View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Repositorio da Producao Cientifica e Intelectual da Unicamp

Matheus Fernandes de Oliveira

Um estudo sobre a implementação de criptossistemas baseados em emparelhamentos bilineares sobre curvas elípticas em cartões inteligentes de oito bits

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação.

Banca Examinadora:
Orientador: Prof. Dr. Marco Aurélio Amaral Henriques
Prof. Dr. Ricardo Dahab - UNICAMP
Prof. Dr. Reginaldo Palazzo Júnior - UNICAMP

Campinas, SP
Outrubro de 2010

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Oliveira, Matheus Fernandes de
OL4e Um estudo sobre a implementação de criptossistemas baseados em emparelhamentos bilineares sobre curvas elípticas em smart cards de oito bits / Matheus Fernandes de Oliveira. – Campinas, SP: [s.n.], 2010.

Orientador: Marco Aurélio Amaral Henriques.
Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Criptografia. 2. Tecnologia da informação - Segurança. 3. Curvas elípticas. 4. Criptografia de chave pública. 5. Formas bilineares. I. Henriques, Marco Aurélio Amaral. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título

Título em Inglês:	A study about implementation of elliptic curve pairing based cryptosystems in 8-bit smart cards
Palavras-chave em Inglês:	Cryptography, Information technology - Security, Elliptic curves, Public-key cryptography, Bilinear forms
Área de concentração:	Engenharia de Computação
Titulação:	Mestre em Engenharia Elétrica
Banca Examinadora:	Ricardo Dahab, Reginaldo Palazzo Júnior
Data da defesa:	01/10/2010
Programa de Pós Graduação:	Engenharia Elétrica

COMISSÃO JULGADORA - TESE DE MESTRADO

Candidato: Matheus Fernandes de Oliveira

Data da Defesa: 1 de outubro de 2010

Título da Tese: "Um estudo sobre a implementação de criptossistemas baseados em emparelhamentos bilineares sobre curvas elípticas em cartões inteligentes de oito bits"

Prof. Dr. Marco Aurélio Amaral Henriques (Presidente): _____

Prof. Dr. Ricardo Dahab: _____

Prof. Dr. Reginaldo Palazzo Júnior: _____

Resumo

Emparelhamentos bilineares sobre curvas elípticas são funções matemáticas que viabilizam o desenvolvimento de uma série de novos protocolos criptográficos, entre eles, os criptossistemas baseados em identidades. Esses criptossistemas representam uma nova forma de se implementar criptografia de chaves públicas na qual são atenuadas ou completamente retiradas as restrições relativas ao tipo, tamanho e formato das chaves públicas. Este trabalho apresenta um estudo sobre a implementação de criptossistemas baseados em emparelhamentos bilineares sobre curvas elípticas em cartões inteligentes de oito bits. O trabalho mostra ainda o desenvolvimento de equações específicas baseadas no método conhecido como *Montgomery's Ladder* para multiplicação escalar de curvas elípticas supersingulares em corpos binários. Estas novas equações tornam o algoritmo mais rápido sem perder suas características de segurança originais. O trabalho apresenta também a técnica de delegação segura de emparelhamentos, na qual um dispositivo computacionalmente restrito, como um cartão inteligente, delega o cálculo do emparelhamento para um dispositivo com maior poder computacional. É proposta uma modificação nesta técnica que diminui o número de operações executadas pelo cartão inteligente.

Palavras-chave: Emparelhamentos bilineares, curvas elípticas, criptografia de chave pública, criptografia baseada em identidades, delegação de emparelhamentos, cartões inteligentes.

Abstract

Bilinear pairings over elliptic curves are mathematical functions that enable the development of a set of new cryptographic protocols, including the so called identity based cryptosystems. These cryptosystems represent a new way to implement public- key cryptography in such a way that the restrictions related to public keys type, size and format are reduced or completely removed. This work presents a study about implementation of pairing based cryptosystems in 8-bit smart cards. It also presents new equations to be used in Montgomery's Ladder algorithm for scalar multiplication of supersingular ellipitic curves over binary fields. These equations make the algorithm faster without compromising its security characteristics. Finally, it discusses the secure delegation of pairing computation, that enables a computationally limited device, like a smart card, to delegate the computation of pairings to a more powerful device. It is proposed a modification in this technique to decrease the number of operations executed by the smart card.

Keywords: Bilinear pairings, elliptic curves, public-key cryptography, identity based cryptography, pairing delegation, smart cards.

Agradecimentos

Ao meu orientador Prof. Marco Aurélio Amaral Henriques, sou grato pela orientação.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Símbolos

d_U	-	chave privada do usuário U
e_U	-	chave pública do usuário U
CA	-	Autoridade Certificadora
$Cert_U$	-	certificado do usuário U assinado pela CA
PKI	-	Infraestrutura de Chave Pública
IBC	-	Criptografia Baseada em Identidades
ECC	-	Criptografia de Curvas Elípticas
$\mathbb{G}$	-	grupo algébrico
q	-	potência de um primo
$\mathbb{F}_q$	-	corpo finito
$E(\mathbb{F}_q)$	-	curva elíptica no corpo finito $\mathbb{F}_q$
$\#E(\mathbb{F}_q)$	-	ordem da curva elítpica $E(\mathbb{F}_q)$
$\mathcal{O}$	-	ponto no infinito
k	-	grau de imersão de uma curva $E(\mathbb{F}_q)$
$e(P,Q)$	-	emparelhamento dos pontos P e Q
ϕ	-	isomorfismo ou mapa de distorção para mapear elementos do corpo base em elementos do corpo de extensão
PKG	-	Centro Gerador de Chaves
H_i	-	Função de *hash*
$\oplus$	-	operação XOR (ou-exclusivo)

Artigos derivados deste trabalho

1. M. F. Oliveira, M. A. A Henriques. “Performance Evaluation of Cryptographic Algorithms in JCOP41 Smart Card”. *Ibero-American Congress on Information Security (CIBSI’07)*, Mar del Plata, Argentina. Novembro 2007.

Capítulo 1

Introdução

Emparelhamento bilinear é uma função matemática que mapeia dois elementos que podem ser de conjuntos distintos ou não em um terceiro elemento de um outro conjunto. Essa função é linear em seus dois argumentos e os elementos envolvidos pertencem a grupos algébricos cíclicos. A escolha adequada desses grupos algébricos permite o cálculo seguro e eficiente dos emparelhamentos bilineares e sua aplicação em diferentes e novos protocolos criptográficos.

Os emparelhamentos bilineares sobre curvas elípticas pertencem a uma classe desses mapeamentos, que recebem como argumentos elementos de certas curvas algébricas conhecidas como curvas elípticas.

Na aplicação de protocolos criptográficos baseados em emparelhamentos bilineares, a segurança depende da maneira como eles são construídos e de um segredo (ou chave) que, em geral, deve ficar sob a posse de um usuário. Dada a importância da chave, é fundamental que não só ela seja mantida em segurança, mas que também qualquer utilização sua no protocolo criptográfico não forneça informações que permitam inferir partes ou até o conteúdo completo da mesma. O comprometimento da chave secreta pode trazer consequências negativas para as partes que utilizam um protocolo criptográfico.

Neste sentido, é oportuna a utilização de cartões inteligentes (*smart cards*) na implementação de protocolos criptográficos, uma vez que tais dispositivos oferecem um hardware conveniente e seguro para armazenamento e processamento de informações sensíveis. Apesar do tamanho reduzido, esse dispositivo consegue oferecer um alto nível de segurança para as informações que armazena, pois todo o processamento só é realizado mediante a autorização do portador do cartão por meio de uma senha conhecida como *Personal Identification Number* (PIN). Os cartões inteligentes possuem ainda mecanismos de segurança que tornam muito difíceis os ataques com o intuito de obter ou modificar as informações armazenadas no cartão. Por esses motivos, os cartões inteligentes constituem um meio conveniente para armazenar chaves criptográficas e realizar qualquer tipo de processamento que as

envolva, como por exemplo, a assinatura de um documento digital.

Os cartões inteligentes atuais apresentam ainda a vantagem de poder armazenar múltiplas aplicações no mesmo *chip*, sem que uma aplicação represente um risco para as demais. Dessa forma, um mesmo cartão inteligente pode oferecer as funcionalidades de armazenar chaves criptográficas, informações do portador, créditos de transporte público e ainda processar transações financeiras, por exemplo. Entre os cartões inteligentes que podem armazenar múltiplas aplicações, os mais empregados e versáteis são aqueles com o sistema operacional Java Card ou com o Multi-application Operating System (MULTOS). Como é costume na indústria e no meio acadêmico, os cartões são referenciados pelo nome do sistema operacional que utilizam, ou seja, um Java Card é um cartão com sistema operacional homônimo.

Ainda que os cartões inteligentes sejam capazes de proporcionar uma série de vantagens, a grande maioria dos dispositivos em uso e comercialmente disponíveis ainda se enquadra na categoria de dispositivos computacionalmente restritos, ou seja, com baixo poder de processamento.

1.1 Abordagens na implementação de emparelhamentos bilineares

A tentativa de implementar algoritmos de criptografia sobre curvas elípticas ou emparelhamentos bilineares em dispositivos com restrições computacionais pode seguir diferentes abordagens. Muitos desses dispositivos são programáveis e pode-se, então, desenvolver software ou firmware para executar as funções criptográficas. Esse é o caso, por exemplo, da implementação de emparelhamentos bilineares em emuladores de cartões inteligentes de arquitetura de 32 bits [1] ou então em redes de sensores [2]. Outra possibilidade de incorporar os emparelhamentos bilineares é pela sua implementação em hardware dedicado [3], por meio de coprocessadores.

Comparativamente, as duas abordagens na implementação de emparelhamentos bilineares apresentam vantagens e desvantagens que devem ser consideradas em sua adoção. A implementação em software, por exemplo, pode ser bem menos custosa, mas pode não oferecer a eficiência necessária para que o algoritmo criptográfico seja executado em um tempo razoável para o usuário. Outra questão é que a quantidade de memória disponível no dispositivo pode não ser suficiente para executar todas as operações do emparelhamento bilinear. A implementação em hardware dedicado, por sua vez, pode ter bom desempenho, mas geralmente exige um grande trabalho no projeto e elaboração de suas unidades funcionais.

1.2 Contribuições deste trabalho

No intuito de viabilizar criptossistemas baseados em emparelhamentos bilineares empregando cartões inteligentes, esta dissertação traz três contribuições principais.

A primeira contribuição é mostrar os resultados práticos de implementação em software de emparelhamentos bilineares em dois tipos de cartões inteligentes de arquitetura de oito bits e comercialmente disponíveis: o Java Card JCOP41 e o cartão MULTOS.

A segunda contribuição é o desenvolvimento de equações específicas que resultam em um método para o cálculo eficiente de multiplicação escalar de pontos de curvas elípticas supersingulares em corpos binários. Esse método apresenta ainda a característica de ser robusto contra ataques do tipo *side-channel*. Essa é uma característica importante para implementações em software ou hardware, já que evita que algum atacante obtenha informações sigilosas com base no tempo e/ou no consumo de energia necessários para efetuar o cálculo de uma multiplicação escalar.

Por fim, mostra-se uma maneira segura e mais eficiente de implementar criptossistemas baseados em emparelhamentos bilineares com cartões inteligentes utilizando a técnica conhecida como delegação de emparelhamentos. Essa técnica permite que os cartões sejam utilizados em criptossistemas baseados em emparelhamentos bilineares sem que esses dispositivos tenham que efetuar o cálculo de um emparelhamento completo. Nesta parte é mostrado como utilizar a delegação de emparelhamentos para prover sigilo na comunicação através de um aparelho celular de forma simples e eficiente. É proposta uma modificação num dos casos de aplicação do protocolo de delegação de emparelhamento, que resulta em um menor número de operações executadas pelo cartão inteligente, sem diminuir a segurança.

1.3 Trabalhos anteriores

O interesse de unir a segurança proporcionada pelos algoritmos de criptografia sobre curvas elípticas e emparelhamentos bilineares com o hardware seguro dos cartões inteligentes motivou uma série de trabalhos de pesquisa e implementação, alguns dos quais são apresentados a seguir. Foram consideradas implementações em software e em hardware de cartões inteligentes com diversas arquiteturas e diferentes fabricantes. Muitos relatos de implementações foram bem sucedidos, conseguindo efetuar os cálculos de maneira eficiente. No entanto, em alguns trabalhos a arquitetura dos cartões inteligentes utilizados se mostrou bastante restritiva e levou a resultados com tempos elevados.

1.3.1 Criptografia sobre curvas elípticas em cartões inteligentes

Utilizando um Java Card com arquitetura de oito bits, Elo [4], Berta et al [5] e Calegari [6] descreveram as tentativas de implementar em software o algoritmo de assinatura digital conhecido como *Elliptic Curve Digital Signature Algorithm* (ECDSA). No entanto, as severas restrições de memória e processamento dos cartões acabaram levando a resultados insatisfatórios do ponto de vista de performance. No trabalho de Calegari [6], por exemplo, uma multiplicação no corpo finito $\mathbb{F}_{2^{167}}$ foi executada em 24,9 segundos no melhor caso, quando empregou-se o método de multiplicação López-Dahab. O cálculo de uma assinatura ECDSA, no entanto, exige várias execuções da operação de multiplicação, o que resultaria num tempo muito elevado para um usuário.

1.3.2 Emparelhamentos bilineares em dispositivos embarcados ou de baixo poder computacional

O aumento do interesse nos criptossistemas baseados em emparelhamentos bilineares sobre curvas elípticas motivou a tentativa de incorporar os algoritmos que viabilizassem tais operações em cartões inteligentes. A empresa Gemplus (hoje conhecida como Gemalto) anunciou em 2004 a implementação de emparelhamentos bilineares em seus cartões [7]. No entanto, o anúncio não veio acompanhado de maiores detalhes, como o algoritmo de emparelhamento escolhido, o tipo de corpo finito ou a curva elíptica empregada. O anúncio não informou também se a referida implementação foi realizada em software ou hardware.

Scott et al [8] e Devegili et al [9] relataram a implementação de diferentes emparelhamentos bilineares em software de um emulador do cartão Philips HiPerSmart, de arquitetura de 32 bits. Os resultados descritos foram eficientes, sendo possível calcular o emparelhamento em um tempo inferior a um segundo, para alguns casos. O bom resultado dessa implementação fez até Scott et al [8] discutirem se a técnica de delegação de emparelhamentos, explicada no Capítulo 5, fazia sentido. No entanto, vale lembrar que foi utilizado nesse trabalho de implementação um emulador de um cartão de arquitetura de 32 bits, diferente dos cartões de oito bits que são utilizados largamente em aplicações bancárias e em universidades, por exemplo.

Uma outra implementação de emparelhamentos bilineares em software sobre um cartão inteligente de arquitetura de 32 bits foi descrita por um grupo de pesquisa formado por engenheiros da STMicroelectronics e da HPLabs [1]. Neste trabalho, foi implementado o emparelhamento BKLS [10] utilizando curvas supersingulares em corpos primos e foi possível efetuar o cálculo de um emparelhamento bilinear em apenas 752ms, mostrando mais uma vez a viabilidade destes cálculos em cartões com arquitetura de 32 bits.

Emparelhamentos bilineares foram implementados com sucesso em nós de uma rede de sensores

MICAz Mote, com processador ATMega128 de 7.3828 MHz de clock, dispositivos que, à semelhança da maioria dos cartões inteligentes, possuem arquitetura de oito bits e apresentam restrições de memória e processamento. Nesses dispositivos foi possível efetuar o cálculo de emparelhamento bilinear em cerca de 2,06 segundos numa implementação empregando C e a linguagem Assembly, conforme reportado por Szczechowiak et al [11] e Aranha et al [12]. Outros bons resultados de implementações de emparelhamentos bilineares em redes de sensores com o mesmo processador ATMega128 podem ser encontrados no trabalho de Szczechowiak et al [13]. Apesar de tanto os cartões inteligentes quanto os nós da rede de sensores serem dispositivos computacionalmente restritos, esses últimos possuem 4KB de memória RAM, ao passo que os cartões inteligentes têm pouco mais que uma centena de bytes de memória RAM. A memória RAM, recurso escasso nos cartões inteligentes e utilizada para vários propósitos, é compartilhada pelo sistema operacional e pelas aplicações, além de ter uma área reservada para receber comandos e enviar respostas. Com isso, restam poucos bytes para armazenar operadores e tabelas na memória RAM dos cartões inteligentes. Além disso, os cartões inteligentes de oito bits comercialmente disponíveis apresentam uma camada de abstração do hardware conhecida como máquina virtual, que introduz uma sobrecarga na execução das aplicações, uma vez que as instruções executadas pelos programas têm que ser interpretadas antes de serem executadas.

Apesar de fornecerem bons resultados para o cálculo dos emparelhamentos bilineares, os cartões inteligentes de arquitetura de 32 bits ainda não estão difundidos no mercado como estão os de oito bits, principalmente em razão dos custos mais elevados dos primeiros. Além disso, os cartões inteligentes de oito bits têm conseguido suprir a baixo custo as necessidades do setor bancário, do transporte público e de outros que procuram um dispositivo seguro para armazenamento e processamento simples de informações. Por essa razão, tornar viável a implementação de emparelhamentos bilineares nesses cartões inteligentes de oito bits é o desafio a ser vencido para possibilitar a disseminação e o aproveitamento das vantagens desses algoritmos criptográficos avançados em um grande parque já instalado.

1.4 Organização

Esta dissertação está organizada da seguinte maneira. O Capítulo 2 apresenta conceitos e notações referentes a criptografia, curvas elípticas, emparelhamentos bilineares e protocolos baseados em emparelhamentos, como os criptossistemas baseados em identidades. No Capítulo 3 é apresentado um método de multiplicação escalar de pontos de curvas elípticas supersingulares em corpos binários que é robusto contra ataques do tipo *side-channel* e que é mais eficiente que o tradicional método *Double-and-Add* com a escolha adequada de coordenadas para representação dos pontos da curva elíptica. O Capítulo 4 contém as informações e os conceitos referentes a cartões inteligentes e apre-

senta os resultados obtidos nas implementações em software nos cartões de oito bits comercialmente disponíveis: Java Card JCOP41 e MULTOS. O conceito de delegação de emparelhamentos e uma proposta de aplicação mais eficiente dessa técnica encontram-se no Capítulo 5. Por fim, o Capítulo 6 apresenta as principais conclusões obtidas e os possíveis encaminhamentos que podem ser tomados a partir deste trabalho.

Capítulo 2

Criptossistemas baseados em emparelhamentos bilineares

A fim de melhor contextualizar o escopo deste trabalho e criar um visão homogênea dos assuntos abordados, neste capítulo serão apresentados conceitos e notações que servirão de base para a dissertação.

2.1 Criptografia

Criptografia consiste no estudo, elaboração e análise de técnicas e aplicações que se baseiam na existência de problemas de difícil solução, de modo a possibilitar a comunicação segura na presença de adversários maliciosos [14].

Criptoanálise é o estudo de como comprometer mecanismos de segurança criados pela criptografia, de maneira a avaliar a robustez dos mesmos.

Em um modelo básico de comunicação, são consideradas as entidades A e B, as quais se comunicam por um canal potencialmente inseguro e vigiado por um adversário C, conforme ilustra a Figura 2.1. Assume-se que C tem acesso a todos os dados transmitidos pelo canal, pode modificá-los e ainda injetar novos dados no canal.

No esquema ilustrado na Figura 2.1, A e B podem ser duas pessoas conversando através da rede de telefone celular e C pode estar tentando captar essa conversação. Ou ainda, A pode representar o *browser* de um usuário que está efetuando uma compra em uma loja virtual representada pelo sítio B. O canal de comunicação é a Internet e o adversário C poderia tentar ler o tráfego entre A e B a fim de obter credenciais dos usuários. Outro exemplo pode levar em consideração que A é um cartão inteligente que é utilizado para autenticar seu portador junto ao servidor B de um banco e o adversário C, neste caso, poderia tentar obter informações sobre a conta do portador a fim de, eventualmente,

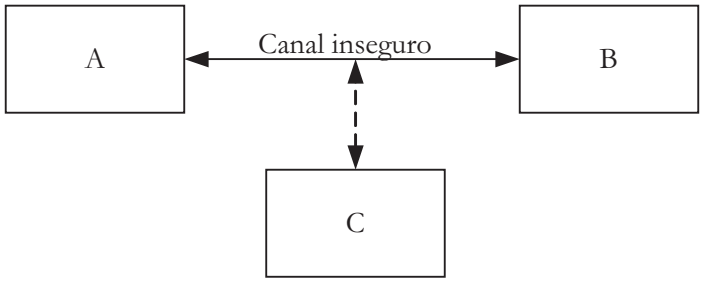


Fig. 2.1: Modelo básico de comunicação.

poder se passar por ele e retirar dinheiro de sua conta.

O estabelecimento de um canal seguro entre as partes comunicantes pode ser realizado de diferentes formas, dependendo de quão sensível é a informação que está sendo trafegada e do quão confiável o canal deve ser.

Quando se deseja estabelecer uma comunicação segura, normalmente deseja-se endereçar as seguintes questões:

- **Sigilo:** os dados devem ser mantidos secretos para todos, exceto para as pessoas autorizadas a saber seu conteúdo. Há variadas maneiras de prover sigilo, que pode ser por meio de proteção física ou por meio de algoritmos matemáticos que tornam os dados ininteligíveis para as partes não autorizadas.

- **Integridade dos dados:** é importante identificar se os dados recebidos passaram por algum tipo de modificação. E a integridade é um serviço que procura detectar uma eventual alteração não autorizada dos dados.

- **Autenticação:** a informação que trafega por um canal deve poder ser autenticada com relação à sua origem, data de criação, conteúdo, data em que foi enviada, entre outros. Ainda com relação à autenticação, uma entidade deve poder comprovar sua identidade ou a identidade recebida da outra entidade com a qual se comunica.

- **Irretratabilidade:** trata-se de um serviço que previne uma entidade de negar ações anteriores. Quando se cria uma disputa em razão de uma entidade negar algumas de suas ações, é necessário haver meios para resolver a situação. Normalmente, para garantir irretratabilidade é necessário envolver uma terceira entidade, considerada confiável pelas demais, a fim de resolver a disputa.

O objetivo fundamental da criptografia é prover sigilo sobre as informações tratadas. Entretanto, combinada com protocolos adequados, a criptografia pode tratar também as demais questões acima tanto do ponto de vista teórico quanto prático.

Cifragem e decifragem de dados são os principais serviços de segurança obtidos quando se emprega criptografia [15]. A cifragem consiste na transformação de dados em uma forma que é praticamente impossível descobrir seu conteúdo sem que se possua uma informação apropriada, que normalmente consiste numa sequência alfa-numérica chamada de chave. Dessa forma, o conteúdo transmitido nos dados cifrados mantém-se ininteligível a todos que não possuem a chave, ainda que eles possam ter acesso aos dados cifrados. A decifragem é a operação reversa da cifragem, pois volta os dados cifrados para sua forma original [16].

Outro serviço muito demandado é a autenticação. Emprega-se autenticação na vida cotidiana quando se assina um documento, por exemplo. E, à medida que muitas decisões e acordos são tratados eletronicamente, a autenticação tem sido provida pelo uso de técnicas de assinatura digital baseadas em algoritmos criptográficos.

A criptografia provê atualmente várias técnicas que permitem elaborar criptossistemas e protocolos seguros que são utilizados nas transmissões de televisão a cabo, em pagamentos eletrônicos, no acesso seguro a sistemas de *email* e em tantas outras situações nas quais sigilo, integridade dos dados, autenticação e irretratabilidade são conceitos que se fazem necessários.

Sistemas criptográficos podem ser divididos em dois tipos principais: sistemas de criptografia simétrica e sistemas de criptografia assimétrica ou de chave pública.

2.1.1 Criptografia simétrica

A criptografia simétrica também é chamada de criptografia de chave secreta. É a forma mais tradicional de criptografia, em que uma única chave é compartilhada pelas entidades A e B e é usada para cifrar e decifrar mensagens. Algoritmos de criptografia simétrica também podem ser utilizados para autenticação de dados, como é o caso da técnica chamada *Message Authentication Code* (MAC).

Protocolos baseados em criptografia simétrica apresentam o inconveniente de que as entidades A e B devem compartilhar uma mesma chave, mesmo que elas possivelmente nunca tenham se encontrado, criando a necessidade de se criar previamente um canal seguro entre as duas partes para transmitir a chave a ser utilizada nas cifragens e decifragens futuras.

Os algoritmos mais comuns da criptografia simétrica são o *Data Encription Standard* (DES) e o *Advanced Encription Standard* (AES) [17].

Uma vantagem da criptografia simétrica é que, para um mesmo nível de segurança, seus algoritmos são executados mais rapidamente que os algoritmos da chamada criptografia assimétrica.

2.1.2 Criptografia assimétrica

A fim de resolver o problema da distribuição de chaves inerente à criptografia simétrica, Whitfield Diffie e Martin Hellman introduziram em 1976 o conceito de criptografia assimétrica ou criptografia de chave pública [18]. A proposta feita por Diffie e Hellman era limitada e só servia para troca segura de chaves. Um mecanismo mais completo só pôde ser efetivado pouco tempo depois, em 1978, com a introdução do algoritmo RSA, cujas iniciais remetem a seus inventores Rivest, Shamir e Adleman [19]. Além do RSA, há também na criptografia de chave pública o esquema de cifragem ElGamal e os algoritmos baseados em curvas elípticas, entre os quais se encontram os chamados emparelhamentos bilineares sobre curvas elípticas, que permitem compor protocolos de cifragem, assinatura digital e troca de chaves.

Nos criptossistemas de chave pública, cada usuário possui um par de chaves (e,d), consistindo na chave pública e e na chave privada d. A chave privada deve ser mantida em sigilo pelo seu dono e a chave pública deve ser divulgada. Nota-se aí a diferença com a criptografia simétrica: a chave trocada entre A e B não precisa ser secreta, apenas autêntica.

O uso do par de chaves ocorre da seguinte forma. Para o usuário A enviar uma mensagem cifrada para B ou para verificar assinaturas digitais de B, ele deve descobrir e em seguida utilizar a chave pública e_B. Para o usuário B decifrar uma mensagem cifrada ou para assinar um documento digital, ele deve empregar sua chave privada d_B. O sigilo do texto cifrado e a autenticidade da assinatura dependem do fato de que apenas B conhece d_B. Nota-se, portanto, que o comprometimento da chave privada d_B pode acarretar como consequências a geração de assinaturas falsas em nome de B ou a decifragem de mensagens sigilosas destinadas para B.

As chaves pública e privada guardam uma relação matemática entre si, mas é computacionalmente inviável determinar a chave privada mesmo conhecendo o algoritmo, os dados cifrados e a chave pública correspondente quando essas chaves empregam um grande número de bits. O par de chaves pode ser gerado pelo próprio usuário ou por uma autoridade central confiável, dependendo do protocolo em que as chaves serão utilizadas. Uma possibilidade segura é gerar o par de chaves num dispositivo de hardware seguro, como um cartão inteligente, de modo que a chave privada nunca seja exposta durante sua existência. No entanto, caso seja condição necessária gerar o par de chaves ou a chave privada numa entidade de confiança, continua sendo importante que a chave privada seja armazenada num dispositivo seguro como o cartão inteligente ou em um módulo de hardware seguro (*Hardware Security Module* – HSM).

Os criptossistemas de chave pública são baseados em funções matemáticas chamadas *trapdoor one-way functions* [16]. Uma *one-way function* é uma função matemática que é significativamente fácil de ser computada em uma direção, mas muito difícil de ser calculada na direção oposta. Uma *trapdoor one-way function*, por sua vez, é uma *one-way function* cuja função inversa é fácil de calcular

dada alguma informação (*trapdoor*), mas é considerada difícil sem essa informação. A chave pública informa sobre uma instância particular da função e a chave privada é o *trapdoor*.

Apesar da criptografia de chaves públicas ter suplantado o problema de distribuição de chaves inerente à criptografia simétrica, surge um outro problema prático quando se considera a necessidade de autenticidade das chaves públicas. Basta imaginar que, se um atacante consegue convencer o usuário A e outros participantes que a chave $e'_B \neq e_B$, de sua escolha, é a chave pública de B, ele pode então decifrar mensagens que deveriam ser sigilosas para B e ainda forjar assinaturas, como se tivessem sido geradas por B. É importante, portanto, que os usuários da criptografia assimétrica possam verificar a autenticidade das chaves públicas dos demais participantes.

