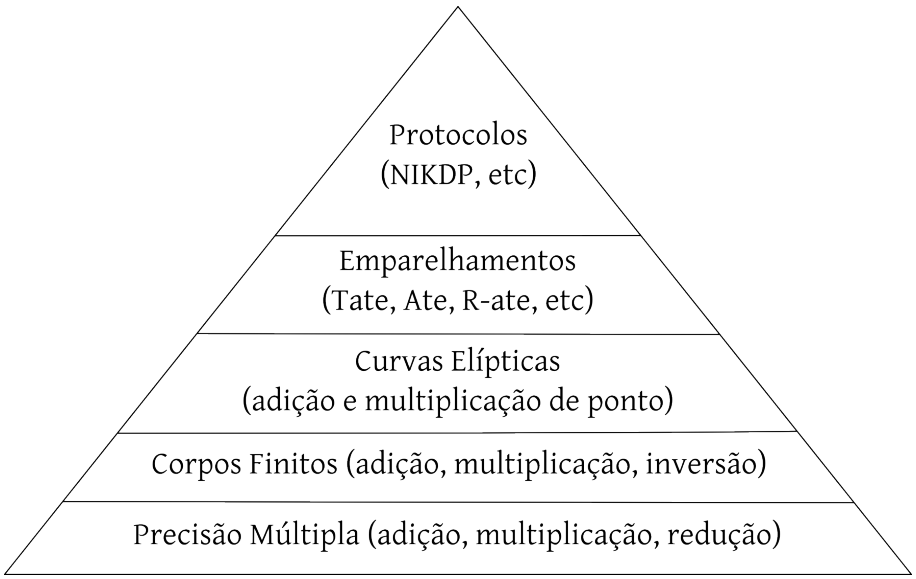


Figura 3.1: Estrutura de uma implementação de Criptografia Baseada em Emparelhamentos

3.1 Introdução à Aritmética de Corpos Finitos Primos

Os números utilizados em protocolos criptográficos têm tamanhos muito maiores do que os tamanhos dos inteiros nativos de grande parte das plataformas computacionais. Assim, é necessária uma forma de se representar tais números usando esses inteiros nativos.

Definição 3.1. *Dada uma base B (inteiro maior que 0), todo inteiro a pode ser representado por uma tupla $(a_{n-1}, a_{n-2}, \ldots, a_0)_B$, tal que $a = a_{n-1}B^{(n-1)} + a_{n-1}B^{(n-2)} + a_1B + a_0$, com $a_i \in [0, B-1]$. Essa representação é denominada* precisão múltipla. *Cada coeficiente a_i é também denominado* dígito.

O modo mais simples de se representar $\mathbb{F}_p$ é através de $\mathbb{Z}_p$, isto é, os números no intervalo $[0, p-1]$. A operação de soma no corpo consiste na soma módulo p e a operação de multiplicação no corpo consiste na multiplicação módulo p. Cada elemento de $a \in \mathbb{F}_p$ é representado em software através de $(a_{n-1}, a_{n-2}, \ldots, a_0)_B$, onde $n = \lceil (\lfloor \log_2 p \rfloor + 1)/W \rceil$, $B = 2^W$ e W é o número de bits do inteiro nativo da plataforma alvo. Por exemplo, um elemento do corpo finito $\mathbb{F}_p$, com p de 256 bits, pode ser representado através de 8 dígitos em uma plataforma de 32 bits.

As operações de soma e subtração em um corpo finito são razoavelmente simples. Basicamente, a soma de dois elementos do corpo finito consiste na soma dígito por dígito dos dois operandos, com propagação de *carry*, seguida da subtração de p caso o resultado seja maior que p. A subtração de dois elementos do corpo finito consiste na subtração dígito por dígito dos dois operandos, com propagação de *borrow*, seguida da adição de p caso o resultado seja menor que zero.

A multiplicação em um corpo finito, por outro lado, é um pouco mais complexa e crucial para o desempenho dos protocolos. Esquemas de CCE e CBE passam cerca de 75% do seu tempo de computação na multiplicação em corpo finito; por esse motivo é fundamental que ela seja implementada da forma mais eficiente possível. De modo geral, a multiplicação em $\mathbb{F}_p$ consiste na multiplicação dos dois operandos de n dígitos da forma usual, seguida da redução do resultado de $2n$ dígitos módulo p, de forma a obter um número de n dígitos novamente. Os algoritmos de multiplicação e redução serão estudados nas próximas seções.

A última operação necessária em corpos finitos é a inversão, isto é, dado um elemento a, calcular c tal que $ac = 1$. Tal cálculo pode ser realizado com variantes do algoritmo de Euclides. Como tal operação não é crítica nos protocolos sendo estudados, refere-se a [47] para mais detalhes.

3.2 Multiplicação de Precisão Múltipla

Neste contexto, a multiplicação consiste no produto de um número a representado por $(a_{n-1}, a_{n-2}, \ldots, a_1, a_0)_B$ por um número b representado por $(b_{n-1}, b_{n-2}, \ldots, b_1, b_0)_B$, de tal forma a obter um resultado c representado por $(c_{2n-1}, c_{2n-2}, \ldots, c_1, c_0)_B$.

3.2.1 Multiplicação *Schoolbook*

O primeiro algoritmo de multiplicação nada mais é o "algoritmo" aprendido na escola para multiplicar (a diferença sendo apenas na base B, que não é 10). Por esse motivo, ele é denominado algoritmo *schoolbook*. Sua versão iterativa está listada no Algoritmo 3.1. Contudo, geralmente os laços do algoritmo são desenrolados por motivos de eficiência; o mesmo se aplica para outros algoritmos de multiplicação.

Algoritmo 3.1 Algoritmo de multiplicação *schoolbook*

Entrada: inteiros positivos a e b com n dígitos na base $B = 2^W$
Saída: produto $c = a \cdot b = (c_{2n-1}, c_{2n-2}, \ldots, c_1, c_0)_B$

```
1:  c ← 0
2:  for i = 0 to n − 1 do
3:      L ← 0
4:      A ← a_i
5:      for j = 0 to n − 1 do
6:          (H, L)_B ← c_{i+j} + A · b_j + L
7:          c_{i+j} ← L
8:          L ← H
9:      end for
10:     c_{i+n} ← L
11: end for
12: return c
```

Pode-se compreender a estrutura do *schoolbook* observando-se a figura 3.2. Nela, quadrados unidos representam os dois dígitos resultantes da multiplicação dos dois dígitos indicados. Dígitos em uma mesma coluna são somados para se obter o dígito resultante da respectiva coluna, com a devida propagação de *carries*. Observe que o algoritmo consiste em tomar um dígito do operando a e multiplicá-lo por todos os dígitos de b, e repetir a operação para os demais dígitos de a.

Analisando-se o Algoritmo 3.1, verifica-se que em cada iteração do *loop* interno são realizadas duas leituras (c_{i+j} e b_j), uma escrita (c_{i+j}) e uma multiplicação ($A \cdot b_j$). Cada iteração do *loop* externo possui uma leitura (a_i) e uma escrita (c_{i+n}) adicionais. Adicionalmente, são necessárias $2n$ escritas para se zerar o resultado no primeiro passo. Assume-se

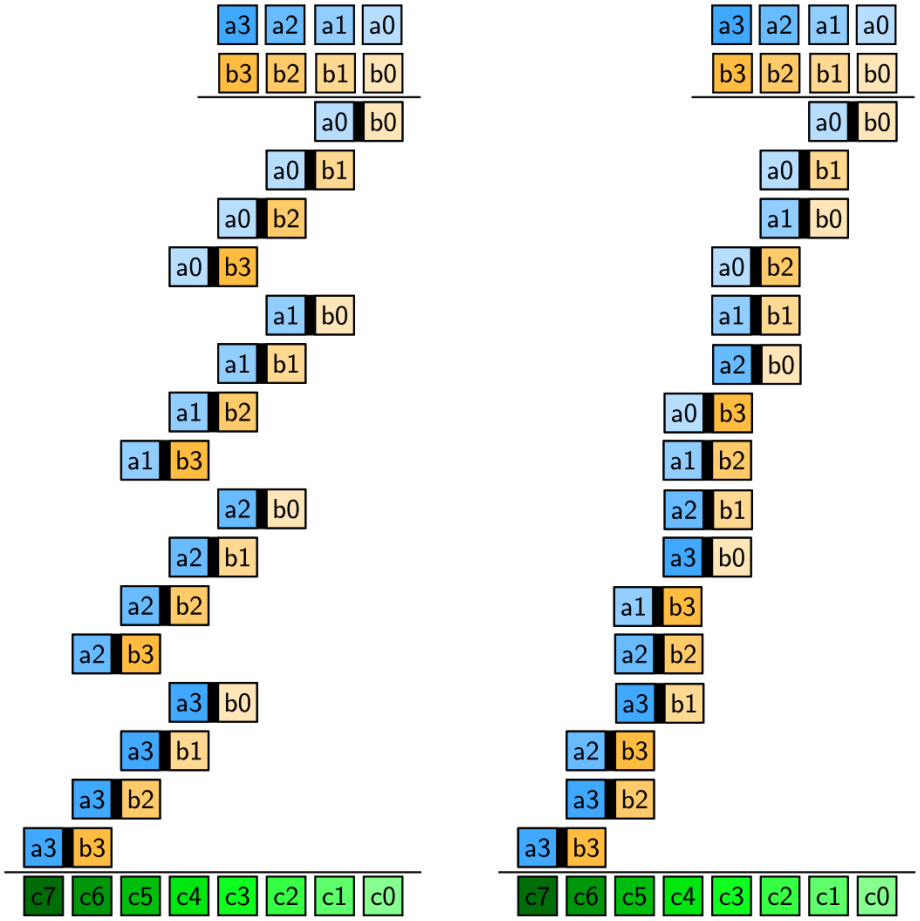


Figura 3.2: Multiplicação *schoolbook* à esquerda e Comba à direita

que variáveis locais como L, H e A estão em registradores e não são contadas suas leituras e escritas. Assim, observa-se que o algoritmo *schoolbook* necessita de $n(2n+1) = 2n^2 + n$ leituras, $n(n+1) + 2n = n^2 + 3n$ escritas e n^2 multiplicações.

O desempenho do *schoolbook* pode ser melhorado usando $n+1$ registradores acumuladores de forma a somente se escrever em c quando uma coluna estivar pronta. Neste caso, são necessárias $n(n+1) = n^2 + n$ leituras e $2n$ escritas.

3.2.2 Multiplicação Comba

A multiplicação Comba foi apresentada pelo astrônomo Paul G. Comba [13]. Conceitualmente, ela nada mais é do que uma alteração do algoritmo *schoolbook* de modo a completar cada coluna do resultado o mais cedo possível, o que permite minimizar o número de escritas na memória. Ela está listada no Algoritmo 3.2 e ilustrada na Figura 3.2.

Algoritmo 3.2 Algoritmo de multiplicação Comba

Entrada: inteiros positivos a e b com n dígitos na base $B = 2^W$
Saída: produto $c = a \cdot b = (c_{2n-1}, c_{2n-2}, \ldots, c_1, c_0)_B$

```
1:  R ← 0
2:  Q ← 0
3:  P ← 0
4:  for k = 0 to 2n − 2 do
5:      for i = max(0, k + 1 − n) to min(k, n − 1) do
6:          j ← k − i
7:          (H, P)_B ← a_i · b_j + P
8:          (R, Q)_B ← H + Q
9:      end for
10:     c_k ← P
11:     P ← Q
12:     Q ← R
13:     R ← 0
14: end for
15: c_{2n−1} ← P
16: return c
```

O algoritmo consiste na multiplicação de pares de dígitos cujos índices somam zero (a_0b_0), seguida da multiplicação de pares de dígitos cujos índices somam um (a_0b_1, a_1b_0), e assim por diante até o último dígito do resultado. As multiplicações são acumuladas em três registradores, com a devida propagação de *carry*, e no final de cada coluna o registrador menos significativo é escrito na memória.

Analisando-se o Algoritmo 3.2, verifica-se que o laço interno é executado n^2 vezes no total e possui duas leituras (a_i e b_j) e uma multiplicação ($a_i \cdot b_j$). O laço externo é executado $2n-1$ vezes e possui adicionalmente uma escrita (c_k). Finalmente, após o laço externo tem-se uma última escrita (c_{2n-1}). Assim, conclui-se que existem $2n^2$ leituras, $2n$ escritas e n^2 multiplicações. Comparado com o *schoolbook*, a multiplicação Comba realiza n leituras a menos, $n^2 + n$ escritas a menos e o mesmo número de multiplicações. Comparado com o *schoolbook* com acumuladores, são necessárias $n^2 - n$ leituras a mais e o mesmo número de escritas e multiplicações.

3.2.3 Multiplicação Híbrida

A multiplicação *schoolbook* com acumuladores possui um número de leituras menor que a multiplicação Comba, porém requer um grande número de registradores. Contudo, a multiplicação Comba precisa de apenas três registradores acumuladores independente do valor de n, e em algumas arquiteturas, isso deixa muitos registradores sem uso. Tal fato

inspirou a criação de uma multiplicação Híbrida que combina as duas técnicas, proposta por Gura et al. [29] e aperfeiçoada por Scott e Szczechowiak [66]. Ela está listada no Algoritmo 3.3 e ilustrada na Figura 3.3.

Algoritmo 3.3 Algoritmo de multiplicação Híbrida

Entrada: inteiros positivos a e b com n dígitos na base $B = 2^W$, tamanho do grupo de dígitos m
Saída: produto $c = a \cdot b = (c_{2n-1}, c_{2n-2}, \ldots, c_1, c_0)_B$

1: $T = (T_{2m}, T_{2m-1}, \ldots, T_1, T_0)_B \leftarrow 0$
2: $n' \leftarrow n/m$
3: **for** $k = 0$ **to** $2n' - 2$ **do**
4: **for** $i = \max(0, k + 1 - n')$ **to** $\min(k, n' - 1)$ **do**
5: $j \leftarrow k - i$
6: $a' \leftarrow (a_{im+(m-1)}, a_{im+(m-2)}, \ldots, a_{im+1}, a_{im})_B$
7: $b' \leftarrow (b_{jm+(m-1)}, b_{jm+(m-2)}, \ldots, b_{jm+1}, b_{jm})_B$
8: $c' \leftarrow \text{Schoolbook}(a', b')$
9: $T \leftarrow T + c'$ {*adição de múltiplos dígitos*}
10: **end for**
11: $(c_{km+(h-1)}, c_{km+(h-2)}, \ldots, c_{km+1}, c_{km})_B \leftarrow (T_{m-1}, T_{m-2}, \ldots, T_1, T_0)_B$
12: $T \leftarrow (0_{m-1}, \ldots, 0_1, 0_0, T_{2m}, T_{2m-1}, \ldots, T_{m+1}, T_m)_B$
13: **end for**
14: $k \leftarrow 2n' - 1$
15: $(c_{km+(m-1)}, c_{km+(m-2)}, \ldots, c_{km+1}, c_{km})_B \leftarrow (T_{m-1}, T_{m-2}, \ldots, T_1, T_0)_B$
16: **return** c

A multiplicação Híbrida pode ser entendida, em alto nível, como uma multiplicação Comba. A diferença é que m dígitos são agrupados e considerados um dígito só. A multiplicação de pares de grupos de dígitos é realizada, por sua vez, com a multiplicação *schoolbook* com acumuladores. O valor de m irá depender do número de registradores da plataforma alvo, e normalmente é dois ou quatro.

Seja $n' = n/m$. Analisando-se o Algoritmo 3.3, observa-se que o *loop* interno é executado $n'^2 = n^2/m^2$ vezes no total. Em cada uma dessas iterações, é realizada uma multiplicação *schoolbook* com acumulador usando m dígitos, portanto possuindo $m^2 + m$ leituras, nenhuma escrita (todas são feitas no acumulador), e m^2 multiplicações. Adicionalmente, em cada uma das $2n' - 1$ iterações do *loop* externo são realizadas m escritas. Finalmente, após o *loop* externo, são realizadas mais m escritas. No total, temos que a multiplicação Híbrida realiza $n'^2(m^2 + m) = n^2\frac{(m+1)}{m}$ leituras, $2n'm = 2n$ escritas e $n'^2m^2 = n^2$ multiplicações. A Tabela 3.1 apresenta uma comparação dos três algoritmos de multiplicação apresentados. Observa-se que a multiplicação Híbrida oferece um compromisso entre número de leituras realizadas e número de acumuladores utilizados.

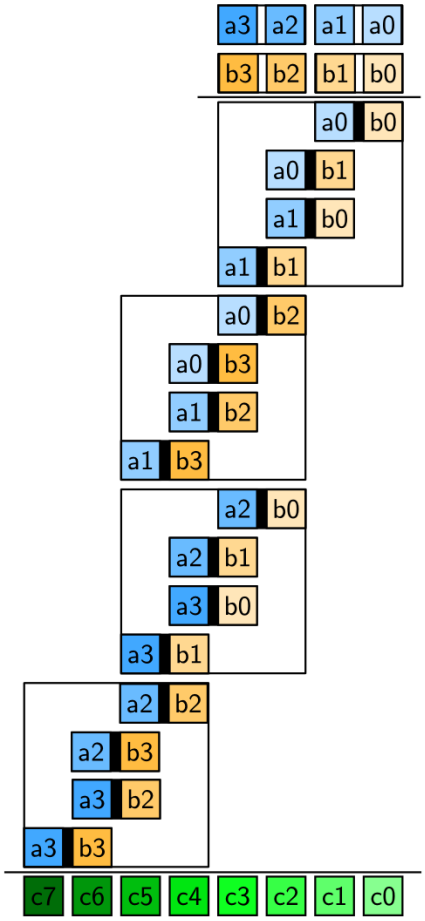


Figura 3.3: Multiplicação Híbrida

Algoritmo	Leituras	Escritas	Multiplicações	Acumuladores
Schoolbook	$(2 + 1/n)n^2$	$n^2 + 3n$	n^2	0
Schoolbook com acumulador	$(1 + 1/n)n^2$	$2n$	n^2	$n + 1$
Comba	$2n^2$	$2n$	n^2	3
Híbrido	$(1 + 1/m)n^2$	$2n$	n^2	$2m + 1$
Híbrido ($m = 2$)	$1{,}5n^2$	$2n$	n^2	5
Híbrido ($m = 4$)	$1{,}25n^2$	$2n$	n^2	9

Tabela 3.1: Comparação das multiplicações *schoolbook*, Comba e Híbrida

3.2.4 Multiplicação Karatsuba-Ofman

Embora os algoritmos de multiplicação apresentados tenham diferenças e respectivas vantagens e desvantagens, todos possuem a mesma complexidade de $O(n^2)$. Contudo, existe um algoritmo proposto por Karatsuba e Ofman [35] que possui complexidade $O(n^{1,58})$ (por brevidade, será denominado de multiplicação Karatsuba).

Suponha que a e b sejam números de s bits e $t = s/2$. Pode-se então representar a como $a = a_1 2^t + a_0$ e $b = b_1 2^t + b_0$, com a_i e b_i de t bits. Assim, tem-se que $a \cdot b = (a_1 b_1)2^{2t} + (a_1 b_0 + a_0 b_1)2^t + (a_0 b_0)$. Porém, é possível verificar que $a_1 b_0 + a_o b_1 = (a_0 + a_1)(b_0 + b_1) - a_0 b_0 - a_1 b_1$, e portanto pode-se calcular o produto dos dois números com apenas três multiplicações de $t \times t$ bits ao invés de quatro, em troca de algumas adições e subtrações a mais.

Pode-se aplicar o algoritmo recursivamente, mas em algum ponto o *overhead* das adições e subtrações supera o que é ganho com uma multiplicação a menos. Neste ponto, pode-se usar um dos algoritmos já citados para multiplicação.

3.2.5 Outros Métodos de Multiplicação

Vale notar a existência de outros métodos de multiplicação, como aqueles baseados na transformada de Fourier. Por exemplo, o algoritmo Schönhage-Strassen [61] possui complexidade $O(n \log n \log \log n)$. Contudo, tais métodos somente são mais eficientes que os já mencionados somente para tamanhos razoavelmente grandes dos operandos.

3.3 Redução Modular

Como mencionado, a multiplicação em um corpo finito é feita em duas etapas: primeiramente, a multiplicação de dois operandos a e b de n dígitos que gera um resultado c de $2n$ dígitos, seguida da redução de c módulo p para se obter o resultado final r de n dígitos. A intuição básica para a redução modular consiste no fato de que é possível subtrair p de um número sem alterar seu valor módulo p. Portanto, pode-se subtrair p de c até se obter um número menor que p. O algoritmo mais básico para se realizar tal operação é uma variação do algoritmo para divisão. Contudo, em muitas plataformas, a divisão é uma operação cara, e em certas plataformas ela nem é suportada. Por esse motivo, existem algoritmos de redução que não utilizam divisões, e um deles será descrito a seguir.