A solução convencional para a autenticidade das chaves públicas é o uso da infraestrutura de chaves públicas ou *Public Key Infrastructure* (PKI), na qual encontra-se uma entidade confiável por todos os usuários, a Autoridade Certificadora ou *Certification Authority* (CA). Essa autoridade define o procedimento a ser seguido pelo usuário A, por exemplo, para apresentar a sua chave pública e_A e provar que ele possui a chave privada d_A correspondente. Isso é geralmente feito por meio de uma requisição de um certificado digital assinada com d_A. Se o usuário A passa por todos os passos estabelecidos pela CA, então ela publica um certificado digital contendo a identidade de A, sua chave pública e_A, outras informações, como a validade, por exemplo, e assina todo esse conteúdo com sua chave privada, gerando o certificado $Cert_A$. Os demais usuários que desejem se comunicar de forma segura com A devem obter a chave pública da CA e verificarem a assinatura contida no $Cert_A$. Uma assinatura válida da CA convence da autenticidade da chave e_A contida em $Cert_A$.

O alto custo computacional dos algoritmos de criptografia de chave pública em comparação com os algoritmos de criptografia simétrica motiva muitas soluções práticas a empregar as duas técnicas, a fim de se beneficiarem da funcionalidade dos primeiros e da eficiência dos últimos.

2.1.3 Criptografia sobre curvas elípticas

A criptografia sobre curvas elípticas (ou *Elliptic Curve Cryptography* – ECC) encaixa-se na chamada criptografia de chaves públicas e baseia-se em funções matemáticas conhecidas como curvas elípticas definidas em corpos finitos. O uso das curvas elípticas para fins criptográficos foi proposto independentemente por Neal Koblitz [20] e Victor S. Miller [21].

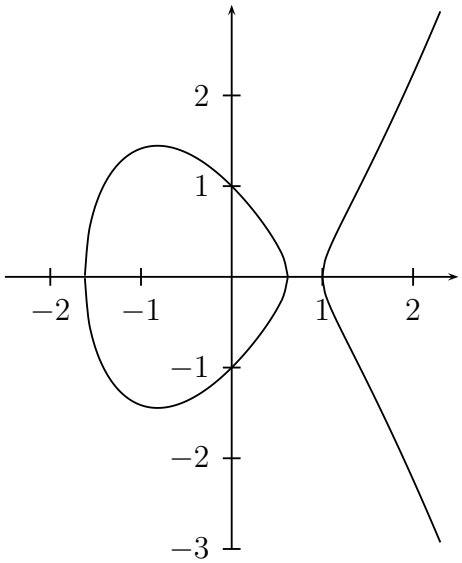
Curvas elípticas são construções matemáticas que empregam conceitos da teoria de números e da geometria algébrica. Uma curva elíptica pode ser definida sobre qualquer corpo, mas para aplicações criptográficas elas costumam ser definidas sobre corpos finitos (corpos binários e corpos primos).

Uma curva elíptica E sobre um corpo algébrico K consiste de todos os elementos (x, y) que satisfazem a equação de Weierstraß,

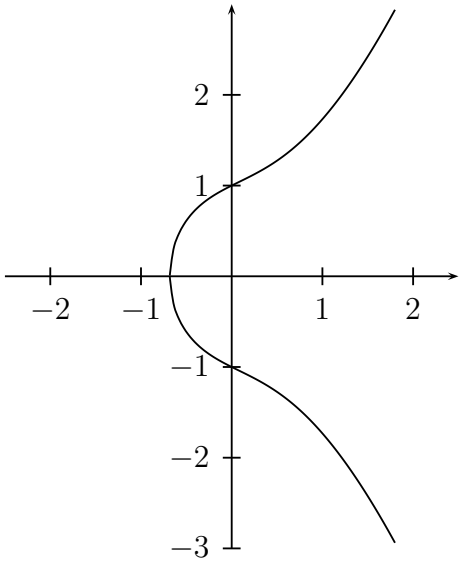
$$E(K) : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_5, \tag{2.1}$$

onde os coeficientes a_1, a_2, a_3, a_4 e $a_6 \in K$. A curva $E(K)$ possui ainda um ponto extra, chamado ponto no infinito, representado por $\mathcal{O}$.

As Figuras 2.2(a) e 2.2(b) ilustram duas curvas elípticas representadas sobre o corpo $\mathbb{R}$ dos números reais.



(a) $E_1(\mathbb{R}) : y^2 = x^3 - 2x + 1$



(b) $E_2(\mathbb{R}) : y^2 = x^3 + x + 1$

Fig. 2.2: Exemplos de curvas elípticas representadas sobre o corpo $\mathbb{R}$ dos números reais.

2.1.4 Aritmética modular e grupos algébricos

Dados os inteiros a, b e n, com $n > 0$, diz-se que a e b são congruentes módulo n se $a - b$ é divisível por n, ou seja, se $\frac{a-b}{n}$ é um inteiro. Denota-se

$$a \equiv b \pmod{n} \tag{2.2}$$

quando a e b são congruentes módulo n.

Um grupo $(\mathbb{G}, *)$ consiste em um conjunto $\mathbb{G}$ (finito ou infinito) e uma operação binária denotada por $*$, chamada de multiplicação do grupo. Para ser considerado um grupo, o conjunto $\mathbb{G}$ e a operação $*$ devem satisfazer algumas condições denominadas axiomas do grupo:

- fechamento: dados a e $b \in \mathbb{G}$, o produto $a * b$ deve resultar num elemento também em $\mathbb{G}$, de modo que $* : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}$, ou seja, $\mathbb{G}$ é fechado sob a operação de multiplicação;

- associatividade: a operação $*$ é associativa, ou seja, $a * (b * c) = (a * b) * c$ para quaisquer a, b e $c \in \mathbb{G}$;

- identidade: existe um elemento identidade $u \in G$ tal que $a * u = u * a = a$ para cada elemento $a \in \mathbb{G}$;

- inverso: cada elemento $a \in \mathbb{G}$ possui um inverso $b \in \mathbb{G}$, satisfazendo $a * b = b * a = u$, o elemento identidade de $\mathbb{G}$.

Se, adicionalmente, a operação $*$ em $\mathbb{G}$ é comutativa, ou seja, $a * b = b * a$ para quaisquer elementos a, $b \in \mathbb{G}$, o grupo é chamado abeliano.

Um grupo é chamado finito se $\mathbb{G}$ possui um número finito de elementos. Esse número é chamado *ordem* do grupo $\mathbb{G}$. Para um inteiro $n > 1$, g^n indica a multiplicação de g por ele mesmo por n vezes. Dessa forma, $g^3 = g * g * g$, enquanto que g^{-n} é o inverso de g^n e g^0 é igual ao elemento identidade u.

Na criptografia de curvas elípticas empregam-se grupos cíclicos. Sendo $\mathbb{G}$ um grupo cíclico, todos os elementos de $\mathbb{G}$ são múltiplos de um elemento particular $g \in \mathbb{G}$. Usando a notação multiplicativa, os elementos do grupo cíclico são:

$$..., g^{-2}, g^{-1}, g^0 = u, g, g^2, ... \tag{2.3}$$

O elemento g é chamado gerador do grupo $\mathbb{G}$.

2.1.5 Corpos Finitos

De uma forma geral, aplicações práticas na criptografia de curvas elípticas têm seus operandos representados em estruturas matemáticas chamadas de corpos finitos.

Corpos são estruturas algébricas que consistem de um conjunto $\mathbb{F}$ com as operações de adição (denotada por $+$) e multiplicação (denotada por $\cdot$), que satisfazem as propriedades:

- $(\mathbb{F}, +)$ é um grupo abeliano aditivo com identidade denotada por 0;
- $(\mathbb{F} \setminus \{0\}, \cdot)$ é um grupo abeliano multiplicativo com identidade denotada por 1;
- a multiplicação distribui sobre a adição: $a \cdot (b + c) = a \cdot b + a \cdot c$, para todo a, b e $c \in \mathbb{F}$.

Se o conjunto $\mathbb{F}$ é finito, então diz-se que o corpo é finito. O número de elementos deste corpo é chamado de ordem do corpo. Um corpo é dito finito de ordem q quando q é potência de um número primo p, ou seja, se $q = p^m$, sendo m um número inteiro e positivo e p é chamado de característica do corpo $\mathbb{F}$. Para $m = 1$ o corpo $\mathbb{F}$ é chamado corpo primo, denotado por $\mathbb{F}_p$, de modo que todas as operações com elementos do corpo devem ser realizadas módulo o primo p. Corpos em que $m > 2$, são chamados corpos de extensão.

Um tipo especial de corpo finito para fins de implementação é o corpo de ordem 2^m, chamado de corpo binário. Usando a representação de base polinomial, os elementos do corpo finito $\mathbb{F}_{2^m}$ são polinômios da forma

$$a_{m-1}z^{m-1} + a_{m-2}z^{m-2} + \cdots + a_1 z + a_0, \tag{2.4}$$

onde os coeficientes a_i pertencem ao corpo finito $\mathbb{F}_2 = 0, 1$.

Os corpos binários são interessantes para implementações em ambientes computacionais, pois os bits setados em registradores ou em palavras da área de memória podem representar os coeficientes de um elemento do corpo $\mathbb{F}_{2^m}$, sem a necessidade de definir estruturas de dados complexas. Além disso, pode-se mostrar que as operações de adição entre dois elementos do corpo $\mathbb{F}_{2^m}$ são efetuadas com uma simples operação de XOR (ou exclusivo) entre esses elementos. As operações de multiplicação no corpo $\mathbb{F}_{2^m}$ são efetuadas módulo um polinômio $f(z)$, chamado de polinômio irredutível de grau m [14].

A Figura 2.3 ilustra os pontos a curva $y^2 = x^3 - 2x + 1$ no corpo finito $\mathbb{F}_{23}$.

2.2 Aritmética de curvas elípticas

Seja uma curva elíptica E definida sobre corpo finito $\mathbb{F}_q$. Os pontos de uma curva elíptica $E(\mathbb{F}_q)$, junto com a operação de adição entre esses pontos, formam um grupo abeliano finito, no qual o ponto

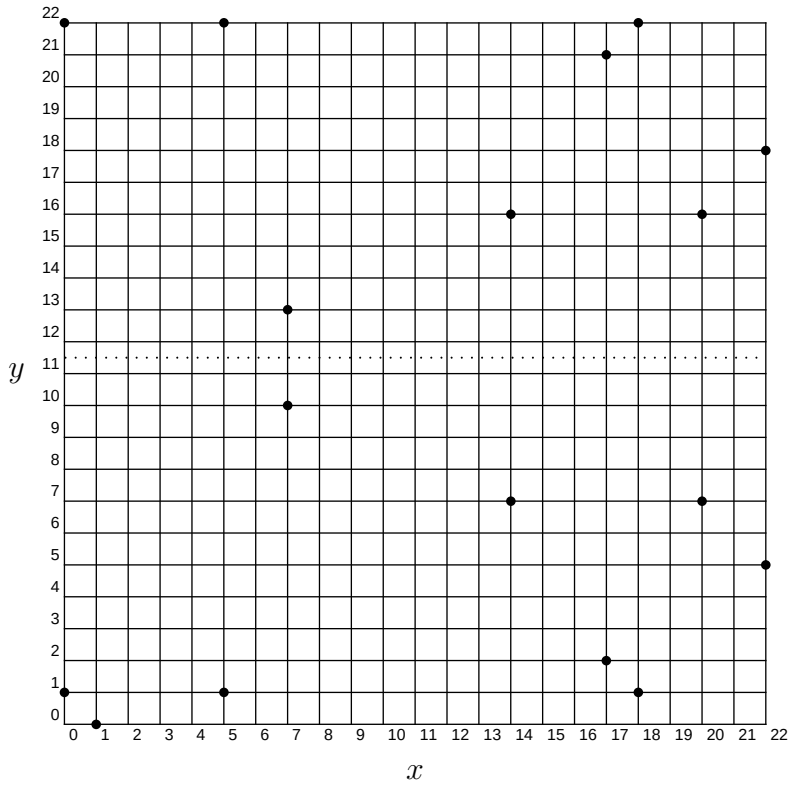
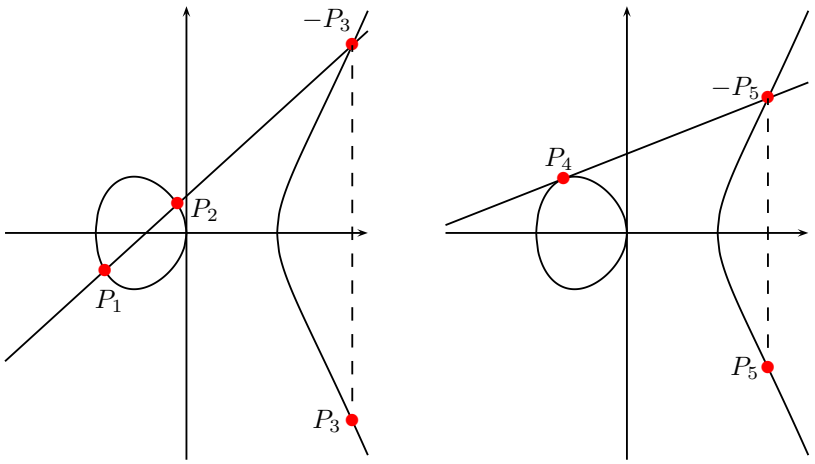


Fig. 2.3: Curva elíptica $y^2 = x^3 - 2x + 1$ representada no corpo finito $\mathbb{F}_{23}$.

$\mathcal{O}$ representa o elemento neutro. O número de pontos da curva $E(\mathbb{F}_q)$ é a ordem da curva, denotado por $\#E(\mathbb{F}_q)$. A operação de adição nesse grupo pode ser descrita geometricamente, como ilustrado na Figura 2.4(a). Sejam P_1 e P_2 dois pontos distintos de uma curva elíptica E. Para somar os dois pontos, traça-se uma reta que passa por P_1, P_2 e por um terceiro ponto da curva, denotado por $-P_3$. Refletindo esse último ponto em relação ao eixo x, obtém-se o ponto $P_1 + P_2 = P_3$. A duplicação de um ponto, ilustrada na Figura 2.4(b), segue dessa definição, tratando os pontos somados como se eles se fundissem em um único ponto: aquele que se deseja duplicar, denotado por P_4. Traça-se uma tangente à curva passando por P_4 e o ponto onde essa reta tangente interceptar a curva elíptica será $-P_5$. Analogamente ao caso da adição, reflete-se o ponto $-P_5$ para obter $2P_4 = P_5$.

Equações específicas para adição e duplicação de pontos de curvas elípticas podem ser derivadas de acordo com a curva E utilizada e o corpo finito sobre o qual essa curva está definida. No Capítulo 3 essas equações serão apresentadas para uma família de curvas elípticas conhecidas como supersingulares.

Os criptossistemas de curvas elípticas se baseiam numa operação conhecida como multiplicação escalar. Para uma curva $E(\mathbb{F}_q)$, a multiplicação do ponto $P \in E$ por um escalar $k \in \mathbb{F}_q$ consiste na adição do ponto P com ele mesmo $(k-1)$ vezes. Para curvas adequadamente escolhidas, a operação kP é simples, mas sua inversa é computacionalmente inviável, como será detalhado na Seção 2.4.



(a) Adição dos pontos P_1 e P_2. (b) Duplicação do ponto P_4.

Fig. 2.4: Exemplos gráficos de adição e duplicação de pontos de curva elíptica.

Diz-se que um ponto $P \in E(\mathbb{F}_q)$ tem ordem r se $rP = \mathcal{O}$.

No Apêndice A, são apresentados dois algoritmos de criptografia sobre curvas elípticas. O esquema de assinatura digital *Elliptic Curve Digital Signature Algorithm* (ECDSA) é apresentado na Seção A.1 e o esquema de troca de chaves conhecido como *Elliptic Curve Diffie-Hellman* (ECDH) é apresentado na Seção A.2.

2.3 Tipos de curvas elípticas

Com base na Equação 2.1, é possível definir variadas curvas elípticas, algumas das quais podem ser empregadas na elaboração de criptossistemas seguros. Neste trabalho, são empregadas as chamadas curvas supersingulares. Uma curva elíptica E definida sobre o corpo finito $\mathbb{F}_q$ de característica p é classificada como supersingular se p divide t, ou seja, se $p|t$, onde $t = q+1-\#E(\mathbb{F}_q)$. No caso em que $p \nmid t$, então a curva $E(\mathbb{F}_q)$ é ordinária [14]. Dentre as curvas ordinárias, algumas podem ser especialmente construídas a fim de levar a implementações eficientes, como as curvas MNT [22], as curvas Barreto-Naehrig (BN) [23], as curvas de Freeman [24] e as curvas KSS [25].

Existem ainda formas de representar as curvas elípticas além da tradicional equação de Weierstraß, apresentada na Equação 2.1, como as curvas de Edwards [26] e de Hesse [27].

2.4 O Problema do Logaritmo Discreto sobre Curvas Elípticas

O problema do logaritmo discreto ou *Discrete Logarithm Problem* (DLP) aplica-se a estruturas matemáticas conhecidas como grupos finitos, conforme definido na Seção 2.1.4. Para um elemento g

do grupo finito $\mathbb{G}$ e um número inteiro n, g^n denota o elemento obtido após sucessivas multiplicações de g por ele mesmo, ou seja, $g^3 = g * g * g$. O problema do logaritmo discreto é encontrar um inteiro x tal que $g^x = h$ dados os elementos $g, h \in \mathbb{G}$. Assim como o problema de fatoração de inteiros [16], o DLP é considerado um problema difícil. A dificuldade nesse caso não se refere à sua concepção, mas aos requisitos computacionais necessários para encontrar uma solução do problema caso a ordem do grupo seja grande. Uma definição mais cformal sobre problemas difíceis e exemplos desses problemas podem ser encontrados no livro de Cormen et al [28].

Quando o grupo $\mathbb{G}$ em questão é formado por pontos de curvas elípticas e utiliza-se a notação aditiva, então o Problema do Logaritmo Discreto em Curvas Elípticas ou *Elliptic Curve Discrete Logarithm Problem* (ECDLP) é descrito da seguinte forma: dados os pontos $Q \in E(\mathbb{F}_q)$ e P um ponto gerador de $E(\mathbb{F}_q)$, encontrar o escalar k tal que $Q = kP$, operação que também é de complexidade alta para curvas apropriadamente escolhidas, apesar do cálculo de kP ser simples.

2.5 Representação de pontos de curvas elípticas

A Equação 2.1, na Seção 2.1.3 apresenta a curva elíptica utilizando coordenadas afins, onde os pontos têm a forma (x, y). No entanto, em situações práticas as operações de adição e de duplicação de pontos de uma curva elíptica podem se tornar muito custosas do ponto de vista computacional quando se utilizam coordenadas afins, principalmente devido às operações de inversão de elementos do corpo finito, como será visto no Capítulo 3.

Com intuito de contornar esse problema, os pontos de curvas elípticas costumam ser representados em outros tipos de coordenadas, chamadas de coordenadas projetivas. Foram propostos diferentes tipos de coordenadas projetivas [29], com o objetivo de tornar mais eficientes as operação com pontos de curvas elípticas. No tipo mais comum de coordenadas projetivas, o ponto $(X : Y : Z)$ na curva E corresponde ao ponto $(X/Z, Y/Z)$ em coordenadas afins quando $Z \neq 0$. O ponto no infinito $\mathcal{O}$ corresponde ao ponto $P_{\mathcal{O}} = (0 : 1 : 0)$ em coordenas projetivas.

No sistema de coordenadas projetivas padrão, a curva elíptica dada pela Equação 2.1 assume a forma dada pela Equação 2.5.

$$E(K) : Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_5Z^3, \tag{2.5}$$

Existem ainda outros tipos de coordenadas, como as coordenadas Jacobianas, de Chudnovsky e López-Dahab [29], que podem ser empregadas em trabalhos de implementação com o objetivo de reduzir o número de operações custosas do ponto de vista computacional.

2.6 Curvas hiperelípticas

Em 1989, poucos anos após a proposta inicial de uso de curvas elípticas para fins criptográficos, Koblitz propôs uma generalização das curvas elípticas com as chamadas curvas hiperelípticas [30]. Uma curva hiperelíptica C sobre um corpo algébrico K é definida pela Equação 2.6.

$$C : y^2 + h(x)y = f(x), \tag{2.6}$$

onde $h(x)$ e $f(x)$ são polinômios cujos coeficientes pertencem ao corpo K. O grau do polinômio $f(x)$ é igual a $2g + 1$ e o grau de $h(x)$ deve ser menor ou igual a g, um inteiro positivo. Para $g = 1$, tem-se uma curva elíptica.

A Figura 2.5 ilustra uma curva hiperelíptica representada sobre o corpo $\mathbb{R}$ dos números reais.

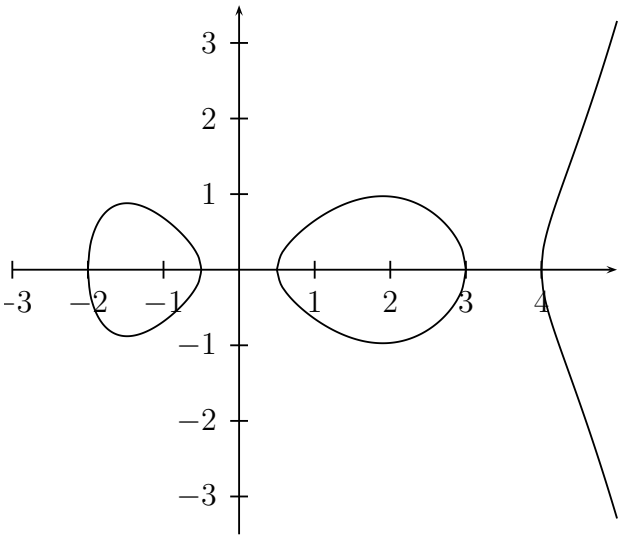


Fig. 2.5: Curva hiperelíptica $C_3 : y^2 = 0.420448207687874 * (x+2)(x+0.5)(x-0.5)(x-3)(x-4)$ representada sobre o corpo $\mathbb{R}$ dos números reais.

Os sistemas de criptografia baseados em curvas hiperelípticas apresentam a vantagem de usar chaves menores em comparação com aqueles baseados nas curvas elípticas, considerando o mesmo nível de segurança. Além de menor espaço de armazenamento, isso pode implicar um processamento mais eficiente. No entanto, os algoritmos e a forma de representar os elementos da criptografia sobre curvas hiperelípticas são bastante complexos e essa eficiência não pôde ser comprovada a princípio nos trabalhos de pesquisa de implementação, tanto em hardware quanto em software [31]. Em razão disso, apesar do curto espaço de tempo que separou as propostas das curvas elípticas e hiperelípticas para fins criptográficos, o desenvolvimento de pesquisa privilegiou as primeiras em detrimento das últimas.

Trabalhos recentes mostraram algoritmos e resultados eficientes para determinadas curvas hiperelípticas, como apontado por Lange [32], Galbraith et al [33] e no trabalho de implementação em hardware por Kim et al [34].

2.7 Emparelhamentos bilineares sobre curvas elípticas

2.7.1 Histórico e definições

No início da década de 90, os emparelhamentos bilineares eram empregados para mostrar que algumas classes de curvas elípticas eram inseguras para fins criptográficos. Os emparelhamentos bilineares de Weil e de Tate eram usados para mostrar que era possível reduzir o problema do logaritmo discreto em algumas curvas para o problema do logaritmo discreto em um corpo finito [1].

Já em 2000, Joux mostrou o uso de emparelhamentos bilineares no protocolo Diffie-Hellman para combinação de chaves entre três entidades em uma única rodada. Também nesse ano, Sakai empregou emparelhamentos bilineares como primitivas para protocolos criptográficos. Desde então, foram encontradas diversas aplicações criptográficas para os emparelhamentos bilineares e a que mais tem se destacado desde então é a chamada Criptografia Baseada em Identidades ou *Identity Based Cryptography* (IBC), a ser descrita na Seção 2.8.2.

Para definir um emparelhamento bilinear, são considerados os seguintes parâmetros:

- $\mathbb{G}_1$ e $\mathbb{G}_2$ são grupos cíclicos aditivos de ordem prima r;
- $\mathbb{G}_T$ é um grupo multiplicativo, também de ordem prima r;
- G_1 e G_2 são elementos geradores dos grupos $\mathbb{G}_1$ e $\mathbb{G}_2$, respectivamente;
- ϕ é um isomorfismo ou mapa de distorção entre $\mathbb{G}_1$ e $\mathbb{G}_2$, tal que $\phi(G_1) = G_2$;
- e é um mapeamento bilinear computável, tal que $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$;

Um emparelhamento bilinear é um mapeamento $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ que possui as propriedades de

- **bilinearidade:** sejam os elementos $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$ e sejam os escalares $a, b \in \mathbb{Z}$. Então as seguintes igualdades são válidas para o emparelhamento bilinear:

$$\begin{aligned} e(aP, bQ) &= e(P, bQ)^a = \\ e(P, Q)^{ab} &= e(bP, Q)^a = \end{aligned}$$

$$\begin{aligned} e(bP, aQ) &= e(abP, Q) = \\ e(P, abQ) &= e(-aP, -bQ); \end{aligned}$$

- **ser não degenerado:** $e(P, Q) \neq u$, onde u é o elemento identidade de $\mathbb{G}_T$, $P \in \mathbb{G}_1$ e $Q \in \mathbb{G}_2$;
- **computabilidade:** o emparelhamento $e(P, Q)$ pode ser eficientemente computado.

A definição acima se aplica a qualquer emparelhamento bilinear. Em se tratando de emparelhamentos bilineares sobre curvas elípticas, existem dois tipos principais: o emparelhamento de Weil [35] e o emparelhamento de Tate [36], ambos envolvendo complexas propriedades matemáticas. Para objetivos práticos, o emparelhamento de Tate costuma ser mais eficientemente implementável [29] e é o tipo adotado neste trabalho.

No emparelhamento de Tate, $\mathbb{G}_1$ é um subgrupo do grupo de pontos de ordem r de uma curva elíptica $E(\mathbb{F}_q)$, $\mathbb{G}_2$ é formado pelo grupo de pontos da curva $E(\mathbb{F}_{q^k})$, onde k é chamado de *grau de imersão* e é definido como o menor valor de k tal que $r|q^k - 1$ e $\mathbb{G}_T$ é um subgrupo do corpo de extensão $\mathbb{F}^*_{q^k}$. A equação da curva é a mesma para $E(\mathbb{F}_q)$ e $E(\mathbb{F}_{q^k})$, mas as coordenadas no primeiro caso encontram-se no corpo $\mathbb{F}_q$ e no segundo, numa extensão $\mathbb{F}_{q^k}$. Dessa maneira, o emparelhamento de Tate pode ser descrito como o mapeamento:

$$e : E(\mathbb{F}_q)[r] \times E(\mathbb{F}_{q^k}) \rightarrow \mathbb{F}^*_{q^k}. \tag{2.7}$$

Nota-se que o primeiro argumento e o valor do emparelhamento têm ambos ordem r, isto é, se $e(P, Q) = \nu$, então $rP = \mathcal{O}$ e $\nu^r = 1$. Para alguns casos é possível definir uma função ϕ, chamada mapa de distorção, tal que $\phi : E(\mathbb{F}_q)[r] \rightarrow E(\mathbb{F}_{q^k})$, o que permite definir o emparelhamento modificado $\hat{e}$, tal que os dois argumentos pertencem ao mesmo grupo:

$$\hat{e} : E(\mathbb{F}_q)[r] \times E(\mathbb{F}_q)[r] \rightarrow \mathbb{F}^*_{q^k}, \tag{2.8}$$

$$\hat{e}(P, Q) = e(P, \phi(Q)). \tag{2.9}$$

Esse emparelhamento é dito simétrico, ou de acordo com a classificação de Galbraith et al, como emparelhamento do Tipo 2 [37].

2.7.2 Aspectos práticos de implementação

Com relação ao grau de imersão, para curvas não-supersingulares, o valor de k geralmente é alto, o que implica um nível elevado de segurança, e ao mesmo tempo um maior custo computacional para calcular o emparelhamento bilinear. Para curvas supersingulares e curvas ordinárias especialmente

construídas, como as curvas MNT, curvas Barreto-Naehrig e as curvas de Edwards, o valor de k é mantido baixo, sem comprometer a segurança, desde que o corpo finito $\mathbb{F}_q$ seja escolhido com extensão apropriada.

Uma curva E para a qual é possível encontrar uma coleção $\langle e, r, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T \rangle$ é uma curva adequada para aplicações criptográficas e é chamada de *pairing-friendly*.

Como os trabalhos de implementação mostrados aqui se baseiam em curvas supersingulares, vamos focar nas características do emparelhamento de Tate para esse tipo de curva. Nas curvas supersingulares, pode-se fazer $\mathbb{G}_1 = \mathbb{G}_2$, o que simplifica a representação dos elementos do corpo finito e exige menor espaço para armazenar os elementos. Essa característica e a possibilidade de usar o grau de imersão k de baixo valor motivaram a escolha das curvas supersingulares, pois tais características são importantes para um dispositivo em que a quantidade de memória é bastante restritiva.

Daqui em diante, o emparelhamento de Tate $\hat{e}(\cdot, \cdot)$ será denotado simplesmente por $e(\cdot, \cdot)$ a fim de não carregar a notação.

2.7.3 Algoritmo para cálculo de emparelhamento bilinear

Com o intuito de ilustrar o cálculo do emparelhamento bilinear com pontos de curva elíptica, será apresentado um exemplo simplificado. Para isso, serão considerados os pontos P e Q, de modo que P é um ponto de ordem r e ambos pertencem à curva elíptica $E(\mathbb{F}_{q^k})$.

O cálculo do emparelhamento bilinear geralmente é dividido em duas partes: o laço de Miller e a exponenciação final, que se faz necessária no emparelhamento de Tate para garantir a unicidade do resultado. A versão básica de cálculo do emparelhamento é mostrada no Algoritmo 2.1.

Algoritmo 2.1 Cálculo de $e(P, Q)$ [10]

Entrada: Ordem r, pontos P e $Q \in E(\mathbb{F}_{q^k})$.
Saída: $e(P, Q)$.

1: $T \leftarrow P$, $f \leftarrow 1$
2: **Para** $i \leftarrow \lfloor \log_2(r) \rfloor$ - 1 **até** 0 **faça**
3: $\quad f = f^2 \cdot l_{T,T}(Q)/v_{T,T}(Q)$
4: $\quad T = 2T$
5: $\quad$ **Se** $r_i = 1$ **então**
6: $\quad\quad f = f \cdot l_{T,P}(Q)/v_{T,P}(Q)$
7: $\quad\quad T = T + P$
8: $f \leftarrow f^{(q^k-1)/r}$
9: **Retorne** f

No Algoritmo 2.1, pode-se observar que o ponto P está sendo implicitamente multiplicado por sua ordem r, usando o clássico método *Double-and-Add*, até que resulte o ponto no infinito $\mathcal{O}$.

O valor $l_{A,B}(Q)$ representa a distância entre o ponto fixo Q e a reta que passa por A e B quando esses dois pontos são adicionados. O valor $v_{A,B}(Q)$ representa a distância entre Q e o resultado soma $A + B$. Utilizando coordenadas afins, pode-se considerar que os pontos Q, A e $A + B$ possuem, respectivamente, as coordenadas (x_Q, y_Q), (x_j, y_j) e (x_{j+1}, y_{j+1}). Considerando ainda que λ_j é a inclinação da reta obtida quando se adiciona B ao ponto A, as Equações 2.10 e 2.11 mostram como obter as distâncias usadas no cálculo do emparelhamento.

$$l_{A,B}(Q) = (y_Q - y_j) - \lambda_j(x_Q - x_j), \tag{2.10}$$

$$v_{A,B}(Q) = (x_Q - x_{j+1}), \tag{2.11}$$

onde $\lambda_j = \frac{y_B - y_j}{x_B - x_j}$.

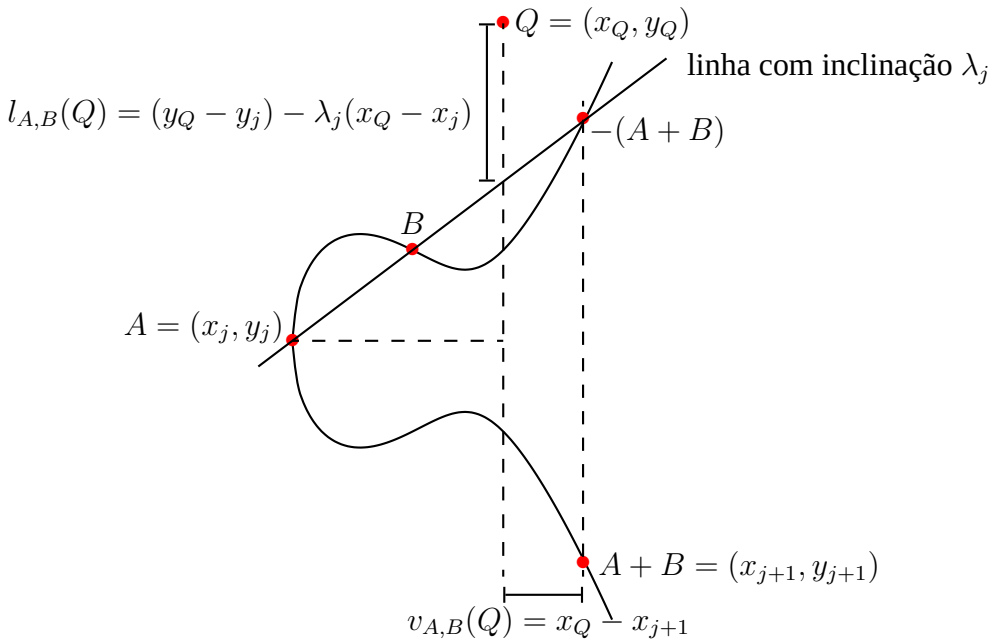
A Figura 2.6 apresenta uma versão gráfica das funções $l_{A,B}(Q)$ e $v_{A,B}(Q)$, para os casos $A \neq B$ e $A = B$.

Conforme observado no início desta seção, o Algoritmo 2.1 apresenta uma versão básica do cálculo de emparelhamento. Observa-se que, na prática, os pontos P e Q não são aleatoriamente escolhidos, já que uma escolha inadequada poderia levar a situações de erro, como no caso em que a reta da adição ou duplicação passa por Q ou quando a distância $v_{A,B}(Q)$ é nula. Para evitar tais situações, e até por motivos de eficiência, os pontos P e Q são linearmente independentes e definidos em curvas elípticas representadas sobre corpos distintos. É o que se observa na definição apresentada na Seção 2.7.1, onde define-se um mapa de distorção $\phi : E(\mathbb{F}_q) \rightarrow E(\mathbb{F}_{q^k})$, que mapeia pontos de ordem r do corpo base para pontos de ordem r no corpo de extensão, de modo a garantir que os pontos P e $\phi(Q)$ sejam sempre linearmente independentes e o emparelhamento $e(P, \phi(Q))$ seja sempre bem definido e não degenerado.

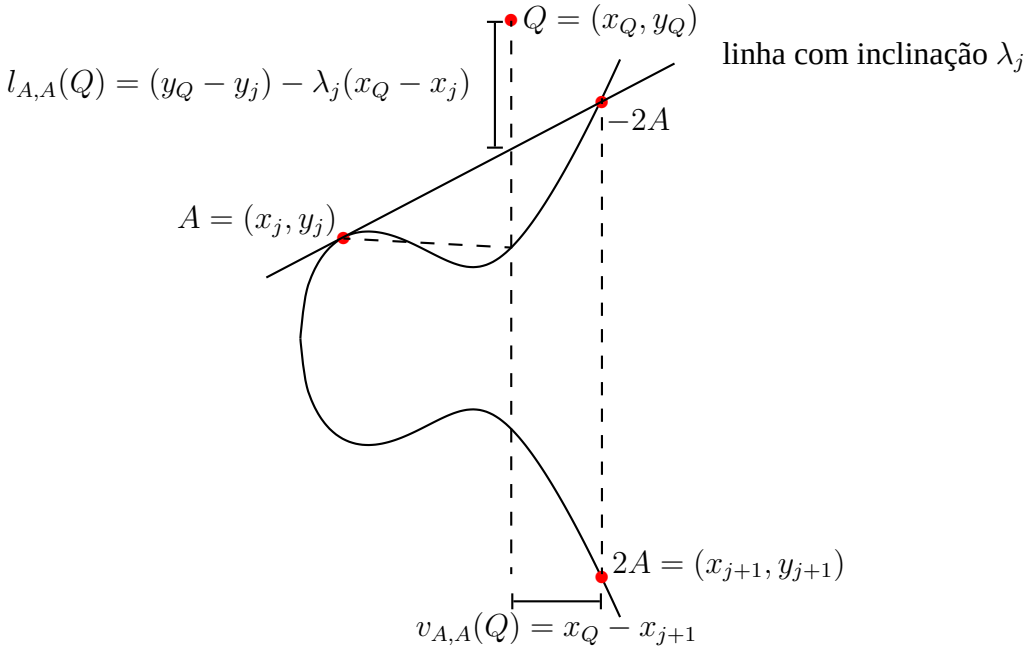
O algoritmo mostrado para cálculo do emparelhamento bilinear é básico e pode ser otimizado em diferentes pontos, como por exemplo, escolhendo um valor para r com baixo peso de Hamming, o que implica em poucas execuções das instruções decorrentes da condição $r_i = 1$. Além disso, dependendo da curva selecionada, é possível criar equações específicas e simplificadas para o algoritmo de emparelhamento. É com base nessas e em outras observações que surgiram propostas como o algoritmo BKLS [10], η (Eta) [38], Ate [39] e R-ate [40], por exemplo.

2.7.4 Segurança nos criptossistemas baseados em emparelhamentos bilineares

A construção de um emparelhamento bilinear traz consigo uma série de implicações com relação à complexidade. Nas discussões a seguir, será considerado um grupo aditivo $\mathbb{G}$ (assim como $\mathbb{G}_1$ e $\mathbb{G}_2$ na Seção 2.7) e um grupo multiplicativo $\mathbb{G}_T$ (também como na Seção 2.7).



(a) Funções $l_{A,B}(Q)$ e $v_{A,B}(Q)$ para $A \neq B$.



(b) Funções $l_{A,B}(Q)$ e $v_{A,B}(Q)$ para $A = B$.

Fig. 2.6: Representação gráfica das funções $l_{A,B}$ e $v_{A,B}$ empregadas durante o cálculo do emparelhamento.

Primeiramente, pode-se perceber que o problema do logaritmo discreto no grupo $\mathbb{G}$, não é mais difícil que o Problema do Logaritmo Discreto em $\mathbb{G}_T$. Considere $Q = aP$, sendo a um número inteiro desconhecido. Resolver o Problema do Logaritmo Discreto implica descobrir a para um dado P e um Q aleatório. Nota-se que:

$$e(P, Q) = e(P, aP) = e(P, P)^a. \tag{2.12}$$

Portanto, pode-se reduzir o Problema do Logaritmo Discreto em $\mathbb{G}$ no Problema do Logaritmo Discreto em $\mathbb{G}_T$. Dados P e $Q \in \mathbb{G}$, e considerando que o mapeamento e pode ser eficientemente computado, pode-se calcular $\log_P(Q)$ da seguinte maneira:

1. determine $\nu_1 = e(P, P)$;
2. determine $\nu_2 = e(P, Q)$;
3. determine $a = log_{\nu_1}(\nu_2)$;
4. a será também $\log_P(Q)$.

A segurança nos criptossistemas baseados em emparelhamentos é baseada nos seguintes problemas.

- **Decision Diffie-Hellman Problem (DDHP)**

  Sejam $P \in \mathbb{G}$ e a, b e $c \in \mathbb{Z}_n$, onde n é a ordem de $\mathbb{G}$. Dados P, aP, bP e $cP \in \mathbb{G}$, decidir se $c = ab \bmod n$. Deve-se lembrar que, considerando do Problema do Logaritmo Discreto intratável, desconhecem-se os valores de a, b e c. Joux e Nguyen [41] observaram que

  $$c = ab \bmod n \Leftrightarrow e(P, cP) = e(aP, bP). \tag{2.13}$$

  O DDHP, portanto, é considerado um problema fácil em grupos onde exista um emparelhamento admissível.

- **Computational Diffie-Hellman Problem (CDHP)**

  Sejam $P \in \mathbb{G}_1$ e a e $b \in \mathbb{Z}_n$, onde n é a ordem de P. Dados P, aP e $bP \in \mathbb{G}$, calcular abP. Nota-se que o CDHP pode ser resolvido sem dificuldades num grupo onde o Problema do Logaritmo Discreto seja tratável.

- **Gap Diffie-Hellman Problem (GDHP)**

  Seja $\mathbb{G}$ um grupo onde o DDHP é tratável. O GDHP consiste em resolver o CDHP em $\mathbb{G}$, sabendo que o DDHP é tratável em $\mathbb{G}$.

- **Bilinear Diffie-Hellman Problem (BDHP)**

  Sejam $P \in \mathbb{G}$ e a, b e $c \in \mathbb{Z}_n$, onde n é a ordem de P. Dados P, aP, bP e $cP \in \mathbb{G}$, calcular $e(P,P)^{abc}$. Trata-se de uma variante do CDHP, pois se existe um algoritmo para resolver o CDHP em $\mathbb{G}$, dados $P \in \mathbb{G}$ e a, b e $c \in \mathbb{Z}_n$, o valor abP é calculado usando esse algoritmo e então o valor de $e(abP, cP) = e(P,P)^{abc}$ será a solução do BDHP.

Para cada um dos quatro problemas citados anteriormente, existem os co-problemas – co-DDHP, co-CDHP, co-GDHP e co-BDHP – que envolvem os dois grupos aditivos $\mathbb{G}_1$ e $\mathbb{G}_2$ no lugar de um único grupo e o emparelhamento $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$

A segurança dos criptossistemas baseados em emparelhamentos bilineares é baseada na intratabilidade do GDHP ou do BDHP, ou de ambos, em um certo grupo $\mathbb{G}$, ou, similarmente, na intratabilidade dos co-problemas correspondentes em um par de grupos $\mathbb{G}_1$ e $\mathbb{G}_2$. Conjectura-se que a escolha apropriada da extensão do corpo finito $\mathbb{F}_q$ e do grau de imersão k torna o GDHP e o BDHP intratáveis e, portanto, garante a segurança dos sistemas baseados em emparelhamentos bilineares.

2.8 Criptossistemas baseados em emparelhamentos bilineares

Os conceitos apresentados até este ponto servem de base para a apresentação dos sistemas criptográficos que fazem uso dos emparelhamentos bilineares sobre curvas elípticas. Existe uma variedade de tais sistemas, como é possível verificar em *The Pairing-Based Crypto Lounge*, uma página na Internet mantida pelo Prof. Paulo Barreto [42].

2.8.1 Arquitetura de software

Os emparelhamentos bilineares sobre curvas elípticas encontram utilidade quando aplicados a protocolos criptográficos, entre os quais se destacam aqueles para troca de chaves, cifragem e decifragem de dados, bem como de assinatura e verificação de documentos digitais.

Para colocar em prática qualquer protocolo criptográfico baseado em emparelhamentos bilineares é indicado adotar uma arquitetura de software modular, na qual os módulos sejam interdependentes e cada módulo seja responsável por operações de um domínio específico. A Figura 2.7 ilustra essa divisão em módulos e a dependência entre eles.

Os módulos apresentados na Figura 2.7 podem ser descritos da seguinte forma:

- **Manipulação de inteiros grandes:** aplicações criptográficas reais utilizam inteiros grandes com o objetivo de se tornarem mais robustas contra diversos ataques, inclusive do tipo força bruta. O número de bits utilizados em palavras de registradores e memórias nos computadores atuais, especialmente em cartões inteligentes, costuma ser menor que o número de bits

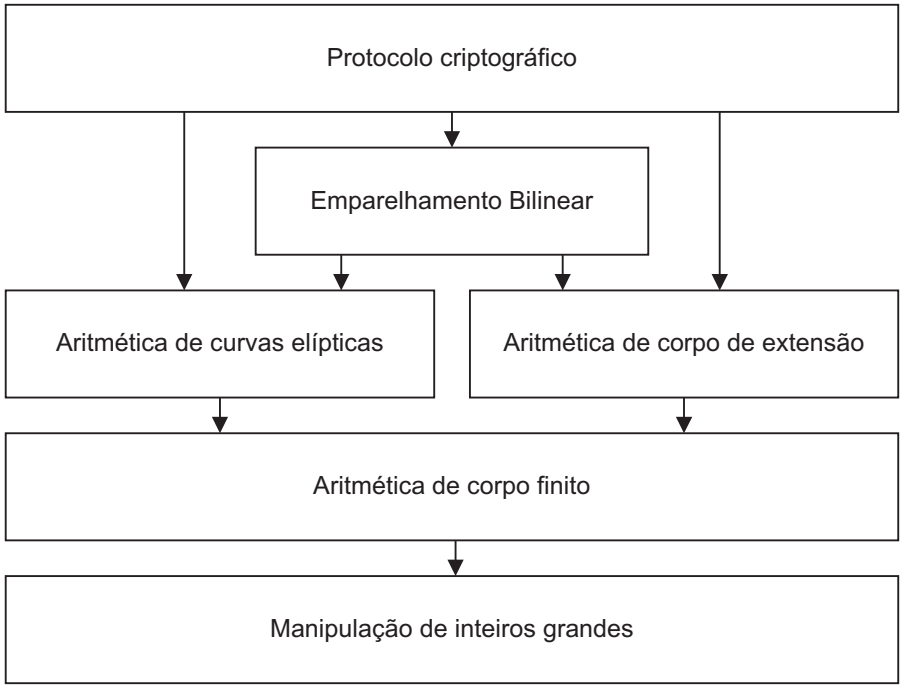


Fig. 2.7: Módulos de software necessários para implementação de criptossistemas baseados em emparelhamentos bilineares.

necessário para representar um operando usado em criptossistemas. Por essa razão, é preciso desenvolver a infraestrutura de software para tratar essa necessidade, criando estruturas de dados para agrupar o número de bits necessário para representar os operandos, bem como as operações básicas necessárias, como por exemplo, copiar e apagar.

- **Aritmética no corpo finito:** como descrito na Seção 2.1.5, aplicações criptográficas com objetivos práticos utilizam corpos finitos, sejam eles binários ou primos. Nessa camada são implementadas operações como adição, multiplicação, redução modular, inversão, raiz quadrada, exponenciação, entre outras.

- **Aritmética de curvas elípticas:** é a parte responsável por realizar as operações básicas com pontos de curvas elípticas, como adição, duplicação e multiplicação escalar.

- **Aritmética no corpo de extensão:** conforme dito na Seção 2.7 e mostrado no Algoritmo 2.1, o resultado do emparelhamento bilinear situa-se no corpo de extensão, determinado de acordo com o tipo da curva e o grau de imersão. Analogamente ao que ocorre para o corpo finito, são implementadas nessa camada de software operações de adição, multiplicação, exponenciação e inversão.

- **Emparelhamento bilinear:** essa é a camada que efetivamente calcula o emparelhamento bilinear e que depende de todas as camadas anteriores. Pode-se notar que uma implementação eficiente do emparelhamento bilinear não depende somente do algoritmo escolhido, mas também da eficiência na implementação das camadas inferiores.

- **Protocolo criptográfico:** é a camada que tem utilidade de fato para um usuário ou uma aplicação. Se implementada de forma eficiente e com boa interface com o usuário, o mesmo não necessita ter conhecimentos profundos sobre curvas elípticas ou emparelhamentos bilineares para usufruir da segurança proveniente do uso desses conceitos. Como exemplos, podem ser citados os protocolos de troca de chaves [43], assinaturas digitais [44] e cifragem de dados [45].

A próxima seção apresenta o ramo em que os emparelhamentos encontram maior aplicação, que é a chamada criptografia baseada em identidades.

2.8.2 Criptografia baseada em identidades

Uma possível aplicação dos emparelhamentos bilineares é na realização da chamada criptografia baseada em identidades, ou *Identity Based Cryptography* (IBC). Com exceção do esquema básico de IBC proposto por Cocks [46], que é baseado em resíduos quadráticos, a grande maioria dos criptossistemas que empregam IBC fazem uso de emparelhamentos bilineares sobre curvas elípticas.

O conceito de IBC foi originalmente proposto por Shamir [47]. A ideia era criar um esquema mais simples de obtenção e utilização de chaves públicas em contraposição aos esquemas tradicionais de infraestrutura de chaves públicas, que utilizam certificados para garantir a autenticidade das chaves públicas, como descrito na Seção 2.1.2.

Em sua publicação original [47], Shamir pressupunha a existência de uma entidade confiável, o Gerador de Chaves Privadas ou *Private Key Generator* (PKG), cuja função é entregar a cada novo usuário A do esquema de IBC um cartão inteligente personalizado com as chaves pública e_A e privada d_A. O cartão teria a capacidade de cifrar e decifrar mensagens, bem como de gerar e e verificar assinaturas digitais. A chave pública e_A, ao contrário dos esquemas tradicionais de PKI, pode ser um identificador escolhido pelo próprio usuário A, como seu nome, CPF, número de telefone, endereço de rede, endereço de email, ou então uma combinação de todos esses identificadores, desde que essa chave identifique unicamente A, de modo que ele não possa negá-la posteriormente e que ela seja facilmente obtida pelos demais usuários.

A chave privada, num esquema de IBC, deve ser computada pelo PKG, já que não há nada especial (secreto) na identidade de um usuário. Se U pudesse computar sua própria chave privada, então ele poderia também computar as chaves privadas correspondente às identidades dos usuários A' e A'', por exemplo, e o sistema não poderia ser considerado seguro.

Shamir observou ainda em seu artigo que IBC permitiria criar um esquema ideal de email: se é possível saber o nome e endereço do usuário A, então é possível enviar mensagens que apenas A pode ler ou verificar assinaturas que apenas A poderia ter gerado. Isso torna os aspectos criptográficos da comunicação praticamente transparentes para os usuários, que podem usar um esquema seguro sem possuírem conhecimentos sobre chaves e protocolos.

Esquemas de IBC podem se adaptar bem a grupos fechados, como empresas, universidades e instituições governamentais, onde pode-se definir uma unidade responsável por atuar como PKG.

Uma outra possibilidade de criar a chave pública num esquema de IBC é concatenar ao identificador uma data. Dessa forma, é possível gerar uma mensagem secreta que só poderá ser decifrada pelo usuário quando chegar essa data, momento em que o PKG gerará a chave privada correspondente. A utilização de datas também pode ser vantajosa porque cria chaves temporárias, o que facilita a revogação de chaves no caso de haver algum comprometimento do PKG ou da chave privada do usuário.

A característica do PKG possuir controle sobre a geração das chaves privadas é chamada de custódia das chaves ou *key escrow*. O PKG pode se fazer passar por qualquer um dos usuários e dessa forma gerar assinaturas falsas ou decifrar mensagens secretas. Apesar de parecerem características negativas, elas podem ser vantajosas e desejáveis em determinados contextos.

Existem outras propostas análogas à IBC, como a *Certificateless Cryptography* [48], que emprega intensamente emparelhamentos bilineares e procura evitar o problema da custódia das chaves, mas que perde a característica de obter a chave pública diretamente da identidade do usuário. Recentemente, surgiram propostas para resolver o problema de *key escrow* em sistemas de IBC [49].

2.8.3 Sistemas baseados em identidades usando emparelhamentos bilineares

Apesar da ideia de IBC ter surgido em 1984 [47], Shamir reconheceu que naquela época o RSA não atendia as condições necessárias para a criação de um sistema baseado em identidades. Ele mesmo comparou com a situação à época da criação da criptografia de chaves públicas: em 1976 o conceito foi definido e as potenciais aplicações foram investigadas, mas sua implementação concreta só veio a ocorrer em 1978.

Em 2001, Boneh e Franklin [45] propuseram a primeira implementação funcional de um esquema de cifragem baseado em identidades, empregando emparelhamentos bilineares. Trata-se de um esquema de cifragem, chamado de *Identity Based Encryption* (IBE), dividido em quatro algoritmos: configuração (*setup*), extração (*extract*), cifragem (*encrypt*) e decifragem (*decrypt*), conforme descrito a seguir.

A descrição a seguir emprega o conceito de função de espalhamento [50]. Idealmente, essa função funciona como um oráculo aleatório, que seria como uma caixa preta que responde a uma consulta

com uma resposta indistinguível de uma sequência aleatória.

Configuração: Dado como entrada o parâmetro de segurança n, gera-se um conjunto de parâmetros chamado $params$, da seguinte forma.

- Geração de um primo q de n bits, um grupo aditivo $\mathbb{G}_1$, um grupo multiplicativo $\mathbb{G}_T$, ambos de ordem q, e um emparelhamento bilinear da forma $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$. Escolha de um ponto P gerador do grupo $\mathbb{G}_1$.

- Seleção de um escalar aleatório $s \in \mathbb{F}_q^*$ e cálculo de $P_{pub} = sP$. O escalar s é mantido em segredo e funciona como chave mestra do PKG e o ponto P_{pub} é divulgado para todos os participantes do esquema de IBC.

- Escolha das funções de *hash* criptográfico:

$$H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1^*, \tag{2.14}$$

$$H_2 : \mathbb{G}_T \rightarrow \{0,1\}^{\ell}, \tag{2.15}$$

  ou seja, H_1 é uma função de espalhamento que tem como entrada uma sequência de bits de tamanho arbitrário e resulta em um ponto do grupo elíptico diferente do ponto $\mathcal{O}$ e a função H_2 tem como entrada o resultado de um emparelhamento entre dois pontos do grupo e responde com uma sequência de bits de tamanho ℓ.

- O espaço das mensagens é $\mathcal{M} = \{0,1\}^{\ell}$ e o espaço dos textos cifrados é $\mathcal{C} = \mathbb{G}_T^* \times \{0,1\}^{\ell}$.

- Os parâmetros públicos do sistema são $params = \langle q, \mathbb{G}_1, \mathbb{G}_T, e, \ell, P, P_{pub}, H_1, H_2 \rangle$.

Extração: Dado o conjunto de caracteres $ID_A \in \{0,1\}^*$ que representa o identificador de um usuário A, o PKG executa os passos:

- Cômputo do ponto $Q_{ID_A} = H_1(ID_A) \in \mathbb{G}_1^*$.

- Cômputo da chave privada $S_{ID_A} = sQ_{ID_A}$, onde s é a chave mestra só conhecida pelo PKG. A chave privada S_{ID_A} é entregue de forma segura ao usuário A.

Cifragem: Para cifrar a mensagem $M \in \mathcal{M}$ usando o identificador ID_A como chave pública, o usuário B executa os passos:

- Cômputo do ponto $Q_{ID_A} = H_1(ID_A) \in \mathbb{G}_1^*$.

- Seleção de um escalar aleatório $r \in \mathbb{F}_q^*$.

- Cômputo do texto cifrado $C = \langle U, V \rangle$, onde[1]:

$$\begin{cases} U = rP \\ V = M \oplus H_2(e(rQ_{ID_A}, P_{pub})). \end{cases}$$

Decifragem: Após receber o texto cifrado $C = \langle U, V \rangle$, o usuário A emprega sua chave privada S_{ID_A} para recuperar o texto em claro M:

- $M = V \oplus H_2(e(S_{ID_A}, U))$.

Nota-se que a mensagem M é de fato recuperada, pois:

$$\begin{aligned} V \oplus H_2(e(S_{ID_A}, U)) &= V \oplus H_2(e(S_{ID_A}, rP)) \\ &= V \oplus H_2(e(sQ_{ID_A}, rP)) \\ &= V \oplus H_2(e(rQ_{ID_A}, sP)) \\ &= V \oplus H_2(e(rQ_{ID_A}, P_{pub})) \\ &= M \oplus H_2(e(rQ_{ID_A}, P_{pub})) \oplus H_2(e(rQ_{ID_A}, P_{pub})) \\ &= M. \end{aligned}$$

A descrição anterior considera que as funções de espalhamento H_1 e H_2 funcionam como oráculos aleatórios. Na prática, no entanto, são utilizadas funções baseadas na funções de *hash* comuns, como aquelas da família SHA [51]. Tais funções perdem a característica de oráculo aleatório, pois sua estrutura interna é conhecida, o que pode ser aproveitado por intrusos mal-intencionados [52].

Existem outras propostas seguras de criptossistemas baseados em emparelhamentos bilineares que não fazem uso de oráculos aleatórios, como se pode ver na proposta de Boneh e Boyen [53], que utiliza a técnica de inversão de expoentes.

2.8.4 Comparação entre sistemas baseados em identidade e PKI

A Tabela 2.1 apresenta alguns parâmetros para comparação entre esquemas ideais de PKI e IBC.

Nota-se que o nível de confiança que uma entidade deposita na autoridade central varia. Numa PKI, por exemplo, caso a CA resolva agir maliciosamente, ela pode emitir certificados falsos, ligando a identidade do usuário a uma chave pública diferente daquela que ele possui. Num modelo ideal de infraestrutura de chaves públicas, esse ataque pode ser detectado com uma consulta a um diretório

[1] A operação $\oplus$ denota XOR (ou exclusivo)

Característica	PKI	IBC
Existência de certificados	sim	não
Geração de chaves em instantes distintos	não	sim
Irretratabilidade de assinaturas	sim	não
Chave pública escolhida livremente	não	sim
Nível de confiança depositado na autoridade central	médio	alto

Tab. 2.1: Comparação entre os modelos PKI e IBC.

público de certificados mantido fora da CA, sendo possível verificar que existem dois certificados dentro do prazo de validade emitidos para uma única entidade, e isso só poderia acontecer legalmente se um dos certificados já tivesse sido revogado. Como existe essa maneira de identificar quando a CA age maliciosamente, então o nível de confiança a ser depositado nela pode ser considerado médio.