3.3.1 Redução Montgomery

Intuitivamente, a redução Montgomery [53] parte do princípio que divisões por potências de $B = 2^W$ (W sendo o número de bits do inteiro da plataforma) são simples. Ao contrário da abordagem clássica, o algoritmo começa *adicionando* múltiplos de p em c de forma a

zerar os n inteiros menos significativos do número c a ser reduzido. Prossegue-se então com uma divisão por $R = B^n$ (ignorando-se tais zeros) e obtém-se um resultado r com n ou $n+1$ inteiros (no segundo caso, subtrai-se o módulo uma vez e se obtém um resultado de n inteiros).

O problema desse método é justificar tal divisão por R, que do modo apresentado é totalmente arbitrária e forneceria um resultado incorreto. A solução é alterar a representação dos inteiros sendo utilizados. A redução Montgomery requer que seus operandos sejam transformados para uma forma especial, denominada *domínio de Montgomery*.

Para se transformar um número a para o domínio de Montgomery, calcula-se $a' = aR \bmod p$. Para se multiplicar dois números $a' \equiv aR \pmod p$ e $b' \equiv bR \pmod p$ no domínio de Montgomery e obter o resultado $r' \equiv abR \pmod p$ também no domínio de Montgomery, multiplicam-se os dois operandos com um algoritmo tradicional obtendo-se $t = a'b' \equiv abR^2 \pmod p$. Em seguida, aplica-se a redução Montgomery, que consiste nos seguintes passos. Adicionam-se múltiplos de p a t de forma a zerar seus n dígitos menos significativos, obtendo-se t'; ainda se terá que $t' \equiv abR^2 \pmod p$. Como se quer $abR \pmod p$, prossegue-se calculando $c' = t'/R \bmod p \equiv abR \pmod p$. Assim, é introduzida a necessidade da divisão por R, necessária para o funcionamento correto do algoritmo. Como os n dígitos menos significativos de t' são zero, a divisão por $R = B^n$ nada mais é do que o descarte de tais zeros.

Com essa visão de alto nível do algoritmo, pode-se descrevê-lo com mais detalhes. A redução Montgomery está listada no Algoritmo 3.4. Pode-se observar que o algoritmo é extremamente parecido com a multiplicação *schoolbook* de um certo número a e $b = p$; a única diferença é que se inicializa o resultado com o número a ser reduzido ao invés de zero; e que os dígitos de a são calculados na linha 4 do algoritmo (em A). O inteiro c a ser reduzido é alterado no processo; isto não é um problema pois normalmente ele é o resultado de uma multiplicação e estará em uma variável temporária. Pode-se provar que, após a linha 10, $r < 2p$. Assim, subtrai-se p se necessário para se obter um número menor que p.

A constante pré-calculada $p' = -1/p_0 \bmod B$ vem do seguinte raciocínio. Em cada iteração i do *loop* externo, deseja-se adicionar um múltiplo Ap de p em $(c_{n+i}, \ldots, c_{i+1}, c_i)_B$ de modo a tornar $c_i = 0$. Pode-se verificar que o valor de A necessário para se realizar isso depende somente de p_0 e c_i. Assim, deseja-se que

$$A \cdot p_0 + c_i \equiv 0 \pmod p .$$

Reorganizando os termos, verifica-se que $A = -c_i/p_0 = c_i \cdot p' \bmod p$.

Seguindo o mesmo raciocínio da multiplicação, também pode-se adaptar a redução Montgomery para utilizar o método Comba. O resultado está listado no Algoritmo 3.5. As diferenças para o Comba são que cada coluna k é inicializada com c_k, além do *carry* da

Algoritmo 3.4 Algoritmo de redução Montgomery baseado em *schoolbook*

Entrada: inteiro positivo c com $2n$ dígitos na base $B = 2^W$, módulo p de n dígitos, constante $p' = -(1/p_0) \bmod B$
Saída: redução $r = c/R \bmod p = (r_{n-1}, r_{n-2}, \ldots, r_1, r_0)_B$

```
1:  c_{2n} ← 0 {adiciona um dígito a c}
2:  for i = 0 to n − 1 do
3:      L ← 0
4:      A ← c_i · p' mod B
5:      for j = 0 to n − 1 do
6:          (H, L)_B ← c_{i+j} + A · p_j + L
7:          c_{i+j} ← L
8:          L ← H
9:      end for
10:     (c_{i+n+1}, c_{i+n})_B ← c_{i+n} + L
11: end for
12: (r_n, r_{n−1}, r_{n−2}, ..., r_1, r_0)_B ← (c_{2n}, c_{2n−1}, ..., c_{n+1}, c_n)_B
13: if r ≥ p then
14:     r ← r − p
15: end if
16: return r
```

coluna anterior, na linha 5; e na linha 10, calcula-se o próximo dígito de a se necessário, logo antes dele ser utilizado pela primeira vez.

Uma otimização sutil para este algoritmo (que também pode ser adaptada para a versão *schoolbook*) é baseada na seguinte observação. Lembre-se que, após as adições do módulo, o resultado c terá suas n palavras menos significativas valendo zero, e na linha 19 tais zeros são descartados. Assim, não é necessário escrever tais zeros de fato na memória, ou seja, pode-se executar a linha 15 somente para $k \geq n$. Pode-se inclusive utilizar os c_i, com $0 < i < n$, para se armazenar os dígitos a_i — pois quando eles são calculados, o valor de c_i já foi lido e não será mais utilizado.

3.4 Multiplicação Modular Híbrida para Curvas BN

Em 2009, Fan et al. [22] detalharam um novo método para a multiplicação em corpo finito (multiplicação seguida de redução) no contexto de curvas BN, implementado em hardware. Tal método será resumido a seguir.

Primeiramente, deve-se notar que o módulo p possui uma forma especial em curvas BN; e pode ser escrito em função de um parâmetro x como $p(x) = 36x^4 + 36x^3 + 24x^2 + 6x + 1$. Tal fato inspira uma nova representação de elementos de $\mathbb{F}_p$, como números na base x, que

Algoritmo 3.5 Algoritmo de redução Montgomery baseado em Comba

Entrada: inteiro positivo c com $2n$ dígitos na base $B = 2^W$, módulo p de n dígitos, constante $p' = -(1/p_0) \bmod B$

Saída: redução $r = c/R \bmod p = (r_{n-1}, r_{n-2}, \ldots, r_1, r_0)_B$

1: $c_{2n} \leftarrow 0$ {*adiciona um dígito a c*}
2: $a = (a_{n-1}, \ldots, a_0)_B \leftarrow 0$
3: $(R, Q, P)_B \leftarrow (0, 0, 0)_B$
4: **for** $k = 0$ **to** $2n - 2$ **do**
5: $\quad (R, Q, P)_B \leftarrow (R, Q, P)_B + c_k$
6: $\quad L \leftarrow \min(k, n-1)$
7: $\quad$ **for** $i = \max(0, k + 1 - n)$ **to** L **do**
8: $\quad\quad j \leftarrow k - i$
9: $\quad\quad$ **if** $i = L$ and $k < n$ **then**
10: $\quad\quad\quad a_i \leftarrow P \cdot p' \bmod B$
11: $\quad\quad$ **end if**
12: $\quad\quad (H, P)_B \leftarrow a_i \cdot p_j + P$
13: $\quad\quad (R, Q)_B \leftarrow H + Q$
14: $\quad$ **end for**
15: $\quad c_k \leftarrow P$
16: $\quad (R, Q, P)_B \leftarrow (0, R, Q)_B$
17: **end for**
18: $(c_{2n}, c_{2n-1})_B \leftarrow (Q, P)_B$
19: $(r_n, r_{n-1}, r_{n-2}, \ldots, r_1, r_0)_B \leftarrow (c_{2n}, c_{2n-1}, \ldots, c_{n+1}, c_n)_B$
20: **if** $r \geq p$ **then**
21: $\quad r \leftarrow r - p$
22: **end if**
23: **return** r

podem ser simultaneamente interpretados como polinômios em x. Como há uma certa liberdade na escolha de x, pode-se usar um valor na forma $x = 2^s + c$, o que facilita os cálculos.

O método proposto em [22], denominado algoritmo de multiplicação modular híbrida para curvas BN (*Hybrid Modular Multiplication Algorithm for BN curves*, HMMB; não possui relação com a multiplicação Híbrida), consiste nos seguintes passos. Para ilustrar o método, considere um $p(x)$ com 256 bits, o que é ideal para curvas BN. O parâmetro x possui cerca de 64 bits. Os operandos $a(x)$ e $b(x)$ possuem cinco coeficientes; quatro de 64 bits e o mais significativo de 32 bits (pois é necessária uma certa redundância na representação). Deseja-se calcular $r(x) = a(x)b(x) \bmod p(x)$. Primeiramente, calcula-se $c(x) = a(x)b(x)$ através de multiplicação polinomial comum. O resultado $c(x)$ será um polinômio de 9 coeficientes, cada coeficiente com cerca de 128 bits. Prossegue-se então com uma redução Montgomery polinomial de forma a reduzir $c(x)$ a 5 coeficientes. O uso de tal redução implica que os elementos $a(x)$ e $b(x)$ estão no domínio Montgomery, isto é, $a'(x) = a(x)R \bmod p(x)$, com $R = x^5$; e que na verdade deseja-se calcular $c'(x) = a'(x)b'(x)/R \bmod p(x)$. Por último, é realizada uma redução dos coeficientes, de forma a tornar seu tamanho próximo de 64 bits. Tal redução nada mais é que o cálculo do quociente e resto da divisão de cada coeficiente por x, trocando o respectivo coeficiente pelo resto e adicionando-se o quociente no próximo coeficiente, como um *carry*. Tal divisão é eficiente se x tiver a forma $2^s + c$, pela existência de um algoritmo especial.

Os autores de [22] detalham uma implementação em hardware de 64 bits, o que é ideal para o HMMB já que os coeficientes possuem tal tamanho. Finalmente, eles comparam os resultados com trabalhos anteriores, reportando um *speedup* de 5 vezes.

3.5 Aritmética de Corpos de Extensão

A aritmética de corpos de extensão segue a instanciação concreta apresentada na seção 2.2, onde $\mathbb{F}_{p^m}$ é representado por $\mathbb{F}_p[u]/f(u)$, isto é, com polinômios de grau $m-1$ módulo um polinômio irredutível $f(u)$ de grau m. Porém, tal representação não é particularmente eficiente, já que após a soma ou multiplicação polinomial é necessário realizar uma redução polinomial, que é um tanto cara.

Para resolver este problema, utilizam-se duas técnicas. A primeira é utilizar polinômios irredutíveis da forma $f(u) = u^2 - \beta$. Quando se trabalha módulo $f(u)$, tem-se que $f(u) = 0$, e neste caso $u^2 = \beta$ e portanto $u = \sqrt{\beta}$. Para $f(u)$ desta forma ser irredutível, basta que β seja um não-resíduo quadrático (não possua raiz quadrada em $\mathbb{F}_p$). Tal técnica pode ser interpretada como a adição da raiz quadrada $\sqrt{\beta}$, que em $\mathbb{F}_p$ não existe, de forma a criar $\mathbb{F}_{p^2}$ onde tal raiz é denominada u. Este é o mesmo método com o qual criam-se os números complexos — adiciona-se a raiz quadrada de -1, que em $\mathbb{R}$ não existe, para se

criar $\mathbb{C}$ onde tal raiz é denominada i.

Para se entender como tal técnica facilita a implementação, considere a multiplicação de dois elementos $a_0 + a_1u$ e $b_0 + b_1u$ em $\mathbb{F}_{p^2}$, onde $a_0, a_1, b_0, b_1 \in \mathbb{F}_p$. Primeiramente, multiplicam-se os elementos polinomialmente, de modo a obter $a_0b_0 + (a_0b_1 + a_1b_0)u + a_1b_1u^2$. Agora, deve-se reduzir polinomialmente tal resultado módulo $f(u) = u^2 - \beta$. Mas como visto, $u^2 = \beta$ e portanto o resultado pode ser computado por $(a_0b_0 + a_1b_1\beta) + (a_0b_1 + a_1b_0)u$. Escolhe-se um valor de β pequeno (-1 ou -2) de forma a permitir a multiplicação por β com adições e subtrações apenas. Assim, a multiplicação em $\mathbb{F}_{p^2}$ é realizada somente com multiplicações e adições em $\mathbb{F}_p$, sem necessidade de redução polinomial.

O mesmo raciocínio pode ser adaptado para extensões de grau maior que 2. Contudo, outra técnica é utilizada de forma a se necessitar somente de extensões de grau 2 e 3. Tal técnica é denominada *torre de extensões*, e é mais facilmente demonstrada com um exemplo.

Considere as curvas BN, onde é necessário construir $\mathbb{F}_{p^{12}}$. Pode-se representar tal corpo com um polinômio irredutível de grau 12, mas existe uma forma mais eficiente. Primeiramente, representa-se $\mathbb{F}_{p^2}$ como $\mathbb{F}_p[u]/(u^2 - \beta)$ onde β é um não-resíduo quadrático em $\mathbb{F}_p$. Então, representa-se $\mathbb{F}_{p^6}$ como $\mathbb{F}_{p^2}[v]/(v^3 - \xi)$, onde ξ é um não-resíduo cúbico em $\mathbb{F}_{p^2}$. Finalmente, representa-se $\mathbb{F}_{p^{12}}$ como $\mathbb{F}_{p^6}[w]/(w^2 - \gamma)$, onde γ é um não-resíduo quadrático em $\mathbb{F}_{p^6}$. Desta forma, operações em $\mathbb{F}_{p^{12}}$ são calculadas em termos de operações em $\mathbb{F}_{p^6}$, e assim por diante até operações em $\mathbb{F}_p$ que já foram estudadas.

Assim, somente é necessário estudar a aritmética de extensões quadráticas e cúbicas. Naturalmente, extensões de graus com fatores diferentes de 2 e 3 (15, por exemplo) não podem ser representadas somente com extensões quadráticas e cúbicas, que são mais simples de se tratar. Contudo, neste trabalho, tais extensões não são necessárias. Os diversos algoritmos para multiplicação e quadrado em extensões quadráticas e cúbicas foram estudados em [16], e são resumidos a seguir. Por completude, também são listadas as fórmulas para a inversão em tais extensões.

3.5.1 Extensão Quadrática

Considere uma extensão quadrática $\mathbb{F}_{q^2}$ (onde $\mathbb{F}_q$ também pode ser uma outra extensão) construída como $\mathbb{F}_q[u]/(u^2 - \beta)$. Um elemento $c \in \mathbb{F}_{q^2}$ é representado por $c_0 + c_1u$, com $c_0, c_1 \in \mathbb{F}_q$. Para a multiplicação $c = ab$ em $\mathbb{F}_{q^2}$, tem-se os seguintes métodos.

Multiplicação *Schoolbook*:

$$c_0 = a_0b_0 + \beta a_1b_1$$
$$c_1 = a_0b_1 + a_1b_0\,.$$

Multiplicação Karatsuba:

$$\begin{aligned} v_0 &= a_0 b_0 \\ v_1 &= a_1 b_1 \\ c_0 &= v_0 + \beta v_1 \\ c_1 &= (a_0 + a_1)(b_0 + b_1) - v_0 - v_1 \,. \end{aligned}$$

Para se computar o quadrado $c = a^2$ em $\mathbb{F}_{q^2}$, tem-se os seguintes métodos.

Quadrado *Schoolbook*:

$$\begin{aligned} c_0 &= a_0^2 + \beta a_1^2 \\ c_1 &= 2a_0 a_1 \,. \end{aligned}$$

Quadrado Karatsuba:

$$\begin{aligned} v_0 &= a_0^2 \\ v_1 &= a_1^2 \\ c_0 &= v_0 + \beta v_1 \\ c_1 &= (a_0 + a_1)^2 - v_0 - v_1 \,. \end{aligned}$$

Quadrado Complexo:

$$\begin{aligned} v_0 &= a_0 a_1 \\ c_0 &= (a_0 + a_1)(a_0 + \beta a_1) - v_0 - \beta v_0 \\ c_1 &= 2v_0 \,. \end{aligned}$$

O quadrado Complexo tem esse nome por vir de uma fórmula bem conhecida para o quadrado de números complexos.

Finalmente, para se computar o inverso $c = a^{-1}$ em $\mathbb{F}_{q^2}$, utiliza-se a fórmula a seguir.

Inversão:

$$\begin{aligned} v_0 &= (a_0^2 - \beta a_1^2)^{-1} \\ c_0 &= a_0 v_0 \\ c_1 &= -a_1 v_0 \,. \end{aligned}$$

Os custos computacionais de todos os métodos apresentados estão resumidos na Tabela 3.2. Refere-se a adições, multiplicações por β, multiplicações e quadrados por A, B, M e S, respectivamente. Verifica-se que a multiplicação Karatsuba é mais eficiente se $M > 3A$, isto é, se uma multiplicação custar mais que três adições, o que geralmente é verdade. O quadrado Karatsuba é mais eficiente que *schoolbook* se $M > S + 2A + B$ e o quadrado Complexo é mais eficiente que Karatsuba se $S > \frac{2}{3}M \approx 0{,}67M$, o que também geralmente é verdade.

Método	M	S	A	B
Multiplicação				
Schoolbook	4		2	1
Karatsuba	3		5	1
Quadrado				
Schoolbook	1	2	2	1
Karatsuba		3	4	2
Complexo	2		4	2

Tabela 3.2: Custos para operações em extensões quadráticas; M = multiplicações, S = quadrados, A = adições, B = multiplicações por β

3.5.2 Extensão Cúbica

Considere uma extensão cúbica $\mathbb{F}_{q^3}$ construída como $\mathbb{F}_q[u]/(u^3 - \beta)$. Um elemento $c \in \mathbb{F}_{q^3}$ é representado por $c_0 + c_1 u + c_2 u^2$, com $c_0, c_1, c_2 \in \mathbb{F}_q$. Para a multiplicação $c = ab$ em $\mathbb{F}_{q^3}$, tem-se os seguintes métodos.

Multiplicação *Schoolbook*:

$$
\begin{aligned}
c_0 &= a_0 b_0 + \beta(a_1 b_2 + a_2 b_1) \\
c_1 &= a_0 b_1 + a_1 b_0 + \beta a_2 b_2 \\
c_2 &= a_0 b_2 + a_1 b_1 + a_2 b_0 \,.
\end{aligned}
$$

Multiplicação Karatsuba:

$$
\begin{aligned}
v_0 &= a_0 b_0 \\
v_1 &= a_1 b_1 \\
v_2 &= a_2 b_2 \\
c_0 &= v_0 + \beta((a_1 + a_2)(b_1 + b_2) - v_1 - v_2) \\
c_1 &= (a_0 + a_1)(b_0 + b_1) - v_0 - v_1 + \beta v_2 \\
c_2 &= (a_0 + a_2)(b_0 + b_2) - v_0 + v_1 - v_2 \,.
\end{aligned}
$$

Existem outros métodos como o Toom-Cook-3 [14, 75], mas que não forneceriam ganho de desempenho relevante no contexto deste trabalho de acordo com [16]. Para o quadrado $c = a^2$ em $\mathbb{F}_{q^3}$, tem-se os seguintes métodos.

Quadrado *Schoolbook*:

$$
\begin{aligned}
c_0 &= a_0^2 + 2\beta a_1 a_2 \\
c_1 &= 2a_0 a_1 + \beta a_2^2 \\
c_2 &= a_1^2 + 2a_0 a_2 \,.
\end{aligned}
$$

Quadrado Karatsuba:

$$
\begin{aligned}
v_0 &= a_0^2 \\
v_1 &= a_1^2 \\
v_2 &= a_2^2 \\
c_0 &= v_0 + \beta((a_1 + a_2)^2 - v_1 - v_2) \\
c_1 &= (a_0 + a_1)^2 - v_0 - v_1 + \beta v_2 \\
c_2 &= (a_0 + a_2)^2 - v_0 + v_1 - v_2 \,.
\end{aligned}
$$

Quadrado CH-SQR2 [11]:

$$
\begin{aligned}
s_0 &= a_0^2 \\
s_1 &= 2a_0a_1 \\
s_2 &= (a_0 - a_1 + a_2)^2 \\
s_3 &= 2a_1a_2 \\
s_4 &= a_2^2 \\
c_0 &= s_0 + \beta s_3 \\
c_1 &= s_1 + \beta s_4 \\
c_2 &= s_1 + s_2 + s_3 - s_0 - s_4 \,.
\end{aligned}
$$

Para se computar $c = a^{-1}$ em $\mathbb{F}_{q^3}$, utiliza-se a seguinte fórmula.