Já para um esquema tradicional de IBC, a Autoridade de Confiança PKG possui acesso irrestrito às chaves privadas de todos os usuários do sistema, podendo forjar assinaturas e decifrar mensagens. Isso evidencia que o nível de confiança depositado nessa entidade deve ser muito elevado.

A ligação forte entre identidade e chave num esquema de IBC pode se benéfico para sistemas de comunicação em que há uma ligação também forte entre o usuário e o identificador da comunicação. Um exemplo disso ocorre em redes móveis, onde um canal de cifragem pode ser criado a partir do conhecimento do endereço físico daquele dispositivo que recebe as mensagens secretas.

Considerando as características de cada modelo, pode-se afirmar que a IBC é mais recomendada para ambientes fechados, como uma empresa, onde uma autoridade que conhece todas as chaves privadas pode ser necessária e tolerada. Além disso, como a chave pública está diretamente relacionada com o identificador da entidade, não há necessidade de certificados, o que viabiliza implementações mais simples para os usuários.

2.9 Conclusões do capítulo

Este capítulo teve o propósito de apresentar os conceitos e a notação que servirão de base para a compreensão dos resultados e das discussões que serão apresentados neste trabalho. Primeiramente, foi apresentado o conceito de criptografia e sua aplicação para prover sigilo, integridade dos dados, autenticação e irretratabilidade. Foi mostrado que os criptossistemas se dividem em sistemas de criptografia simétrica e de criptografia assimétrica ou de chave pública, modalidade dentro da qual se encaixa a criptografia de curvas elípticas.

Os conceitos de aritmética modular, grupos algébricos e corpos finitos foram mostrados a fim de fundamentar a aritmética de curvas elípticas, em especial as operações de adição e duplicação de pontos de uma curva, que são essenciais para a construção de sistemas criptográficos. Tais sistemas

baseiam sua segurança no chamado Problema do Logaritmo Discreto sobre Curvas Elípticas, que é considerado difícil.

Foram apresentadas também as funções matemáticas conhecidas como emparelhamentos bilineares, as quais utilizam como parâmetros pontos de curvas elípticas. Essas funções apresentam como principal propriedade a bilinearidade, que viabiliza a construção de criptossistemas inovadores e seguros, entre os quais se encontra a chamada criptografia baseada em identidades ou IBC.

Com foco na aplicação dos emparelhamentos bilineares, foi apresentada a arquitetura de software empregada na construção de criptossistemas e a aplicação dos emparelhamentos na construção de um esquema de cifragem baseado em identidades. Por fim, foi apresentada uma comparação entre os criptossistemas de IBC e de chave pública típica (PKI), a fim de evidenciar suas características e respectivas vantagens.

No próximo capítulo será apresentada uma forma eficiente de se implementar multiplicação escalar para curvas supersingulares sobre corpos binários.

Capítulo 3

Multiplicação escalar em curvas supersingulares sobre corpos binários

Conforme explicado no Capítulo 1, um dos objetivos deste trabalho é mostrar maneiras eficientes e seguras de viabilizar protocolos baseados em emparelhamentos bilineares utilizando cartões inteligentes.

Muitos protocolos baseados em emparelhamentos bilineares exigem, além do próprio cálculo do emparelhamento, a multiplicação escalar de pontos da curva elíptica. Um exemplo é o protocolo de cifragem baseado em identidades mostrado no Capítulo 2. Existem ainda os criptossistemas baseados em emparelhamentos que utilizam a técnica de delegação de emparelhamentos, que será detalhada no Capítulo 5. De maneira simplificada, esta técnica consiste em retirar a carga de calcular um emparelhamento, que é uma operação custosa, de um dispositivo com baixo poder computacional, como um cartão inteligente, e delegar esse cálculo para um dispositivo com maior poder computacional, de modo que as informações passadas do cartão para o meio externo não revelem dados sigilosos armazenados no cartão. Para aplicar a técnica de delegação de emparelhamentos, é importante que o cartão inteligente consiga efetuar a multiplicação escalar de pontos da curva elíptica de forma eficiente e segura.

Durante a fase de implementação dos emparelhamentos bilineares em cartões inteligentes optou-se por trabalhar com a família de curvas supersingulares sobre corpos binários em razão dos algoritmos mais eficientes que podem ser desenvolvidos para essas curvas. O trabalho com as curvas elípticas supersingulares nos permitiu perceber a possibilidade de desenvolver equações específicas para realizar a operação de multiplicação escalar. Para o caso de serem utilizadas coordenadas projetivas, o método proposto é mais eficiente que outros métodos de multiplicação escalar, como o *Double-and-Add*, a serem detalhados nas Seções 3.3 e 3.4. O método é baseado no algoritmo conhecido como *Montgomery's Ladder* e herda desse método a robustez contra ataques do tipo *side-channel*, mostra-

dos na Seção 4.1.9. Por fim, o método não exige pré-computação e pode calcular uma multiplicação escalar utilizando apenas a coordenada x do ponto, o que pode ser vantajoso para um dispositivo com pouca memória RAM, como um cartão inteligente, por exemplo. Apesar do trabalho de Saeki [54] ter chegado a equações semelhantes às mostradas aqui para o caso de coordenadas afins, nosso trabalho mostra como obter a coordenada y ao final da multiplicação escalar e apresenta também o desenvolvimento de equações para coordenadas projetivas. Como será visto, o método de multiplicação aplicado com coordenadas projetivas é mais eficiente que outros métodos de multiplicação escalar, o que não ocorre em situações práticas quando são empregadas coordenadas afins.

3.1 Curvas supersingulares

Neste capítulo serão consideradas as curvas supersingulares sobre corpos binários $\mathbb{F}_{2^m}$, que obedecem a Equação 3.1.

$$E(\mathbb{F}_{2^m}) : y^2 + y = x^3 + x + c, \tag{3.1}$$

onde m é ímpar e $c = 0$ ou 1.

Por muitos anos, curvas elípticas supersingulares foram negligenciadas, pois acreditava-se que o Problema do Logaritmo Discreto nessa família de curvas poderia ser resolvido de forma mais fácil do que nas curvas ordinárias [15]. No entanto, como apontado por Menezes, não se conhecem fraquezas para curvas supersingulares cuidadosamente selecionadas [55]. Como exemplo, para a curva dada pela Equação 3.1, tem-se um grau de imersão $k = 4$, de modo que o problema do Logaritmo Discreto nessa curva é equivalente ao problema do Logaritmo Discreto no corpo de extensão $\mathbb{F}_{2^{4m}}$. Isso significa que para construir criptossistemas seguros utilizando curvas supersingulares é preciso trabalhar com corpos $\mathbb{F}_{2^m}$ onde o valor de m deve ser maior do que no caso de curvas ordinárias, para um mesmo nível de segurança.

Para curvas não-supersingulares, o valor de k geralmente é alto, o que implica um nível elevado de segurança, e ao mesmo tempo um maior custo computacional para calcular o emparelhamento bilinear. Para curvas supersingulares e outras curvas especialmente construídas, chamadas de *pairing friendly*, o valor de k é mantido baixo, sem comprometer a segurança, desde que o corpo finito $\mathbb{F}_q$ seja escolhido com extensão apropriada.

As curvas supersingulares oferecem a vantagem de utilizarem grau de imersão k sempre menor que 6, ou seja, são valores pequenos. Como explicado no Capítulo 2, ter um grau de imersão pequeno é uma característica desejável para criptossistemas baseados em emparelhamentos bilineares, pois permitem o desenvolvimento de algoritmos eficientes para o cálculo do emparelhamento, como se pode ver na proposta de Barreto et al [38] e nos algoritmos citados no Capítulo 4, como o emparelhamento η. Também as implementações utilizando curvas supersingulares têm apresentado bons

resultados mesmo em dispositivos de baixo poder computacional, como se pode ver nos trabalhos de Bertonib [1] e de Aranha et al [12].

Por fim, as curvas supersingulares têm sido utilizadas em aplicações reais, e mereceram a criação pelo Internet Engineering Task Force (IETF) da RFC 5091: "Identity-Based Cryptography Standard (IBCS) #1: Supersingular Curve Implementations of the BF and BB1 Cryptosystems" [56], que descreve a aritmética empregada na aplicação *Voltage Identity Based Encryption*™.

Dentre os fatores que devem ser considerados na construção e implementação dos criptossistemas que empregam curvas elípticas supersingulares, a escolha de um método de multiplicação escalar adequado é fundamental para garantir segurança e eficiência. As equações propostas aqui procuram endereçar essas necessidades.

3.2 Histórico

A fim de melhorar a eficiência e a segurança dos criptossistemas de curvas elípticas, muitos métodos de multiplicação escalar já foram propostos. Em 1999, López e Dahab [57] propuseram um método para multiplicação escalar eficiente de pontos de curvas não supersingulares em corpos binários. Okeya et al [58] e Tetsuya et al [59] propuseram um método similar, mas para corpos primos. Fischer et al mostrou a implementação em um coprocessador para cartões inteligentes do método de multiplicação escalar para curvas elípticas em corpos primos [60]. Aqui desenvolvemos equações específicas para um método análogo ao método de López e Dahab [57], mas que se aplica a curvas supersingulares em corpos binários, como aquelas definidas pela Equação 3.1.

3.3 Método Double-and-Add

O método *Double-and-Add* é bastante utilizado para multiplicação escalar por sua simplicidade e por não exigir pré-computação. Ele é baseado no método de exponenciação que repete as operações de quadrado e multiplicação [61]. Uma versão desse método é apresentada no Algoritmo 3.1, onde o escalar s, cuja representação binária tem $n = \lceil \log_2 s \rceil$ bits, multiplica um ponto de curva elíptica P e é processado da direita para esquerda. Uma versão análoga processa o escalar da esquerda para direita, de modo que a ordem das operações de adição e duplicação ficam invertidas com relação ao Algoritmo 3.1.

Pode-se notar no Algoritmo 3.1 que o número de adições depende do número de bits setados no escalar s. A porcentagem de bits setados em relação ao total de bits é denominada peso de Hamming. Se A denota uma adição e D uma duplicação e se assumirmos que o peso de Hamming em s é igual

Algoritmo 3.1 Multiplicação escalar usando Double-and-Add [14]

Entrada: Ponto P_0, escalar positivo $s = (s_{n-1}, s_{n-2}, \ldots, s_0)_2$.
Saída: Ponto sP_0.
1: $R \leftarrow \mathcal{O}$.
2: **Para** $i \leftarrow 0$ **até** $n-1$ **faça**
3: **Se** $s_i = 1$ **então**
4: $R \leftarrow R + P_0$.
5: $P_0 \leftarrow 2P_0$.
6: **Retorne** R.

a 50%, pode-se esperar que o cálculo sP exija

$$\frac{n}{2}A + nD. \tag{3.2}$$

3.4 Métodos *NAF* e *w-NAF*

O escalar s pode ter uma representação chamada *non-adjacent form* (*NAF*). A maneira de obter a representação NAF de um escalar pode ser encontrada no livro *Guide to Elliptic Curve Cryptography* [14]. O custo de uma multiplicação escalar empregando o método *NAF* é de

$$\frac{n}{3}A + nD. \tag{3.3}$$

Vale lembrar que é preciso computar a nova representação do escalar antes de efetuar uma multiplicação escalar utilizando o método *NAF*.

Os métodos baseados em janela oferecem uma alternativa para efetuar a multiplicação escalar. Tais métodos exigem a pré-computação de alguns pontos antes que a operação de multiplicação escalar seja de fato realizada. Um desses métodos é o chamado *w-NAF*, que se baseia na representação *NAF* do escalar e em janelas de largura w. O custo de uma multiplicação escalar empregando o método *w-NAF* é de

$$\left(1D + (2^{w-2} - 1)A\right) + \left(\frac{n}{w+1}A + nD\right). \tag{3.4}$$

3.5 Método *Montgomery's Ladder*

O método de multiplicação escalar proposto é inspirado no algoritmo denominado *Montgomery's Ladder* [62], originalmente proposto para efetuar a operação de exponenciação, assim como no caso do método *Double-and-Add*. Esse algoritmo pode ser facilmente adaptado para efetuar multiplicação escalar de um ponto de curva elíptica P_0 por um escalar s, conforme ilustrado no Algoritmo 3.2.

Algoritmo 3.2 Multiplicação escalar usando o método *Montgomery's Ladder* [62]

Entrada: Ponto P_0, escalar positivo $s = (s_{n-1}, s_{n-2}, \ldots, s_0)_2$.
Saída: Ponto sP_0.

```
1: R ← P_0, S ← 2P_0.
2: Para i ← n − 2 até 0 faça
3:    Se s_i = 0 então
4:       S ← R + S, R ← 2R.
5:    Senão
6:       R ← R + S, S ← 2S.
7: Retorne R.
```

Uma característica importante do Algoritmo 3.2 é que para qualquer escalar s positivo de n bits será efetuado o mesmo número de operações: uma adição e uma duplicação de um ponto da curva elíptica. À semelhança do que foi feito para o *Double-and-Add*, pode-se esperar

$$(n-1)A + nD. \tag{3.5}$$

Essa característica torna o método de multiplicação empregando a técnica *Montgomery's Ladder* mais robusto contra ataques do tipo *side-channel*, pois o número de operações executado em cada iteração i não depende do bit s_i, como no caso do *Double-and-Add*.

Pode-se observar ainda no Algoritmo 3.2 que os pontos auxiliares S e R mantêm a diferença $S - R = P_0$, como apresentado no Lema 1 no Apêndice B. Essa propriedade será explorada na modificação do método *Montgomery's Ladder* para multiplicação escalar em curvas elípticas supersingulares.

3.6 Aritmética de curvas elípticas supersingulares

Conforme discutido na Seção 2.2, é possível derivar equações específicas para operar os pontos de curvas elípticas. Nesta seção, apresentamos as operações referentes à curva dada pela Equação 3.1, no corpo finito $\mathbb{F}_{2^m}$, considerando coordenadas afins.

1. **Identidade:** $P + \mathcal{O} = \mathcal{O} + P$ para todo $P \in E(\mathbb{F}_{2^m})$;

2. **Negativo:** Seja $P_1 = (x_1, y_1) \in E(\mathbb{F}_{2^m})$, tem-se que $(x_1, y_1) + (x_1, y_1 + 1) = \mathcal{O}$. O ponto $(x_1, y_1 + 1)$ é denotado por $-P_1$ e é chamado de *negativo* de P_1;

3. **Adição de pontos:** Seja $P_1 = (x_1, y_1) \in E(\mathbb{F}_{2^m})$ e $P_2 = (x_2, y_2) \in E(\mathbb{F}_{2^m})$, onde $P_1 \neq \pm P_2$.

Então $P_1 + P_2 = (x_3, y_3)$, onde:

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2}; \tag{3.6}$$

$$x_3 = \lambda^2 + x_1 + x_2; \tag{3.7}$$

$$y_3 = \lambda(x_1 + x_3) + y_1 + 1. \tag{3.8}$$

4. **Duplicação de ponto:** Seja $P_1 = (x_1, y_1) \in E(\mathbb{F}_{2^m})$, onde $P_1 \neq -P_1$. Então $2P_1 = P_3 = (x_3, y_3)$, onde

$$x_3 = \left({x_1}^2 + 1\right)^2 = x^4 + 1, \tag{3.9}$$

já que termos com coeficiente 2 são nulos na aritmética de corpo binário,

$$y_3 = \left({x_1}^2 + 1\right)(x_1 + x_3) + y_1 + 1. \tag{3.10}$$

3.7 Proposta de método de multiplicação escalar para curvas supersingulares

Sejam os pontos de curva elíptica supersingular em coordenadas afins $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$ e $P_3 = (x_3, y_3) \in E(\mathbb{F}_{2^m})$, tal que $P_1 + P_2 = P_3$. As coordenadas x_3 e y_3 podem ser calculadas usando as Equações 3.7 e 3.8.

Considerando um outro ponto $P_4 = (x_4, y_4)$ pertencente à mesma curva supersingular $E(\mathbb{F}_{2^m})$, o qual é resultado da subtração $P_2 - P_1$, tal que $P_4 = P_2 + (-P_1)$. Essa subtração pode ser vista como a adição do ponto P_2 e o negativo de P_1, ou seja, $(x_4, y_4) = (x_2, y_2) + (x_1, y_1 + 1)$, e então tem-se que a coordenada x_4 é dada pela Equação 3.11.

$$x_4 = (\lambda')^2 + x_1 + x_2, \tag{3.11}$$

onde $\lambda' = \left(\frac{y_1+1+y_2}{x_1+x_2}\right)$.

Pode-se notar que $\lambda' = \lambda + \frac{1}{x_1+x_2}$. Dessa forma, é possível reescrever a Equação 3.11 da seguinte maneira:

$$x_4 = \lambda^2 + \frac{1}{x_1^2 + x_2^2} + x_1 + x_2. \tag{3.12}$$

Empregando a Equação 3.7 na Equação 3.12, obtém-se:

$$x_4 = x_3 + \frac{1}{x_1^2 + x_2^2}, \tag{3.13}$$

que depende somente das coordenadas x e que pode ser reescrita como

$$x_3 = x_4 + \frac{1}{x_1^2 + x_2^2}. \tag{3.14}$$

Agora considerando o Algoritmo 3.2, pode-se perceber que, em cada iteração, sempre é executada uma adição e uma duplicação de pontos da curva elíptica. Como mostrado na Equação 3.9, a coordenada x do ponto que está sendo duplicado pode ser calculada usando apenas a coordenada x desse ponto antes da operação de duplicação. A Equação 3.14, por sua vez, mostra que a coordenada x do ponto resultante da adição dos pontos P_1 e P_2 é função de x_1, x_2 e x_4. Como x_4 é coordenada do ponto $P_4 = P_2 - P_1$, e pelo Lema 1, $P_4 = P_0 = (x_0, y_0)$, o que significa que $x_4 = x_0$. A Equação 3.14 pode, portanto, ser reescrita como

$$x_3 = x_0 + \frac{1}{x_1^2 + x_2^2}. \tag{3.15}$$

Note que a Equação 3.15 depende apenas das coordenadas x dos pontos P_0, P_1 e P_2. Dessa forma, o Algoritmo 3.2 pode facilmente ser adaptado para calcular a multiplicação escalar do ponto P_0 usando apenas as coordenadas x desse ponto e dos pontos auxiliares P_1 e P_2, conforme será descrito nas próximas equações. Essa característica pode ser vantajosa para dispositivos com pouca memória, como cartões inteligentes.

Considerando os pontos $P_5 = sP_0 = (x_5, y_5)$ e $P_6 = (k+1)P_0 = (x_6, y_6)$, ao final da execução do Algoritmo 3.2, pode-se esperar que $x_1 = x_5$ e $x_2 = x_6$. As próximas equações mostram como obter a coordenada y_5.

Sabe-se que $P_6 = (k+1)P_0 = kP_0 + P_0 = P_5 + P_0$. Da Equação 3.7, tem-se que:

$$x_6 = \left(\frac{y_5 + y_0}{x_5 + x_0}\right)^2 + x_5 + x_0 \tag{3.16}$$

$$= \frac{y_5^2 + y_0^2 + x_5^3 + x_0^3 + x_5^2 x_0 + x_5 x_0^2}{x_5^2 + x_0^2}. \tag{3.17}$$

Como os pontos kP e P pertencem ambos à curva supersingular dada pela Equação 3.1, pode-se afirmar que:

$$\begin{cases} & y_0^2 + y_0 = x_0^3 + x_0 + c \\ + & y_5^2 + y_5 = x_5^3 + x_5 + c \\ \hline & y_5^2 + y_P^2 + x_5^3 + x_P^3 = y_P + y_5 + x_P + x_5. \end{cases}$$

Pode-se empregar o resultado anterior na Equação 3.17, resultando em

$$x_6 = \frac{y_0 + y_5 + x_0 + x_5 + x_5^2 x_0 + x_5 x_0^2}{x_5^2 + x_0^2}, \tag{3.18}$$

que pode ser reescrita como

$$y_5 = y_0 + (x_5 + x_0)(1 + x_0 x_5 + x_6(x_5 + x_0)). \tag{3.19}$$

A Equação 3.19 mostra que y_5 pode ser calculado usando apenas a coordenada y do ponto P_0 e as coordenadas x obtidas ao final da execução do Algoritmo 3.2.

Com base nas equações anteriores, é possível adaptar o Algoritmo 3.2 e criar um novo algoritmo para calcular a multiplicação escalar de um ponto da curva elíptica supersingular num corpo $\mathbb{F}_{2^m}$, como mostrado no Algoritmo 3.3.

Algoritmo 3.3 Multiplicação escalar em curva supersingular usando coordenadas afins.

Entrada: Ponto $P_0 = (x_0, y_0) \in E(\mathbb{F}_{2^m})$, escalar positivo $s = (s_{n-1}, s_{n-2}, \ldots, s_0)_2$.
Saída: Ponto $sP_0 = R = (x_R, y_R) \in E(\mathbb{F}_{2^m})$.

1: $x_R \leftarrow x_0$, $x_S \leftarrow x_0^4 + 1$.
2: **Para** $i \leftarrow n-2$ **até** 0 **do faça**
3: **Se** $s_i = 0$ **então**
4: $x_S \leftarrow x_0 + \frac{1}{x_R^2 + x_S^2}$.
5: $x_R \leftarrow x_R^4 + 1$.
6: **Senão**
7: $x_R \leftarrow x_0 + \frac{1}{x_R^2 + x_S^2}$.
8: $x_S \leftarrow x_S^4 + 1$.
9: $y_R \leftarrow y_0 + (x_R + x_0)(1 + x_0 x_R + x_S(x_R + x_0))$.
10: **Retorne** (x_R, y_R).

O Algoritmo 3.3 será comparado aos anteriores na Seção 3.9.

3.8 Versão com coordenadas projetivas

Nesta seção será mostrado como o método de multiplicação escalar da Seção 3.7 pode ser adaptado para usar coordenadas projetivas.

Considerando que as operações de adição e quadrado de elementos do corpo finito podem ser eficientemente computadas em corpos binários [14], pode-se perceber nas equações mostradas na Seção 3.6 que a multiplicação e a inversão são as operações mais caras do ponto de vista computacional quando se deseja computar a adição ou a duplicação de pontos da curva elípticas.

É possível eliminar a operação de inversão durante o cálculo da adição de pontos usando o sistema de coordenadas projetivas. Sejam os pontos $P_1 = (X_1 : Y_1 : Z_1)$, onde $Z_1 = 1$ e $P_2 = (X_2 : Y_2 : Z_2) \in E(\mathbb{F}_{2^m})$ e suponha que $P_1, P_2 \neq \mathcal{O}$, $P_1 \neq P_2$ e $P_1 \neq -P_2$. Então $P_3 = P_1 + P_2 = (X_3 : Y_3 : Z_3)$, tal que:

$$X_3 = \alpha^2\beta Z_2 + \beta^4 \tag{3.20}$$

$$Y_3 = (1 + Y_1)Z_3 + \alpha^3 Z_2 + \alpha\beta^2 X_2 \tag{3.21}$$

$$Z_3 = \beta^3 Z_2, \tag{3.22}$$

onde $\alpha = (Y_1 Z_2 + Y_2)$ e $\beta = (X_1 Z_2 + X_2)$. É possível observar que as Equações 3.20, 3.21 e 3.22 demandam 9 multiplicações de elementos do corpo binário. Para duplicar um ponto, são necessárias 6 multiplicações. Quando se empregam coordenadas projetivas, o resultado $(X_3 : Y_3 : Z_3)$ pode ser facilmente convertido para coordenadas afins multiplicando cada coordenada por Z_3^{-1}.

Equações 3.9 e 3.14 podem ser modificadas para utilizar coordenadas projetivas.

- **Duplicação**: da Equação 3.9,

$$\frac{X_3}{Z_3} = \frac{X_1^4}{Z_1^4} + 1 \tag{3.23}$$

$$= \frac{X_1^4 + Z_1^4}{Z_1^4}, \tag{3.24}$$

  o que significa que $X_3 = (X_1^4 + Z_1^4)$ e $Z_3 = Z_1^4$.

- **Adição**: da Equação 3.15,

$$\frac{X_3}{Z_3} = x_0 + \frac{1}{\frac{X_1^2}{Z_1^2} + \frac{X_2^2}{Z_2^2}} \tag{3.25}$$

$$= x_0 + \frac{Z_1^2 Z_2^2}{X_1^2 Z_2^2 + X_2^2 Z_1^2} \tag{3.26}$$

$$= \frac{x_0 t + Z_1^2 Z_2^2}{t}, \tag{3.27}$$

  onde $t = X_1^2 Z_2^2 + X_2^2 Z_1^2$. Dessa forma, tem-se

$$X_3 = \left(x_0 t + Z_1^2 Z_2^2\right), \tag{3.28}$$

$$Z_3 = t. \tag{3.29}$$

Assim como foi desenvolvida a Equação 3.19 para obter a coordenada y do ponto $kP_0 = P_5 =$

(x_5, y_5) para coordenadas afins, também é possível obter y_5 diretamente no caso das coordenadas projetivas. Ao final das iterações do algoritmo de multiplicação escalar pode-se fazer $x_5 = X_1/Z_1$, $x_6 = X_2/Z_2$ e usar essas relações para adaptar a Equação 3.19 e obter 3.30.

$$y_5 = y_0 + \frac{(X_1 + x_0 Z_1)(x_0(X_1 Z_2 + X_2 Z_1) + X_1 X_2 + Z_1 Z_2)}{Z_1^2 Z_2} \tag{3.30}$$

O Algoritmo 3.4 mostra como implementar a multiplicação do ponto P_0 pelo escalar s empregando coordenadas projetivas.

Algoritmo 3.4 Multiplicação escalar em curva supersingular usando coordenadas projetivas.

Entrada: Ponto $P_0 = (x_0, y_0) \in E(\mathbb{F}_{2^m})$, escalar positivo $s = (s_{n-1}, s_{n-2}, \ldots, s_0)_2$.
Saída: Ponto $sP_0 = R = (x_R, y_R) \in E(\mathbb{F}_{2^m})$.
1: $X_R \leftarrow x_0$, $Z_R \leftarrow 1$, $Z_S \leftarrow Z_R^4$, $X_S \leftarrow X_R^4 + Z_S$.
2: **Para** $i \leftarrow n-2$ **até** 0 **do faça**
3: $t \leftarrow (X_R Z_S + X_S Z_R)^2$.
4: **Se** $s_i = 0$ **então**
5: $X_R \leftarrow x_0 t + Z_R^2 Z_S^2$, $Z_R \leftarrow t$,
6: $Z_S \leftarrow Z_S^4$, $X_2 \leftarrow X_S^4 + Z_S$.
7: **Senão**
8: $X_S \leftarrow x_0 t + Z_R^2 Z_2^2$, $Z_2 \leftarrow t$,
9: $Z_R \leftarrow Z_R^4$, $X_R \leftarrow X_R^4 + Z_R$.
10: $x_R = X_R/Z_R$
11: $y_5 = y_0 + \frac{(X_R + x_0 Z_R)(x_0(X_R Z_S + X_S Z_R) + X_R X_S + Z_R Z_S)}{Z_R^2 Z_S}$.
12: **Retorne** (x_R, y_R).

Pode-se perceber no Algoritmo 3.4 que em uma das multiplicações efetuadas em cada iteração há sempre o fator x_0. Essa observação pode levar a implementações eficientes usando técnicas de pré-computação, como mostradas em [29]. Lembra-se que a pré-computação não é necessária para empregar o Algoritmo 3.4.

3.9 Comparação com *Double-and-Add*

As operações executadas em cada iteração dos Algoritmos 3.3 (linhas 4, 5 or 7, 8) e 3.4 (linhas 5, 6 or 8, 9) são independentes e podem ser paralelizadas. Dessa forma, cada iteração pode ser executada de forma mais rápida, levando a um menor tempo de execução do algoritmo de multiplicação escalar. A Figura 3.1 ilustra como seria essa paralelização parcial durante as iterações do Algoritmo 3.3. Uma estratégia análoga poderia ser empregada no Algoritmo 3.4.

Tanto o método proposto aqui quanto o *Double-and-Add* são métodos para multiplicação escalar

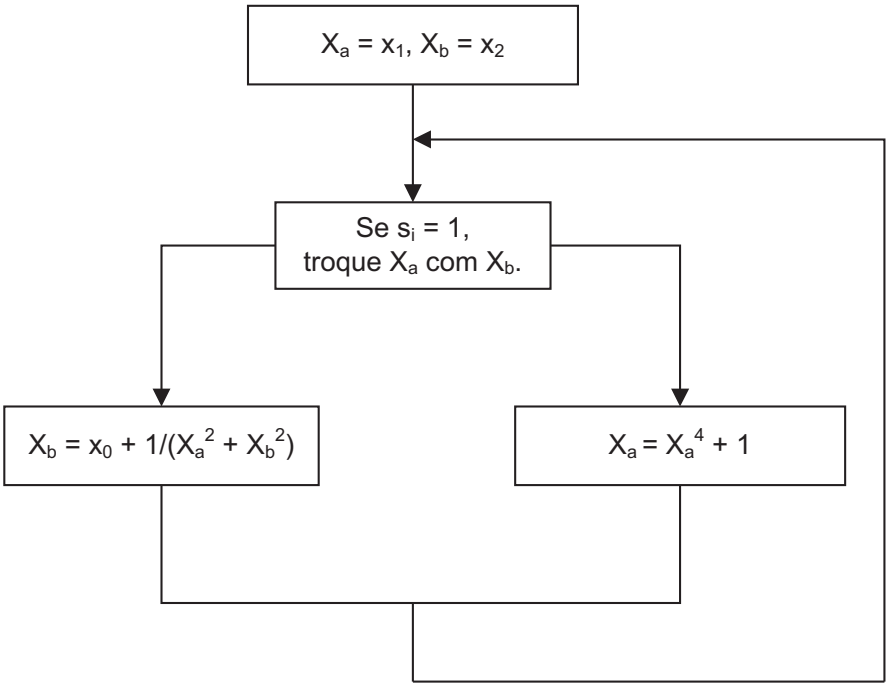


Fig. 3.1: Paralelização de operações durante iterações do Algoritmo 3.3.

que não exigem pré-computação. Para comparar esses dois métodos, será considerado o escalar s com $n = \lceil \log_2 s \rceil$.

No método *Double-and-Add*, a duplicação de pontos é realizada em todas as iterações e se $k_i = 1$, é necessário calcular uma adição de pontos, como mostrado na Seção 3.3. No método proposto, cada iteração do algoritmo requer o mesmo número de operações, independentemente do valor de s_i. Ao final da execução do algoritmo, é preciso efetuar algumas outras operações caso de deseje saber o valor da coordenada y.

Sabe-se que as operações de adição na aritmética de corpo binário apresentam um custo computacional baixo, já que elas dependem apenas do número de palavras de máquina que representam o elemento do corpo binário [14]. De forma semelhante, a operação de quadrado no corpo binário também tem um baixo custo computacional e pode usar tabelas. Considerando então que as operações de quadrado e adição no corpo binário têm um custo computacional baixo quando comparadas as operações de multiplicação (M) e inversão (I), apenas essas duas últimas serão consideradas em nossa comparação. Também será considerado que a representação binária de s tem um peso de Hamming de 50%, na média.

As Tabelas 3.1 e Tab. 3.2 mostram a comparação do método proposto com o método *Double-and-Add* usando coordenadas afins e projetivas, respectivamente.

Considerando o tempo gasto para calcular a multiplicação escalar de um ponto, nota-se na Tabela

Tab. 3.1: Comparação entre os métodos usando coordenadas afins.

Método	Operações necessárias
Double-and-Add	$(3n/2)$M + $(n/2)$I
Método proposto	3M + $(n-1)$I

3.1 que, para valores grandes de n, o método proposto é mais rápido que o tradicional *Double-and-Add* se I < 3 M, ou seja, quando a operação de inversão I tem um custo computacional menor que três vezes o custo da operação de multiplicação M, o que é difícil de encontrar em implementações práticas [14].

Tab. 3.2: Comparação entre os métodos usando coordenadas projetivas.

Método	Operações necessárias
Double-and-Add	$(21n/2)$M + 1I
Método proposto	$4(n+1)$M + 2I

Na Tabela 3.2, pode-se considerar valores de n grandes o suficiente para ignorar o custo dos termos que não são multiplicados por n, de tal forma que o método proposto é mais rápido que o método *Double-and-Add* todas as vezes em que $(13n/2)\text{M} > 1\text{I}$, o que é sempre verdade para situações práticas, já que n assume valores grandes (tipicamente, $n > 100$). Nas comparações apresentadas na Tabela 3.2, também foi considerado que o algoritmo de inversão inclui a multiplicação do numerador pelo valor invertido do denominador. Se essa suposição não for atendida, deve-se acrescentar o custo de uma multiplicação M para cada inversão I.

A Figura 3.2 ilustra o crescimento do número de operações executadas numa multiplicação escalar com o aumento do número de bits n do escalar, para os dois casos apresentados na Tabela 3.2.

Pode-se notar na Figura 3.2 que as vantagens do método de multiplicação escalar baseado no algoritmo *Montgomery's Ladder* em relação ao *Double-and-Add* ficam mais evidentes à medida que o tamanho dos escalares aumenta.

É importante observar que, enquanto o custo do método *Double-and-Add* depende do peso de Hamming do escalar s, o método proposto requer o mesmo número de operações para todos os valores de s.

Os algoritmos 3.3 e 3.4 foram ambos implementados, testados e seus resultados verificados para validar as equações apresentadas neste trabalho.

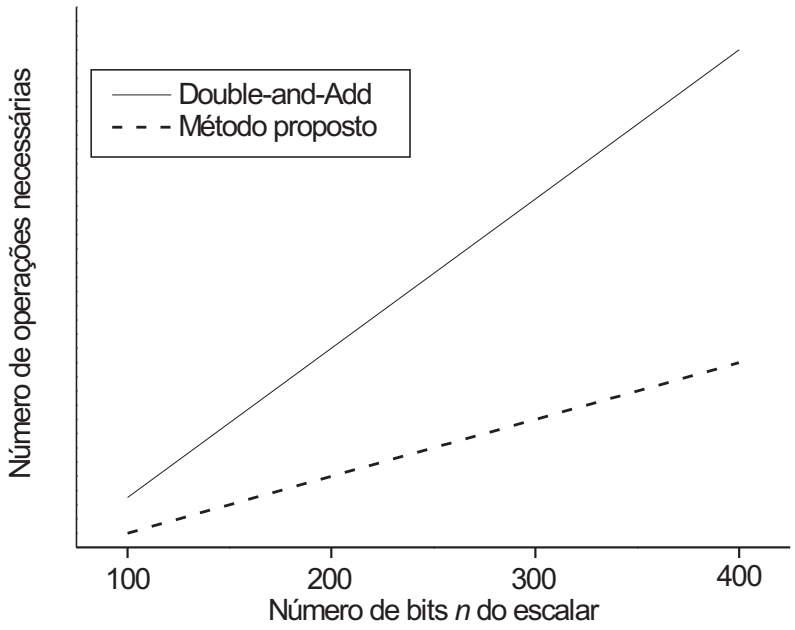


Fig. 3.2: Crescimento do número de operações da multiplicação escalar com o aumento de n para o caso de coordenadas projetivas.

3.10 Comparação com os métodos *NAF* e *w-NAF*

É possível também comparar o método proposto com outros métodos para multiplicação escalar, como o *NAF* e o *w-NAF*. Primeiramente, serão considerados esses métodos com coordenadas afins, conforme apresentado na Tabela 3.3. Pode-se perceber que, quando são empregadas coordenadas

Tab. 3.3: Comparação entre os métodos usando coordenadas afins.

Método	**Operações necessárias**
NAF	$(4n/3)$M + $(n/3)$I
w-NAF	$(6n/5+4)$M + $(n/5+3)$I
Método proposto	3M + $(n-1)$I

afins, o método proposto somente é mais vantajoso que os métodos *NAF* e *w-NAF* quando I < 2M ou I < $(3/2)$M, respectivamente, o que é difícil de ocorrer em situações reais.

Para coordenadas projetivas, o custo esperado para o cálculo da multiplicação escalar encontra-se na Tabela 3.4. Adotou-se para o método *w-NAF* uma janela de largura $w = 4$, que pode ser considerado um tamanho adequado para o caso de uma implementação em cartões inteligentes de 8 bits.

Tab. 3.4: Comparação entre os métodos usando coordenadas projetivas.

Método	Operações necessárias
NAF	$9n$M + 1I
w-NAF	$(39n/5 + 33)$M + 1I
Método proposto	$4(n + 1)$M + 2I

Como é possível perceber na Tabela 3.4, quando são empregadas coordenadas projetivas, o custo do método proposto é menor que os métodos *NAF* e *w-NAF* para $5n$M > I e $(19n/5)$M > 1I, respectivamente, o que é sempre verdadeiro para situações práticas.

Existe ainda a possibilidade de comparação com outros métodos, como o *fixed-base comb* [14], mas nosso objetivo é considerar os casos mais utilizados na prática.

3.11 Outras otimizações

Conforme apresentado na Seção 3.6, a duplicação de um ponto de curva supersingular é uma operação relativamente barata do ponto de vista computacional. Dessa forma, pode-se escrever o escalar s como:

$$s = s_1 + 2^{(n/2)} s_2, \tag{3.31}$$

onde s_1 e s_2 têm ambos o número de bits igual a $n/2$. Desse modo, o cálculo de sP_0 pode ser efetuado como duas multiplicações escalares paralelas:

$$sP_0 = s_1 P_0 + s_2 Q, \tag{3.32}$$

onde $Q = 2^{(n/2)} P_0$.

Mesmo nessa nova forma de efetuar o cálculo da multiplicação escalar, as equações mostradas aqui podem ser aplicadas, especialmente no caso de serem empregadas as coordenadas projetivas, aproveitando assim a eficiência e a robustez contra ataques do tipo *side-channel*.

3.12 Conclusões do capítulo

Este capítulo apresentou um método eficiente e seguro de multiplicação escalar em curvas supersingulares sobre corpos binários baseado no algoritmo *Montgomery's Ladder*. Em coordenadas afins, a eficiência do método em comparação com outros métodos de multiplicação escalar, como o tradicional *Double-and-Add*, depende do custo relativo das operações de inversão e multiplicação no corpo binário, e normalmente é inviável na prática. Já em coordenadas projetivas, e levando em

conta situações práticas, o método proposto é mais rápido que os demais métodos considerados. A adaptação do método *Montgomery's Ladder* para multiplicação escalar de curvas supersingulares apresenta como características a robustez contra ataques do tipo *side-channel* e a possibilidade de ser parcialmente paralelizado. Essas são características importantes, considerando a utilização do método em aplicações de segurança. O método não exige pré-computação e usa durante suas iterações apenas a coordenada x do ponto sendo multiplicado, o que se torna uma vantagem para dispositivos com pouca quantidade de memória, como cartões inteligentes.

Capítulo 4

Implementação de emparelhamentos bilineares em cartões inteligentes de oito bits

Conforme descrito no Capítulo 1, um dos objetivos deste trabalho é apresentar os resultados da implementação de emparelhamentos bilineares em software em dois tipos de cartões inteligentes comercialmente disponíveis e de arquitetura de 8 bits: um Java Card um cartão MULTOS.

A implementação dos emparelhamentos bilineares nesses cartões expande a possibilidade de difundir e aproveitar de forma mais ampla as vantagens dessas funções matemáticas em protocolos criptográficos inovadores, como os sistemas baseados em identidade (*Identity Based Cryptography* – IBC), que serão detalhados neste capítulo.

Não é de nosso conhecimento qualquer resultado de implementação de emparelhamentos bilineares ou de protocolos baseados em emparelhamentos utilizando cartões inteligentes de 8 bits.

4.1 Cartões inteligentes

Na década de 50, nos Estados Unidos, foi iniciado o uso de cartões de plástico para efetuar pagamentos, no lugar do dinheiro. Inicialmente, essa prática era restrita a um pequeno público privilegiado, mas foi se popularizando com o tempo e trazendo com isso problemas de segurança. Os cartões de plástico continham apenas o nome e, em alguns casos, a assinatura do usuário. Dessa forma, cartões podiam ser facilmente fraudados e ladrões podiam usar sem grandes problemas os cartões roubados.

Uma maneira de incrementar a segurança dos primeiros cartões foi através das tarjas magnéticas, que armazenavam dados dos usuários e que podiam ser lidas pelos terminais de compras. No entanto, a segurança introduzida pela tarjas é bastante frágil, haja vista a facilidade com que elas podem ser alteradas.

Os avanços na microeletrônica permitiram incorporar armazenamento e processamento em um único *chip*, em uma área de poucos milímetros quadrados, criando os chamados cartões inteligentes.

Um cartão inteligente consiste em um dispositivo do tamanho de um cartão de crédito comum, que contém um ou mais circuitos integrados e que pode empregar uma ou mais das seguintes tecnologias: tarja magnética, código de barras (linear ou bi-dimensional), comunicação por rádio, biometria, cifragem e autenticação, além de identificação visual por foto (bi ou tri-dimensional) [63]. O *chip* de circuito integrado do cartão inteligente pode atuar como um microcontrolador. Os dados armazenados na memória do *chip* podem ser acessados por diferentes aplicações. A memória também armazena o sistema operacional do *chip*, o qual também pode oferecer algoritmos criptográficos implementados em sua memória ou em coprocessadores para prover segurança aos dados armazenados. Quando usado em conjunto com aplicações apropriadas, os cartões inteligentes podem prover um elevado nível de segurança, bem como interoperabilidade entre diferentes serviços e aplicações.

Os cartões inteligentes de contato são aqueles que necessitam de um dispositivo leitor para receberem alimentação elétrica, bem como receber e transmitir dados. Existem também os cartões sem contato, que comunicam-se por ondas de rádio de curto alcance, entre os quais o tipo mais conhecido é o Mifare [63]. Os primeiros cartões sem contato apresentaram falhas graves de segurança [64], que foram corrigidas nos modelos mais novos. Existem ainda os cartões que incorporam ambas interfaces com e sem contato.

É possível encontrar cartões inteligentes empregados em sistemas de bancos, universidades, empresas privadas, transporte público e ainda em celulares. Seu uso está bastante difundido graças à sua capacidade de armazenamento e de processamento, aliada a uma forte segurança contra violações e fraudes.

4.1.1 Alguns benefícios dos cartões inteligentes

Os cartões inteligentes que possuem funcionalidades criptográficas podem proporcionar alguns benefícios para seus usuários, como os listados a seguir.

- As chaves criptográficas são armazenadas em um hardware seguro e resistente a diversos tipos de ataques, ao contrário de outras mídias menos seguras de armazenamentos de dados, como os discos rígidos dos computadores pessoais. A maior proteção às chaves proporcionada pelos cartões inteligentes facilita a implementação de vários serviços de segurança.

- Ao contrário do que ocorre num computador pessoal, as operações criptográficas executadas num cartão inteligente são protegidas do sistema operacional e, portanto, não são susceptíveis a ataques como *buffer overflow* e *dump* de memória, os quais podem revelar o conteúdo das chaves criptográficas ou outras informações sigilosas.

- O mesmo cartão inteligente que armazena chaves e executa operações criptográficas pode armazenar também outras aplicações e funcionar como moedeiro eletrônico, por exemplo.

- Um cartão inteligente pode ser transportado com facilidade pelo usuário, o que confere mobilidade e facilidade de empregar funções criptográficas sempre que for necessário.

4.1.2 *Personal Identification Number* (PIN)

As informações contidas num cartão inteligente são protegidas pelo *Personal Identification Number* (PIN), que consiste numa sequência de dígitos conhecidos apenas pelo dono do cartão inteligente. Somente após a inserção correta do PIN, o cartão fica liberado para fornecer informações ou executar programas. Note, portanto, que as chaves criptográficas armazenadas no cartão somente podem ser utilizadas por quem tem a posse do cartão e sabe o seu PIN.

O uso de PIN oferece uma proteção maior do que o uso de senhas ou suas derivações, como *hashes*. As senhas estão sujeitas a ataques do tipo dicionário, quando um atacante tenta algumas palavras conhecidas de um dicionário ou outras senhas comumente utilizadas na tentativa de invadir o conteúdo do cartão. Como muitos usuários preferem usar senhas que podem ser facilmente lembradas, os ataques do tipo dicionário têm boas chances de serem bem sucedidos. Dessa forma, a força de uma senha depende de seu tamanho, de como ela é protegida pelo usuário e como ela é difícil de ser adivinhada. Vale lembrar que mesmo boas senhas, com todas essas características, podem estar sujeitas a ataques do tipo força bruta.

Nesse sentido, o uso de PIN apresenta a vantagem de que ataques de força bruta só podem ser tentados por alguém que tenha a posse do cartão inteligente. Além disso, mesmo que um atacante tenha o cartão em suas mãos, este pode ser configurado para travar após algumas (geralmente três) tentativas sem sucesso de adivinhar o PIN.

Como o PIN geralmente possui quatro a oito dígitos, fica mais fácil para o usuário guardar sua composição. Isso é mais difícil de acontecer para boas senhas, que em geral são longas e com uma composição complexa, envolvendo letras maiúsculas, minúsculas, dígitos e sinais de pontuação.

4.1.3 Estrutura de hardware

O *chip* de um cartão inteligente possui componentes típicos de um microcontrolador, que permitem armazenar e processar dados. Tais componentes, ilustrados na Figura 4.1, são:

- **ROM**: a máscara da memória ROM contém o sistema operacional, que é escrito quando o *chip* é fabricado. O conteúdo da memória ROM é, portanto, o mesmo para todos os *chips* produzidos numa leva de produção e não pode ser alterado durante o ciclo de vida do *chip*;

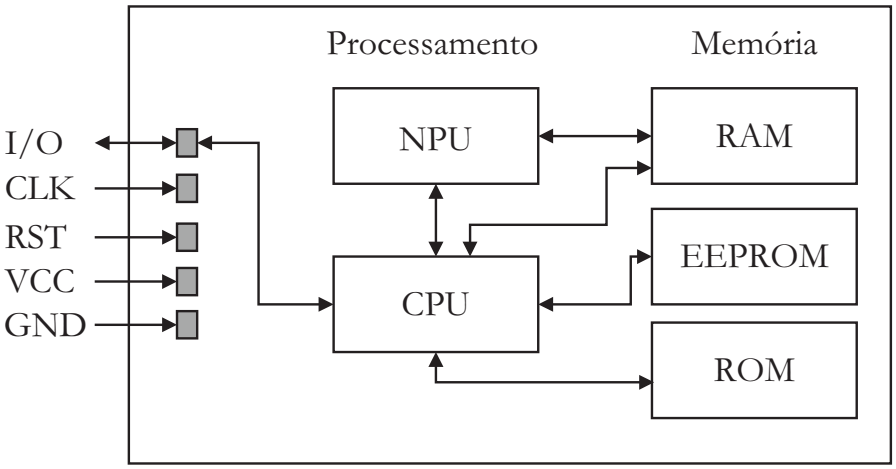


Fig. 4.1: Componentes de hardware de um *chip* de cartão inteligente.

- **EEPROM ou flash**: memória não-volátil que armazena dados (identificação do dono, chaves criptográficas) e programas (gravação e recuperação de informações, geração e verificação de assinaturas digitais, armazenamento de certificados digitais). A leitura ou escrita da memória EEPROM é controlada pelo sistema operacional do *chip*;

- **RAM**: memória de trabalho temporária usada pelo processador para execução de atividades do sistema operacional e dos programas. Como se trata de uma memória volátil, todo seu conteúdo é perdido quando o cartão deixa de ser energizado;

- **CPU**: processador responsável por executar as atividades do sistema operacional e os aplicativos. Os *chips* de cartões inteligentes normalmente não dão suporte a *threads* e, portanto, somente um programa pode ser executado por vez. Maiores detalhes sobre os processadores para cartões inteligentes encontram-se na Seção 4.1.4;

- **NPU**: coprocessador, que consiste em um processador dedicado para efetuar operações matemáticas, a fim de tornar mais rápidos os algoritmos de criptografia. Detalhes sobre os coprocessadores em cartões inteligentes encontram-se na Seção 4.1.5.

4.1.4 Processadores

Os processadores usados em cartões inteligentes costumam usar como referência processadores de propósito geral empregados com sucesso e por um certo tempo em dispositivos diversos e que se mostraram confiáveis na prática. Normalmente, não são desenvolvidos novos processadores específicos para cartões inteligentes devido ao custo e à conveniência, uma vez que a adoção de um processador já conhecido permite o aproveitamento de bibliotecas, compiladores e ferramentas de

desenvolvimento. Dessa forma, cartões inteligentes tendem a não utilizar o estado da arte em termos de processadores.

Muitos cartões inteligentes encontrados em aplicações de bancos e universidades, por exemplo, têm uma arquitetura de 8 bits e baseiam-se no conjunto de instruções do Motorola 6805 ou Intel 8051 [63]. Esses são processadores CISC (*complex-instruction-set computer*), o que significa que eles oferecem um conjunto de instruções vasto e utilizam vários ciclos de relógio para executar as instruções de máquina. Os processadores de cartões inteligentes da família de 8 bits normalmente possuem extensões que possibilitam endereçar regiões de memória de até 128 KB. Apesar dessa estratégia trazer a vantagem de permitir o armazenamento maior de informações no cartão, o acesso não-linear às áreas de memória implica que a área de código e de dados fica distribuída em vários bancos de memória, o que exige que o processador tenha um trabalho adicional para acessar essas áreas.

No nível mais alto de desempenho dos microcontroladores de cartões inteligentes encontram-se os processadores de 32 bits. Esses processadores oferecem um poder de processamento compatível com as necessidades de aplicações mais complexas, como operações criptográficas. Tais vantagens precisam ser balanceadas com o fato de que esses processadores consomem mais energia e ocupam uma área maior do *chip*. Existe ainda o fator econômico, pois os processadores de 8 bits ainda atendem muitas das demandas relacionadas a cartões inteligentes com *chips* de custo bem inferior aos de 32 bits.

4.1.5 Coprocessadores

Os coprocessadores são unidades aritméticas inseridas no *chip* de cartões inteligentes e que são especialmente desenvolvidas com o intuito de efetuar operações criptográficas, principalmente as de algoritmos de criptografia de chaves públicas, como o RSA e as de algoritmos de curvas elípticas. Essas unidades aritméticas normalmente são capazes de executar as operações básicas necessárias para esses algoritmos, como multiplicação modular e exponenciação modular utilizando números grandes. A velocidade desses componentes deve-se bastante a registradores grandes empregados em sua arquitetura (de até 140 bits [63]). Dessa forma, alguns coprocessadores conseguem atingir um desempenho equivalente ou até superior em comparação com um computador pessoal.

O coprocessador é invocado pelo processador, que pode passar os parâmetros de entrada diretamente em registradores ou então enviar ponteiros para áreas de memória onde se encontram tais parâmetros. O coprocessador executa as operações solicitadas, armazena o resultado obtido na memória RAM e, ao final, o controle do programa volta para o processador. Os coprocessadores mais comuns encontrados nos cartões inteligentes atuais são aqueles que executam operações básicas dos algoritmos DES, 3DES, AES, na criptografia simétrica e RSA, ECDSA e ECDH, na criptografia de

chaves públicas, sendo que os dois últimos ainda não são facilmente encontrados em cartões inteligentes comercialmente disponíveis. É comum encontrar *chips* que suportam chaves RSA de até 1024 bits e os mais modernos já suportam chaves de 2048 bits.

4.1.6 Estrutura lógica

A estrutura mostrada na Figura 4.1 permite que os cartões inteligentes sejam utilizados de forma bem flexível. Normalmente, os cartões inteligentes modernos permitem que diversas aplicações sejam armazenadas em um único cartão. Dessa forma, a memória ROM armazena apenas os componentes mais básicos do sistema operacional e os demais componentes são armazenados na memória EEPROM após a fabricação do cartão. Isso ocorre com os cartões Java Card e MULTOS, por exemplo. Tais cartões possuem uma máquina virtual que abstrai o tipo de hardware utilizado e que é incorporada no cartão após sua fabricação.

A Figura 4.2 ilustra a estrutura lógica encontrada em grande parte dos cartões inteligentes modernos e é detalhada a seguir.

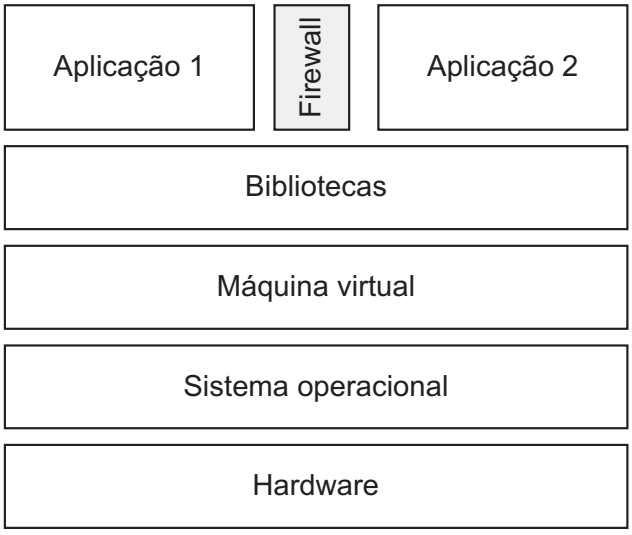


Fig. 4.2: Arquitetura de um cartão inteligente com suporte a múltiplas aplicações.

- **Hardware**: essa camada se refere à plataforma física descrita na Figura 4.1, que armazena os dados e que executa todas instruções do sistema operacional e dos programas carregados no cartão.

- **Sistema Operacional**: realiza o gerenciamento de recursos do cartão inteligente, como memória, processadores, coprocessadores criptográficos e recursos de entrada/saída. O sistema

operacional também é responsável por controlar a adição, execução e remoção de programas no cartão. Informações mais detalhadas sobre os sistemas operacionais Java Card e MULTOS encontram-se na Seção 4.1.7.

- **Máquina virtual**: essa camada proporciona a abstração do hardware, de modo que os programas para cartões inteligentes podem ser desenvolvidos independentemente do *chip* no qual serão executados. Dessa forma, um programa desenvolvido e testado em um *chip* de um fabricante pode funcionar sem problemas no *chip* de um outro fabricante, desde que ambos utilizem a mesma máquina virtual. Essa característica é apontada como uma vantagem, pois uma aplicação para cartões inteligentes não fica atrelada a um fabricante específico. Por outro lado, como a máquina virtual deve interpretar todas as instruções, ela acaba introduzindo uma sobrecarga na execução dos programas. Num Java Card, essa camada é conhecida como Máquina Virtual Java Card e é responsável por executar os chamados *bytecodes* Java Card. Nos cartões MULTOS, existe a *Application Abstract Machine* (AAM), que interpreta a MULTOS Executable Language (MEL), um tipo de linguagem *Assembly*.

- ***Application Programming Interface* (API)**: os cartões inteligentes podem ser programados usando uma linguagem de alto nível. A programação dos Java Cards utiliza um subconjunto da API Java, disponibilizado em bibliotecas pela empresa Sun no pacote *Java Card Development Kit* (JCDK). Trata-se de um subconjunto bem restrito da linguagem Java, que não inclui *threads*, *strings* e matrizes multi-dimensionais, por exemplo. A mais nova especificação do Java Card, a versão 3.0 [65], não impõe tantas restrições à programação dos cartões inteligentes. No entanto, os fabricantes ainda não disponibilizaram *chips* compatíveis com essa nova especificação. Os programas para os cartões MULTOS podem ser desenvolvidos nas linguagens C, Java e MEL, utilizando as bibliotecas específicas e compiladores disponibilizados em um sítio mantido por diversos fabricantes de cartões inteligentes [66].

- **Aplicações e *Firewall***: múltiplas aplicações podem residir num mesmo cartão inteligente e todas são protegidas por *firewalls*. Cada programa residente no *chip* possui uma área de memória muito bem definida para abrigar os segmentos de código e dados. Um programa tem acesso total a suas áreas de código e dados, mas não pode acessar diretamente as áreas de um outro programa. Caso um programa tente acessar uma área de memória fora de sua área reservada, o sistema operacional força o fim do processamento. Isso garante que uma aplicação não consiga acessar ou modificar dados que pertençam a outras aplicações residentes no cartão inteligente.

4.1.7 Sistemas operacionais para cartões inteligentes

De maneira geral, um sistema operacional é responsável por gerenciar os recursos de uma máquina e proporcionar aos programas do usuário uma interface para o hardware dessa máquina. Dessa maneira, não se faz necessário que um programa tenha que endereçar diretamente o hardware. Essa característica pode ser vista como um benefício, pois um programa adquire uma certa portabilidade. Foi almejando essa característica que foram desenvolvidos os sistemas operacionais para cartões inteligentes.

Os primeiros sistemas operacionais para cartões inteligentes surgiram na década de 90 e desde então foram se aprimorando [63], consolidando o termo COS (*card operating system*) para os sistemas operacionais para cartões inteligentes que foram surgindo. Neste trabalho serão abordados os sistemas operacionais Java Card e MULTOS, devido à maior disponibilidade comercial dessas plataformas. Esses dois sistemas operacionais são classificados como plataforma aberta, pois permitem que programas sejam carregados no cartão sem o envolvimento dos fabricantes de *chips* de cartões inteligentes.