Inversão:

$$
\begin{aligned}
v_0 &= a_0^2 - \beta a_1a_2 \\
v_1 &= \beta a_2^2 - a_0a_1 \\
v_2 &= a_1^2 - a_0a_2 \\
v_3 &= (\beta a_1v_2 + a_0v_0 + \beta a_2v_1)^{-1} \\
c_0 &= v_0v_3 \\
c_1 &= v_0v_3 \\
c_2 &= v_2v_3 \,.
\end{aligned}
$$

Também existem outros métodos como CH-SQR1 e CH-SQR3 [11], mas eles também não fornecem ganho de desempenho significativo. Os custos computacional dos métodos apresentados estão resumidos na Tabela 3.3. A multiplicação Karatsuba é mais eficiente se $M > \frac{7}{3}A \approx 2{,}3A$. O quadrado Karatsuba é mais eficiente que o *schoolbook* se $M > S + \frac{7}{3}A \approx S + 2{,}3A$ e o CH-SQR-2 é mais eficiente que o Karatsuba se $M < \frac{3}{2}S + \frac{3}{2}A = 1{,}5S + 1{,}5A$.

3.5.3 *Lazy Reduction*

Existe uma otimização em corpos de extensões que fornece ganhos significativos de desempenho. Considere uma extensão quadrática sobre $\mathbb{F}_p$. Como visto, na fórmula da multiplicação Karatsuba tem-se que $c_1 = v_2 - v_0 - v_1$ com $v_2 = (a_0 + a_1)(b_0 + b_1)$. Cada

Método	M	S	A	B
Multiplicação				
Schoolbook	9		6	2
Karatsuba	6		13	2
Quadrado				
Schoolbook	3	3	6	2
Karatsuba		6	13	2
CH-SQR2	2	3	10	2

Tabela 3.3: Custos para operações em extensões cúbicas; M = multiplicações, S = quadrados, A = adições, B = multiplicações por β

v_i é resultado de uma multiplicação em $\mathbb{F}_p$, portanto o cálculo de c_1 requer três multiplicações em $\mathbb{F}_p$, onde cada multiplicação é composta por uma multiplicação tradicional e uma redução.

Considere então a seguinte alternativa, denominada *lazy reduction* [42]. Computam-se v_i' que são o resultado das multiplicações correspondentes mas sem a posterior redução modular. Prossegue-se com o cálculo de $c_1' = v_2' - v_0' - v_1'$. Finalmente, reduz-se c_1' módulo p para se obter c_1; assim, economizam-se duas reduções modulares. O mesmo pode ser aplicado para o cálculo de c_0. Seja M o custo da multiplicação tradicional e R o custo da redução; tem-se que o cálculo de $c = ab$ requer $3M + 3R$ do modo tradicional e $3M + 2R$ com *lazy reduction*. Como uma redução custa aproximadamente o mesmo que uma multiplicação, conclui-se que tal técnica fornece um ganho de aproximadamente $1/6 \approx 17\%$ do custo original.

Contudo, tal técnica necessita de uma operação que até então não existia: a soma ou subtração dos v_i', que possuem precisão dupla. Seja n o número de dígitos de p na base $B = 2^W$. Tem-se que todos os v_i possuem no máximo $2n$ dígitos. Portanto, tal operação de soma ou subtração de precisão dupla deve preservar tal limite máximo. Assim, pode-se simplesmente somar os dois operandos do modo usual e então proceder da seguinte forma. Na soma, se o resultado for maior ou igual que pB^n, subtrai-se pB^n do resultado (já que qualquer múltiplo de p é congruente a 0 módulo p). Na subtração, se o resultado for menor que 0, adiciona-se pB^n. Naturalmente, as operações de adição e subtração de precisão dupla custam aproximadamente o dobro das operações normais. Porém, tal custo adicional é pequeno em comparação com o que se ganha usando o *lazy reduction*.

Finalmente, verifica-se que esta técnica também pode ser usada na multiplicação Karatsuba em extensões cúbicas de $\mathbb{F}_p$, onde o custo passa de $6M + 6R$ para $6M + 3R$, fornecendo um ganho de aproximadamente 25%.

3.5.4 Mapa de Frobenius

O mapa de Frobenius π_p para um corpo finito $\mathbb{F}_q$ de característica p é definido por $\pi_p(x) = x^p$. Tal mapa é muito utilizado no cálculo de emparelhamentos por ser muito eficiente de se computar, portanto aqui será descrito com mais detalhes porque tal eficiência é adquirida. Primeiramente, considere o seguinte teorema.

Teorema 3.2. *[78, Teorema C.1] Para qualquer $x \in \mathbb{F}_q$, $x^q = x$ e $x^{q-1} = 1$.*

Tal teorema implica que para $a \in \mathbb{F}_p$, $\pi_p(a) = a^p = a$. O próximo teorema demonstra que o mapa de Frobenius possui uma propriedade curiosa, que é apelidada de "sonho dos calouros"[1] por ser, em outros contextos, um erro comum cometido por estudantes.

Teorema 3.3. *[78, Proposição C.2, parte 2] Sejam a, b elementos de F_q, cuja característica é p. Tem-se que $(a+b)^p = a^p + b^p$.*

Agora considere o comportamento do mapa em extensões quadráticas. Seja $(a_0 + a_1 u) \in \mathbb{F}_{q^2}$, com a extensão representada por $\mathbb{F}_q[u]/(u^2 - \beta)$. De acordo com o Teorema 3.3, tem-se que $(a_0 + a_1 u)^p = a_0^p + (a_1 u)^p = a_0^p + a_1^p u^{p-1} u$. Assim, pode-se calcular o mapa de Frobenius na extensão quadrática de modo eficiente aplicando o mapa recursivamente em a_0 e a_1 e então multiplicando a_1^p pelo valor u^{p-1}, que pode ser pré-calculado. Em particular, se p for ímpar, tem-se que $u^{p-1} = (u^2)^{(p-1)/2} = \beta^{(p-1)/2}$. Também é interessante notar que se $q = p$, então $\beta^{(p-1)/2} = -1$ pelo critério de Euler [68, Teorema 2.21], que diz que ao se elevar um não-resíduo quadrático em $\mathbb{F}_p$ por $(p-1)/2$ obtém-se -1; e β é um não-resíduo quadrático por definição. Neste caso, tem-se que $(a_0 + a_1 u)^p = a_0 - a_1 u$.

Para extensões cúbicas, procede-se da mesma forma, obtendo-se $(a_0 + a_1 u + a_2 u^2)^p = a_0^p + a_1^p u^{p-1} u + a_2^p (u^{p-1})^2 u^2$. Neste caso pode-se pré-calcular u^{p-1} e elevá-lo ao quadrado para obter o fator de a_2, ou se pré-calcula ambos, dependendo do desempenho necessário.

3.6 Multiplicação de Ponto

A operação fundamental em protocolos baseados em curvas elípticas é a multiplicação de ponto, que também é utilizada em protocolos baseados em emparelhamentos. Seja $E(K)$ uma curva elíptica e $P \in E(K), k \in \mathbb{Z}$. A multiplicação de ponto consiste em calcular kP, isto é, adicionar k parcelas de P utilizando-se a operação de grupo em curvas elípticas. Vale lembrar que se k for negativo, tem-se que $kP = (-k)(-P)$, portanto a seguir assume-se que k seja positivo.

[1] `http://en.wikipedia.org/wiki/Freshman's_dream`

3.6.1 Multiplicação Binária

O algoritmo mais simples para se calcular kP seria de fato calcular k somas, porém tal operação é computacionalmente cara já que k, por exemplo, é da ordem de 2^{160} bits para o nível de segurança de 80 bits. Já o algoritmo binário se baseia na recursão

$$kP = \begin{cases} \infty & \text{se } k = 0, \\ 2(k/2)P & \text{se } k \text{ par}, \\ (k-1)P + P & \text{se } k \text{ ímpar}. \end{cases}$$

Pode-se transformar tal recursão no Algoritmo 3.6, que percorre de modo iterativo os bits de k para decidir se deve somente duplicar o ponto ou também somá-lo com P. Assim, o número de passos do algoritmo binário é proporcional a $\log_2 k$, sendo portanto muito mais eficiente que a abordagem "ingênua".

Algoritmo 3.6 Algoritmo binário de multiplicação de ponto

Entrada: $k \in \mathbb{N}$, $P \in E(\mathbb{F}_q)$
Saída: kP

```
1: (k_{t-1}, ..., k_1, k_0)_2 ← k
2: Q ← ∞
3: for i = t − 1 to 0 do
4:     Q ← 2Q
5:     if k_i = 1 then
6:         Q ← Q + P
7:     end if
8: end for
9: return Q
```

3.6.2 wNAF

Denomina-se *método wNAF* o método que utiliza a representação wNAF do multiplicador k. O wNAF [70] é uma generalização do NAF (*non-adjacent form*, forma não-adjacente). Tais métodos são baseados no fato que o inverso de um ponto P, isto é, $-P$, pode ser calculado eficientemente. O NAF de k é a sua representação na base $\{-1, 0, 1\}$, isto é, $k = \sum k_i 2^i$, com $k_i \in \{-1, 0, 1\}$. Pode-se converter k na sua representação NAF de modo eficiente [30, Algoritmo 3.30], e converte-se o Algoritmo 3.6 para o NAF adicionando-se que se $k_i = -1$, então $Q \leftarrow Q - P$. A grande vantagem do uso do NAF é que, enquanto um número aleatório k de t bits possui em média $t/2$ bits de valor 1, o NAF de k possui $t/3$ [54], reduzindo portanto o número de adições necessárias.

O wNAF é uma generalização do NAF, onde se representa o multiplicador como $k = \sum k_i 2^i$, com k_i ímpar e $k_i \in [-2^{w-1}, 2^{w-1}]$; denota-se tal representação por $w\text{NAF}(k)$. O wNAF também pode ser eficientemente computado [30, Algoritmo 3.35]. Através da pré-computação de $P_j = jP$ para $j \in \{1, 3, \ldots, 2^{w-1} - 1\}$, pode-se alterar o Algoritmo 3.6, obtendo-se o Algoritmo 3.7. É importante notar que a tabela de pré-computação possui 2^{w-2} pontos e que deve ser calculada a cada multiplicação de ponto.

Algoritmo 3.7 Algoritmo wNAF de multiplicação de ponto

Entrada: $k \in \mathbb{N}$, $P \in E(\mathbb{F}_q)$, $w \in \mathbb{N}$
Saída: kP

1: $(k_{t-1}, \ldots, k_1, k_0)_2 \leftarrow w\text{NAF}(k)$
2: Calcule tabela $P_j = jP$ para $j \in \{1, 3, \ldots, 2^{w-1} - 1\}$
3: $Q \leftarrow \infty$
4: **for** $i = t - 1$ **to** 0 **do**
5: $\quad Q \leftarrow 2Q$
6: $\quad$ **if** $k_i > 0$ **then**
7: $\quad\quad Q \leftarrow Q + P_{k_i}$
8: $\quad$ **end if**
9: $\quad$ **if** $k_i < 0$ **then**
10: $\quad\quad Q \leftarrow Q - P_{-k_i}$
11: $\quad$ **end if**
12: **end for**
13: **return** Q

3.6.3 Comb

O método Comb [43] também utiliza uma tabela de pré-computação, mas é voltado para multiplicação de um ponto P fixo, e portanto é mais eficiente que o wNAF. Tal método é útil pois a assinatura no ECDSA, por exemplo, envolve a multiplicação de um ponto fixo que é um parâmetro global do esquema. Como o ponto é fixo, a tabela pode ser computada previamente e guardada na memória, não sendo necessária computá-la a cada multiplicação, como normalmente acontece no wNAF.

Seja t o número máximo de bits de $k = (k_{t-1}, k_{t-2}, \ldots, k_1, k_0)_2$, w o parâmetro desejado (quanto maior, maior é a tabela de pré-computação e mais eficiente é o algoritmo) e $d = \lceil t/w \rceil$. O algoritmo Comb baseia-se na observação que kP pode ser escrito como

$$
\begin{aligned}
kP &= k_0P + k_1 2P + k_2 2^2 P + \ldots + k_{t-1}2^{t-1}P \\
&= k_0P + k_d 2^d P + k_{2d} 2^{2d} P + \ldots + k_{(w-1)d} 2^{(w-1)d} P \\
&+ 2(k_1P + k_{d+1} 2^d P + k_{2d+1} 2^{2d} P + \ldots + k_{(w-1)d+1} 2^{(w-1)d} P) \\
&+ 2^2(k_2P + k_{d+2} 2^d P + k_{2d+2} 2^{2d} P + \ldots + k_{(w-1)d+2} 2^{(w-1)d} P \\
&+ \ldots \\
&+ 2^{d-1}(k_{d-1}P + k_{d+(d-1)} 2^d P + k_{2d+(d-1)} 2^{2d} P + \ldots + k_{(w-1)d+(d-1)} 2^{(w-1)d} P) .
\end{aligned}
$$

Denote por T_a a soma $a_0P + a_1 2^d P + a_2 2^{2d} P + \ldots + a_{w-1} 2^{(w-1)d} P$. Assim, pré-calculando T_a para todos números a de w bits, pode-se calcular a soma acima linha por linha, de baixo para cima. Processa-se w bits de k por linha, totalizando d passos. O método Comb está listado no Algoritmo 3.8. Note que a tabela contém 2^w pontos.

Algoritmo 3.8 Algoritmo Comb de multiplicação de ponto

Entrada: $k \in \mathbb{N}$, $P \in E(\mathbb{F}_q)$, $w \in \mathbb{N}$, T_a para todos a de w bits
Saída: kP

1: $(k_{t-1}, \ldots, k_1, k_0)_2 \leftarrow k$
2: $d \leftarrow \lceil t/w \rceil$
3: $Q \leftarrow \infty$
4: **for** $i = d - 1$ **to** 0 **do**
5: $\quad Q \leftarrow 2Q$
6: $\quad a \leftarrow (k_{(w-1)d+i}, \ldots, k_{2d+i}, k_{d+i}, k_i)_2$
7: $\quad Q \leftarrow Q + T_a$
8: **end for**
9: **return** Q

3.6.4 *Interleaving*

Em muitos protocolos é necessário computar a soma $kP + \ell Q$, inclusive na verificação do ECDSA. Neste caso, pode-se efetuar as duas multiplicações simultaneamente, de forma a compartilhar as duplicações de ponto. Tal método é denominado *Shamir's trick*. Pode-se também combinar tal método com o wNAF, inclusive permitindo o uso de diferentes w para codificar k e ℓ, resultando no chamado método *interleaving* (entrelaçamento) [26, 52]. Tal técnica é útil quando P é fixo mas Q é aleatório, onde pode-se utilizar uma janela maior para P já que sua tabela não precisa ser computada com antecedência e pode ser deixada guardada na memória. O *interleaving* de wNAFs está detalhado no Algoritmo 3.9.

Algoritmo 3.9 Algoritmo Interleaving de wNAFs para multiplicação simultânea de pontos

Entrada: $k, \ell \in \mathbb{N}$, $P, Q \in E(\mathbb{F}_q)$, $w_k, w_\ell \in \mathbb{N}$
Saída: $kP + \ell P$

1: $(k_{t-1}, \ldots, k_1, k_0) \leftarrow w_k\text{NAF}(k)$
2: $(\ell_{t-1}, \ldots, \ell_1, \ell_0) \leftarrow w_l\text{NAF}(\ell)$
3: Leia/calcule tabela $P_j = jP$ para $j \in \{1, 3, \ldots, 2^{w_k-1} - 1\}$
4: Leia/calcule tabela $Q_j = jQ$ para $j \in \{1, 3, \ldots, 2^{w_\ell-1} - 1\}$
5: $T \leftarrow \infty$
6: **for** $i = t - 1$ **to** 0 **do**
7: $\quad T \leftarrow 2T$
8: $\quad$ **if** $k_i > 0$ **then**
9: $\quad\quad T \leftarrow T + P_{k_i}$
10: $\quad$ **end if**
11: $\quad$ **if** $k_i < 0$ **then**
12: $\quad\quad T \leftarrow T - P_{-k_i}$
13: $\quad$ **end if**
14: $\quad$ **if** $\ell_i > 0$ **then**
15: $\quad\quad T \leftarrow T + Q_{\ell_i}$
16: $\quad$ **end if**
17: $\quad$ **if** $\ell_i < 0$ **then**
18: $\quad\quad T \leftarrow T - Q_{-\ell_i}$
19: $\quad$ **end if**
20: **end for**
21: **return** T

3.7 Otimizações de Emparelhamentos Bilineares

Os fundamentos do algoritmo de Miller e o emparelhamento de Tate já foram descritos na seção 2.4. Contudo, existem muitas otimizações e novos emparelhamentos que, embora não sejam fundamentais para a compreensão, são vitais para implementações eficientes. Tais otimizações e novos emparelhamentos serão descritos nesta seção.

3.7.1 *Twists*

Como mencionado, o emparelhamento de Tate mapeia pontos $P \in \mathbb{G}_1 \subseteq E(\mathbb{F}_q)[r]$ e $Q \in \mathbb{G}_2 \subseteq E(\mathbb{F}_{q^k})$ em um elemento $f \in \mathbb{G}_T \subseteq \mathbb{F}_{q^k}$. O algoritmo de Miller para o emparelhamento de Tate requer a aritmética em $\mathbb{G}_1$ e $\mathbb{F}_{q^k}$ (os elementos de $\mathbb{G}_2$ apenas servem como entrada para as funções das retas), porém existem outros emparelhamentos que serão descritos adiante onde o papel dos grupos $\mathbb{G}_1$ e $\mathbb{G}_2$ é invertido. Neste caso, a aritmética em $\mathbb{G}_2$ passa a ser necessária e bastante cara em relação à aritmética $\mathbb{G}_1$. Sabe-se, porém, que no lugar de $E(\mathbb{F}_{q^k})$ como $\mathbb{G}_2$ é possível usar curvas sobre corpos de extensão menor do que k e que possuem um isomorfismo eficientemente computável para a curva sobre a extensão k.

Definição 3.4. *[31, Definição 1] Sejam $E/\mathbb{F}_q$ e $E'/\mathbb{F}_q$ duas curvas elípticas. A curva $E'/\mathbb{F}_q$ é denominada um twist* de grau *d de $E/\mathbb{F}_q$ se existe um isomorfismo $\phi_d\colon E'(\mathbb{F}_q) \to E(\mathbb{F}_{q^d})$ e d é mínimo.*

Existem somente *twists* de grau $d = 2, 3, 4, 6$ [31, Seção 4.1]. Para uma curva elíptica $E/K\colon y^2 = x^3 + ax + b$ sempre existem *twists* quadráticos ($d = 2$); existem *twists* quárticos ($d = 4$) se e somente se $a \neq 0$ e $b = 0$; e existem *twists* cúbicos e sêxticos ($d = 3, 6$) se e somente se $a = 0$, $b \neq 0$.

Dada uma curva elíptica $E/\mathbb{F}_q$ e um valor $D \in \mathbb{F}_q^*$ que seja um não-resíduo d-ésimo, o *twist* de grau d (se existir) e o respectivo isomorfismo ϕ_d são dados por

$$
\begin{aligned}
&d = 2\colon y^2 = x^3 + a/D^2 x + b/D^4, && \phi_d\colon E'(\mathbb{F}_q) \to E(\mathbb{F}_{q^d})\colon (x, y) \mapsto (Dx, D^{3/2} y)\\
&d = 4\colon y^2 = x^3 + a/Dx, && \phi_d\colon E'(\mathbb{F}_q) \to E(\mathbb{F}_{q^d})\colon (x, y) \mapsto (D^{1/2} x, D^{3/4} y)\\
&d = 3\colon y^2 = x^3 + b/D^2, && \phi_d\colon E'(\mathbb{F}_q) \to E(\mathbb{F}_{q^d})\colon (x, y) \mapsto (D^{2/3} x, Dy)\\
&d = 6\colon y^2 = x^3 + b/D, && \phi_d\colon E'(\mathbb{F}_q) \to E(\mathbb{F}_{q^d})\colon (x, y) \mapsto (D^{1/3} x, D^{1/2} y)
\end{aligned}
$$

Em emparelhamentos, *twists* podem ser aplicados da seguinte forma. Considere o emparelhamento de Tate $\tau_r\colon E(\mathbb{F}_{q^k})[r] \times E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k}) \to \mu_r$. Como mencionado, na prática utiliza-se $\tau_r\colon E(\mathbb{F}_q)[r] \times E(\mathbb{F}_{q^k}) \to \mu_r$. Suponha que E tenha um *twist* de grau

d e que $k = d \cdot e$ para algum inteiro e (isto é, k é divisível por d). Tem-se então que o emparelhamento $\tau'_r \colon E(\mathbb{F}_q)[r] \times E'(\mathbb{F}_{q^e}) \to \mu_r$ computado por $\tau'_r(P, Q') = \tau_r(P, \phi_d(Q'))$ é naturalmente um emparelhamento bilinear. É importante notar que o isomorfismo ϕ_d não precisa ser computado antes de se calcular o emparelhamento (o que não traria nenhum ganho de desempenho) e que pode ser calculado o mais tarde possível. Isso ficará mais claro ao se usar variantes do emparelhamento Ate.

Vale notar que às vezes é usada outra notação para o mapa ϕ_d. Seja $D' = \sqrt[d]{D}$ (note que D', por definição, não pode estar em $\mathbb{F}_q$, mas sim em alguma extensão sua). Então pode-se verificar que os três mapas assumem a mesma forma $\phi_d \colon (x, y) \mapsto (D'^2 x, D'^3 y)$.