Java Card

Um dos COS mais difundidos é o Java Card. Assim como programas desenvolvidos para a linguagem Java, programas desenvolvidos para a plataforma Java Card são traduzidos em *bytecodes* Java por um compilador. Tais *bytecodes* consistem em código objeto independente de um processador real, como se fossem destinados a um processador virtual, que consiste na máquina virtual Java. Essa máquina virtual é responsável por interpretar os *bytecodes* e encaminhar o código nativo para o processador real. A vantagem vista nesse cenário é que apenas a máquina virtual precisa ser portada para um processador específico e, uma vez que isso tenha sido realizado, programas Java poderão ser executados independentemente da plataforma.

Mesmo sendo bastante utilizado, o sistema operacional Java Card recebe algumas críticas. Além de demandar muita memória dos cartões, ele também apresenta, em geral, baixo desempenho na execução dos programas. De acordo com o Smart Card Handbook [63], pode-se esperar que um programa Java seja de 2 a 3 vezes mais lento que um programa desenvolvido em código nativo de um cartão inteligente.

MULTOS

O Multi-application Operating System (MULTOS) guarda muitas semelhanças com o Java Card. Ele também faz uso de uma máquina virtual, que interpreta o código objeto denominado MEL (MULTOS Executable Language). Os programas para essa plataforma podem ser desenvolvidos usando

diretamente as instruções MEL ou então com as linguagens de alto nível C e Java.

Uma das características que tornam o MULTOS diferente dos demais COS é que ele implementa o chamado Secure Trusted Environment Provisioning (STEP). Trata-se de um mecanismo patenteado que permite que a entidade que distribui os cartões, como um banco, por exemplo, tenha controle total da personalização e das atualizações dos cartões MULTOS, mesmo que eles estejam em campo, ou seja, na posse dos usuários finais. Isso é possível pelo uso de certificados digitais, que são enviados para o cartão e, se considerados válidos, permitem o carregamento ou remoção de programas. Com esse esquema baseado em certificados, o carregamento e a remoção de programas num cartão MULTOS pode ocorrer de forma controlada, em qualquer ponto do seu ciclo de vida.

4.1.8 Padrões de cartões inteligentes

Um dos requisitos que viabilizaram o uso difundido dos cartões inteligentes em aplicações cotidianas foi a criação de padrões internacionais regendo diversos aspectos físicos e funcionais desses dispositivos.

Um cartão inteligente normalmente é um componente de um sistema grande e complexo, de modo que as interfaces entre o cartão e o restante do sistema devem ser especificadas de forma precisa para evitar problemas de incompatibilidade. Esse trabalho de definição de interfaces poderia ser feito a cada novo sistema, mas além de introduzir novos custos, o seu resultado poderia implicar que os usuários tivessem que portar um cartão específico para cada tipo de aplicação baseada em cartões inteligentes. Para evitar esse tipo de situação, foram criados padrões a fim de possibilitar o desenvolvimento de cartões inteligentes que pudessem atender tipos variados de aplicações.

ISO/IEC 7816

As características e funções fundamentais dos cartões inteligentes com contato são especificadas pela International Organization for Standardization (ISO) e pela International Electrotechnical Commission (IEC), através da família de padrões ISO/IEC 7816. Tais padrões especificam, por exemplo, as características físicas, a dimensão e localização dos contatos, a interface elétrica, os protocolos de transmissão e recebimento de dados, a organização dos dados no cartão, as permissões de acesso a esses dados, os métodos de segurança que devem ser suportados e os comandos para troca de informações com o ambiente externo. Na família de padrões ISO/IEC 7816, um dos mais importantes é o ISO/IEC 7816-4, que trata da organização dos dados no cartão, dos mecanismos de segurança, como permissões de acesso, e estrutura dos comandos e respostas para comunicação com o mundo externo ao cartão inteligente.

***Applications Protocol Data Units* - APDUs**

Uma APDU pode ser vista como um envelope que armazena um comando completo enviado para o cartão, ou uma resposta a esse comando. Os dados que trafegam entre o cartão inteligente e o meio externo, representado pelo terminal, seguem o formato das APDUs. Faz-se uma distinção entre as APDUs de comando, que representam comandos enviados para o cartão, e as APDUs de resposta, que representam as respostas a esses comandos que o cartão envia para o terminal.

4.1.9 Ataques contra cartões inteligentes

Como foi apontado anteriormente, uma das principais características de um cartão inteligente é que ele proporciona um ambiente seguro para armazenamento de dados e execução de programas. Se não houvesse proteção contra acessos indevidos aos dados de um cartão inteligente, esse dispositivo não seria muito diferente de um microcontrolador qualquer.

Sabe-se que é muito difícil criar um sistema ou um dispositivo com uma segurança perfeita e robusto contra qualquer tipo de ataque, e isso se aplica também a cartões inteligentes. Ainda que esses dispositivos sejam projetados com vários recursos de segurança, se o esforço empregado para efetuar um ataque é levado a um nível alto o suficiente, é possível ganhar acesso aos e manipular os dados de um cartão inteligente.

Uma questão que se levanta quando se trata de ataques a um sistema ou dispositivo é que qualquer atacante faz uma análise de custo/benefício e avalia se o resultado almejado no ataque compensa o tempo, o dinheiro e o esforço investidos.

Os ataques contra cartões inteligentes são classificados em três níveis [63]:

- ataques no nível social;
- ataques no nível físico;
- ataques no nível lógico.

Ataques no nível social ocorrem, por exemplo, quando um usuário tem seu cartão roubado e é forçado a fornecer seu PIN. Ocorre ainda quando o usuário é induzido a revelar o PIN em resposta a mensagens ou sítios falsos.

Os ataques no nível físico ocorrem quando obtém-se acesso ao hardware do cartão inteligente. Tais ataques podem ser estáticos, pois ocorrem quando o cartão não está energizado. Esses ataques podem ocorrer quando deseja-se conhecer o conteúdo dos *chips* de memória do cartão, por exemplo. Para isso, podem ser utilizados microscópios e computadores com grande poder de processamento. Isso pode ocorrer não só para as memória não-voláteis, como a ROM e a EEPROM, mas também

para a memória RAM, pois seu conteúdo não necessariamente é perdido quando o cartão deixa de ser energizado. Sabe-se que o conteúdo da RAM pode ser mantido se o *chip* estiver a baixas temperaturas ou quando um mesmo conteúdo fica nesta memória por muito tempo [63]. Os *chips* mais modernos têm usado técnicas como embaralhamento ou cifragem do conteúdo das memórias a fim de coibir esse tipo de ataque.

Cartões inteligentes também podem estar sujeitos a ataques no nível físico do tipo dinâmicos, que ocorrem quando o cartão está operando. Um dispositivo como o cartão inteligente é composto de uma série de circuitos lógicos que são chaveados de maneiras diferentes dependendo da complexidade das operações executadas. Esses chaveamentos causam variações no consumo de energia, de modo que torna-se possível relacionar o consumo de energia com as operações executadas nos diferentes componentes do *chip*: CPU, coprocessadores criptográficos, barramentos e memórias. Para o caso particular das operações criptográficas, para uma mesma instrução, o consumo de energia é alterado de acordo com os valores das chaves e dos dados sendo processados. Um monitoramento do consumo de energia durante essas operações, seguido de um tratamento estatístico, pode levar à obtenção de informações sobre dados sensíveis que foram manipulados. Um ataque como esse pode ser executado com um aparato relativamente simples: um osciloscópio digital, um computador e uma leitora de cartões inteligentes modificada, que se comunica com o osciloscópio [29].

Como maneira de prevenir os ataques no nível físico, alguns processadores e coprocessadores de cartões inteligentes têm introduzido componentes sem função definida e operações aleatórias, mas que não afetam os resultados dos cálculos efetuados e que servem para confundir um eventual atacante. Os programadores também podem ajudar a combater esses ataques evitando desvios condicionais ou efetuando o mesmo número de operações em todos os casos, como ocorre, por exemplo, no método de exponenciação chamado *Montgomery's Ladder* [29].

Por fim, ainda existem os ataques no nível lógico, que se baseiam no raciocínio lógico e em alguns cálculos. Nessa categoria se incluem as técnicas de criptoanálise e ataques que exploram falhas conhecidas dos cartões inteligentes [67]. Assim como as técnicas de criptoanálise, os ataques de nível lógico podem ser classificados em passivos e ativos [63]. Os ataques passivos baseiam-se na análise dos resultados fornecidos pelo cartão inteligente a partir de determinadas entradas. Num ataque ativo, o atacante manipula os dados que são transmitidos para o cartão e analisa as respostas correspondentes.

4.1.10 Principais dificuldades na programação de cartões inteligentes

Além das restrições de hardware, a programação de cartões inteligentes apresenta outras dificuldades, como, por exemplo[4]:

- a depuração dos programas costuma ser demorada, pois os erros devem ser interpretados a partir das APDUs de resposta do cartão;
- os dados são transmitidos para o cartão ou fornecidos por ele por meio de APDUs que limitam o comprimento de dados a 256 bytes. Para carregar programas, tabelas e certificados é preciso quebrar os dados em várias partes e só então enviá-los, uma parte por vez, por meio de APDUs;
- a máquina virtual encontrada nos cartões inteligentes de múltiplas aplicações tem como objetivo resolver muitos dos problemas associados com ambientes de desenvolvimento específicos, oferecendo uma API que é seguida por todos os cartões que incorporam o padrão proposto para a máquina virtual, como apresentado na Seção 4.1.6. No entanto, esse fato não é sempre verdade, já que algumas aplicações podem usar primitivas não implementadas em todos os cartões, ainda que de um mesmo fabricante;
- outro fator dependente de cada cartão é a quantidade de memória disponível, tanto a memória não-volátil (EEPROM e flash) quanto a memória volátil (RAM), de modo que não se pode garantir que haverá uma quantidade de memória disponível quando se tratam de *chips* de fabricantes diferentes ou versões distintas de um mesmo fabricante;
- a máquina virtual introduz *overheads* nas aplicações, já que todas as instruções devem ser interpretadas antes de serem executadas pelo processador do cartão inteligente;
- não há muitas informações disponíveis sobre a implementação da máquina virtual. Isso pode levantar algumas dúvidas com relação à segurança que essas máquinas dizem proporcionar, bem como a possibilidade de existir falhas e *trap doors* em suas implementações, que permitam vazar o conteúdo de dados sigilosos armazenados no cartão.

Uma outra questão que se levanta é tem a ver com os custos dos cartões inteligentes. A Tabela 4.1 exibe os preços dos cartões inteligentes de arquitetura de 8 bits utilizados neste trabalho de implementação. Os preços foram obtidos através do sítio comercial USA Smart Card [68] e do consórcio MULTOS [66] em julho de 2010.

Cartão inteligente	**Preço (US$)**
JCOP 41	15,00
MULTOS	4,00

Tab. 4.1: Preços dos cartões inteligentes de 8 bits.

Os cartões inteligentes de 32 bits comercialmente disponíveis durante a execução deste trabalho eram na verdade emuladores dos *chips*, cujos preços estão exibidos na Tabela 4.2 e foram obtidos no

sítio da Ashling [69]. Recentemente, a empresa ARM anunciou o lançamento de um novo processador de 32 bits para cartões inteligentes [70].

Cartão inteligente	Preço (US$)
Vitra Networked Emulator	9.200,00
Genia Networked Emulator	4.750,00
Opella Emulator	985,00

Tab. 4.2: Preços de emuladores de cartões inteligentes de 32 bits.

Deve-se acrescentar aos preços mostrados na Tabela 4.2 o custo de US$ 2.465,00 referente ao software para compilar e depurar as aplicações, bem como carregá-las no emulador, já que nenhuma das opções de *chip* adota o sistema operacional Java Card ou MULTOS. Além disso, os emuladores e software para os cartões inteligentes de 32 bits são disponibilizados somente mediante um contrato de sigilo, o *Non-Disclosure Agreement* (NDA).

Nota-se que os cartões de 8 e de 32 bits apresentam diferenças significativas nos seguintes aspectos:

- **arquitetura**: a arquitetura de 32 bits proporciona um dispositivo com mais recursos de memória e processamento;

- **disponibilidade comercial**: os cartões de 8 bits são os cartões encontrados em aplicações de bancos e transporte público, por exemplo, e são comercialmente disponibilizados. Já os cartões de 32 bits são na verdade emuladores de *chips* e apresentam um custo muito elevado em comparação com os cartões inteligentes de 8 bits;

- **sistema operacional**: os cartões de arquitetura de 8 bits (JCOP e MULTOS) utilizam sistemas operacionais padronizados, que permite agregar às aplicações que os utilizam a característica de independência de fabricante;

- **utilização de software livre**: o software necessário para compilar e depurar aplicações, bem como carregá-las nos cartões de arquitetura de 8 bits pode ser facilmente encontrado na Internet e utilizado pelos desenvolvedores, desde que suas licenças sejam respeitadas. O software necessário para utilizar os cartões de 32 bits depende de um contrato e tem um custo elevado.

4.2 Motivações para implementação de emparelhamentos bilineares em cartões inteligentes de oito bits

Os conceitos referentes a cartões inteligentes apresentados neste capítulo e sobre criptossistemas baseados em emparelhamentos bilineares sobre curvas elípticas no Capítulo 2 fornecem uma base para apresentar os resultados da implementação de emparelhamentos bilineares em cartões inteligentes comercialmente disponíveis.

Grande parte das PKIs no Brasil e no mundo inteiro utilizam o RSA como algoritmo criptográfico padrão para criptografia de chave pública. Essa escolha tem motivação histórica e técnica. Como visto no Capítulo 2, o algoritmo RSA foi proposto em 1978 [19], enquanto a criptografia sobre curvas elípticas só foi proposta em 1985 [20], o que permitiu que o RSA fosse utilizado com exclusividade por um grande período e se firmasse como algoritmo seguro e eficiente para a criptografia de chaves públicas. A motivação técnica para o uso do RSA quando se empregam cartões inteligentes é a incorporação de *chips* de coprocessadores criptográficos para RSA na grande maioria dos cartões inteligentes. Esses coprocessadores possibilitam que os cartões executem eficientemente as operações de geração e verificação das assinaturas digitais.

É importante que os cartões inteligentes sejam capazes de efetuar operações criptográficas de forma segura e eficiente a fim de criar condições que viabilizem a implantação de PKIs e de outros sistemas criptográficos que utilizam criptografia baseada em emparelhamentos bilineares sobre curvas elípticas nesses dispositivos. Com esse propósito, o suporte ao cálculo de emparelhamentos em cartões inteligentes pode ser viabilizado por implementação em hardware ou software.

Uma implementação em hardware tem como resultado um coprocessador que deve ser inserido no *chip* com os demais componentes e que, portanto, deve considerar:

- características específicas do algoritmo de emparelhamento bilinear, como os parâmetros de entrada e o formato desses parâmetros, por exemplo;

- área: um *chip* de cartão inteligente tem uma área normalmente limitada a 25 mm^2, dos quais a maior parte é ocupada pelos componentes de memória. Restam poucos mm^2 para um coprocessador e, consequentemente, é limitado o número de operações que ele pode realizar;

- consumo de energia: a energia disponível para um *chip* de cartão inteligente é bastante limitada e um coprocessador deve considerar essa restrição;

- o projeto e integração de um coprocessador a um *chip* de cartão inteligente podem implicar em custos elevados.

Uma implementação em software também sofre restrições, como a quantidade de memória disponível, a limitação no tamanho de variáveis, entre outros fatores, conforme dito na Seção 4.1.10. No entanto, tais restrições não são tão fortes quanto aquelas que se aplicam a uma implementação em hardware. Além disso, a implementação em software pode utilizar um cartão inteligente comercialmente disponível, o que confere maior rapidez em comparação ao projeto, produção e incorporação de um coprocessador. Por isso, antes de partir para uma implementação de emparelhamentos bilineares em hardware, é importante investigar se uma implementação de emparelhamentos bilineares em software consegue atender os requisitos de segurança e eficiência necessários para uma aplicação criptográfica.

4.3 Materiais utilizados

Para implementar emparelhamentos bilineares em software, foram utilizados dois tipos de cartões inteligentes comercialmente disponíveis, ambos de arquitetura de 8 bits, que é o tipo mais encontrado (conforme visto na Seção 4.1.4) e que mais desafios apresenta:

- **Java Card JCOP41:** Esse cartão faz parte da família de cartões inteligentes SmartMX, produzido pela NXP, uma empresa de semicondutores criada pela Philips. Ele possui 72 KB de memória EEPROM utilizada para memória de dados e programas, 160 KB de memória ROM, 4608 bytes de memória RAM, coprocessador FameXE para operações de criptografia de chave pública, coprocessador para TDES e *clock* interno de até 30 MHz. O coprocessador FameXE do JCOP41 é capaz de efetuar operações do algoritmo RSA utilizando chaves de até 2048 bits. Esse coprocessador também é capaz de executar os algoritmos de criptografia de curvas elípticas *Elliptic Curve Digital Signature Algorithm* (ECDSA) e *Elliptic Curve Diffie-Hellman* (ECDH) para chaves de até 193 bits.

- **MULTOS 4.0:** Trata-se de um cartão fabricado com base na máscara AE45C 4.032f pela Hitachi/Renesas. Ele possui 32.5 KB de memória EEPROM utilizada para memória de dados e programas, 96 KB de memória ROM, 800 bytes de memória RAM pública, utilizada pelas APDUs de comando e resposta, 512 bytes de memória RAM dinâmica, utilizada para pilha e variáveis de sessão, coprocessadores para criptografia de chave pública e TDES, e *clock* de até 8 MHz. O coprocessador para criptografia de chave pública suporta chaves RSA de até 1024 bits e também oferece suporte a operações de multiplicação, inversão e exponenciação modulares.

O carregamento de aplicações nos cartões JCOP41 pode ser efetuado por meio da ferramenta JCOP Tools, desenvolvida pela IBM e que funciona como *plugin* para o IDE Eclipse [71] em ambiente Windows. A utilização dessa ferramenta depende de uma chave de autenticação, fornecida

gratuitamente pela IBM até 2008. Desde então, a distribuição de tal chave ficaria a cargo da NXP Semiconductors [72], produtora dos *chips* dos cartões JCOP. No entanto, não foi encontrado no sítio da empresa qualquer referência à distribuição da chave para o JCOP Tools. Como o cartão JCOP segue os padrões Java Card e Global Platform, é possível utilizar o kit de desenvolvimento para cartões inteligentes (JCDK) da Sun e ferramentas de software livre, como o GPShell, para compilar, depurar, carregar e remover aplicações nos cartões JCOP. Durante os trabalhos realizados todas as ferramentas citadas foram utilizadas com sucesso.

Os softwares necessários para compilar e manipular aplicações nos cartões MULTOS são todos disponibilizados gratuitamente no sítio da empresa [66]. Para efetuar o carregamento ou a remoção de aplicações no cartão MULTOS, é necessário gerar certificados digitais através do sítio da empresa. A adoção de certificados é uma boa solução de segurança para controlar aplicações prontas para utilização, mas torna-se bastante incômoda durante o processo de desenvolvimento, já que para cada alteração no programa é necessário efetuar uma nova requisição de dois certificados: um para remoção da versão anterior do programa e um para carregamento da nova versão.

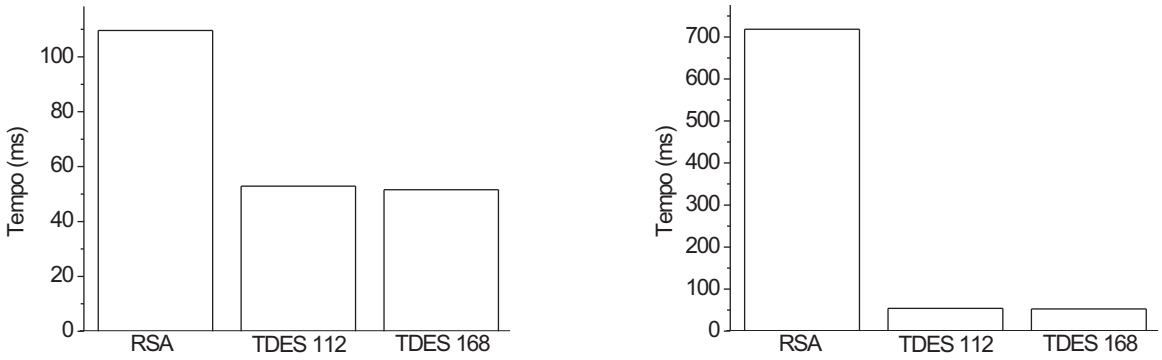
4.4 Testes preliminares com o cartão JCOP41

Com o intuito de criar familiaridade com o ambiente de desenvolvimento dos cartões inteligentes utilizados, foram efetuados diferentes testes com os cartões. Esta seção apresenta os resultados dos testes de desempenho feitos com o Java Card JCOP41. Todos os resultados apresentados foram obtidos pela média dos tempos de execução de cada operação em pelo menos 10 repetições. Não foram observadas variações nos tempos de execução e por isso o erro de cada medida foi desconsiderado.

4.4.1 Cifragem e decifragem

Primeiramente, foi avaliado o desempenho dos algoritmos de cifragem disponíveis no cartão JCOP41: o TDES e o RSA, ambos utilizando os coprocessadores disponíveis no cartão. De acordo com Hankerson et al [14], se o o tempo é a única medida empregada para avaliar a eficiência e robustez de um algoritmo, então o algoritmo TDES com chave de 112 bits apresenta o mesmo nível de segurança do RSA com chaves de 2048 bits. A Figura 4.3(a) mostra os tempos obtidos para a cifragem de uma mensagem de 128 bits utilizando o algoritmo RSA com chave de 2048 bits e o TDES com chaves de 112 e 168 bits. Apesar do algoritmo TDES usar efetivamente duas ou três chaves de 56 bits, a API Java Card requer duas ou três chaves de 64 bits. A Figura 4.3 apresenta os tempos para a operação de decifragem.

Os testes confirmam que as operações de cifragem e decifragem utilizando algoritmos de crip-



(a) Tempos para operação de cifragem.

(b) Tempos para operação de decifragem.

Fig. 4.3: Tempos para cifragem e decifragem empregando RSA e TDES.

tografia de chave pública são mais custosas quando comparados com os algoritmos de criptografia simétrica. Considerando a pequena diferença de tempos para o TDES com chaves de 112 e 168 bits, pode-se concluir que a segunda opção deveria ser preferida sempre que possível, uma vez que ela é mais robusta contra criptoanálise.

Nas Figuras 4.3(a) e 4.3(b) é interessante notar que, enquanto a cifragem utilizando RSA leva um tempo aproximadamente 110% maior que o TDES, a diferença sobe para 1200% na decifragem. Essa disparidade é causada pelo fato de que a cifragem é executada utilizando a chave pública 65537, conhecida como Número de Fermat, que é significativamente menor que a chave privada empregada para decifrar a mensagem. Ainda assim, os tempos para cifragem e decifragem para os dois algoritmos fica abaixo de um segundo para essa mensagem de 128 bits.

4.4.2 Assinatura digital e troca de chaves

Esta seção apresenta os resultados para operações de assinatura digital e troca de chaves fazendo uso de algoritmos de criptografia de chave pública implementados no hardware do cartão, ou seja, em seu coprocessador para operações de criptografia de chave pública. Conforme dito na Seção 4.3, o *chip* do cartão JCOP41 possui o coprocessador FameXE para operações de criptografia de chaves públicas, que inclui o RSA e dois algoritmos de curvas elípticas: o algoritmo de troca de chave ECDH e o algoritmo de assinatura digital ECDSA, de acordo com o padrão P1363 [73]. O coprocessador do cartão JCOP41 contempla apenas a versão dos referidos algoritmos para corpos binários e curvas ordinárias e segue a interface de programação do Java Card [65]. Os detalhes do algoritmo ECDSA, tanto para geração quanto para verificação de assinatura, encontram-se no Apêndice A.1. E os detalhes do ECDH encontram-se no Apêndice A.2.

No intuito de avaliar o desempenho dos algoritmos de ECC implementados no coprocessador FameXE, foram comparados os tempos desses algoritmos com operações análogas usando o algoritmo RSA, também implementado no mesmo coprocessador. Para efetuar as comparações, foi considerado o mesmo nível de segurança. De acordo com o documento SEC 2: Recommended Elliptic Curve Domain Parameters [74], RSA e ECC apresentam o mesmo nível de segurança para diferentes tamanhos de chaves, como mostrado na Tabela 4.3.

Tab. 4.3: Tamanhos de chaves e níveis de segurança equivalente para RSA e ECC.

Nível	chave RSA (bits)	chave ECC (bits)	RSA/ECC
1	512	113	5:1
2	704	131	5:1
3	1024	163	6:1
4	1536	193	8:1
5	2240	233	10:1
6	2304	239	10:1
7	3456	283	12:1

A fim de efetuar as comparações entre os algoritmos de ECC e o RSA, foram empregados quatro tamanhos de chaves disponíveis na API Java Card: 113, 131, 163 e 193 bits. Apesar de serem suportadas diversos tamanhos de chaves RSA, a API Java Card disponibiliza apenas as chaves RSA correspondentes aos níveis de segurança 1, 3 e 4 na Tabela 4.3. Para o nível 2, foi empregada a chave RSA de 768 bits, uma vez que esse era o tamanho que mais se aproximava aos 704 bits desse nível.

Observa-se ainda que as operações com o algoritmo RSA e os algoritmos de curvas elípticas foram todas efetuadas seguindo a API Java Card. Para o caso do ECDSA e do ECDH, a execução do protocolo envolve uma série de operações básicas no corpo binário, como adição, multiplicação, quadrado, inversão e redução modular. No entanto, apesar dessas operações básicas serem executadas, o programador pode utilizar apenas as operações de alto nível. Na assinatura ECDSA, por exemplo, o par de chaves público e privada é criado e, para o cálculo da assinatura digital, são executados dois métodos:

- `void init(Key theKey, byte theMode)`, método que inicializa as estruturas internas do cartão para efetuar operações relacionadas à assinatura digital. O parâmetro `theKey` é a chave privada previamente inicializada e `theMode` especifica se a operação é de geração ou verificação de assinatura;

- `short sign(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short sigOffset)`, método para efetuar a assinatura, onde `inBuff` é o *buffer* onde está a mensagem a ser assinada, `inOffset` é a posição no `inBuff` a partir da qual encontra-se a mensagem a ser assinada, `inLength` é o tamanho dessa mensagem,

`sigBuff` é o *buffer* onde deverá ser colocado o resultado da assinatura e `sigOffset` é a posição a partir da qual o resultado da assinatura deve ser armazenado no *buffer* `sigBuff`.

- `boolean verify(byte[] inBuff, short inOffset, short inLength, byte[] sigBuff, short sigOffset, short sigLength)`, método para verificar uma assinatura, onde `inBuff` é o *buffer* onde está a mensagem assinada a ser verificada, `inOffset` é a posição no *buffer* `inBuff` a partir da qual encontra-se a mensagem assinada, `inLength` é o tamanho dessa mensagem, `sigBuff` é o *buffer* onde se encontram os dados que geraram a assinatura `sigOffset` é a posição a partir da qual encontram-se os dados que geraram a assinatura no *buffer* `sigBuff`.

De maneira análoga, o protocolo de troca de chaves ECDH também pode ser executado pelos métodos:

- `void init(PrivateKey privKey)`, onde `privKey` é a chave privada previamente gerada e armazenada no cartão;
- `short generateSecret(byte[] publicData, short publicOffset, short publicLength, byte[] secret, short secretOffset)`, onde a área de memória `publicData` é o *buffer* que armazena a chave pública da outra entidade com a qual se deseja efetuar a troca de chaves, `publicOffset` é a posição do *buffer* `publicData` a partir da qual encontra-se essa chave pública, `publicLength` é o tamanho dessa chave pública, `secret` é o *buffer* onde deve ser armazenado o resultado obtido da execução do protocolo ECDH e `secretOffset` é a posição desse *buffer* a partir da qual deve ser armazenado o resultado.

Nota-se que os métodos disponibilizados para utilização do coprocessador são todos de alto nível, tornando impeditivo utilizá-lo para executar operações básicas separadamente, como adição e redução modular, por exemplo.

Resultados para o ECDSA

A Figura 4.4 apresenta os tempos para geração de chaves para algoritmos de ECC e para o RSA, considerando diferentes tamanhos de chaves, e, consequentemente, diferentes níveis de segurança, conforme apresentado na Tabela 4.3.