3.7.2 Eliminação de Denominador

O algoritmo de Miller (Algoritmo 2.1) requer divisões pelos resultados das funções $V_P(Q)$. Divisões por elementos de corpos finitos envolvem o cálculo de inversos, que são bastante caros. Porém, ao se utilizar curvas *twists* em $\mathbb{G}_2$, tais divisões podem ser eliminadas, como será descrito a seguir. Primeiramente, são necessários alguns teoremas.

Lema 3.5. *[4, Lema 5] Para qualquer d tal que $d|k$ e $d < k$, tem-se que $q^d - 1 \mid (q^k - 1)/r$.*

Teorema 3.6. *Pode-se multiplicar $f_{r,P}(Q)$ por qualquer $x \in \mathbb{F}_{q^d}$ com $x \neq 0$, $d \mid k$, $d < k$ sem alterar o valor do emparelhamento.*

Demonstração. A exponenciação final do emparelhamento de Tate consiste em elevar o resultado do algoritmo de Miller ao valor $(q^k - 1)/r$. Analisando-se o algoritmo de Miller, pode-se verificar que ele consiste numa série de multiplicações pelas funções das retas. Portanto, pode-se considerar que cada fator da multiplicação é elevado por $(q^k - 1)/r$ (já que $(a \cdot b)^x = a^x b^x$). Finalmente, suponha que um desses fatores seja $x \in \mathbb{F}_{q^d}$. De acordo com o lema 3.5, podemos escrever $(q^k - 1)/r = (q^d - 1)z$ para certo z. Assim, tem-se que a exponenciação final fará com que x seja elevado a $(x^{q^d-1})^z$. Mas de acordo com o Teorema 3.2, $(x^{q^d-1})^z = 1^z = 1$. Logo, qualquer fator $x \in \mathbb{F}_{q^d}$ é irrelevante ao resultado final do emparelhamento. □

Um corpo $\mathbb{F}_{q^e}$ é denominado *subcorpo próprio* de $\mathbb{F}_{q^k}$ se $e \mid k$ e $e < k$. Assim, em outras palavras, o Teorema 3.6 diz que é possível multiplicar o resultado do algoritmo de Miller por qualquer elemento de um subcorpo próprio de $\mathbb{F}_{q^k}$ que não seja zero.

Sejam d, e como descritos na seção anterior. De acordo com a seção 2.4.5, cada função de reta vertical V_P possui forma $V_P(Q) = x_Q - x_P$. No emparelhamento de Tate, tem-se que $P \in E(\mathbb{F}_q)$ e portanto $x_P \in \mathbb{F}_q$, um subcorpo próprio de $\mathbb{F}_{q^k}$. Ao se usar *twists*, o ponto $Q = (x_Q, y_Q)$ é computado por $\phi_d(Q')$. O seguinte teorema mostrará que neste caso, x_Q é um elemento de um subcorpo próprio de $\mathbb{F}_{q^k}$ e portanto $V_R(Q) = x_Q - x_P$ também o é, sendo assim um fator irrelevante.

Teorema 3.7. *Seja $E/\mathbb{F}_q$ uma curva com grau de mergulho k em relação a r. Seja $E'(\mathbb{F}_{q^e})$ um* twist *de grau $d \neq 3$ baseado em D e escreva $k = d \cdot e$. Para todo $Q' \in E'(\mathbb{F}_{q^e})$, tem-se que $x_{\phi_d(Q')} \in \mathbb{F}_{(q^e)^t}$ para certo t tal que $\mathbb{F}_{(q^e)^t}$ seja um subcorpo próprio de $\mathbb{F}_{q^k}$.*

Demonstração. Se $d = 2$, $x_Q = Dx_{Q'}$. Tem-se que $D, x_{Q'} \in \mathbb{F}_{q^e}$ e portanto $x_Q \in \mathbb{F}_{q^e}$.

Se $d = 4$, suponha que $\mathbb{F}_{(q^e)^2}$ seja construído como $\mathbb{F}_{q^e}[X]/(X^2 - D)$. Tem-se que $x_Q = D^{1/2}x_{Q'}$. Em $\mathbb{F}_{(q^e)^2}$, $X^2 = \xi$, portanto $D^{1/2} = X \in \mathbb{F}_{(q^e)^2}$ e sabe-se que $x_{Q'} \in \mathbb{F}_{q^e}$. Logo, $x_Q \in \mathbb{F}_{(q^e)^2}$.

Se $d = 6$, suponha que $\mathbb{F}_{(q^e)^3}$ seja construído como $\mathbb{F}_{q^e}[X]/(X^3 - D)$. Tem-se que $x_Q = D^{1/3}x_{Q'}$. Em $\mathbb{F}_{(q^e)^3}$, $X^3 = D$, portanto $D^{1/3} = X \in \mathbb{F}_{(q^e)^3}$ e sabe-se que $x_{Q'} \in \mathbb{F}_{q^e}$. Logo, $x_Q \in \mathbb{F}_{(q^e)^3}$.

Nos três casos, tem-se que x_Q é um elemento de um subcorpo próprio de $\mathbb{F}_{q^k}$. □

De acordo com o Teorema 3.7, o valor $1/V_P(\phi_d(Q')) = 1/(x_{\phi_d(Q')} - x_P)$ é um elemento de um subcorpo próprio de $\mathbb{F}_{q^k}$ se $d \neq 3$, e pelo Teorema 3.6 ele pode ser ignorado no cálculo do emparelhamento. Assim, ao se usar curvas *twists* pode-se simplificar o algoritmo de Miller como listado no Algoritmo 3.10. Se $d = 3$, pode-se utilizar uma outra abordagem demonstrada em [44] onde se trocam as divisões por $V_P(Q)$ por algumas multiplicações.

Algoritmo 3.10 Algoritmo de Miller com eliminação de denominador

Entrada: $P \in E(\mathbb{F}_q)[r]$, $Q' \in E'(\mathbb{F}_{q^e})$, $r \in \mathbb{N}$
Saída: $f'_{r,P}(Q') \in \mathbb{F}_{q^k}$

1: $(r_n, r_{n-1}, \ldots, r_0)_2 \leftarrow r$
2: $x \leftarrow 1$
3: $Z \leftarrow P$
4: **for** $i = n - 1$ **to** 0 **do**
5: $\quad x \leftarrow x^2 \cdot T_Z(\phi_d(Q'))$
6: $\quad Z \leftarrow 2Z$
7: $\quad$ **if** $r_i = 1$ **then**
8: $\quad\quad x \leftarrow x \cdot L_{Z,P}(\phi_d(Q'))$
9: $\quad\quad Z \leftarrow Z + P$
10: $\quad$ **end if**
11: **end for**
12: **return** x

Exemplo 3.8. *Considere a curva $E\colon y^2 = x^3+3x+4$ sobre $\mathbb{F}_{13}$. Ela possui $n = 14$ pontos. Para $r = 7$, o seu grau de mergulho é $k = 2$. Suponha que o corpo $\mathbb{F}_{13^2}$ seja construído como $\mathbb{F}_{13}[u]/(a^2 - 2)$. Para $D = 2$, $E(\mathbb{F}_q)$ possui um* twist *quadrático $E'(\mathbb{F}_q)\colon y^2 =$*

$x^3 + 4x + 7$. *Temos que o emparelhamento simplificado de Tate para* E *é definido em* $\tau_7' \colon E(\mathbb{F}_{13})[r] \times E'(\mathbb{F}_{13}) \to \mu_7$. *O isomorfismo* ϕ_2 *é calculado por* $(x, y) \mapsto (2x, 2yu)$ *(pois* $u^2 = 2$, *logo* $2^{1/2} = u$, *e* $D^{3/2} = u^3 = 2u$*).*

3.7.3 Emparelhamento Ate

Em 2003, Duursma e Lee [20] introduziram uma técnica para curvas hiperelípticas que reduz o número de iterações no *loop* do algoritmo de Miller. Posteriormente, Barreto et al. [3] generalizaram tal construção para variedades algébricas supersingulares (onde se incluem curvas elípticas), criando o denominado emparelhamento eta. Finalmente, Hess et al. [31] estenderam e simplificaram o emparelhamento eta criando o emparelhamento Ate, permitindo o uso de curvas ordinárias. Para compreender o emparelhamento Ate, serão necessários alguns resultados preliminares, que serão descritos a seguir.

Definição 3.9. *Seja* $E/\mathbb{F}_q$ *uma curva elíptica. O* mapa de Frobenius de q-ésima potência $\pi_q \colon E(\mathbb{F}_{q^e}) \mapsto E(\mathbb{F}_{q^e})$ *é definido por* $(x, y) \mapsto (x^q, y^q)$ *para qualquer* $e \geq 1$.

Esse mapa de Frobenius para curvas elípticas (que aplica uma potência do mapa de Frobenius para corpos finitos nas coordenadas do ponto) é ilustrado a seguir.

Exemplo 3.10. *Seja* $E/\mathbb{F}_{13} \colon y^2 = x^3 + 3x + 4$ *com* $k = 2$, $r = 7$, $\mathbb{F}_{13^2}$ *construído como* $\mathbb{F}_{13}[u]/(u^2 - 2)$. *Para* $P = (11, 4)$, $\pi_{13}(P) = P$ *(já que para* $x \in \mathbb{F}_p$, $x^p = x$*). Para* $Q = (u + 10, 5u + 11)$, $\pi_{13}(Q) = (12u + 10, 8u + 11)$ *e* $\pi_{13}^2(Q) = \pi_{13}(\pi_{13}(Q)) = Q$ *(pelo mesmo motivo).*

O mapa traço, definido a seguir, é baseado no mapa de Frobenius e tem a curiosa propriedade de mapear pontos na curva sobre um corpo de extensão de volta para a curva sobre o corpo original.

Definição 3.11. *Seja* $E/\mathbb{F}_q$ *uma curva elíptica com grau de mergulho* k *em relação a* r. *O* mapa traço $\mathrm{tr} \colon E(\mathbb{F}_{q^k}) \mapsto E(\mathbb{F}_q)$ *é definido por* $\mathrm{tr}(P) = P + \pi_q(P) + \ldots + \pi_q^{k-1}(P)$.

Exemplo 3.12. *Para a mesma curva e pontos do exemplo anterior, tem-se que* $\mathrm{tr}(Q) = Q + \pi_q(Q) = (u + 10, 5u + 11) + (12u + 10, 8u + 11) = (5, 1)$.

Com isso, pode-se definir o subgrupo de traço zero, que é utilizado no emparelhamento Ate.

Definição 3.13. *Seja* $E/\mathbb{F}_q$ *uma curva elíptica com grau de mergulho* k *em relação a* r. *O* subgrupo de traço zero *de* $E(\mathbb{F}_{q^k})[r]$ *é definido por* $\{P \in E(\mathbb{F}_{q^k})[r] \mid \mathrm{tr}(P) = \infty\}$.

Teorema 3.14. *[46, Seção 3.3] Seja* $E/\mathbb{F}_q$ *uma curva elíptica com grau de mergulho* k *em relação a* r. *Seja* $\mathbb{T}$ *o subgrupo de traço zero de* $E(\mathbb{F}_{q^k})[r]$. *Então* $\mathbb{T}$ *é um grupo cíclico de ordem* r, *e para todo* $P \in \mathbb{T}$ *tem-se que* $\pi_q(P) = qP$.

Teorema 3.15. *[46, Seção 6.4] Seja $E/\mathbb{F}_q$ uma curva elíptica com grau de mergulho k em relação a r. Seja $E'/\mathbb{F}_{q^e}$ um* twist *de grau d de $E/\mathbb{F}_{q^e}$ com $k = d \cdot e$. Tem-se que o conjunto $\{\phi_d(P') \mid P' \in E'(\mathbb{F}_{q^e})[r]\}$ é o subgrupo de traço zero de $E(\mathbb{F}_{q^k})[r]$.*

Em resumo, o subgrupo de traço zero é um grupo cíclico de ordem r, e os pontos nas curvas de *twist* correspondem a pontos neste subgrupo. Portanto, sempre que curvas de *twist* são utilizadas, o grupo $\mathbb{G}_2$ acaba sendo o subgrupo de traço zero. Além disso, para pontos do subgrupo de traço zero, tem-se que $qP = \pi_q(P)$. Agora é possível definir o emparelhamento Ate, através do seguinte teorema.

Teorema 3.16. *[31, Teorema 1] Seja $E/\mathbb{F}_q$ uma curva elíptica, n a ordem de $E(\mathbb{F}_q)$, r um primo grande com $r \mid n$, k o grau de mergulho de $E(\mathbb{F}_q)$ em relação a r e t o traço de Frobenius. Para $T = t - 1$, $Q \in \mathbb{G}_2 = E(\mathbb{F}_{q^k})[r] \cap Ker(\pi_q - [q])$ e $P \in G_1 = E(\mathbb{F}_{q^k})[r] \cap Ker(\pi_q - [1])$, tem-se que:*

- $\text{ate}(Q, P) = f_{T,Q}(P)$ *define um emparelhamento bilinear, denominado emparelhamento Ate;*
- *seja $N = \gcd(T^k - 1, q^k - 1)$ e $T^k - 1 = LN$, com k sendo o grau de mergulho, então*
$$\tau_r(Q, P)^L = f_{T,Q}(P)^{c(q^k-1)/N}$$
onde $c = \sum_{i=0}^{k-1} T^{k-1-i} q^i \equiv kq^{k-1} \pmod{r}$;
- *para $r \nmid L$, o emparelhamento Ate é não degenerado.*

Comparado ao Tate, as principais diferenças do emparelhamento Ate é que o algoritmo de Miller percorre os bits de $T = t-1$ ao invés de r, e a ordem dos argumentos é invertida. Como, pelo Teorema 2.13, o traço de Frobenius tem aproximadamente metade do tamanho de q, tem-se que o tamanho do *loop* de Miller é cortado pela metade.

O emparelhamento de Ate somente funciona para grupos específicos. Considere o grupo $\mathbb{G}_1 = E(\mathbb{F}_{q^k})[r] \cap Ker(\pi_q - [1])$. O kernel de uma função são os valores do domínio que são mapeados para a identidade; e [1] é o mapa da unidade ($[1]P = P$). Assim, tem-se que $Ker(\pi_q - [1]) = \{P \mid \pi_q(P) - P = \infty\} = \{P \mid \pi_q(P) = P\}$. Como visto, $\pi_q(P) = P$ se e somente se $P \in E(\mathbb{F}_q)$. Portanto, $\mathbb{G}_1 = E(\mathbb{F}_q)[r]$, que era o mesmo grupo sendo usado até então. Seguindo o mesmo raciocínio, tem-se que $Ker(\pi_q - [q]) = \{P \mid \pi_q(P) = qP\}$. Assim, pelo Teorema 3.14, $\mathbb{G}_2$ é precisamente o subgrupo de traço zero de $E(\mathbb{F}_{q^k})[r]$.

A importância do uso de curvas *twists* fica mais claro no emparelhamento Ate. Como $\mathbb{G}_2$ é usado como primeiro argumento, o algoritmo de Miller necessita da aritmética da curva elíptica correspondente. Assim, usar uma curva *twist* é muito mais eficiente, pois

pode-se efetuar tal aritmética no *twist* e somente calcula-se ϕ_d dos pontos quando eles servem de parâmetro para as funções das retas.

Pode-se verificar que a eliminação de denominador também se aplica ao emparelhamento Ate. Assim, lista-se o algoritmo de Miller para Ate e suas variantes no Algoritmo 3.11. Note que o antigo parâmetro r agora é denominado s, já que não é mais utilizado r e sim $t-1$; além disso, em variantes do Ate descritas a seguir, outros valores são utilizados para s.

Algoritmo 3.11 Algoritmo de Miller com eliminação de denominador para variantes do Ate

Entrada: $P \in E(\mathbb{F}_q)[r]$, $Q' \in E'(\mathbb{F}_{q^e})$, $s \in \mathbb{Z}$
Saída: $f'_{s,Q'}(P) \in \mathbb{F}_{q^k}$

1: $(s_n, s_{n-1}, \ldots, s_0)_2 \leftarrow s$
2: $x \leftarrow 1$
3: $Z' \leftarrow Q'$
4: **for** $i = n-1$ **to** 0 **do**
5: $\quad x \leftarrow x^2 \cdot T_{\phi_d(Z')}(P)$
6: $\quad Z' \leftarrow 2Z'$
7: $\quad$ **if** $s_i = 1$ **then**
8: $\quad\quad x \leftarrow x \cdot L_{\phi_d(Z'),\phi_d(Q')}(P)$
9: $\quad\quad Z \leftarrow Z' + Q'$
10: $\quad$ **end if**
11: **end for**
12: **return** x

Exemplo 3.17. *Seja* $E/\mathbb{F}_{13}\colon y^2 = x^3 + 3x + 4$ *uma curva elíptica com grau de mergulho* $k = 2$ *em relação a* $r = 7$, $\mathbb{F}_{13^2}$ *construído como* $\mathbb{F}_{13}/(u^2 - 2)$. *O emparelhamento Ate para esta curva é degenerado (o resultado é sempre* 1*), pois* $t = 0$, $T = -1$, $N = 168$ *e portanto* $L = 0$ *e assim* $r \mid L$.

Exemplo 3.18. *Seja* $E/\mathbb{F}_{19}\colon y^2 = x^3 + 3$ *uma curva elíptica com grau de mergulho* $k = 12$ *em relação a* $r = 13$. *Como* $t = 7$, $T = 6$, $N = 455$, $L = 4784137$ *e assim* $r \nmid L$, *o emparelhamento Ate para esta curva não é degenerado. O* loop *de Miller para Tate percorreria os bits de* $r = 13$, *mas o Ate percorre os bits de* $T = 6$.

Exemplo 3.19. *Um exemplo concreto no nível de segurança de 128 bits. Seja* $E/\mathbb{F}_p\colon y^2 = x^3 + 22$ *uma curva elíptica com grau de mergulho* $k = 12$ *em relação a* $r = \#E(\mathbb{F}_p)$ *para o seguinte* p. *Tem-se que*

```
p = 0x2523648240000001BA344D8000000008612100000000013A700000000000013
```

```
r = 0x2523648240000001BA344D8000000007FF9F800000000010A10000000000000D
T = 0x6181800000000030600000000000006,
```

e T possui a metade do tamanho de r.

3.7.4 Emparelhamentos Ótimos

Em 2008, Lee et al. [37] publicaram um novo emparelhamento denominado R-ate. Enquanto o Ate é capaz de dividir o tamanho *loop* de Miller por 2, o R-ate consegue dividir por $\varphi(k)$, onde φ é a função tociente de Euler; por exemplo, $\varphi(12) = 4$. Logo depois, Vercauteren [76] mostrou outro emparelhamento denominado Optimal Ate com a mesma propriedade; introduziu o conceito de emparelhamento ótimo (capaz de dividir o *loop* e Miller por $\varphi(k)$); e conjecturou que tal divisão é a melhor possível. Finalmente, Hess [32] demonstrou que tal conjectura era verdadeira. Paralelamente, também foram propostos outros emparelhamentos ótimos, como o Xate [57]. Nesta seção, será descrito como são feitos os cálculos de tais emparelhamentos. Provas de suas correções serão omitidas e podem ser encontradas nos artigos originais.

Nos três emparelhamentos, o primeiro parâmetro passado para a função de Miller pode ser negativo. Neste caso, utiliza-se a igualdade $f_{s,Q}(P) = 1/(f_{-s,Q}(P) \cdot V_{sQ}(P))$ [76, Seção 2.3]. Ao se utilizar curvas *twist*, devido à eliminação de denominador, não é necessário computar a função da reta vertical e tem-se que $f_{s,Q}(P) = 1/f_{-s,Q}(P)$.

R-ate

Embora possa ser adaptado para várias famílias de curvas, o emparelhamento R-ate será descrito para curvas BN. Seja x o parâmetro de uma curva BN sobre $\mathbb{F}_p$ e $f = f_{6x+2,Q}(P)$. Então tem-se que o emparelhamento R-ate é dado por

$$R_r(Q,P) = (f \cdot (f \cdot L_{(6x+2)Q,Q}(P))^p \cdot L_{\pi_p((6x+2)Q+Q),(6x+2)Q}(P))^{(p^k-1)/r}.$$

É importante notar que $(6x+2)Q$ é computado implicitamente junto com f pelo algoritmo de Miller; $(6x+2)Q + Q$ é computado implicitamente no cálculo da função da reta $L_{(6x+2)Q,Q}(P)$; a potência por p é facilmente calculada com auxílio do mapa de Frobenius; e finalmente, π_p também é facilmente calculado com o mapa de Frobenius.