A observação da Figura 4.5 mostra que o tempo de geração do par de chaves é menor que um segundo para os algoritmos de ECC em todos os níveis considerados. Já no caso do RSA, esse tempo apresenta um crescimento exponencial com o aumento dos níveis de segurança, chegando a

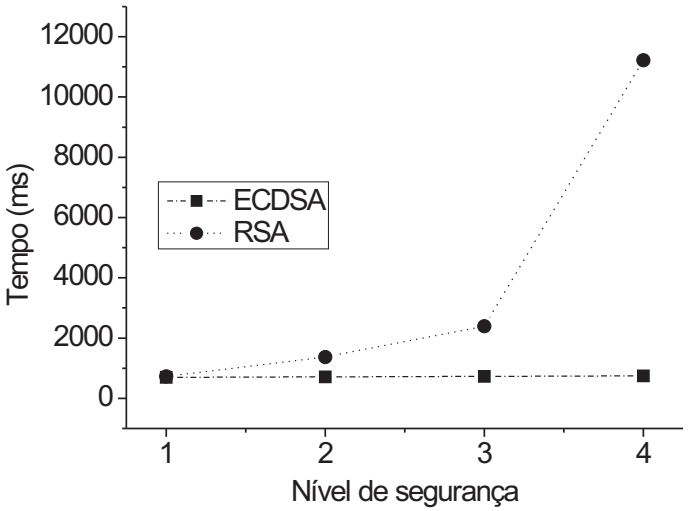


Fig. 4.4: Tempos de geração do par de chaves para diferentes níveis de segurança.

aproximadamente 12s no nível 4. Deve-se lembrar que, geralmente, a criação do par de chaves não é uma operação que se repete com frequência.

A Figura 4.5 apresenta os tempos de execução para gerar uma assinatura de uma mensagem de 160 bits, resultado da aplicação do algoritmo de *hash* SHA-1 [51] em uma mensagem arbitrária.

Pode-se notar na Figura 4.5 que, assim como na Figura 4.4, o tempo de execução para geração de uma assinatura pelo ECDSA é menor que o tempo do RSA em todos os níveis de segurança avaliados. O tempo de execução para o RSA é 82% e 250% maior que do tempo do ECDSA para os níveis de segurança 3 e 4, respectivamente.

Observa-se na Figura 4.6 que a verificação de assinatura apresentou um comportamento distinto daquele observado nas operações de geração de chaves (Figura 4.4) e de assinatura (Figura 4.5). Nota-se um comportamento estranho para os níveis 1 e 2, talvez porque a implementação do RSA no coprocessador do JCOP41 tenha sido otimizada para alguns tamanhos de chaves.

Nota-se pelos resultados que a vantagem do ECDSA sobre o RSA não foi tão grande no caso da verificação por dois principais motivos. O primeiro está relacionado com o tamanho da chave pública no caso do RSA, usada como parâmetro de entrada na verificação de assinatura. Conforme discutido na Seção 4.4.1, trata-se de uma chave pequena, o que reduz o custo das operações para o RSA. O segundo motivo tem a ver com as operações executadas durante uma verificação de assinatura ECDSA, que conforme apresentado no Apêndice A, são operações que podem ser consideradas de alto custo computacional: uma inversão, duas multiplicações escalares e uma adição de pontos da curva elíptica. Ainda assim, observa-se que para os maiores níveis de segurança (níveis 3 e 4) os

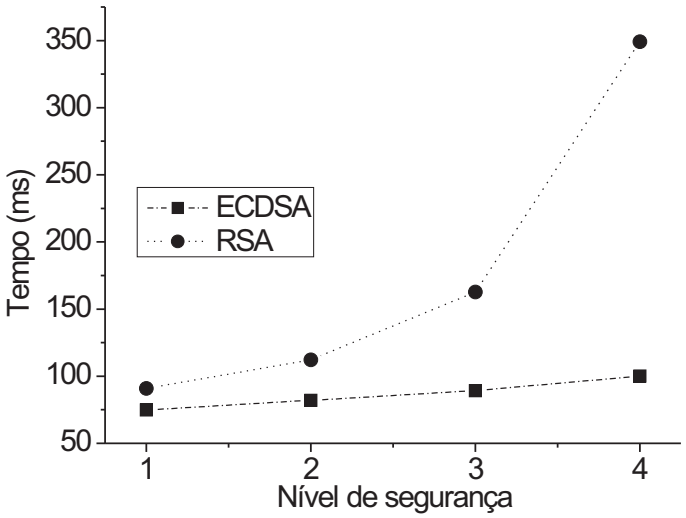


Fig. 4.5: Tempos de assinatura para diferentes níveis de segurança.

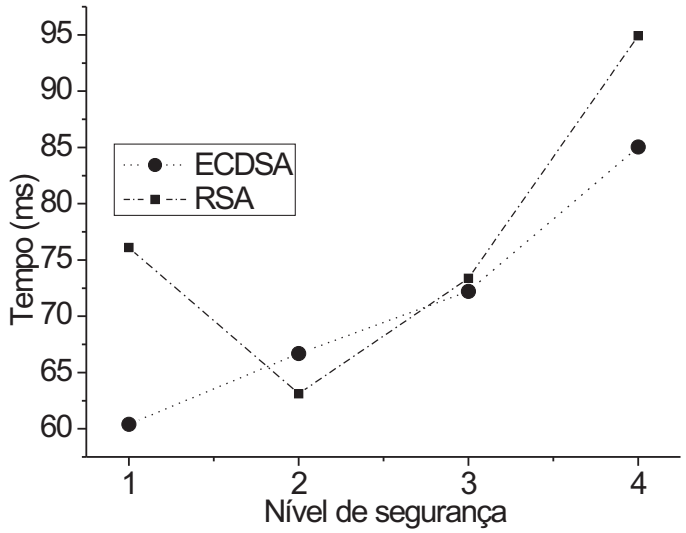


Fig. 4.6: Tempos de verificação de assinatura para diferentes níveis de segurança.

tempos de execução para o ECDSA são menores que para o RSA.

Como a operação de verificação de assinaturas digitais emprega a chave pública como parâmetro, não há um ganho de segurança ao executá-la no cartão inteligente, um dispositivo com limitações de processamento e memória. Portanto, pode-se exportar essa chave e utilizá-la, por exemplo, em um PC para efetuar a verificação de assinaturas, provavelmente em tempo inferior àqueles apresentados na Figura 4.6.

Resultados para o ECDH

Para efetuar uma comparação nos tempos de execução para a troca de chaves, considerou-se para o caso do RSA um esquema como descrito a seguir. O usuário A gera uma string aleatória de 64 bits, que em seguida é cifrada com a chave pública do destinatário B. O conteúdo recebido é então decifrado por B usando sua chave privada e o resultado funciona com uma chave simétrica. Os tempos de execução para efetuar uma troca de chaves usando esse esquema e usando o ECDH encontram-se na Figura 4.7.

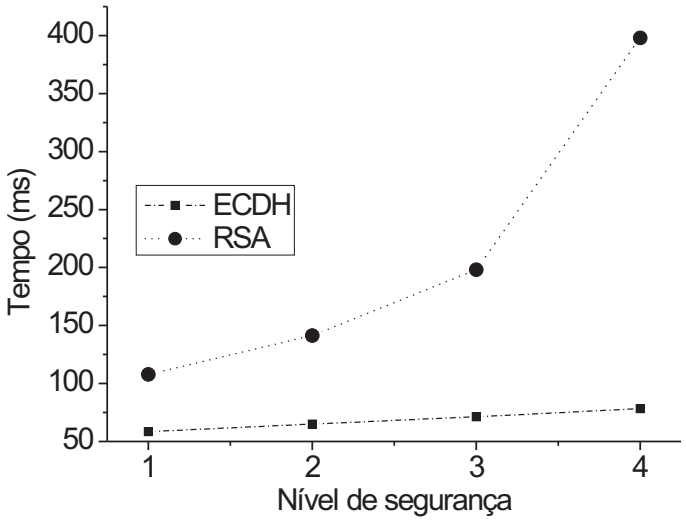


Fig. 4.7: Tempos de troca de chaves para diferentes níveis de segurança.

Os resultados apresentados na Figura 4.7 são semelhantes àqueles apresentados na Figura 4.5 e servem para corroborar a vantagem dos algoritmos de ECC sobre o RSA do ponto de vista de desempenho, especialmente quando os tamanhos das chaves vão ficando maiores.

4.5 Implementação de emparelhamentos bilineares nos cartões

Após os testes preliminares, o trabalho foi direcionado para a implementação dos emparelhamentos bilineares nos cartões Java Card JCOP41 e MULTOS. Os resultados obtidos são apresentados a seguir.

4.5.1 Emparelhamento η no Java Card

Os testes que fizeram uso do coprocessador FameXE do cartão JCOP41 mostraram bons resultados e proporcionaram uma maior familiaridade com esse Java Card. O próximo passo consistiu em implementar um algoritmo de emparelhamento bilinear nesse cartão. Conforme observado na Seção 4.4.2, as operações do coprocessador são disponibilizadas apenas em alto nível e, portanto, para implementar o emparelhamento bilinear foi preciso implementar os módulos de software que servem de base para o emparelhamento, conforme ilustrado na Figura 2.7. A tentativa de viabilizar emparelhamentos bilineares em cartões inteligentes foi feita por meio da implementação do chamado emparelhamento η (eta), proposto por Barreto et al [38], por ser considerado um emparelhamento eficiente para corpos binários. Existem outros algoritmos, como o emparelhamento Ate [39], mas o emparelhamento η serviu como base para que, se bem sucedido, outros algoritmos fossem implementados e avaliados.

Considerou-se a curva elíptica supersingular $y^2 + y = x^3 + x + b$, onde $b = \{0, 1\}$, para a qual se pôde aplicar o emparelhamento η.

Utilizou-se o corpo binário $\mathbb{F}_{2^{271}}$ e o grau de imersão, neste caso, é $k = 4$. A escolha do corpo binário foi motivada pela facilidade de representar e manipular elementos desse tipo de corpo no cartão inteligente. O corpo binário também permite que operações como adição e quadrado sejam executadas eficientemente. A extensão 271 foi empregada porque oferece um grau de segurança equivalente ao RSA com chaves de 1024 bits. É preciso observar que, por se tratar de uma curva supersingular, a equivalência de níveis de segurança não é a mesma apresentada na Tabela 4.3. Neste caso, multiplica-se a extensão do corpo base pelo grau de imersão da curva elíptica e o resultado é comparado com o tamanho das chaves RSA [11]. Essa equivalência foi questionada recentemente, mas ainda não há trabalhos publicados sobre o assunto. Outro fator que motivou a adoção da extensão 271 é a possibilidade de empregar um polinômio irredutível $f(z) = z^{271} + z^{207} + z^{175} + z^{111} + 1$, especialmente construído, que permite executar eficientemente as operações de raiz quadrada e redução modular com operações de ou-exclusivo e um número muito reduzido de *shifts*, conforme proposto por Scott et al [75].

Primeiramente, foram implementadas as operações do corpo binário. Como trata-se de um Java Card, foi utilizada a linguagem Java na implementação.

Uma dificuldade encontrada na implementação das operações básicas foi o fato de que todas as operações são realizadas com sinal pelo Java Card. Algumas operações no corpo base, como redução modular, envolvem muitas operações de deslocamentos (*shift*). Para variáveis com sinal, o resultado de um deslocamento pode sair diferente do esperado, e para contornar esse problema foi preciso aplicar máscaras de bits antes de algumas operações, o que introduz um processamento a mais, que poderia ser evitado numa linguagem como C, por exemplo, que permite definir variáveis sem sinal.

Na implementação a operação de adição, empregou-se a instrução de ou-exclusivo, uma vez que trata-se de um corpo binário. Para a redução modular, empregou-se o algoritmo baseado em *shifts* e ou-exclusivo, como aqueles apresentados por Hankerson et al [14].

Foram testadas duas implementações para a multiplicação modular. A primeira é baseada no método López-Dahab e utiliza uma tabela relativamente pequena de 16 bytes. A segunda implementação emprega uma tabela pré-computada de 16 elementos, com cada elemento utilizando 18 bytes, e é baseada na combinação dos métodos López-Dahab e Karatsuba-Ofman [76]. Os tempos obtidos para o último caso foram 12% menores que os tempos da primeira implementação e por isso foi adotado o segundo método de multiplicação modular.

A operação de quadrado utilizou uma tabela pré-computada para 4 bits. A implementação da raiz quadrada empregou o algoritmo proposto por Scott para o corpo binário de extensão 271 [77]. Por fim, a implementação da inversão empregou o algoritmo extendido de Euclides [14]. Os resultados obtidos para as operações no corpo finito encontram-se na Tabela 4.4.

Tab. 4.4: Desempenho do JCOP41 em operações básicas.

Operação	**Tempo (s)**
Adição	0.4
Redução modular	2.5
Multiplicação	5.3
Quadrado	2.8
Raiz quadrada	4.3
Inversão	166.0

Nota-se na Tabela 4.4 que os tempos obtidos para as operações básicas do emparelhamento bilinear, que são as operações no corpo binário, foram muito elevados. O tempo para execução do emparelhamento sem a exponenciação final completa foi de aproximadamente 840 segundos. A exponenciação final não pôde ser executada pelo cartão em razão de um erro de operação. Foram empregados para exponenciação tanto o método *Square-and-Multiply* quanto o método de exponenciação com mapas de Frobenius [29]. Também tentou-se executar a operação de inversão separada do resto do cálculo do emparelhamento, mas ainda assim o cartão continuou com erro de operação. Acredita-se que esse erro seja devido a algum problema na máquina virtual Java e no gerenciamento

de memória. Pode-se perceber, portanto, que o tempo de um emparelhamento completo seria extremamente alto, o que torna impraticável propor para um usuário utilizar um protocolo que utilize emparelhamento bilinear totalmente implementado em software nesse tipo de cartão inteligente.

Em razão do baixo desempenho observado já nas operações básicas, não foram experimentados outros algoritmos de emparelhamento no cartão JCOP41.

Esse baixo desempenho se contrapõe aos ótimos resultados obtidos na implementação em software relatada por Aranha et al [78], que fez uso de um computador com arquitetura Intel de 45nm e de instruções especiais para operar com vetores e multiprocessamento para reduzir a latência no cálculo do emparelhamento bilinear.

Os resultados apresentados nesta seção, em comparação com os bons resultados apresentados nas Seções 4.4.2, permitem levantar uma observação interessante sobre os resultados da implementação em software das operações básicas para corpos binários mostradas na Tabela 4.4. Como pode ser visto no Algoritmo A.1, mostrado no Apêndice A.1, para a geração de uma assinatura ECDSA é necessário efetuar várias operações no corpo binário (linha 7), e outras tantas são necessárias para executar a multiplicação escalar (linha 2). Ainda assim, o tempo de execução do ECDSA mostrado na Figura 4.5 não ultrapassa os 100ms, para o caso do maior nível de segurança considerado (4). Isso mostra que, mesmo estando na mesma plataforma, o coprocessador tem um papel fundamental para permitir que as operações básicas sejam executadas de forma eficiente, permitindo que o algoritmo criptográfico tenha um tempo de execução aceitável do ponto de vista do usuário. Portanto, caso houvesse possibilidade de acesso às operações básicas no coprocessador do cartão, os tempos apresentados na Tabela 4.4 poderiam ser bem menores.

4.5.2 Emparelhamento BKLS no cartão MULTOS

O baixo desempenho obtido para implementações em software para o cartão JCOP41 motivou a procura por um outro cartão inteligente para uma implementação de emparelhamento bilinear. Dentre os cartões programáveis e comercialmente disponíveis, havia o Java Card e o MULTOS. Como o Java Card já havia sido explorado com o JCOP41, optou-se então pelo cartão MULTOS. Uma vantagem desse cartão em relação ao JCOP41 é que sua interface de programação permite acessar alguns recursos do hardware, no caso o coprocessador que executa as operações modulares, conforme mostrado na Seção 4.3. Essa característica poderia beneficiar uma implementação em corpos primos, diferente da implementação em corpos binários mostrada na Seção 4.5.1.

Nessa etapa de implementação foi adotado o algoritmo BKLS, proposto por Barreto et al [10]. Esse algoritmo se aplica a determinadas curvas supersingulares, que segundo os autores apresentam características interessantes para aplicações em emparelhamentos bilineares. Mesmo tendo surgido o algoritmo Ate, proposto por Hess et al [39], com algumas otimizações em relação ao algoritmo

BKLS, dois fatos motivaram a implementação do algoritmo BKLS no cartão MULTOS. O primeiro fato foi o uso de corpo primos $\mathbb{F}_p$ nesse algoritmo, que possibilita aproveitar o coprocessador do cartão MULTOS com suporte a operações aritméticas em corpos primos, como multiplicação e exponenciação modulares. O segundo fato foi a possibilidade de comparação com duas implementações do mesmo algoritmo também em cartões inteligentes: a primeira feita por Scott et al [8], que utilizou o cartão Philips HiPerSmart de arquitetura de 32 bits, e a segunda por um grupo de pesquisadores da STMicroelectronics e da HPLabs [1], que utilizou um cartão inteligente de 32 bits da plataforma ST22, da STMicroelectronics. Nas referidas implementações, foi utilizada a curva elíptica

$$E(\mathbb{F}_p) : y^2 = x^3 + x, \tag{4.1}$$

sobre o corpo primo $\mathbb{F}_p$, com $p \equiv 3 \bmod 4$, o que permite o uso do grau de imersão $k = 2$ e a aplicação do mapa de distorção $\phi(x, y) = (-x, iy)$, onde $i^2 \equiv -1$, como nos números complexos.

A implementação descrita a seguir utilizou um primo p de 512 bits e a curva supersingular $E(\mathbb{F}_p)$ da Equação 4.1, com ordem $\#E(\mathbb{F}_p) = p + 1 = rc$, onde r é a ordem do ponto P, o primeiro argumento do emparelhamento, e o co-fator c é usado na exponenciação final do emparelhamento. Como tem-se $k = 2$, o expoente é $\frac{p^2-1}{r}$, que pode ser reescrito como $(p-1)c$.

Os parâmetros p, r e c utilizados na implementação do algoritmo BKLS foram os mesmos usados na implementação da biblioteca MIRACL [79], que oferece um conjunto de ferramentas de software que suportam algoritmos de criptografia simétrica e de chave pública, incluindo algoritmos de criptografia de curvas elípticas em corpos primos e binários:

$$\begin{aligned}
p = \ & 0x \ \ 8BA2A5229BD9C57CFC8ACEC76DFDBF3E3E1952C6B3193EC \\
& F5C571FB502FC5DF410F9267E9F2A605BB0F76F52A79E8043B \\
& F4AF0EF2E9FA78B0F1E2CDFC4E8549B; \\
r = \ & 0x \ \ 117454A4537B38AF9F9159D8EDBFB7E7C7C2E48760E930A461 \\
& D5F451F9D9210DC70095F4B241FF57F1BB0549C; \\
c = \ & 0x \ \ 800000000000000000000000000000020001.
\end{aligned}$$

A implementação do emparelhamento BKLS no cartão MULTOS exigiu uma série de cuidados e estratégias para contornar a escassez de memória RAM. Tal escassez implica que nem todos parâmetros de entrada e variáveis auxiliares do algoritmo podem ocupar ao mesmo tempo a memória de programa. Dessa forma, durante a execução do algoritmo, algumas variáveis precisaram ser copiadas da RAM para a EEPROM e vice-versa. Essa necessidade de gerenciar o uso de memória RAM acabou, portanto, introduzindo processamento adicional que não está relacionado diretamente com o

cômputo do emparelhamento.

A Tabela 4.5 mostra os resultados obtidos nos trabalhos de referência e o resultado da implementação no cartão MULTOS.

Tab. 4.5: Tempo de execução do algoritmo BKLS em diferentes cartões inteligentes.

Smart Card	**Arquitetura**	***Clock***	**Tempo**
Philips HiPerSmart[8]	32-bit	21 MHz	0,470 s
SmartJ ST22[1]	32-bit	33 MHz	0,752 s
MULTOS	8-bit	8 MHz	80 s

É possível ver na Tabela 4.5 que o tempo de execução do algoritmo BKLS no cartão MULTOS é extremamente alto, especialmente quando comparado com as implementações em outros cartões inteligentes, o que o torna inviável para um usuário utilizar um criptossistema que dependa do cômputo do emparelhamento no cartão. Isso mostra que a implementação de funções básicas, como multiplicação e exponenciação modulares no coprocessador do cartão MULTOS ainda não são suficientes para obter bons resultados em uma implementação em software de emparelhamentos bilineares nesse tipo de cartão.

Algumas razões do baixo desempenho obtido no cartão MULTOS são a baixa quantidade de memória RAM, o *clock* baixo e a restrição proporcionada por uma arquitetura de 8 bits, em comparação com os cartões de arquitetura de 32 bits.

Considerando uma situação onde apenas o tipo de arquitetura (8 ou 32 bits) e a frequência de *clock* sejam determinantes para o tempo de execução, se o cartão MULTOS tivesse uma arquitetura de 32 bits e um *clock* de 21 MHz, como o cartão Philips HiPerSmart, poderia ser esperado um tempo de execução de $\left(\frac{8}{32}\right) \cdot \left(\frac{8}{21}\right) \cdot 80 = 7,6s$. De forma análoga, para um *clock* de 33 MHz, o tempo de execução estimado seria de $\left(\frac{8}{32}\right) \cdot \left(\frac{8}{33}\right) \cdot 80 = 4,8s$. Obviamente, há outros fatores, como a velocidade dos barramentos, que afetam os tempos de execução, mas mesmo com esses ajustes nota-se que os tempos ainda estariam elevados.

4.5.3 Emparelhamento η no cartão MULTOS

De posse do cartão MULTOS, procurou-se desenvolver uma implementação do emparelhamento η, como aquela realizada no cartão JCOP41. Primeiramente, foram implementadas as operações no corpo binário utilizando a linguagem C. Os resultados obtidos encontram-se na Tabela 4.6.

Pode-se notar na Tabela 4.6 que os resultados para o cartão MULTOS são melhores que os resultados do JCOP41, muito provavelmente porque a implementação em C leva a um código mais eficiente em comparação com a implementação em Java. Outro motivo que pode levar a esses melhores resultados é a máquina virtual, que difere nos dois cartões.

Tab. 4.6: Tempos de execução (em segundos) para JCOP41 e MULTOS.

Operação	**JCOP41**	**MULTOS**
Adição	0.4	0.1
Redução modular	2.5	0.6
Multiplicação	5.3	4.2
Quadrado	2.8	1.2
Raiz quadrada	4.3	1.2
Inversão	166.0	120.0

Além da implementação em C feita no cartão MULTOS, fez-se também uma experiência implementando toda a rotina de quadrado e redução modular na linguagem Assembly disponível para esse cartão (MEL). A operação de quadrado utilizou uma tabela de valores pré-computados para 4 bits. Obteve-se um tempo de 0.9s, uma redução de 0.3s em comparação com a implementação em C. Nota-se que, apesar do esforço, o ganho obtido com a utilização de uma linguagem de baixo nível não foi muito significativo, muito provavelmente porque o compilador do MULTOS gera um código eficiente.

O tempo para a execução do emparelhamento sem a exponenciação final completa foi de 760 segundos, que também é muito elevado. A exponenciação final não foi executada porque a memória RAM do cartão não foi suficiente para armazenar as variáveis e a pilha usada pelo programa. Pode-se notar que os tempos obtidos tanto para o JCOP41 quanto para o MULTOS foram extremamente altos, especialmente quando comparados com as mesmas operações implementadas em PCs e até outros dispositivos embarcados, nos quais os tempos de execução das operações básicas têm uma ordem de mili ou microsegundos.

4.5.4 Análise do desempenho da implementação em software

Os resultados descritos nas Seções 4.5.1, 4.5.2 e 4.5.3 mostraram que as tentativas de implementar emparelhamentos em cartões inteligentes de 8 bits comercialmente disponíveis acabaram resultando em tempos extremamente elevados e proibitivos do ponto de vista do usuário.

Por outro lado, os dados da Tabela 4.5 mostram que é possível obter resultados satisfatórios para implementações em software parecidas às que estão sendo descritas neste trabalho, desde que sejam em arquitetura de 32 bits e frequência de *clock* superior àquela encontrada nos cartões MULTOS e JCOP41. Apesar de não terem sido encontradas informações sobre a quantidade de memória RAM disponível nos cartões inteligentes de 32 bits, provavelmente os *chips* devem conter uma maior quantidade dessa memória para programas em comparação com os poucos bytes disponíveis nos cartões de 8 bits. Esses fatores mostram que a arquitetura de 8 bits é muito restritiva para o desenvolvimento de programas complexos, como o cálculo de emparelhamentos bilineares.

Outra questão a ser discutida diz respeito à baixa eficiência da máquina virtual existente nos cartões multi-aplicações comercialmente disponíveis, como o MULTOS e o JCOP41. Conforme observado na Seção 4.1.10, esta máquina virtual não interfere nos tempos de execução dos cartões de arquitetura de 32 bits considerados na Tabela 4.5, por se tratarem de emuladores. Sabe-se que a máquina virtual introduz uma sobrecarga durante a execução dos programas, já que as instruções devem ser interpretadas em tempo de execução. Logo, é provável que os tempos reais em uma arquitetura de 32 bits sejam maiores do que os mostrados na Tabela 4.6.

O acesso apenas em alto nível das funções do coprocessador, conforme apresentado na Seção 4.4.2, torna impeditivo utilizar esse hardware para executar separadamente as operações básicas, como adição e redução modular, por exemplo. Uma maneira de resolver isso seria permitir o aproveitamento do coprocessador que encontra-se no cartão inteligente para disponibilizar métodos como os seguintes, nos quais considera-se que o corpo finito como um corpo binário:

- `void init(byte theMode)`, onde o parâmetro `theMode` indicaria qual a extensão do corpo binário;
- `void add(byte[] op1, byte[] op2, byte[] result)`, para adição de elementos do corpo finito, onde os parâmetros `op1` e `op2` são os *buffers* onde se encontrariam os operandos a serem somados e `result` é o *buffer* onde o resultado da soma seria armazenado. A assinatura desse e dos demais métodos consideraria que os valores encontram-se na posição inicial dos *buffers*, sem necessidade de indicar um *offset*.
- `void multiply(byte[] op1, byte[] op2, byte[] result)`, para multiplicação de elementos do corpo finito. Esse método já efetuaria a redução modular.
- `void square(byte[] op1, byte[] result)`, para efetuar a operação de quadrado.
- `void squareRoot(byte[] op1, byte[] result)`, para efetuar a operação de raiz quadrada.
- `void invert(byte[] op1, byte[] result)`, para efetuar a inversão de elementos do corpo finito.

Deve ser notado que, com a possibilidade de utilização destes métodos propostos, seria disponibilizada uma base para operações complexas, como ilustrado na Figura 2.7. Isso permitiria uma flexibilização no uso dos coprocessadores e viabilizaria a utilização eficiente de cartões inteligentes para protocolos baseados em emparelhamentos bilineares, num enfoque que utiliza parte de implementação em software e parte da implementação em hardware do coprocessador.

4.6 Conclusões do capítulo

Este capítulo apresenta resultados de implementações nos cartões Java Card JCOP41 e MULTOS, ambos de arquitetura de 8 bits. Primeiramente, foram apresentadas informações sobre cartões inteligentes, como estrutura física e lógica, além de características de segurança do cartão, que o tornam um dispositivo conveniente para aplicações diversas. Como o objetivo do trabalho é mostrar implementações de software em cartões inteligentes, foi dado enfoque nas questões relativas ao sistema operacional dos cartões e suas restrições de memória e processamento, que impõem uma série de dificuldades para o desenvolvimento de programas para esses dispositivos.

Com o intuito de apresentar o funcionamento dos cartões, em especial do Java Card JCOP41, foram apresentados testes preliminares que fazem uso do coprocessador do cartão. Nesses testes, foi apresentado o desempenho do JCOP41 em operações criptográficas variadas, como geração de chaves e assinatura digital, que permitiram traçar uma comparação entre RSA e algoritmos de criptografia de curvas elípticas (ECDSA e ECDH). De maneira geral, observou-se um melhor desempenho dos algoritmos de curvas elípticas em relação ao RSA e uma tendência dessa vantagem aumentar à medida que aumentam o tamanho das chaves e, consequentemente, o nível de segurança.

Em seguida, foram apresentados os resultados da implementação dos emparelhamentos bilineares nos cartões inteligentes de arquitetura de 8 bits. Foram empregados os algoritmos η e BKLS, mas em ambos os casos o tempo obtido foi bastante elevado. Foram identificadas como causas para o baixo desempenho a baixa quantidade de memória, as restrições de processamento impostas pela arquitetura de hardware e a sobrecarga imposta pela máquina virtual dos cartões analisados.

Para contornar os problemas de desempenho evidenciados com a implementação dos emparelhamentos bilineares, foi proposta a disponibilização de métodos que permitam o acesso das aplicações às operações básicas do coprocessador do cartão. Tendo em vista os bons resultados obtidos em operações como o ECDSA e o ECDH, a disponibilização desses métodos poderia levar a tempos menores do que aqueles apresentados neste capítulo.

Capítulo 5

Delegação de emparelhamentos

5.1 Delegação de emparelhamentos

Os resultados obtidos na implementação em software de emparelhamentos bilineares completos em cartões inteligentes de 8 bits mostrou que esses dispositivos apresentam características muito restritivas no que se refere à quantidade de memória e poder de processamento.