Optimal Ate

Analogamente ao R-ate, este emparelhamento pode ser adaptado para inúmeras famílias de curvas, mas será descrita sua versão para curvas BN. Seja x o parâmetro da curva BN sobre $\mathbb{F}_p$ e $f = f_{6x+2,Q}(P)$. Então tem-se que o emparelhamento Optimal Ate é dado por

$$\mathrm{OptAte}_r(Q,P) = (f \cdot L_{Q_3,-Q_2}(P) \cdot L_{-Q_2+Q_3,Q_1}(P) \cdot L_{Q_1-Q_2+Q_3,(6x+2)Q}(P))^{(p^k-1)/r},$$

onde $Q_i = \pi_p^i(Q)$. As mesmas observações do R-ate se aplicam.

Xate

Novamente, descreve-se este emparelhamento no contexto de curvas BN. Seja x o parâmetro da curva BN sobre $\mathbb{F}_p$ e $f = f_{-x,Q}(P)$. Então tem-se que o emparelhamento Xate é dado por

$$\begin{aligned}\chi_r(Q,P) = (f^{1+p+p^3+p^{10}} &\cdot L_{-xQ,\pi_p(-xQ)}(P) \cdot L_{\pi_p^3(-xQ),\pi_p^{10}(-xQ)}(P) \\ &\cdot L_{-xQ+\pi_p(-xQ),\pi_p^3(-xQ)+\pi_p^{10}(-xQ)}(P))^{(p^k-1)/r}.\end{aligned}$$

É interessante ressaltar que enquanto o Optimal Ate e R-ate percorrem os bits de $6x+2$ no *loop* de Miller, o Xate percorre os bits de x. Ao se criar curvas BN, prefere-se utilizar valores de x com baixo peso de Hamming, pois assim tem-se menos adições no *loop* de Miller e menos multiplicações na exponenciação final, como será visto. Contudo, normalmente $6x+2$ terá um peso de Hamming um pouco maior para um x com peso de Hamming pequeno. Assim, é de se esperar que o emparelhamento Xate seja um pouco mais eficiente, desde que as computações adicionais (que são um pouco mais complexas do que as do Optimal Ate e R-ate) não anulem tal ganho.

3.7.5 Exponenciação Final

A exponenciação final consiste em elevar o resultado do algoritmo de Miller a $(p^k - 1)/r$. Em curvas MNT e BN, $\rho = 1$ e portanto $r \approx p$. Tem-se então que tal expoente final tem aproximadamente $n(k-1)$ bits, onde n é o número de bits de p. Com um algoritmo tradicional de exponenciação baseado em elevar ao quadrado e multiplicar, seriam necessárias $n(k-1)$ iterações. Embora seja factível, tal computação pode ser bastante melhorada.

Assuma que k é par, o que é verdade para os casos estudados neste trabalho. Seja $d = k/2$. Pode-se verificar que o expoente final pode ser decomposto em três fatores, tornando-se

$$(p^k - 1)/r = (p^d - 1) \cdot ((p^d + 1)/\Phi_k(p)) \cdot (\Phi_k(p)/r),$$

onde Φ_k é o k-ésimo polinômio ciclotômico. A potência ao primeiro fator é fácil de se calcular, pois para elevar a p^d pode-se usar o mapa de Frobenius. O segundo fator

também é fácil pelo mesmo motivo, pois a divisão sempre resultará num polinômio de p. O terceiro fator é o mais caro de se computar, mas pode ser simplificado em famílias de curvas elípticas como observado em [65]. Cada família abordada neste trabalho será estudada a seguir.

Curvas MNT, $k = 4$

Como visto, nesta família p é dado por $p(x) = x^2 + x + 1$ e $r = n(x) = (x^2 + 2x + 2)$ para a família A e $r = n(x) = x^2 + 1$ para a família B. Tem-se que $\Phi_4(p) = p^2 + 1$. Note que o segundo fator do expoente final é 1. O terceiro fator é $(p^2 + 1)/r$.

Para a família A, obtém-se que o terceiro fator pode ser escrito como $(p(x)^2+1)/n(x) = x^2+1 = p-x$. Ou seja, elevar a esse fator envolve uma aplicação do mapa de Frobenius e elevar a $-x$. Para a família B, tem-se que $(p(x)^2 + 1)/n(x) = x^2 + 2x + 2 = p + x + 1$. De modo parecido, elevar a esse fator envolve uma aplicação do mapa de Frobenius e elevar a x.

Para famílias de curvas MNT com cofator $h > 1$, procede-se de maneira análoga mas escrevendo $r = n(x)/h$.

Curvas MNT, $k = 6$

Tem-se que $p(x) = 4x^2 + 1$, $r = n(x) = 4x^2 - 2x + 1$ e $\Phi_6(p) = p^2 - p + 1$. O segundo fator é dado por $p + 1$ e o terceiro por $(p^2 - p + 1)/r$. Assim, o terceiro fator pode ser escrito como $(p(x)^2 - p(x) + 1)/n(x) = 4x^2 + 2x + 1 = p + 2x$. Ou seja, elevar esse fator envolve uma aplicação do mapa de Frobenius, elevar a x e calcular um quadrado.

Curvas BN, $k = 12$

Nesta família, $p(x) = 36x^4+36x^3+24x^2+6x+1$ e $r = n(x) = 36x^4+36x^3+18x^2+6x+1$. Tem-se que $\Phi_{12}(p) = p^4 - p^2 + 1$. O segundo fator é dado por $p^2 + 1$ e o terceiro por $(p^4 - p^2 + 1)/r$.

Para se simplificar o terceiro o fator, o processo é mais complicado, e refere-se a [65] para mais detalhes. Em resumo, computa-se

$$
\begin{aligned}
y_0 &= m^p m^{p^2} m^{p^3} & y_3 &= (m^x)^p \\
y_1 &= 1/m & y_4 &= m^x/((m^{x^2})^p) \\
y_2 &= (m^{x^2})^{p^2} & y_5 &= 1/m^{x^2} \\
& & y_6 &= m^{x^3}(m^{x^3})^p\,,
\end{aligned}
$$

onde pode-se verificar que somente são necessárias três potências de x, mapas de Frobenius e algumas inversões (que são baratas neste ponto do algoritmo, como será visto). Finalmente, obtém-se o resultado final com $y_0 y_1^2 y_2^6 y_3^{12} y_4^{18} y_5^{30} y_6^{36}$, que pode ser computado com duas variáveis temporárias, 9 multiplicações e 4 quadrados [65].

Elementos Unitários

Existe uma última otimização para a exponenciação final que é bastante eficaz. Ela parte da observação que um elemento de $\mathbb{F}_{p^k}$, representado por $\mathbb{F}_{p^d}[w](w^2 - \beta)$ com $d = k/2$, possui uma propriedade especial após ser elevado pelo primeiro fator $(p^d - 1)$ da exponenciação final. Seja $c = c_0 + c_1 w \in \mathbb{F}_{p^k}$ um elemento que foi previamente elevado a $(p^d - 1)$. Então tem-se que

$$c_0^2 - \beta c_1^2 = 1\,.$$

Pode-se provar tal fato observando que $c = a^{(p^d-1)} = (a_0 + a_1 w)^{(p^d-1)} = (a_0 - a_1 w)^{p^d}/(a_0 + a_1 w) = (a_0 - a_1 w)/(a_0 + a_1 w)$ devido ao mapa de Frobenius, e então utilizando a fórmula para inverso em extensão quadrática para se obter c_0 e c_1 e finalmente mostrar que a equação é válida.

Um elemento que possui essa propriedade é denominado *elemento unitário* [62, Seção 3]. Pode-se verificar que esta propriedade se mantém sob exponenciações subsequentes. A grande vantagem fornecida por ela é que a inversão de um elemento unitário é dada por $(c_0 + c_1 w)^{-1} = (c_0 - c_1 w)$. Inclusive pode-se mover o efeito da exponenciação para trás, fazendo com que o inverso de qualquer número que *será* elevado a $(p^d - 1)$ possa ser calculado dessa forma, pois

$$\left(\frac{1}{a_0 + a_1 w}\right)^{(p^d-1)} = \frac{a_0 + a_1 w}{a_0 - a_1 w} = (a_0 - a_1 w)^{(p^d-1)}\,.$$

Elementos unitários também possuem quadrados mais eficientes. Considere $(c_0 + c_1 w) = (a_0 + a_1 w)^2$. Sabe-se que, usando quadrado Karatsuba, $c_0 = a_0^2 + \beta a_1^2$ e $c_1 = (a_0 + a_1)^2 - a_0^2 - a_1^2$. Se a for unitário, $a_0^2 = \beta a_1^2 + 1$. Substituindo, pode-se computar c com $c_0 = 2a_1^2\beta + 1$ e $c_1 = (a_0 + a_1)^2 - a_1^2 - \beta a_1^2 - 1$, o que requer 2 quadrados, 5 somas, 2 multiplicações por β e duas somas de constantes (-1). Em comparação ao quadrado Complexo, isso resulta na troca de duas multiplicações por dois quadrados e uma soma a mais.

3.7.6 Duplicação e Soma Encapsuladas de Pontos

No algoritmo de Miller, os cálculos das funções das retas sempre vêm acompanhado de uma duplicação ou adição de ponto: o cálculo de $T_Z(Q)$ é seguido do cálculo de $2Z$ e o

cálculo de $L_{Z,P}(Q)$ é seguido do cálculo de $Z + P$. Foi observado que tais computações possuem alguns elementos em comum [10], e portanto elas podem ser mais eficientes se efetuadas conjuntamente. Tal técnica é denominada *duplicação e soma encapsuladas de pontos*.

Duplicação

Considere a duplicação de um ponto $Z = (x_1, y_1, z_1)$ em coordenadas Jacobianas. As coordenadas de $2Z = (x_3, y_3, z_3)$ são dadas por

$$\begin{aligned} x_3 &= (3x_1^2 + az_1^4)^2 - 8x_1y_1^2 \\ y_3 &= (3x_1^2 + az_1^4)(4x_1y_1^2 - x_3) - 8y_1^4 \\ z_3 &= 2y_1z_1 \end{aligned}$$

e podem ser computadas usando variáveis temporárias da seguinte forma:

$$\begin{aligned} t_1 &\leftarrow y_1^2 & t_5 &\leftarrow 3x_1^2 + az_1^4 \\ t_2 &\leftarrow 4x_1t_1 & x_3 &\leftarrow t_5^2 - 2t_2 \\ t_3 &\leftarrow 8t_1^2 & y_3 &\leftarrow t_5(t_2 - x_3) - t_3 \\ t_4 &\leftarrow z_1^2 & z_3 &\leftarrow 2y_1z_1\,. \end{aligned}$$

Se $a = -3$, então pode-se calcular t_5 com $t_5 \leftarrow 3(x_1 - t_4)(x_1 + t_4)$, totalizando quatro quadrados e quatro multiplicações (4S + 4M). Se $a = 0$, então $t_5 \leftarrow 3x_1^2$ e tem-se 4S + 3M.

Agora, considere o cálculo da função $T_Z(Q)$ para verificar se é possível reutilizar algumas das variáveis temporárias do cálculo de $2Z$. Como visto, $T_Z(Q) = (y_Q - y_1) - \lambda(x_Q - x_1)$, com $\lambda = (3x_1^2 + a)/(2y_1)$. Pode-se converter tal função em uma versão Jacobiana fazendo a troca de variáveis $x_1 \mapsto x_1/z_1^2$, $y_1 \mapsto y_1/z_1^3$, obtendo-se $T_Z(Q) = (y_Q - y_z/z_z^3) - \lambda(x_s - x_1/z_1^2)$ com $\lambda = (3x_1^2 + az_1^4)/(2y_1z_1)$. Note que $\lambda = t_5/z_3$, portanto podemos reaproveitar t_5 e z_3. O resto da função T_Z será estudado adiante, caso a caso.

Adição

Para a adição de um ponto $R = (x_1, y_1, z_1)$ em coordenadas Jacobianas com um ponto $P = (x_P, y_p)$ em coordenadas afins, tem-se que $R + P = (x_3, y_3, z_3)$ em coordenadas Jacobianas pode ser computado com

$$
\begin{aligned}
t_1 &\leftarrow z_1^2 & t_7 &\leftarrow t_5^2 \\
t_2 &\leftarrow z_1 t_1 & t_8 &\leftarrow t_5 t_7 \\
t_3 &\leftarrow x_P t_1 & t_9 &\leftarrow x_1 t_7 \\
t_4 &\leftarrow y_P t_2 & x_3 &\leftarrow t_6^2 - (t_8 + 2t_9) \\
t_5 &\leftarrow t_3 - x_1 & y_3 &\leftarrow t_6(t_9 - x_3) - y_1 t_8 \\
t_6 &\leftarrow t_4 - y_1 & z_3 &\leftarrow z_1 t_5 \,,
\end{aligned}
$$

totalizando 3S + 8M. A função relacionada à reta entre R e P é dada por $L_{R,P}(Q) = (y_Q - y_P) - \lambda(x_Q - x_P)$ com $\lambda = (y_P - y_1)/(x_P - x_1)$. Convertendo-se para coordenadas Jacobianas, $L_{R,P}$ mantém-se igual, trocando-se apenas o valor λ para $\lambda = (y_P z_1^3 - y_1)/(z(x_P z_1^2 - x_1))$. Note que $\lambda = t_6/z_3$.

Os demais detalhes são específicos para o grau de mergulho e tipo de emparelhamento (Tate ou variantes de Ate), portanto serão estudados a seguir para cada caso relevante neste trabalho.

Curvas MNT, Tate, $k = 4$

Neste caso, o emparelhamento de Tate usando *twist* mapeia pontos $P \in E(\mathbb{F}_p)$ e $Q' \in E'(\mathbb{F}_{p^2})$ em um elemento de $\mathbb{F}_{p^4}$. Suponha que o corpo $\mathbb{F}_{p^2}$ seja construído como $\mathbb{F}_p[u](u^2 - \beta)$ e $\mathbb{F}_{p^4}$ como $\mathbb{F}_{p^2}[v](v^2 - u)$. Escolhendo-se $D = u$, o isomorfismo ϕ_2 é dado por $(x, y) \mapsto (xu, yuv)$. O mesmo raciocínio pode ser aplicado para outras representações e outros valores de D, mas tais escolhas são as mais eficientes.

Assim, para a duplicação, tem-se

$$
\begin{aligned}
T_P(Q') &= T_P(\phi_2(Q')) = T_P(x_{Q'}u, y_{Q'}uv) \\
&= (y_{Q'}uv - \frac{y_1}{z_1^3}) - \frac{t_5}{z_3}(x_{Q'}u - \frac{x_1}{z_1^2}) \,,
\end{aligned}
$$

Para eliminar os denominadores, pode-se multiplicar os dois lados por $z_3 z_1^2 = 2y_1 z_1^3$, efetivamente trocando-se a função $T_P(Q')$ por outra função $T_P^*(Q') = z_3 z_1^2 T_P(Q')$. De acordo com o Teorema 3.6, o fator $z_3 z_1^2$ será eliminado pela exponenciação final já que $y_1, z_1 \in \mathbb{F}_p$, portanto, neste contexto, $T_P^*(Q') = T_P(Q')$. Escreva $x_{Q'} = (x_{Q'0} + x_{Q'1}u)$ e similarmente para $y_{Q'}$. Assim, tem-se

$$\begin{aligned} T_P^*(Q') &= (z_3 z_1^2 y_{Q'} uv - 2y_1^2) - t_5(z_1^2 x_{Q'} u - x_1) \\ &= (z_3 t_4 y_{Q'} uv - 2t_1) - t_5(t_4 x_{Q'} u - x_1) \\ &= (-2t_1 + t_5 x_1) + (-t_4 t_5 x_{Q'})u + (z_3 t_4 y_{Q'})uv \\ &= (-2t_1 + t_5 x_1 - t_4 t_5 x_{Q'1}\beta) + (-t_4 t_5 x_{Q'0})u + (z_3 t_4 y_{Q'1}\beta)v + (z_3 t_4 y_{Q'0})uv\,. \end{aligned}$$

Portanto, $c = (c_0 + c_1 u) + (c_2 + c_3 u)v = T_P^*(Q') \in \mathbb{F}_{p^4}$ é dado por

$$\begin{aligned} c_0 &= -2t_1 + t_5 x_1 - t_4 t_5 x_{Q'1}\beta \\ c_1 &= -t_4 t_5 x_{Q'0} \\ c_2 &= z_3 t_4 y_{Q'1}\beta \\ c_3 &= z_3 t_4 y_{Q'0}\beta\,, \end{aligned}$$

totalizando 11M + 4S para $a = -3$.

Para a soma, tem-se

$$\begin{aligned} L_{R,P}(Q') = L_{R,P}(\phi_2(Q')) &= L_{R,P}(x_{Q'} u, y_{Q'} v) \\ &= (y_{Q'} uv - y_P) - \frac{t_6}{z_3}(x_{Q'} u - x_P)\,. \end{aligned}$$

Elimina-se o denominador multiplicando-se os dois lados por z_3, trocando $L_{R,P}(Q')$ por $L_{R,P}^*(Q') = z_3 L_{R,P}(Q')$. Novamente, z_3 será eliminado pela exponenciação final. Assim,

$$\begin{aligned} L_{R,P}^*(Q') &= (z_3 y_{Q'} uv - z_3 y_P) - t_6(x_{Q'} u - x_P) \\ &= (t_6 x_P - z_3 y_P) + (-t_6 x_{Q'})u + (z_3 y_{Q'})uv \\ &= (t_6 x_P - z_3 y_P - t_6 x_{Q'1}\beta) + (-t_6 x_{Q'0})u + (z_3 y_{Q'1}\beta)v + (z_3 y_{Q'0})uv\,. \end{aligned}$$

Portanto, $c = (c_0 + c_1 u) + (c_2 + c_3 u)v = T_{R,P}^*(Q') \in \mathbb{F}_{p^4}$ é dado por

$$\begin{aligned} c_0 &= t_6 x_P - z_3 y_P - t_6 x_{Q'1}\beta \\ c_1 &= -t_6 x_{Q'0} \\ c_2 &= z_3 y_{Q'1}\beta \\ c_3 &= z_3 y_{Q'0}\,, \end{aligned}$$

totalizando 14M + 3S.

Curvas MNT, Tate, $k = 6$

Neste caso, o emparelhamento de Tate usando *twist* mapeia pontos $P \in E(\mathbb{F}_p)$ e $Q' \in E'(\mathbb{F}_{p^3})$ em um elemento de $\mathbb{F}_{p^6}$. Suponha que o corpo $\mathbb{F}_{p^3}$ seja construído como $\mathbb{F}_p[u](u^3 - \beta)$ e $\mathbb{F}_{p^6}$ como $\mathbb{F}_{p^3}[v](v^2 + u)$. Escolhendo-se $D = -u$, o isomorfismo ϕ_2 é dado por $(x, y) \mapsto (-xu, -yuv)$.

Derivando as fórmulas de forma análoga ao caso $k = 4$, obtém-se que $c = (c_0 + c_1u + c_2u^2) + (c_3 + c_4u + c_5u^2)v = T^*_P(Q') \in \mathbb{F}_{p^6}$ é dado por

$$\begin{aligned}
c_0 &= -2t_1 + t_5x_1 + t_4t_5x_{Q'2}\beta \\
c_1 &= t_4t_5x_{Q'0} \\
c_2 &= t_4t_5x_{Q'1} \\
c_3 &= -z_3t_4y_{Q'2}\beta \\
c_4 &= -z_3t_4y_{Q'0} \\
c_5 &= -z_3t_4y_{Q'1}\,,
\end{aligned}$$

totalizando $13M + 4S$ para $a = -3$, enquanto $c = (c_0 + c_1u + c_2u^2) + (c_3 + c_4u + c_5u^2)v = T^*_{R,P}(Q') \in \mathbb{F}_{p^6}$ é dado por

$$\begin{aligned}
c_0 &= t_6x_P - z_3y_P + t_6x_{Q'2}\beta \\
c_1 &= t_6x_{Q'0} \\
c_2 &= t_6x_{Q'1} \\
c_3 &= -z_3y_{Q'2}\beta \\
c_4 &= -z_3y_{Q'0} \\
c_5 &= -z_3y_{Q'1}\,,
\end{aligned}$$

totalizando $16M + 3S$.

Curvas BN, Família Ate, $k = 12$

Neste caso, o emparelhamento Ate (ou variantes) mapeia pontos $Q' \in E(\mathbb{F}_{p^2})$ e $P \in E(\mathbb{F}_p)$ em um elemento de $\mathbb{F}_{p^{12}}$. Suponha que o corpo $\mathbb{F}_{p^2}$ seja construído como $\mathbb{F}_p[u](u^2 - \beta)$, $\mathbb{F}_{p^6}$ como $\mathbb{F}_{p^2}[v](v^3 - \xi)$ e $\mathbb{F}_{p^{12}}$ como $\mathbb{F}_{p^6}[w](w^2 - v)$ com ξ um não-resíduo sêxtico em $\mathbb{F}_{p^2}$. Escolhendo-se $D = \xi$, o isomorfismo ϕ_6 é dado por $(x, y) \mapsto (xv, yvw)$. (Essa é a abordagem é sugerida em [17].)

Pode-se derivar as fórmulas de forma análoga às curvas MNT, lembrando que neste caso os parâmetros P e Q' têm papéis trocados. Obtém-se que $c = (c_0 + c_1v + c_2v^2) + (c_3 + c_4v + c_5v^2)w = T^*_{Q'}(P) \in \mathbb{F}_{p^{12}}$, com $c_i \in \mathbb{F}_{p^2}$, é dado por

$$
\begin{aligned}
c_0 &= z_3 t_4 y_P \\
c_1 &= 0 \\
c_2 &= 0 \\
c_3 &= -t_5 t_4 x_P \\
c_4 &= t_5 x_{Q'} - 2t_1 \\
c_5 &= 0\,,
\end{aligned}
$$

totalizando $8M+4S$, enquanto $c = (c_0+c_1v+c_2v^2)+(c_3+c_4v+c_5v^2)w = L^*_{R,Q'}(P) \in \mathbb{F}_{p^{12}}$, com $c_i \in \mathbb{F}_{p^2}$, é dado por

$$
\begin{aligned}
c_0 &= z_3 y_P \\
c_1 &= 0 \\
c_2 &= 0 \\
c_3 &= -t_6 x_P \\
c_4 &= t_6 x_{Q'} - z_3 y_{Q'} \\
c_5 &= 0\,,
\end{aligned}
$$

totalizando $12M+3S$. É importante notar que ambos os resultados são esparsos, portanto pode-se melhorar o desempenho do algoritmo de Miller utilizando uma função específica para multiplicar um elemento de $\mathbb{F}_{p^{12}}$ por um elemento esparso de $\mathbb{F}_{p^{12}}$. A Tabela 3.4 apresenta um resumo dos custos da duplicação e soma encapsuladas de pontos.

	Duplicação		Soma	
Curva	M	S	M	S
MNT, $k=4$, $a=-3$	11	4	14	3
MNT, $k=6$, $a=-3$	13	4	16	3
BN	8	4	12	3

Tabela 3.4: Custos da duplicação e soma encapsuladas de pontos; M = multiplicações, S = quadrados

3.8 *Hashing* em $\mathbb{G}_1$ e $\mathbb{G}_2$

A maior parte dos protocolos de CBE, incluindo o NIKDP, necessitam de duas funções de *hash* H_1 e H_2 que mapeiam identidades (que são *strings* arbitrárias) em pontos de $\mathbb{G}_1$

e $\mathbb{G}_2$, respectivamente. Nesta seção, será estudado como tais funções são implementadas.

A abordagem mais simples consiste em fixar dois pontos $G_1 \in \mathbb{G}_1$, $G_2 \in \mathbb{G}_2$. Seja r a ordem de ambos os grupos. Então, pode-se tomar $H_1(\mathrm{ID}) = H(\mathrm{ID}) \cdot G_2$ e $H_2(\mathrm{ID}) = H(\mathrm{ID}) \cdot G_2$, onde H é uma função de *hash* que mapeia identidades em números no intervalo $[0, r-1]$, que é mais simples de se definir. Contudo, existe um grande problema com tal construção, e a maioria dos protocolos seriam simples de se quebrar se ela fosse utilizada. Considere o protocolo NIKDP, e suponha que Alice calcula uma chave para se comunicar com Beto através da chave combinada $e(S_{1\mathrm{A}}, H_2(\mathrm{ID_B}))$. Denote por k_X o multiplicador $H(\mathrm{ID_X})$. Pode-se verificar que a chave combinada pode ser escrita como $e(k_\mathrm{A} s G_1, k_\mathrm{B} G_2)$, onde todos os valores são públicos, com exceção de s. Em particular, se for possível para um atacante calcular sG_1, ele poderá calcular a chave. Suponha, então, que Eva seja uma participante do sistema. Assim, ela pode obter sua chave privada $S_{1\mathrm{E}}$ do centro. Pode-se escrever $S_{1\mathrm{E}} = k_\mathrm{E} s G_1$, e multiplicando tal valor por $k_\mathrm{E}^{-1} \pmod r$, Eva obtém sG_1 (analogamente, também obtém sG_2) e consegue calcular a chave combinada entre qualquer par de participantes do sistema.

Assim, são necessárias funções de *hash* que mapeiem identidades em pontos P de forma que não se saiba escrever $P = kG$ para algum ponto base fixo G. Primeiramente, pode-se mapear identidades no grupo $E(K)$ de uma curva elíptica mapeando-se a identidade em um valor x de K, e calculando-se $y = \sqrt{x^3 + ax + b}$. Caso a raiz quadrada não exista, pode-se incrementar x e tentar novamente. Em [33, 15] o processo é mais detalhado, inclusive fornecem-se algoritmos determinísticos para o *hash* quando $p \equiv 2 \pmod 3$.

Resta apenas um ponto a ser esclarecido, que surge ao se considerar que $\mathbb{G}_1 = E(K)[r]$, isto é, os pontos de r-torção em $E(K)$. Como r é primo, então $\mathbb{G}_1$ consiste nos pontos de ordem r mais o ponto no infinito. Sabe-se como mapear uma identidade em $E(K)$, mas falta saber como garantir que o ponto tenha ordem r. Seja n a ordem de $E(K)$ e $c = n/r$, também denominado cofator. Então, pode-se garantir que o ponto mapeado tenha ordem r calculando-se $cH_1(\mathrm{ID})$. Em $\mathbb{G}_2$ procede-se de maneira análoga. Se for utilizada curva *twist* de grau d, então o cofator consiste em $c = n'/r$, onde n' é a ordem do *twist* $E'(\mathbb{F}_{q^{k/d}})$ que pode ser calculada de acordo com [31, Proposição 2]. Neste caso, é importante notar que o cofator é maior (do tamanho de p se o twist for em $E(\mathbb{F}_{p^2})$, por exemplo), portanto H_2 é mais cara que H_1. Mesmo assim, alguns trabalhos como [64] fornecem técnicas para se melhorar a performance de H_2. Vale lembrar que, se for utilizado o emparelhamento de Tate, não é necessário multiplicar pelo cofator em $\mathbb{G}_2$ já que qualquer ponto é representante de uma classe lateral.

Capítulo 4

Implementação

Neste capítulo será descrita a implementação em software realizada e novas otimizações propostas, que representam o resultado final do trabalho descrito nesta dissertação. A arquitetura alvo é a família de microcontroladores MSP430 da Texas Instruments, utilizada em sensores como o Tmote Sky da Moteiv[1], o TelosB da Crossbow[2] e o TinyNode da Shockfish[3].

Foram implementados o esquema ECDSA nos níveis de segurança de 80 e 128 bits; assim como o esquema NIKDP nos níveis de 64 bits (curva MNT com $k = 4$), 70 bits (curva MNT com $k = 6$), 80 e 128 bits (curvas BN).

4.1 A Família MSP430

A MSP430 é uma família de microcontroladores com CPUs de 16 bits, onde cada membro da família varia em *clock*, espaço de ROM, RAM e periféricos; demais características são iguais para todos. Os sensores Tmote Sky e TelosB usam o microcontrolador MSP430F1611 que possui 8 MHz de *clock*, 10 KB de RAM e 48 KB de ROM.

A MSP430 possui 16 registradores — 12 deles de propósito geral — nomeados `Rn`, onde `n` é o número do registrador. O `R0` é o *program counter* (`PC`) que aponta para a próxima instrução a ser executada; o `R1` é o *stack pointer* (`SP`) que aponta para o topo da pilha; o `R2` é o *status register* (`SR`) com as *flags* de *carry* (`C`), *overflow* (`V`), negativo (`N`), zero (`Z`), entre outros; e o `R3` sempre vale zero.

Existem 27 instruções, listadas no apêndice A; em particular notam-se *shifts* de somente um bit e ausência de divisão e multiplicação. As mais importantes, utilizadas nas

[1] `http://www.moteiv.com/`
[2] `http://www.xbow.com/`
[3] `http://www.tinynode.com/`

seções a seguir, estão listadas na Tabela 4.1. Muitas delas possuem versões orientadas a bytes que são especificadas usando-se o sufixo "`.b`" (por exemplo, `mov.b`).

Instrução	Operação
`mov` *src*`,`*dst*	`dst = src`
`add` *src*`,`*dst*	`dst += src` (soma)
`addc` *src*`,`*dst*	`dst += src + C` (soma com *carry*)
`adc` *dst*	`dst += C` (adiciona *carry*)

Tabela 4.1: Instruções principais da família MSP430

Existem quatro modos de endereçamento:

Register direct: o valor de um registrador. Exemplo: `adc r7`.

Register indexed: o valor no endereço de memória apontado pelo registrador mais um *offset*. Exemplo: `adc 2(r7)` adiciona o *carry* no endereço `r7 + 2`. Requer uma palavra de *offset* após a instrução.

Register indirect: o valor no endereço de memória apontado pelo registrador. Exemplo: `adc @r7`.

Register indirect with post-increment: igual ao *register indirect*, mas incrementa o registrador após a operação. Exemplo: `adc @r7+`.

Em instruções com dois operandos, os modos *register indirect* e *indirect with post-increment* não são disponíveis para o operando de destino. O número de ciclos que cada instrução leva para executar pode ser calculado de forma relativamente simples. Primeiramente, é necessário um ciclo para se ler a instrução propriamente dita. Adiciona-se um ciclo para cada origem em memória (leitura) e dois ciclos para cada destino em memória (escrita) — a leitura do destino necessária em instruções como `add` está inclusa nesses dois ciclos. Adiciona-se um ciclo para cada palavra de *offset* utilizada no modo *register indexed*. As únicas exceções à essas regrais gerais são: instruções de dois operandos sem palavras de *offset* que escrevem no PC, que precisam de um ciclo a mais; *jumps* que sempre levam dois ciclos; e as instruções `push` (dois ciclos a mais), `call` (4 ciclos em *direct* e *indexed*, senão 5 ciclos) e `reti` (5 ciclos).

Pode-se usar as seguintes constantes imediatas sem o uso de palavras de *offset*: -1, 0, 1, 2, 4 e 8. Assim, por exemplo, zerar um registrador do jeito tradicional com `mov #0,r7` leva apenas um ciclo. Internamente, este recurso é implementado através de uma codificação que utiliza modos de endereçamento que não fazem sentido para `R2` e `R3`.

Alguns modelos da família MSP430, incluindo o MSP430F1611, possuem um multiplicador em hardware como um periférico mapeado em memória que é vital para a implementação eficiente da aritmética de corpos finitos. Ele suporta quatro operações: multiplicar, multiplicar com sinal, multiplicar e acumular, e multiplicar e acumular com sinal. Para se utilizá-lo, opera-se da seguinte forma. Primeiramente, escreve-se o primeiro operando em um de quatro endereços específicos na memória; o endereço usado especifica a operação desejada. Tais endereços são apelidados de `MPY` (multiplicar), `MPYS` (multiplicar com sinal), `MAC` (multiplicar e acumular) e `MACS` (multiplicar e acumular com sinal). Prossegue-se escrevendo o segundo operando no endereço `OP2`; tal escrita também indica o começo da operação. Dois ciclos depois, o resultado de 32 bits fica disponível em dois endereços da memória: a parte baixa em `RESLO` e a parte alta em `RESHI`. Nas operações com acumulação, o *carry* é escrito no endereço `SUMEXT`.

4.2 Ferramentas

Utilizou-se o compilador MSPGCC versão 3.2.3. Para executar os programas, foi utilizado o simulador MSPsim [21]; observou-se a presença de alguns *bugs* neste simulador com respeito às escritas nas *flags* do *status register* em algumas instruções e também no multiplicador em hardware. Por ser um projeto *open source*, foi possível consertar tais *bugs*, gerando *patches* que foram enviados ao projeto[4]. Por esse motivo, não se utilizou nenhuma versão específica do simulador e sim uma versão baseada no controle de versão SVN do projeto, revisão 582.

Para se testar a correção dos algoritmos implementados, foram gerados vetores de teste com o auxílio do Sage [71], um sistema de software matemático *open source*.

A implementação foi baseada na biblioteca MIRACL [5], mas a maioria dos algoritmos foi implementada separadamente. De tal biblioteca, foi utilizada a rotina de inversão em corpo finito e outras funções auxiliares. Posteriormente, integrou-se nossa implementação com a biblioteca *open source* RELIC [6].

4.3 Parâmetros

Esta seção visa detalhar e justificar as escolhas de parâmetros utilizadas na implementação.

[4] `http://sourceforge.net/tracker/?group_id=207133&atid=1000388`
[5] `http://www.shamus.ie`
[6] `http://code.google.com/p/relic-toolkit/`

4.3.1 CCE

Implementou-se o esquema ECDSA por ser um dos mais populares e padronizados [8, 56] esquemas na CCE. Dois níveis de segurança foram utilizados: 80 e 128 bits, que utilizam corpos finitos de 160 e 256 bits respectivamente. O primeiro é considerado o mínimo aceitável (inclusive já sendo abandonado pelo NIST) e o segundo considerado seguro por um bom tempo (até 2040 [69]). As curvas utilizadas foram a secp160r1 [8] e a P-256 [56].

Para a aritmética de corpo finito, foram utilizadas a multiplicação Karatsuba e Comba e a redução especial específica para as curvas citadas. Para multiplicação de ponto, implementou-se o wNAF para ponto aleatório, Comb para ponto fixo e *Interleaving* para pontos simultâneos.

4.3.2 CBE

Para emparelhamentos, foram implementadas curvas MNT com $k = 4, 6$ sobre corpos de 160 bits e curvas BN sobre corpos de 160 e 256 bits. Curvas MNT com $k = 4$ foram implementadas em [72], o trabalho mais rápido para MSP430 até então, e portanto permitem comparações imediatas. Contudo, elas fornecem segurança de cerca de 64 bits (devido ao tamanho do corpo de extensão $\mathbb{F}_{p^4}$, com 640 bits). O uso de $k = 6$ ameniza o problema, elevando o nível de segurança para cerca de 70 bits. Também foram utilizadas curvas BN com corpos de 160 bits para fornecer segurança de 80 bits, e curvas BN sobre corpos de 256 bits para fornecer 128 bits de segurança.

Para curvas MNT, utilizou-se o emparelhamento de Tate. O motivo para tal escolha é que o *hash* em $\mathbb{G}_2$ é mais simples, como mencionado. O uso de variantes do Ate não traria ganhos substanciais já que $\varphi(k) = 2$ para $k = 4, 6$ e portanto pode-se cortar o *loop* de Miller somente pela metade; mas em compensação se torna necessária a aritmética em $E(\mathbb{F}_{p^2})$ que é substancialmente mais cara e negaria os ganhos com o corte do *loop*. Para curvas BN, contudo, o uso de variantes ótimas de Ate é vantajoso porque o *loop* é cortado em um quarto.

A curva MNT com $k = 4$ utilizada é a mesma de [72], dada por $E/\mathbb{F}_p\colon y^2 = x^3 - 3x + b$, a ordem de $E(\mathbb{F}_P)$ sendo $n = hr$, com

```
p = 0xE3F367D542C82027F33DC5F3245769E676A5755D
b = 0xDABC0397E45200C4DF4CF67714DB64EB866BA034
r = 0x6B455E0A014F1E30EAEF7300BD4BB4258290FC5
h = 0x22.
```

A curva MNT com $k = 6$ utilizada é dada por $E/\mathbb{F}_p\colon y^2 = x^3 - 3x + b$, a ordem de $E(\mathbb{F}_p)$ sendo $n = hr$, com

```
p = 0x7DDCA613A2E3DDB1749D0195BB9F14CF44626303
b = 0x21C3F3AC7864D1F1F99273D0F828D3657D8CFD4E
r = 0x3EEE5309D171EED8BA4E12DEF44414FD17D369B7
h = 0x2.
```

A curva BN sobre corpo de 160 bits utilizada é dada por $E/\mathbb{F}_p$: $y^2 = x^3 + 3$ e gerada por x, a ordem de $E(\mathbb{F}_p)$ sendo n, com

```
x = 0x4000000031
p = 0x240000006ED000007FE9C000419FEC800CA035C7
n = 0x240000006ED000007FE96000419F59800C9FFD81.
```

A curva BN sobre corpo de 256 bits utilizada é a sugerida em [57], $E/\mathbb{F}_p$: $y^2 = x^3 + 22$ e gerada por x, a ordem de $E(\mathbb{F}_p)$ sendo n, com

```
x = -0x4080000000000001
p = 0x2523648240000001BA344D8000000008612100000000013A700000000000013
n = 0x2523648240000001BA344D8000000007FF9F800000000010A10000000000000D.
```

Nas duas últimas curvas, x possui peso de Hamming de apenas quatro e três respectivamente, o que acelera o cálculo do emparelhamento. Também pode-se verificar que nessas curvas os primos utilizados têm na verdade 158 bits e 254 bits respectivamente. Isso acarreta na perda de um bit de segurança em cada curva, o que pode ser considerado aceitável. Por clareza, tal detalhe será omitido adiante.

Para a aritmética de corpos finitos, foram usadas a multiplicação Karatsuba e Comba e redução Montgomery.

4.4 Aritmética de Corpos Finitos

Elementos de $\mathbb{F}_p$ são representados por vetores de inteiros de 16 bits. A soma e subtração de tais elementos pode ser implementada de modo bastante eficiente com as instruções `addc` e `subc` com auxílio do modo de endereçamento *register indirect with post-increment*. Já a multiplicação e redução são mais complexas e serão descritas a seguir.

4.4.1 Multiplicação

A implementação mais rápida de multiplicação para a MSP430 conhecida até então era a descrita em [66, 72], onde é utilizado o método Híbrido com grupos de tamanho 2. Tal método parece ser o mais adequado sempre que existem registradores o suficiente na

plataforma, como é o caso da MSP430. Porém, tal abordagem não considera todos os recursos disponíveis da plataforma.

Estudando-se a multiplicação Comba, pode-se observar que cada coluna do resultado é computada através de múltiplos passos de multiplicar e acumular. Seguindo a notação da seção 3.2.2, para cada coluna de índice k, multiplicam-se e acumulam-se os pares a_i, b_j tal que $i + j = k$. A observação chave para se melhorar o desempenho da multiplicação é que tal passo de multiplicar e acumular é precisamente o que é fornecido pela operação `MAC` do multiplicador em hardware da MSP430.

Suponha que o ponteiro para o operando a esteja em `r15` e o ponteiro do operando b esteja em `r14`, e os acumuladores sejam `r4`, `r5` e `r6`, do menos para o mais significativo. Considere o passo de multiplicar e acumular de $a_0 b_1$. Tal passo, na multiplicação Comba normal, está ilustrado na Figura 4.1 e consiste no seguinte código *assembly*:

```
mov 0(r14),&__MPY
mov 2(r15),&__OP2
add &__RESLO,r4
addc &__RESHI,r5
adc r6
```

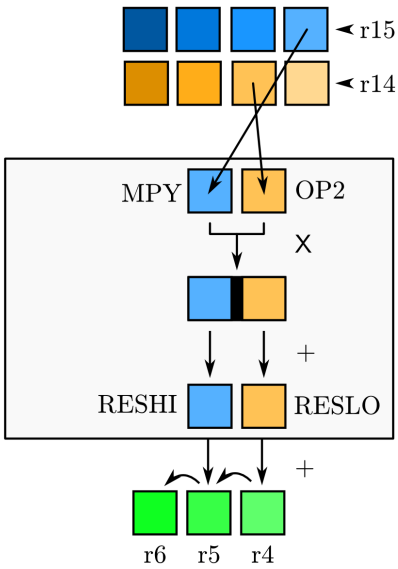


Figura 4.1: Passo na multiplicação Comba, computando $a_0 b_1$

O mesmo passo, utilizando MAC, está ilustrado na Figura 4.2 e consiste no seguinte código *assembly*:

```
mov 0(r14),&__MAC
mov 2(r15),&__OP2
addc &__SUMEXT,r6
```

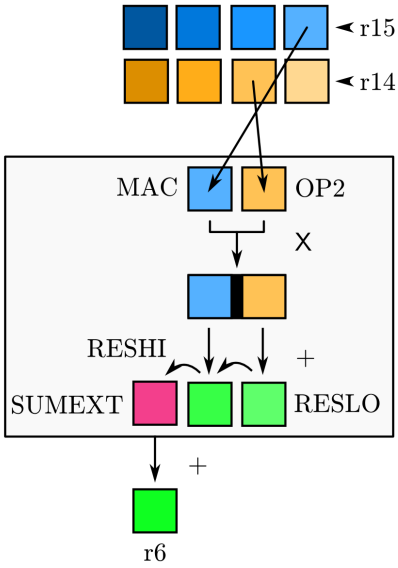


Figura 4.2: Passo na multiplicação Comba usando MAC, computando $a_0 b_1$

A versão utilizando MAC possui duas instruções, uma leitura e um endereço em palavra de *offset* a menos em relação à versão comum, utilizando portanto 4 ciclos a menos. Como cada passo é executado n^2 vezes, com n sendo o número de inteiros dos operandos, então tal ganho se torna considerável. Naturalmente, nota-se que tal comparação é com a multiplicação Comba normal, e não com a multiplicação Híbrida, que não possui um conceito de "passo" tão bem definido, dificultando comparações diretas. Contudo, resultados experimentais confirmam que o uso de Comba com MAC (a seguir denominado apenas Comba MAC) fornece um desempenho melhor do que o uso de Híbrida. Também pode-se verificar o ganho em ciclos examinando-se a Tabela 4.2, que apresenta o número de instruções e ciclos necessários para se multiplicar operandos de 160 bits. Pode-se observar que o número de adições é bastante reduzido, já que o multiplicador em hardware fica responsável pela maioria delas. É interessante notar que a importância do passo de multiplicar e acumular já havia sido ressaltada em [28], onde se propõe que seja adicionada uma instrução de multiplicar e acumular em novas arquiteturas RISC.

O uso do Comba MAC, sozinho, não fornece ganhos o suficiente em relação à multiplicação Híbrida. Contudo, o Comba MAC deixa oito registradores livres. Tais registradores podem ser usados como um *cache* para os operandos. Um desses registradores

		Comba MAC		Híbrida em [72]	
Instrução	CPI	Instruções	Ciclos	Instruções	Ciclos
add @reg,reg	2	99	198		
Outras adições				309	709
mov x(reg),&label	6	20	120	45	270
mov reg,x(reg)	4			20	80
mov reg,reg	1			27	27
mov reg,&label	4	89	356	100	400
mov x(reg),reg	3	13	39	45	135
mov @reg+,&label	5	100	500		
mov @reg,&label	5	29	145		
mov @reg,x(reg)	5	20	100		
Outros			128		167
Total			1586		1746

Tabela 4.2: Comparação de números de instruções para a multiplicação Comba MAC e Híbrida para números de 160 bits

pode ser usado para guardar o endereço de `SUMEXT`, permitindo a modificação da adição de *carry* no passo do Comba MAC para `addc @r4,r6` (supondo que `r4` guarda o endereço do `SUMEXT`), economizando um ciclo adicional a cada passo. Tal técnica exige a reordenação das instruções, visto que o *carry* da acumulação da operação MAC demora dois ciclos para ser escrito, e desta forma ele seria lido antes disso. Pode-se simplesmente escrever o primeiro operando do próximo passo e então ler o *carry*, e tal problema é contornado. Finalmente, uma última otimização refere-se no modo de endereçamento da leitura do primeiro operando em cada passo. Observando-se o algoritmo Comba, verifica-se que o primeiro operando é acessado de forma linear. Por exemplo, para a terceira coluna ($k = 2$), tem-se as multiplicações a_0b_2, a_1b_1, a_2b_0; observe que o acesso ao operando a é linear. Assim, pode-se trocar a primeira linha do passo Comba com MAC para `mov @r14+,&__MAC`, economizando mais um ciclo a cada passo. O registrador é ajustado no fim de cada coluna para apontar para o lugar correto no início da próxima coluna.

Tais otimizações, apesar de não muito complexas, se tornam complicadas de se lidar ao se escrever o código totalmente desenrolado. Por esse motivo, foi escrito um pequeno programa em Python responsável por gerar todo o código em *assembly* da multiplicação, de modo parametrizado pelo número de inteiros dos operandos, aplicando todas as otimizações descritas.

4.4.2 Redução Modular

Como mencionado na seção 3.3.1, a redução Montgomery possui a mesma estrutura que uma multiplicação Comba, com a diferença que um dos operandos é calculado no percorrer do algoritmo. Por esse motivo, a mesma otimização da multiplicação Comba usando o MAC se aplica à redução Montgomery, com ganhos significativos.

A técnica de multiplicação modular híbrida para curvas BN [22] foi implementada, mas sem resultados satisfatórios, como será discutido na próxima seção. Contudo, descobriu-se uma outra otimização também somente aplicável a curvas BN, porém restrita às curvas gerada pelos parâmetros $x = 2^{38} + 2^5 + 2^4 + 1$ (primo de 158 bits) e $x = -2^{62} - 2^{55} - 1$ (primo de 254 bits), esse último sugerido em [57]. Estudando-se os primos $p(x)$ gerados por tais parâmetros, pode-se verificar que eles possuem as seguintes formas na base 2^{16}, com dígitos na base 16, respectivamente:

```
(2400,0000,6ed0,0000,7fe9,c000,419f,ec80,0ca0,35c7),

(2523,6482,4000,0001,BA34,4D80,0000,0008,
 6121,0000,0000,0013,A700,0000,0000,0013).
```

Os dois dígitos zero no primeiro primo e os cinco dígitos zero no segundo permitem um ganho significativo na redução Montgomery. Por exemplo, para o segundo primo, antes eram necessários $16^2 = 256$ passos de multiplicar e acumular. Ignorando-se os passos envolvendo multiplicações por zero, são necessários $256 - 16 \cdot 5 = 176$ passos, uma redução de 31%. O dígito 1 do segundo primo também permite mais ganhos, visto que passos envolvendo ele se tornam simples adições.

Outros dígitos também inspiram otimizações. Por exemplo, `0x4000`, cuja multiplicação consiste em apenas dois *shifts* à direita. Porém, não se obteve resultados satisfatórios para tais elementos devido à falta de registradores para se calcular tais *shifts*; pois registradores livres são usados como *cache*, e o ganho obtido com essas otimizações não compensou a perda com o *cache*. A otimização de `0x0008`, cuja multiplicação consiste em três *shifts* à esquerda, já é cara o suficiente para custar o mesmo que a multiplicação comum. Assim, aparenta não ser possível explorar os demais dígitos de tais primos nesta plataforma.

Capítulo 5

Resultados

Nesta seção serão apresentados os resultados de nossa implementação em software, em especial os desempenhos dos algoritmos e protocolos implementados. As medições foram realizadas no simulador MSPsim, que mede exatamente o número de ciclos gastos por função. Os valores em segundos foram derivados supondo um *clock* de 8 milhões de hertz; o *clock* exato é dependente do modelo específico da família MSP430 e da voltagem sendo utilizada, mas são próximos deste valor. Tempos de outros trabalhos também foram derivados da mesma forma a partir do número de ciclos reportados pelos autores.

Vale notar que sensores mais modernos, como o TinyNode, utilizam microcontroladores MSP430 com 16 MHz de *clock*. Assim, o tempo necessário para o cálculo dos algoritmos em tais sensores é cortado pela metade.

O tamanho do código dos programas foi medido com o utilitário `msp430-size` do MSPGCC. O uso de RAM máximo foi estimado usando o simulador MSPsim, que apresenta um gráfico do uso de RAM ao decorrer da execução do programa.

5.1 Aritmética de Corpos Finitos

Os tempos para multiplicação e quadrado estão listados na Tabela 5.1. Pode-se observar que o Comba MAC fornece, de fato, um desempenho melhor em relação à multiplicação Híbrida; em particular, há um ganho de 9,2% em relação a [66]. Para números de 256 bits, pode-se utilizar um nível de Karatsuba (portanto com multiplicador de 128 bits) para se obter um ganho de cerca de 100 ciclos (2,5%); níveis adicionais não valeriam a pena pois a margem de ganho já não é muito grande no primeiro nível.

Os tempos para redução modular estão listados na Tabela 5.2. O tempo da redução do trabalho [72] foi estimado subtraindo-se o tempo de multiplicação presente em [66] do tempo total da multiplicação em corpo finito presente em [72]. Pode-se observar que o Montgomery MAC é realmente eficiente e fornece um ganho de 44,5% em relação

Algoritmo	Ciclos	Tempo (ms)
Multiplicação de 160 bits		
Híbrida em [66]	1 746	0,22
Comba MAC	1 586	0,20
Quadrado de 160 bits		
Comba MAC	1 371	0,19
Multiplicação de 256 bits		
Híbrida (Karatsuba, Comba de 128 bits)	4 025	0,50
Comba MAC (Karatsuba, Comba de 128 bits)	3 597	0,45
Comba MAC (Comba de 256 bits)	3 689	0,46
Quadrado de 256 bits		
Comba MAC (Karatsuba, Comba de 128 bits)	2 960	0,37

Tabela 5.1: Tempos da multiplicação e quadrado

a [72]. Usando-se os primos esparsos para curvas BN descritos previamente obtém-se uma economia de 236 ciclos em 160 bits e 926 ciclos em 256 bits, gerando um ganho de 14,2% e 25,7% respectivamente em relação ao Montgomery MAC genérico.

Finalmente, os tempos para multiplicação em corpo finito estão listados na Tabela 5.3, onde novamente se confirma a eficácia da utilização do MAC. Observe que o desempenho do método HMMB foi um pouco pior que o Comba MAC. De fato, a redução no HMMB ficou mais rápida (2755 ciclos); porém a sua multiplicação é mais cara (4986 ciclos). Tal aumento é causado por dois motivos. O primeiro é o aumento dos operandos (que requerem 18 inteiros ao invés de 16), o que causa um impacto considerável devido à natureza quadrática da multiplicação. O segundo é o fato do resultado intermediário da multiplicação do HMMB possuir o dobro do tamanho (aproximadamente 64 inteiros ao invés de 32); como as operações envolvidas no final de cada coluna (escrever resultado na memória, fazer "*shift*" dos acumuladores que também estão na memória, devido ao MAC) têm um custo razoável, o fato de haver o dobro delas causa um impacto significativo. Assim, considera-se que o HMMB não é vantajoso para esta plataforma, preferindo-se o uso do primo esparso, já que ambos são específicos para curvas BN.

5.2 Criptografia de Curvas Elípticas

Os tempos para operações no corpo e para multiplicação de ponto no contexto de Criptografia de Curvas Elípticas estão listados na Tabela 5.4. Foi utilizado Comba MAC de 160 bits e redução especial para a curva secp160r1 e Karatsuba com Comba MAC de 128

Algoritmo	Ciclos	Tempo (ms)
Módulo primo de 160 bits		
Montgomery em [72] (estimado)	2 988	0,37
Montgomery MAC	1 659	0,21
Montgomery MAC, primo esparso	1 423	0,18
SECG (primo: $2^{160} - 2^{31} - 1$)	342	0,04
Módulo primo de 256 bits		
Montgomery	4 761	0,60
Montgomery MAC	3 600	0,45
Montgomery MAC, primo esparso	2 674	0,33
NIST (primo: $2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$)	709	0,09

Tabela 5.2: Tempos da redução modular

Algoritmo	Ciclos	Tempo (ms)
Multiplicação de 160 bits		
Híbrida em [72]	4 734	0,59
Comba MAC	3 263	0,41
Comba MAC, primo esparso	3 027	0,38
Quadrado de 160 bits		
Comba MAC	3 046	0,38
Comba MAC, primo esparso	2 805	0,35
Inverso de 160 bits		
Euclides estendido	187 575	23,45
Multiplicação de 256 bits		
Híbrida	8 855	1,11
Comba MAC	7 198	0,90
Comba MAC, primo esparso	6 272	0,78
HMMB MAC	7 797	0,97
Quadrado de 256 bits		
Comba MAC	6 581	0,82
Comba MAC, primo esparso	5 655	0,71
Inverso de 256 bits		
Euclides estendido	380 254	47,53

Tabela 5.3: Tempos da multiplicação em corpo finito (usando redução Montgomery)

	secp160r1		P-256	
Algoritmo	Ciclos	Tempo (ms)	Ciclos	Tempo (ms)
Operações no corpo				
Multiplicação	1 952	0,24	4 327	0,54
Quadrado	1 734	0,22	3 679	0,46
Inversão	187 575	19,27	292 170	36,52
Multiplicação de ponto aleatório				
4NAF	4 417 661	552	13 372 271	1 672
5NAF	4 433 104	554	13 188 903	1 649
Escada de Montgomery	6 319 383	790	20 476 234	2 560
Desconhecido de [77]		800		
Multiplicação de ponto fixo				
Comb, $w = 4$	1 831 063	229	5 688 793	711
Comb, $w = 4$ em [73]		720		
Sliding window, $w = 4$ em [77]		720		
Multiplicação simultânea				
Interleaving ($w_k = 4$, $w_l = 5$)	5 204 544	651	15 784 176	1 973

Tabela 5.4: Tempos das operações no corpo e multiplicação de ponto

bits e redução especial para a curva P-256. Pode-se observar que o método 5NAF não fornece ganho significativo em relação ao 4NAF, portanto este último é mais vantajoso por requerer menos espaço de memória. Os tempos obtidos são melhores que os presentes em [77], porém seus autores ressaltam que sua implementação não era muito otimizada. A multiplicação de ponto com a escada de Montgomery [45] (sem relação com a redução Montgomery) é resistente a certos ataques por canais secundários, porém como pode ser visto, é mais cara.

Os tempos para os métodos do ECDSA estão listados na Tabela 5.5. Pode-se perceber que a mudança do nível de segurança de 80 bits para 128 bits causa um impacto razoável nos tempos de execução, aproximadamente os triplicando.

O espaço em ROM e uso máximo de RAM de um programa que assina e verifica uma pequena mensagem estão listados na Tabela 5.6. Verifica-se que ambos são menores que os valores em [73]; mas é difícil determinar exatamente o motivo de tal redução. É importante notar que para a curva P-256, o tamanho do programa é menor usando Karatsuba com Comba de 128 bits do que usando Comba de 256 bits. Portanto, além de um pequeno ganho de desempenho com o uso de um nível de Karatsuba, também há um ganho de cerca de 12,9% no tamanho da ROM.

	secp160r1		P-256	
Algoritmo	Ciclos	Tempo (ms)	Ciclos	Tempo (ms)
Geração de chave	1 849 903	231	5 682 433	710
Assinar	2 166 906	270	5 969 593	746
Verificar	5 488 568	686	16 139 555	2 017

Tabela 5.5: Tempos do ECDSA

Versão	ROM (KB)	RAM (KB)
secp160r1		
Comba de 160 bits	23,5	2,5
Comba de 160 bits em [73]	31,3	2,9
P-256		
Karatsuba com Comba de 128 bits	25,7	3,5
Comba de 256 bits	29,5	3,5

Tabela 5.6: Tamanho de ROM e máximo de RAM utilizada em programas usando curvas elípticas

5.3 Criptografia Baseada em Emparelhamentos

Para curvas MNT, os tempos dos cálculos de emparelhamentos estão listados na Tabela 5.7. Deve-se ressaltar que em [72] foi desativada a otimização do compilador para se gerar os programas; os autores justificam esta escolha por terem determinado que o desempenho dos programas, ao ativarem a otimização, variava bastante de acordo com o compilador utilizado, e assim o tempo sem otimização permitiria comparações mais justas. Porém, pode-se verificar que o desempenho de código compilado sem otimização é extremamente dependente da estrutura do código. Por exemplo, o custo de chamadas de função é muito mais caro ao se desativar a otimização porque todos os parâmetros são escritos na pilha. Acredita-se, portanto, que fornecer os tempos com a otimização ativada permite comparações mais justas, e eventuais discrepâncias entre compiladores podem ser facilmente esclarecidas fornecendo os tempos obtidos nos diferentes compiladores utilizados. Para permitir a comparação com [72], são fornecidos os tempos dos emparelhamentos obtidos com a otimização desligada. Verifica-se então que o uso do MAC fornece um ganho de 21,6% no cálculo do emparelhamento com $k = 4$.

Os tempos do cálculo de emparelhamento para curvas BN estão listados na Tabela 5.8. Observa-se prontamente que o custo da elevação do nível de segurança para 128 bits é bastante caro, praticamente triplicando o tempo obtido com curvas MNT com $k = 6$,

Algoritmo	Otimização	Ciclos	Tempo (s)
Curva MNT, $k = 4$			
Tate em [72]	Não	37 739 040	4,717
Tate (MAC)	Não	29 597 568	3,700
Tate (MAC)	Sim	25 916 270	3,240
Curva MNT, $k = 6$			
Tate (MAC)	Sim	39 963 255	5,000

Tabela 5.7: Tempos do cálculo de emparelhamento em curvas MNT

Emparelhamento	Ciclos	Tempo (s)
Corpo de 160 bits		
Optimal Ate	40 494 840	5,062
R-ate	40 455 059	5,057
Xate	40 626 975	5,078
Corpo de 256 bits		
Optimal Ate	104 407 586	13,051
R-ate	117 514 219	13,042
Xate	103 098 195	12,887

Tabela 5.8: Tempos do cálculo de emparelhamento em curvas BN

que fornecem cerca de 70 bits de segurança. Ainda assim, tal fator é próximo ao fator obtido na elevação do nível de segurança para a Criptografia de Curvas Elípticas. Ao se considerar que esquemas baseados em emparelhamentos escalam de forma mais parecida com sistemas tipo RSA do que com sistema de curvas elípticas [25], conclui-se que tal valores são razoavelmente satisfatórios. Tal desempenho, contudo, ainda não aparenta ser factível para o uso em sensores. Uma conclusão mais detalhada dependeria do cenário específico onde o esquema criptográfico baseado em emparelhamentos fosse utilizado.

É interessante comparar o desempenho da curva BN sobre um corpo de 160 bits com a curva MNT com $k = 6$. A curva BN, apesar do seu emparelhamento ser aproximadamente 60 milissegundos mais caro, fornece 80 bits de segurança ao invés dos 70 bits fornecidos pela curva MNT. Assim, nesta plataforma, o uso de curvas MNT só aparenta ser adequado para o nível de segurança 64 bits. O desempenho da curva BN é surpreendente visto que o corpo de extensão, neste caso, possui 1920 bits, muito além dos 1248 bits necessários para fornecer 80 bits de segurança. Ainda assim, mesmo com tal desperdício computacional, o desempenho das curvas BN é satisfatório devido à grande quantidade de otimizações existentes para elas.

Versão	ROM (KB)	RAM (KB)
Curva MNT		
Comba de 160 bits	28,9	2,3
Comba de 160 bits em [72]	34,9	3,4
Curva BN		
Karatsuba com Comba de 128 bits	32,3	4,7
Comba de 256 bits	36,2	4,7

Tabela 5.9: Tamanho de ROM e máximo de RAM utilizada em programas usando emparelhamentos

Pode-se observar que os três tipos de emparelhamentos utilizados resultam em tempos muito próximos, com o Xate sendo ligeiramente mais rápido com o corpo de 256 bits. Tal resultado é coerente com o fato dos três emparelhamentos serem ótimos. O Xate não é mais rápido no corpo de 160 bits pois as multiplicações adicionais acabam anulando as vantagens de se usar x no *loop* de Miller, como mencionado.

Por último, tem-se os tamanhos de ROM e uso máximo de RAM para um programa que calcula um emparelhamento na Tabela 5.9. Verifica-se novamente a vantagem do uso de um nível de Karatsuba em curvas BN. Visto que o MSP430F1611 possui 48 KB de ROM e 10 KB de RAM, conclui-se que o uso de ROM aparenta ser um tanto alto; já o uso de RAM nem tanto visto que a maior parte é alocada na pilha e liberada logo após o cálculo do emparelhamento. Novamente, conclusões mais detalhadas dependeriam da análise de aplicações específicas.

A Figura 5.1 condensa a maior parte dos resultados obtidos.

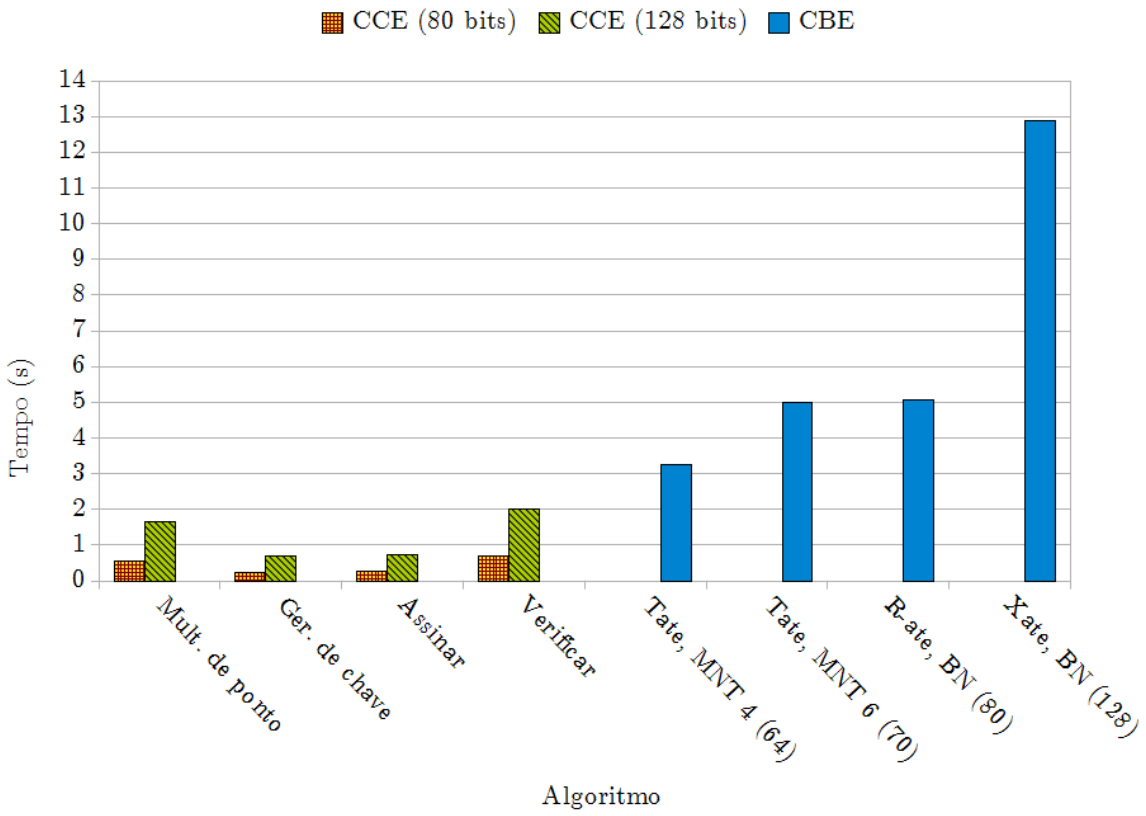


Figura 5.1: Tempos dos diversos algoritmos implementados

Capítulo 6

Conclusões e Trabalhos Futuros

Neste trabalho foi apresentada uma implementação em software de criptografia baseada em identidades com emparelhamentos para a plataforma MSP430, utilizada em redes de sensores sem fio. Suas principais contribuições consistem em:

- um estudo dos fundamentos, algoritmos e otimizações utilizados em emparelhamentos, que na sua maior parte está bastante fragmentada na literatura;
- uma nova técnica específica para a MSP430 que acelera a multiplicação e redução modular (e consequentemente acelera o cálculo de emparelhamentos), explorando o multiplicador em hardware presente em múltiplos membros da família MSP430;
- uma nova técnica para certas curvas BN para acelerar a redução modular, explorando o formato esparso do primo do corpo base da curva, que pode ser adaptada para outras plataformas de 8 e 16 bits;
- uma implementação eficiente do estado da arte de emparelhamentos para a MSP430.

Parte de tais contribuições foram detalhas em artigo [27] aceito na *10th International Conference on Cryptology in India* (INDOCRYPT 2009).

O uso de criptografia baseada em identidades em redes de sensores sem fio é promissora, porém ainda são necessários trabalhos no sentido de otimizar seu desempenho. Especificamente, o nível de segurança de 128 bits aparenta ser proibitivo para tais dispositivos.

Pode-se citar os seguintes possíveis trabalhos futuros:

- implementação de curvas binárias para o MSP430; pois aparentemente fornecem desempenho melhor em níveis de segurança mais baixos [72];

- implementação na plataforma MSP430X[1], capaz de endereçar 20 bits — portanto permitindo mais ROM e RAM — e que possui novas instruções (contudo, ferramentas que suportam tal plataforma ainda estão em estado experimental);
- implementação de emparelhamentos para o nível de segurança de 80 bits usando curvas BN para outras plataformas;
- busca por famílias de curvas com $k = 6$ e twists sêxticos ou $k = 8$ e twists quárticos e $\rho = 1$, até então não conhecidas [23], que seriam mais adequadas para o nível de segurança de 80 bits.

[1] `http://focus.ti.com/lit/ml/slap109/slap109.pdf`

Apêndice A

Instruções da MSP430

As instruções presentes na família MSP430 estão listadas nas tabelas A.1, A.2 e A.3. Instruções de *jump* presumem que houve uma comparação com `CMP` ***src,dst*** previamente. Instruções marcadas com um asterisco são "emuladas", isto é, podem ser escritas em função das outras instruções (por exemplo, `POP op` é equivalente a `MOV @SP+,op`) — contudo, o *assembler* suporta a escrita de tais instruções emuladas diretamente. Somente as instruções emuladas mais importantes estão listadas. As *flags* de status são `Z` (zero), `C` (*carry*), `N` (negativo) e `V` (*overflow*).

Instrução	Operação
`jne/jnz` ***label***	Salta se `Z == 0` (***dst*** $!=$ ***src***)
`jeq/jz` ***label***	Salta se `Z == 1` (***dst*** $==$ ***src***)
`jnc/jlo` ***label***	Salta se `C == 0` (***dst*** $<$ ***src*** sem sinal)
`jc/jhs` ***label***	Salta se `C == 1` (***dst*** $>=$ ***src*** sem sinal)
`jn` ***label***	Salta se `N == 1` (***dst*** $<$ `0`)
`jge` ***label***	Salta se `N == V` (***dst*** $>=$ ***src*** com sinal)
`jl` ***label***	Salta se `N != V` (***dst*** $<$ ***src*** com sinal)
`jmp` ***label***	Salto incondicional
`*br` ***label***	Salto absoluto

Tabela A.1: Instruções de *jump* presentes na família MSP430

Instrução	Operação
`rrc` *op*	*Shift* de *op* para a direita com *carry*. *carry* vira bit mais significativo; bit menos significativo é escrito no *carry*
`rra` *op*	*Shift* aritmético de *op* para a direita. Bit mais significativo não é alterado; bit menos significativo é escrito no *carry*
*`rla` *op*	*Shift* de *op* para a esquerda. Bit menos significativo é zerado
*`rlc` *op*	*Shift* de *op* para a esquerda. *Carry* vira bit menos significativo; bit mais significativo é escrito no *carry*
`swpb` *op*	Troca os bytes de *op*
`sxt` *op*	Estende sinal de *op*
*`inv` *op*	Inverte bits de *op*
*`clr` *op*	Zera *op*
*`tst` *op*	Escreve *flags* de acordo com *op*
*`adc` *op*	Adiciona *carry* em *op*
*`sbc` *op*	Subtrai *carry* de *op*
`push` *op*	Empilha *op*
*`pop` *op*	Desempilha em *op*
`call` *op*	Chama função no endereço *op* (empilha `PC`; escreve *op* no `PC`)
`reti`	Retorna de interrupção (desempilha `SP`; desempilha `PC`)
*`ret`	Retorna de função (desempilha `PC`)
*`nop`	Não faz nada

Tabela A.2: Instruções de até um operando presentes na família MSP430

Instrução	Operação
`mov` *src*,*dst*	`dst = src`
`add` *src*,*dst*	`dst += src`
`addc` *src*,*dst*	`dst += src + C`
`sub` *src*,*dst*	`dst -= src`
`subc` *src*,*dst*	`dst -= src + ~C`
`cmp` *src*,*dst*	`dst - src` (apenas escreve *flags* de status)
`bit` *src*,*dst*	`dst & src` (apenas escreve *flags* de status)
`and` *src*,*dst*	`dst &= src`
`bis` *src*,*dst*	`dst |= src`
`xor` *src*,*dst*	`dst ^= src`
`bic` *src*,*dst*	`dst &= ~C`
`dadd` *src*,*dst*	`dst &= src + C` (usando codificação binária decimal)

Tabela A.3: Instruções de dois operandos presentes na família MSP430

Referências Bibliográficas

[1] Leonard Adleman. A subexponential algorithm for the discrete logarithm problem with applications to cryptography. In *SFCS '79: Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, pages 55–60. IEEE Computer Society, 1979.

[2] R. Balasubramanian and Neal Koblitz. The improbability that an elliptic curve has subexponential discrete log problem under the Menezes—Okamoto—Vanstone algorithm. *Journal of Cryptology*, 11(2):141–145, 1998.

[3] Paulo S. L. M. Barreto, Steven Galbraith, Colm Ó hÉigeartaigh, and Michael Scott. Efficient pairing computation on supersingular abelian varieties. *Designs, Codes and Cryptography*, 42(3):239–271, 2007.

[4] Paulo S. L. M. Barreto, Ben Lynn, and Michael Scott. On the selection of pairing-friendly groups. In *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 17–25. Springer Berlin / Heidelberg, 2004.

[5] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In *Selected Areas in Cryptography*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer Berlin / Heidelberg, 2006.

[6] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 213–229. Springer Berlin / Heidelberg, 2001.

[7] Daniel R. L. Brown. Generic groups, collision resistance, and ECDSA. *Designs, Codes and Cryptography*, 35(1):119–152, 2005.

[8] Certicom Research. SEC 2: Recommended elliptic curve domain parameters, 2006. `http://www.secg.org/`.

[9] Sanjit Chatterjee and Alfred Menezes. On cryptographic protocols employing asymmetric pairings – the role of ψ revisited. Cryptology ePrint Archive, Report 2009/480, 2009. `http://eprint.iacr.org/`.

[10] Sanjit Chatterjee, Palash Sarkar, and Rana Barua. Efficient computation of Tate pairing in projective coordinate over general characteristic fields. In *Information Security and Cryptology — ICISC 2004*, volume 3506 of *Lecture Notes in Computer Science*, pages 168–181. Springer Berlin / Heidelberg, 2005.

[11] Jaewook Chung and M. Anwar Hasan. Asymmetric squaring formulae. In *ARITH '07: Proceedings of the 18th IEEE Symposium on Computer Arithmetic*, pages 113–122, Washington, DC, USA, 2007. IEEE Computer Society.

[12] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *Cryptography and Coding*, volume 2260 of *Lecture Notes in Computer Science*, pages 360–363. Springer Berlin / Heidelberg, 2001.

[13] P. G. Comba. Exponentiation cryptosystems on the IBM PC. *IBM Systems Journal*, 29(4):526–538, 1990.

[14] S. A. Cook. *On the Minimum Computation Time of Functions*. PhD thesis, Harvard University, 1966.

[15] Jean-Sébastien Coron and Thomas Icart. An indifferentiable hash function into elliptic curves. Cryptology ePrint Archive, Report 2009/340, 2009. `http://eprint.iacr.org/`.

[16] Augusto Jun Devegili, Colm Ó hÉigeartaigh, Michael Scott, and Ricardo Dahab. Multiplication and squaring on pairing-friendly fields. Cryptology ePrint Archive, Report 2006/471, 2006. `http://eprint.iacr.org/`.

[17] Augusto Jun Devegili, Michael Scott, and Ricardo Dahab. Implementing cryptographic pairings over Barreto-Naehrig curves. In *Pairing-Based Cryptography — Pairing 2007*, volume 4575 of *Lecture Notes in Computer Science*, pages 197–207. Springer Berlin / Heidelberg, 2007.

[18] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[19] Régis Dupont and Andreas Enge. Provably secure non-interactive key distribution based on pairings. *Discrete Applied Mathematics*, 154(2):270–276, 2006.

[20] Iwan Duursma and Hyang-Sook Lee. Tate pairing implementation for hyperelliptic curves $y^2 = x^p - x + d$. In *Advances in Cryptology — ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 111–123. Springer Berlin / Heidelberg, 2003.

[21] Joakim Eriksson, Adam Dunkels, Niclas Finne, Fredrik Österlind, and Thiemo Voigt. MSPsim – an extensible simulator for MSP430-equipped sensor boards. In *Proceedings of the European Conference on Wireless Sensor Networks (EWSN), Poster/Demo session*, Delft, The Netherlands, January 2007.

[22] Junfeng Fan, Frederik Vercauteren, and Ingrid Verbauwhede. Faster $\mathbb{F}_p$-arithmetic for cryptographic pairings on Barreto-Naehrig curves. In *Cryptographic Hardware and Embedded Systems — CHES 2009*, volume 5747 of *Lecture Notes in Computer Science*, pages 240–253. Springer Berlin / Heidelberg, 2009.

[23] David Freeman, Michael Scott, and Edlyn Teske. A taxonomy of pairing-friendly elliptic curves. *Journal of Cryptology*, 23(2):224–280, 2010.

[24] Steven D. Galbraith, J. McKee, and P. Valença. Ordinary abelian varieties having small embedding degree. *Finite Fields and Their Applications*, 13(4):800–814, 2007.

[25] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[26] Robert P. Gallant, Robert J. Lambert, and Scott A. Vanstone. Faster point multiplication on elliptic curves with efficient endomorphisms. In *Advances in Cryptology — CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200. Springer Berlin / Heidelberg, 2001.

[27] Conrado P. L. Gouvêa and Julio López. Software implementation of pairing-based cryptography on sensor networks using the MSP430 microcontroller. In *Progress in Cryptology — INDOCRYPT 2009*, volume 5922 of *Lecture Notes in Computer Science*, pages 248–262. Springer Berlin / Heidelberg, 2009.

[28] J. Großschädl. Instruction set extension for long integer modulo arithmetic on RISC-based smart cards. In *Proceedings of the 14th Symposium on Computer Architecture and High Performance Computing (SCAB-PAD'02)*, pages 13–19. IEEE Computer Society, 2002.

[29] Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and Sheueling Chang Shantz. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In *Cryptographic Hardware and Embedded Systems — CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 925–943. Springer Berlin / Heidelberg, 2004.

[30] Darrel Hankerson, Alfred Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, 2004.

[31] F. Hess, N. Smart, and F. Vercauteren. The Eta pairing revisited. *IEEE Transactions on Information Theory*, 52(10):4595–4602, 2006.

[32] Florian Hess. Pairing lattices. In *Pairing-Based Cryptography — Pairing 2008*, volume 5209 of *Lecture Notes in Computer Science*, pages 18–38. Springer Berlin / Heidelberg, 2008.

[33] Thomas Icart. How to hash into elliptic curves. In *Advances in Cryptology — CRYPTO 2009*, volume 5677 of *Lecture Notes in Computer Science*, pages 303–316. Springer Berlin / Heidelberg, 2009.

[34] Antoine Joux. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, 17(4):263–276, 2004.

[35] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics Doklady*, 7:595, 1963.

[36] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[37] Eunjeong Lee, Hyang-Sook Lee, and Cheol-Min Park. Efficient and generalized pairing computation on abelian varieties. Cryptology ePrint Archive, Report 2008/040, 2008. `http://eprint.iacr.org/`.

[38] A.K. Lenstra and E.R. Verheul. Selecting Cryptographic Key Sizes. *Journal of Cryptology*, 14(4):255–293, 2001.

[39] Arjen K. Lenstra. Key Lengths. In *Handbook of Information Security*. John Wiley & Sons, 2004.

[40] H. W. Lenstra, Jr. Factoring integers with elliptic curves. *The Annals of Mathematics*, 126(3):649–673, 1987.

[41] Stephen Lichtenbaum. Duality theorems for curves over p-adic fields. *Inventiones Mathematicae*, 7(2):120–136, 1969.

[42] Chae Hoon Lim and Hyo Sun Hwang. Fast implementation of elliptic curve arithmetic in GF(p^n). In *Public Key Cryptography*, volume 1751 of *Lecture Notes in Computer Science*, pages 405–421. Springer Berlin / Heidelberg, 2000.

[43] Chae Hoon Lim and Pil Joong Lee. More flexible exponentiation with precomputation. In *Advances in Cryptology — CRYPTO'94*, volume 839 of *Lecture Notes in Computer Science*, pages 95–107. Springer Berlin / Heidelberg, 1994.

[44] Xibin Lin, Chang-An Zhao, Fangguo Zhang, and Yanming Wang. Computing the Ate pairing on elliptic curves with embedding degree $k = 9$. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 91(9):2387–2393, 2008.

[45] Julio López and Ricardo Dahab. Fast multiplication on elliptic curves over GF(2^m) without precomputation. In *Cryptographic Hardware and Embedded Systems*, volume 1717 of *Lecture Notes in Computer Science*, page 724. Springer Berlin / Heidelberg, 1999.

[46] Ben Lynn. *On The Implementation of Pairing-Based Cryptosystems*. PhD thesis, Stanford University, 2007.

[47] A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[48] Alfred J. Menezes, Tatsuaki Okamoto, and Scott A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, 1993.

[49] Victor S. Miller. Short programs for functions on curves. *Unpublished manuscript*, 97:101–102, 1986.

[50] Victor S. Miller. Use of elliptic curves in cryptography. In *Advances in Cryptology — CRYPTO'85 Proceedings*, pages 417–426, 1986.

[51] Atsuko Miyaji, Masaki Nakabayashi, and Shunzou Takano. New explicit conditions of elliptic curve traces for FR-reduction. *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, 84(5):1234–1243, 2001.

[52] Bodo Möller. Algorithms for multi-exponentiation. In *Selected Areas in Cryptography*, volume 2259 of *Lecture Notes in Computer Science*, pages 165–180. Springer Berlin / Heidelberg, 2001.

[53] Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[54] François Morain and Jorge Olivos. Speeding up the computations on an elliptic curve using addition-subtraction chains. *Theoretical Informatics and Applications*, 24:531–544, 1990.

[55] National Institute of Standards and Technology. Recommendation for key management, March 2007. `http://www.itl.nist.gov`.

[56] National Institute of Standards and Technology. FIPS 186-3: Digital signature standard (DSS), June 2009. `http://www.itl.nist.gov`.

[57] Yasuyuki Nogami, Masataka Akane, Yumi Sakemi, Hidehiro Kato, and Yoshitaka Morikawa. Integer variable χ-based Ate pairing. In *Pairing-Based Cryptography — Pairing 2008*, volume 5209 of *Lecture Notes in Computer Science*, pages 178–191. Springer Berlin / Heidelberg, 2008.

[58] Leonardo B. Oliveira, Diego F. Aranha, Eduardo Morais, Felipe Daguano, Julio López, and Ricardo Dahab. TinyTate: Computing the Tate pairing in resource-constrained sensor nodes. In *Sixth IEEE International Symposium on Network Computing and Applications — NCA 2007*, pages 318–323, 2007.

[59] J. M. Pollard. Monte Carlo methods for index computation (mod p). *Mathematics of Computation*, 32(143):918–924, 1978.

[60] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems based on pairing. In *The 2000 Symposium on Cryptography and Information Security, Okinawa, Japan*, 2000.

[61] A. Schönhage and V. Strassen. Fast multiplication of large numbers. *Computing*, 7:281–292, 1971.

[62] Michael Scott and Paulo S. L. M. Barreto. Compressed pairings. In *Advances in Cryptology — CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2004. `http://eprint.iacr.org/`.

[63] Michael Scott and Paulo S. L. M. Barreto. Generating more MNT elliptic curves. *Designs, Codes and Cryptography*, 38:209–217, 2006. `http://eprint.iacr.org/`.

[64] Michael Scott, Naomi Benger, Manuel Charlemagne, Luis J. Dominguez Perez, and Ezekiel J. Kachisa. Fast hashing to g_2 on pairing-friendly curves. In *Pairing-Based Cryptography — Pairing 2009*, volume 5671 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2009.

[65] Michael Scott, Naomi Benger, Manuel Charlemagne, Luis J. Dominguez Perez, and Ezekiel J. Kachisa. On the final exponentiation for calculating pairings on ordinary elliptic curves. In *Pairing-Based Cryptography — Pairing 2009*, volume 5671 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2009. `http://eprint.iacr.org/`.

[66] Michael Scott and Piotr Szczechowiak. Optimizing multiprecision multiplication for public key cryptography. Cryptology ePrint Archive, Report 2007/299, 2007. `http://eprint.iacr.org/`.

[67] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Advances in Cryptology*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer Berlin / Heidelberg, 1985.

[68] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2008. `http://www.shoup.net/ntb/`.

[69] Nigel Smart et al. ECRYPT2 yearly report on algorithms and keysizes (2008-2009). Technical report, European Network of Excellence in Cryptology II, 2009.

[70] Jerome A. Solinas. Efficient arithmetic on Koblitz curves. *Designs, Codes and Cryptography*, 19(2):195–249, 2000.

[71] W. A. Stein et al. *Sage Mathematics Software (Version 4.2.1)*. The Sage Development Team, 2009. `http://www.sagemath.org`.

[72] Piotr Szczechowiak, Anton Kargl, Michael Scott, and Martin Collier. On the application of pairing based cryptography to wireless sensor networks. In *Proceedings of the second ACM conference on Wireless network security*, pages 1–12. ACM New York, NY, USA, 2009.

[73] Piotr Szczechowiak, Leonardo B. Oliveira, Michael Scott, Martin Collier, and Ricardo Dahab. NanoECC: Testing the limits of elliptic curve cryptography in sensor networks. In *Wireless Sensor Networks*, volume 4913 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2008.

[74] J. Tate. WC-groups over p-adic fields. *Séminaire Bourbaki*, 1556, 1957.

[75] A.L. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Mathematics Doklady*, 3(4):714–716, 1963.

[76] F. Vercauteren. Optimal pairings. Cryptology ePrint Archive, Report 2008/096, 2008. `http://eprint.iacr.org/`.

[77] H. Wang and Q. Li. Efficient implementation of public key cryptosystems on mote sensors (short paper). In *Information and Communications Security*, volume 4307 of *Lecture Notes in Computer Science*, pages 519–528. Springer Berlin / Heidelberg, 2006.

[78] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography*. Chapman Hall/CRC, 2008.

[79] André Weil. On the Riemann hypothesis in function-fields. In *Proceedings of the National Academy of Sciences*, volume 27, pages 345–347, 1941.