Uma maneira de diminuir a carga sobre os cartões inteligentes é a utilização de uma técnica conhecida como delegação de emparelhamentos ou *Pairing Delegation*, proposta por Chevallier-Mames et. al [80]. Essa técnica leva em consideração a existência de implementações eficientes de emparelhamentos bilineares em dispositivos como PCs e celulares, como descrito por Kawahara et al [81]. Normalmente, os cartões inteligentes estão ligados a esse tipo de dispositivo que, comparativamente, possui maior poder de processamento e quantidade de memória. A técnica também considera que dispositivos como cartões inteligentes são capazes de executar operações simples com elementos de corpos finitos e pontos de grupos elípticos.

A partir deste ponto, o dispositivo computacionalmente restrito será referenciado como *cartão* e o dispositivo mais robusto do ponto de vista computacional, como *terminal*.

Na delegação segura de emparelhamentos, o cartão delega o cômputo do emparelhamento bilinear $e(aP, bQ)$ para um terminal, de modo que são válidas as seguintes condições:

1. após o término do protocolo com um terminal honesto, o cartão obtém $e(P, Q)$;

2. o terminal não obtém nenhuma informação sobre os pontos P e Q;

3. o cartão é capaz de detectar quando o dispositivo mais robusto está trapaceando.

Neste capítulo serão empregados os conceitos e notações referentes a emparelhamentos bilineares apresentados no Capítulo 2.

5.2 Princípio básico

Na delegação do cômputo do emparelhamento $e(P, Q)$, pode ser considerado o seguinte cenário. O cartão gera dois números aleatórios a e b, calcula aP e bQ no grupo elíptico e demanda ao terminal o cálculo do emparelhamento:

$$\alpha = e(aP, bQ). \tag{5.1}$$

Para recuperar o valor de $e(P, Q)$, o cartão efetua o cálculo:

$$e(P, Q) = \alpha^{(ab)^{-1}}. \tag{5.2}$$

No entanto, observando as condições necessárias para uma delegação segura de emparelhamento mostradas na Seção 5.1, nota-se que, apesar do terminal não obter informações sobre os pontos P e Q, o cartão não consegue identificar se o terminal está trapaceando. Ou seja, se o terminal oferecesse como saída o valor α^r para algum $r \neq 1$, o cartão obteria $e(P, Q)^r$, ao invés de $e(P, Q)$, sem conseguir identificar a ação mal intencionada do terminal.

5.3 Protocolo para delegação segura de emparelhamento

Conforme dito na Seção 5.1, uma das condições básicas para ocorrer a delegação segura de emparelhamento é que o cartão consiga detectar quando o terminal está trapaceando e interrompa o protocolo com mensagem de erro.

Considerando a definição de emparelhamentos dada no Capítulo 2, o cartão e o terminal recebem como parâmetros de entrada:

- a definição dos grupos $\mathbb{G}_1$, $\mathbb{G}_2$ e $\mathbb{G}_T$, todos de ordem prima p;
- os pontos geradores G_1 e G_2;
- a definição do emparelhamento bilinear $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$;
- o valor do emparelhamento $e(G_1, G_2)$.

Para o cartão computar o emparelhamento $e(P, Q)$, ele deve interagir com o terminal conforme descrito nos seguintes passos.

1. O cartão gera dois escalares aleatórios g_1 e $g_2 \in \mathbb{F}_p$, calcula g_1P e g_2Q e demanda o cálculo de três emparelhamentos para o terminal:

$$\alpha_1 \;=\; e(g_1P, G_2) \tag{5.3}$$

$$\alpha_2 = e(G_1, g_2Q) \tag{5.4}$$

$$\alpha_3 = e(g_1P, g_2Q). \tag{5.5}$$

2. O cartão verifica que α_1, α_2 e $\alpha_3 \in \mathbb{G}_T$ através da exponenciação $\alpha_i^p = 1$, para $i = 1$, 2 e 3, já que $\mathbb{G}_T$ é um grupo de ordem p. Caso ocorra algum erro, o cartão termina a execução com mensagem de erro.

3. O cartão gera dois escalares aleatórios r_1 e $r_2 \in \mathbb{F}_p$, calcula $P + r_1G_1$ e $Q + r_2G_2$ e demanda o cálculo do emparelhamento para o terminal:

$$\alpha_4 = e(P + r_1G_1, Q + r_2G_2). \tag{5.6}$$

4. O cartão finalmente calcula

$$\alpha_4' = \alpha_3^{(g_1g_2)^{-1}} \cdot \alpha_1^{g_1^{-1}r_2} \cdot \alpha_2^{g_2^{-1}r_1} \cdot e(G_1, G_2)^{r_1r_2} \tag{5.7}$$

   e verifica se $\alpha_4' = \alpha_4$. Caso seja verificada a igualdade, o cartão responde com $\alpha_3^{(g_1g_2)^{-1}} = e(P, Q)$. Caso contrário, o cartão termina a execução e apresenta mensagem de erro.

Considerando a igualdade que é verificada no passo 4 da delegação, pode-se mostrar que é de fato esperado que $\alpha_4' = \alpha_4$, pois:

$$\alpha_1 = e(g_1P, G_2) = e(P, G_2)^{g_1}, \tag{5.8}$$

$$\alpha_2 = e(G_1, g_2Q) = e(G_1, Q)^{g_2}, \tag{5.9}$$

$$\alpha_3 = e(g_1P, g_2Q) = e(P, Q)^{g_1g_2}, \tag{5.10}$$

$$\alpha_4 = e(P + r_1G_1, Q + r_2G_2) \tag{5.11}$$

$$= e(P + r_1G_1, Q) \cdot e(P + r_1G_1, r_2G_2) \tag{5.12}$$

$$= e(P, Q) \cdot e(r_1G_1, Q) \cdot e(P, r_2G_2) \cdot e(r_1G_1, r_2G_2) \tag{5.13}$$

$$= e(P, Q) \cdot e(P, r_2G_2) \cdot e(r_1G_1, Q) \cdot e(r_1G_1, r_2G_2) \tag{5.14}$$

$$= e(P, Q) \cdot e(P, G_2)^{r_2} \cdot e(G_1, Q)^{r_1} \cdot e(G_1, G_2)^{r_1r_2} \tag{5.15}$$

$$= \alpha_3^{(g_1g_2)^{-1}} \cdot \alpha_1^{g_1^{-1}r_2} \cdot \alpha_2^{g_2^{-1}r_1} \cdot e(G_1, G_2)^{r_1r_2} \tag{5.16}$$

$$= \alpha_4'. \tag{5.17}$$

É possível observar que os pontos g_1P, g_2Q, $P + r_1G_1$ e $Q + r_2G_2$, entregues ao terminal, são todos pontos aleatórios e independentemente distribuídos nos grupos $\mathbb{G}_1$ e $\mathbb{G}_2$, o que implica que os pontos P e Q são mantidos em segredo.

Uma prova mais completa da técnica de delegação do emparelhamento pode ser encontrada no trabalho de Chevallier-Mames et al [80].

A descrição da delegação de emparelhamento considera que os dois pontos P e Q devem ser mantidos em sigilo. No entanto, há casos de protocolos criptográficos nos quais um desses pontos pode ser público ou constante, o que implica em menos operações no cartão e no terminal e permite gerar variantes de delegação de emparelhamentos, sem comprometer a segurança do protocolo, como mostrado no trabalho de Kang et al [82] e no trabalho não publicado de Tsai [83].

Existe também uma extensão da técnica de delegação de emparelhamentos conhecida como *Batch Pairing Delegation* [84], que oferece vantagens quando se faz necessário o cálculo de vários emparelhamentos bilineares, como nos casos de assinatura em lote. Se uma assinatura desse tipo pode ser verificada, então pode-se esperar com grande probabilidade que todas as assinaturas envolvidas são válidas.

A Seção 5.4 mostra uma forma de se aplicar delegação de emparelhamentos entre um aparelho celular e seu cartão inteligente usando o protocolo de Kang et al [82] e a Seção 5.5 propõe uma modificação neste protocolo que o torna mais eficiente e mais adequado a um contexto de pouco poder computacional disponível, como ocorre com cartões inteligentes.

5.4 Aplicação de delegação de emparelhamentos com protocolo simplificado

O aparelho de telefone celular tornou-se um dispositivo eletrônico com uso bastante difundido e muitos deles possuem internamente um cartão inteligente, que é o *Subscriber Identity Module* (SIM). Muitos celulares são capazes de executar programas cada vez mais complexos, como aplicações de multimídia, sendo então possível pensar no celular como o terminal em um cenário de delegação de emparelhamentos.

Nesta seção será apresentada uma proposta para utilizar a delegação de emparelhamentos a fim de cifrar mensagens enviadas a partir do celular. A proposta considera que o cartão é capaz de computar eficientemente a exponenciação no corpo $\mathbb{G}_T$, a inversão no corpo base $\mathbb{G}_1$ e a multiplicação escalar de pontos da curva $E(\mathbb{F}_p)$. A possibilidade de usar eficientemente emparelhamentos bilineares em celulares e dispositivos similares seria uma maneira de atender à demanda cada vez maior de segurança nas comunicações móveis.

O protocolo apresentado aqui é baseado no esquema de cifragem IBC proposto por Boneh e Franklin [45] e já apresentado na Seção 2.8.3. Normalmente, os trabalhos sobre delegação de emparelhamentos não mostram os detalhes da interação entre o cartão e o terminal da forma como será apresentado aqui, o que pode comprometer a compreensão do protocolo de delegação. No cenário

a ser apresentado, será considerado que os usuários A e B pertencem a uma empresa e precisam enviar mensagens secretar entre si. Nessa empresa, o papel do PKG é desempenhado por uma seção dedicada da mesma.

Na etapa de configuração, os parâmetros públicos são definidos e o PKG entrega para cada usuário um cartão contendo esses parâmetros e sua chave privada S_{ID}, conforme ilustrado na Figura 5.1.

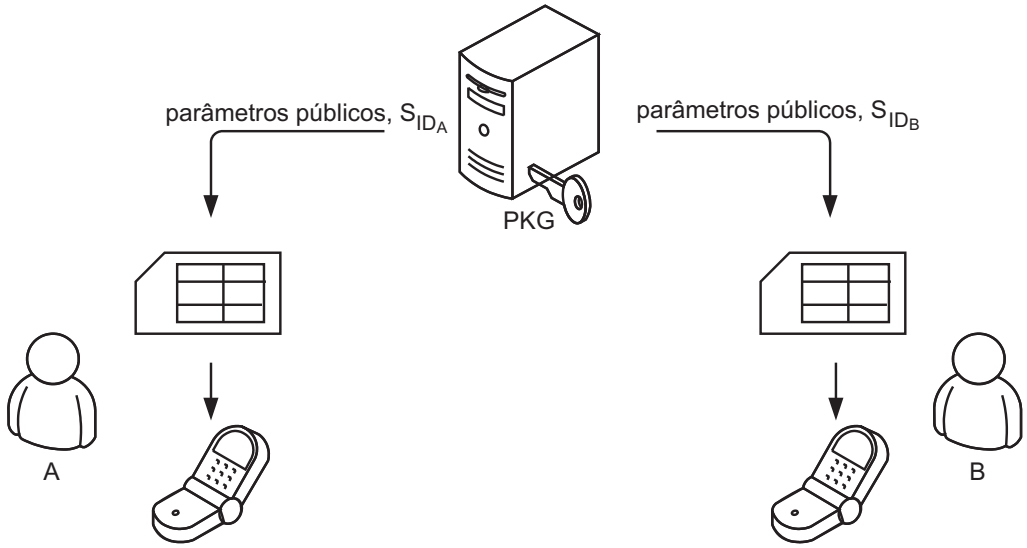


Fig. 5.1: Entrega dos parâmetros públicos e das chaves privadas.

Quando o usuário A deseja enviar uma mensagem sigilosa para o usuário B, ele utiliza a identidade ID_B e os parâmetros públicos para gerar o texto cifrado $C = \langle U, V \rangle$, conforme descrito na Seção 2.8.3. O identificador que é utilizado como chave pública pode ser o próprio número do celular, ou então uma composição desse número com o ano, por exemplo. A cifragem envolve apenas parâmetros públicos e, portanto, não é necessário utilizar a delegação de emparelhamentos. Conforme mostrado na Seção 2.8.3, é necessário utilizar durante a cifragem um valor r aleatório, o qual pode ser fornecido pelo cartão. No mais, o próprio terminal pode executar a operação de cifragem, como ilustrado na Figura 5.2, onde o usuário A cifra uma mensagem para o usuário B, usando como chave pública a identidade ID_B.

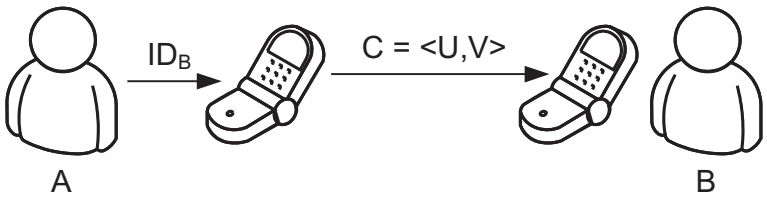


Fig. 5.2: Cifragem e envio da mensagem cifrada para o usuário B.

Após receber o texto cifrado $C = \langle U, V \rangle$, o usuário B emprega a chave privada S_{ID_B} armazenada em seu cartão para recuperar o texto em claro. Para isso, faz-se necessário calcular o emparelhamento $e(S_{ID_B}, U)$, onde o primeiro parâmetro, a chave privada S_{ID_B} é um ponto privado e o segundo parâmetro, o ponto U, é público. Portanto, quando o cartão delega o cálculo do emparelhamento, é preciso proteger apenas o primeiro parâmetro, o que reduz o número de operações necessárias em relação ao protocolo apresentado na Seção 5.3.

Nesse caso, Kang et al [82] sugere que a interação entre o cartão e o terminal para a delegação do emparelhamento ocorra da seguinte forma.

1. O cartão gera um escalar aleatório $g_1 \in \mathbb{F}_p$ e demanda o cálculo de três emparelhamentos para o terminal:

$$\alpha_1 = e(g_1 S_{ID_B}, G_2) \tag{5.18}$$
$$\alpha_2 = e(G_1, U) \tag{5.19}$$
$$\alpha_3 = e(g_1 S_{ID_B}, U), \tag{5.20}$$

   o que exige uma multiplicação escalar no cartão.

2. O cartão verifica que α_1, α_2 e $\alpha_3 \in \mathbb{G}_T$ através da exponenciação $\alpha_i^p = 1$, para $i = 1$, 2 e 3, já que $\mathbb{G}_T$ é um grupo de ordem p. Caso ocorra algum erro, o cartão termina a execução. Esta etapa exige que o cartão calcule três exponenciações no corpo finito $\mathbb{G}_T$.

3. O cartão gera os escalares aleatórios c_1, r_1 e $r_2 \in \mathbb{F}_p$ e demanda o cálculo do emparelhamento para o terminal:

$$\alpha_4 = e(S_{ID_B} + r_1 G_1, c_1 U + r_2 G_2) \tag{5.21}$$
$$= e(S_{ID_B}, U) \cdot e(S_{ID_B}, G_2)^{r_2} \cdot e(G_1, U)^{r_1} \cdot e(G_1, G_2)^{r_1 r_2}. \tag{5.22}$$

   Esse passo exige três multiplicações escalares no cartão.

4. Ao receber α_4, o cartão verifica se $\alpha_4 \in \mathbb{G}_T$, o que exige mais uma exponenciação.

5. O cartão finalmente calcula

$$\alpha_4' = \alpha_3^{(g_1)^{-1} c_1} \cdot \alpha_1^{g_1^{-1} r_2} \cdot \alpha_2^{c_1 r_1} \cdot e(G_1, G_2)^{r_1 r_2} \tag{5.23}$$
$$= e(S_{ID_B}, U) \cdot e(S_{ID_B}, G_2)^{r_2} \cdot e(G_1, U)^{r_1} \cdot e(G_1, G_2)^{r_1 r_2} \tag{5.24}$$

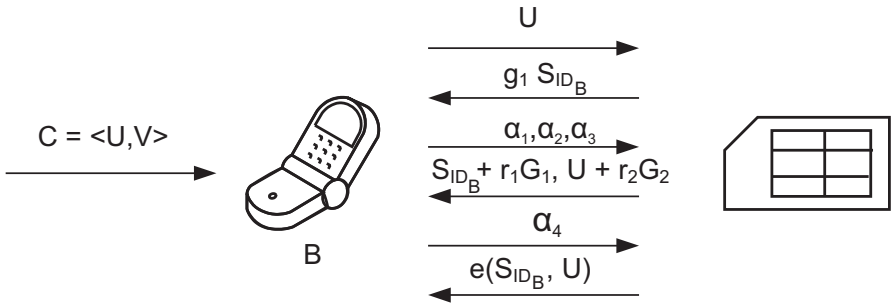


Fig. 5.3: Protocolo simplificado de delegação de emparelhamentos.

e verifica se $\alpha'_4 = \alpha_4$. Caso seja verificada a igualdade, o cartão envia para o terminal o resultado $\alpha_3^{(g_1)^{-1}} = e(S_{ID_B}, U)$. Caso contrário, o cartão indica o erro e termina a execução. Pode-se notar que esse último passo exige mais cinco exponenciações no cartão.

O protocolo exige um total de quatro multiplicações escalares e nove exponenciações.

A interação entre o cartão e o terminal está ilustrada na Figura 5.3.

5.5 Proposta de modificação no protocolo simplificado de delegação de emparelhamentos

Considerando o mesmo caso apresentado na Seção 5.4, pode-se observar que apenas o primeiro argumento do emparelhamento solicitado ao cartão precisa ser protegido, e assim mais uma redução no número de cálculos demandados ao cartão pode ser obtida além daquela no método de Kang et al [82]. Portanto, propomos aqui uma modificação sobre o protocolo simplificado de delegação de emparelhamento conforme mostrado a seguir.

1. O cartão gera um escalar aleatório $g_1 \in \mathbb{F}_p$ e demanda o cálculo de dois emparelhamentos para o terminal:

$$\gamma_1 = e(g_1 S_{ID_B}, U) = \alpha_3 \tag{5.25}$$
$$\gamma_2 = e(G_1, U) = \alpha_2, \tag{5.26}$$

   o que exige uma multiplicação escalar no cartão.

2. O cartão verifica que γ_1 e $\gamma_2 \in \mathbb{G}_T$ através da exponenciação $\gamma_i^p = 1$, para $i = 1$ e 2. Caso ocorra algum erro, o cartão termina a execução. Esta etapa exige que o cartão compute duas exponenciações no corpo finito $\mathbb{G}_T$.

3. O cartão gera o escalar aleatório $r_1 \in \mathbb{F}_p$ e demanda o cálculo do emparelhamento para o terminal:

$$\gamma_3 = e(S_{ID_B} + r_1 G_1, U) \quad (5.27)$$
$$= e(S_{ID_B}, U) \cdot e(G_1, U)^{r_1}. \quad (5.28)$$

   Esse passo exige uma multiplicação escalar no cartão.

4. Ao receber γ_3, o cartão verifica se $\gamma_3 \in \mathbb{G}_T$, o que exige uma exponenciação.

5. O cartão finalmente calcula

$$\gamma_3' = \gamma_1^{g_1^{-1}} \gamma_2^{r_1} \quad (5.29)$$
$$= e(S_{ID_B}, U) \cdot e(G_1, U)^{r_1} \quad (5.30)$$

   e verifica se $\gamma_3' = \gamma_3$. Caso seja verificada a igualdade, o cartão responde com $\gamma_1^{g_1^{-1}} = e(S_{ID_B}, U)$. Caso contrário, o cartão indica o erro e termina a execução. Note que este último passo exige duas exponenciações no cartão.

Com a modificação proposta, o novo protocolo de delegação demanda duas multiplicações escalares e cinco exponenciações, o que representa uma redução de duas multiplicações escalares e quatro exponenciações em relação ao método de Kang et al [82].

Com relação à segurança, vamos considerar que o terminal tenha tentado agir maliciosamente, fornecendo no primeiro passo da delegação os resultados:

$$\gamma_1 = e(g_1 S_{ID_B}, U) \cdot e(G_1, G_2)^{\beta_1} \quad (5.31)$$
$$\gamma_2 = e(G_1, U) \cdot e(G_1, G_2)^{\beta_2}, \quad (5.32)$$

onde β_1 e β_2 são diferentes de 0.

De maneira análoga, o valor de γ_3 fornecido por esse terminal malicioso seria da forma:

$$\gamma_3 = e(S_{ID_B} + r_1 G_1, U) \cdot e(G_1, G_2)^{\beta_3}. \quad (5.33)$$

Desse modo, o cálculo de γ_3 efetuado pelo cartão levaria ao seguinte resultado:

$$\gamma_3' = \gamma_1^{g_1^{-1}} \gamma_2^{r_1} \quad (5.34)$$
$$= e(S_{ID_B} + r_1 G_1, U) \cdot e(G_1, G_2)^{\beta_1 g_1^{-1} + \beta_2 r_1}. \quad (5.35)$$

Analisando as Equações 5.33 e 5.35, nota-se que, para o terminal conseguir trapacear o cartão, deve-se ter $\beta_3 = \beta_1 g_1^{-1} + \beta_2 r_1$. Para isso, é preciso que o terminal conheça os valores de g_1 e r_1, o que, no entanto, é considerado inviável, uma vez que, para obter tais valores, o terminal deve calcular o logaritmo discreto no grupo elíptico $E(\mathbb{F}_p)$, como se pode ver na Equação 5.25, ou no grupo multiplicativo $\mathbb{G}_T$, como pode ser visto na Equação 5.28.

Da mesma forma que o protocolo apresentado aqui pode ser usado para a decifragem IBC, ele pode ser empregado em operações criptográficas baseadas em emparelhamentos bilineares que possuam a característica de ter um ponto público e outro privado. Deve-se lembrar, no entanto, que a técnica considera que o cartão consegue executar eficientemente as operações de exponenciação no corpo $\mathbb{G}_T$ e multiplicação escalar de pontos da curva elíptica $E(\mathbb{F}_p)$. No novo protocolo de delegação eficiente proposto aqui, são requeridas no total duas multiplicações escalares e cinco exponenciações. Essa análise segue o mesmo tipo de contabilização de número de operações encontrado na literatura de delegação de emparelhamentos. Uma análise mais aprofundada, no entanto, deveria contabilizar também o custo de operações como a inversão dos escalares aleatórios, como o expoente g_1^{-1} de α_3 no passo 2, a soma de pontos da curva elíptica, como o cálculo de $S_{ID_B} + r_1 G_1$ e o produto $r_1 r_2$ do expoente no passo 5 do protocolo simplificado de Kang et al [82].

Outra observação que pode ser feita a respeito do protocolo apresentado aqui é que, apesar dos cartões possuírem pouca memória volátil, eles possuem geralmente uma quantidade razoável de memória não-volátil. Como visto na Seção 4.3, os cartões MULTOS e JCOP41 possuem, respectivamente, 32.5 KB e 72 KB de memória EEPROM. Considerando a robustez dos *chips* de cartões inteligentes, essa memória poderia ser utilizada para armazenar tabelas de escalares como g_1 e r_1 no exemplo anterior e seus inversos, como g_1^{-1} no passo 5. Dessa forma, seria reduzido o número de operações que o cartão precisaria executar. Caso se trate de um esquema de IBC em que as chaves são renovadas periodicamente, esses valores pré-computados poderiam ser fornecidos, por exemplo, pelo PKG, junto com as novas chaves privadas. Essa é uma proposta que deve considerar um compromisso entre segurança e eficiência, pois ao longo do tempo corre-se o risco de usar valores repetidos e caso os valores pré-computados vazem de alguma forma, a segurança das comunicações sigilosas anteriores fica comprometida.

5.6 Considerações sobre a técnica de delegação de emparelhamentos

As restrições encontradas na implementação de emparelhamentos bilineares mostradas no Capítulo 4 indicam que uma implementação em software desta técnica nos cartões JCOP41 ou MULTOS levariam a tempos elevados. Quando as operações no corpo base já são ineficientes, como mostrado

nas Seções 4.5.1 a 4.5.3, então pode-se esperar tempos bastante elevados para as operações de mais alto nível, como multiplicação escalar e exponenciação, usadas no protocolo de delegação de emparelhamentos.

Numa situação em que as operações básicas podem ser executadas eficientemente, deve-se questionar se a técnica de delegação é realmente vantajosa ou não, considerando que operações como multiplicação escalar, inversão e exponenciação no corpo de extensão podem ser consideradas custosas do ponto de vista computacional.

A técnica de delegação de emparelhamentos considera que o tempo gasto para o terminal executar as operações que lhe são delegadas é muito menor que o tempo gasto pelo cartão nas demais operações, de modo que o tempo total de aplicação da delegação é definido principalmente por esse último. Essa suposição, no entanto, precisa ser verificada em cada caso, uma vez que ela depende da eficiência do cartão e do terminal associado.

O tempo de comunicação entre o terminal e cartão é também negligenciado na técnica de delegação de emparelhamento. Em implementações práticas, no entanto, essa comunicação pode demandar tempo suficiente para também ter impacto no tempo total da aplicação de delegação de emparelhamento.

Um outro fator que pode pesar na adoção da técnica de delegação de emparelhamentos é a necessidade de um terminal, como um celular, por exemplo. No entanto, deve ser ressaltado que o uso do celular encontra-se bastante difundido, o que permite utilizar o cartão sem necessidade de dispositivos adicionais como uma leitora de cartões.

A técnica de delegação de emparelhamentos apresentada aqui se aplica bem a contextos onde o cartão inteligente está conectado a um dispositivo de maior poder computacional. Num cenário em que o cartão é substituído, por exemplo, por um nó de uma rede de sensores, a aplicação da delegação de emparelhamentos pode ser inviável, uma vez que a comunicação do sensor com sua base implica, de maneira geral, num consumo muito grande de tempo e de energia.

5.7 Estimativa de tempo

Para fazer uma estimativa sobre o tempo na execução do protocolo de delegação de emparelhamento, vamos considerar nesta seção uma implementação empregando a biblioteca MIRACL [79], em um computador pessoal de arquitetura Intel de 32 bits e *clock* de 2GHz, onde foi considerado um corpo primo de 160 bits e uma curva MNT com grau de imersão $k = 6$. A Tabela 5.1 exibe os tempos obtidos para operações no corpo de extensão e com pontos de curva elíptica:

O tempo para multiplicação escalar considera um ponto da curva elíptica no corpo base e um escalar aleatório de 160 bits. A exponenciação considera um resultado de emparelhamento e um

Operação	Tempo (ms)
Emparelhamento Tate	6.2
Multiplicação escalar	0.6
Exponenciação	0.8

Tab. 5.1: Operações no corpo de extensão e com pontos de curva elíptica.

expoente aleatório de 160 bits.

Vamos considerar agora um cartão capaz de executar as operações citadas em tempo proporcional aos tempos mostrados na Tabela 5.1, ou seja, o emparelhamento é executado em $6.2u$, por exemplo, onde u é a unidade de tempo que parametriza o tempo do cartão em relação ao tempo do computador.

As operações que o cartão executa no protocolo simplificado de delegação proposto por Kang et al [82], mostrado na Seção 5.4, resultam num total de quatro multiplicações escalares e nove exponenciações e o tempo total esperado para a delegação de emparelhamento seria de

$$4 \cdot 0.6u + 9 \cdot 0.8u = 9.6u. \tag{5.36}$$

Ou seja, pode-se esperar um tempo pelo menos 54% maior do que a execução do emparelhamento completo no cartão inteligente, que seria de $6.2u$, não havendo portanto vantagem no uso da delegação.

Agora levando em conta a modificação no protocolo de delegação proposta neste trabalho, pode-se esperar duas multiplicações escalares e 5 exponenciações, de modo que o tempo de

$$2 \cdot 0.6u + 5 \cdot 0.8u = 5.2u, \tag{5.37}$$

que representa um tempo 45% menor que o tempo do protocolo de delegação de Kang e que é 16% menor que o tempo para cálculo do emparelhamento completo ($6.2u$). Ou seja, a modificação proposta faz com que a utilização da técnica de delegação possa ser mais vantajosa do que o cálculo do emparelhamento completo no cartão.

Considerando que ainda existem os tempos internos do terminal e da comunicação entre o cartão e terminal, percebe-se que a técnica de delegação pode ou não trazer vantagens para um criptossistema baseado em emparelhamentos bilineares, dependendo dos algoritmos, implementações e plataformas computacionais empregados em cada caso.

5.8 Conclusões do capítulo

Este capítulo apresentou uma técnica conhecida como delegação de emparelhamentos, que foi proposta com o intuito de diminuir a carga de processamento no cartão que integra um criptossistema baseado em emparelhamentos bilineares. Num protocolo de delegação, o cartão efetua algumas operações e delega as mais custosas para um terminal. Para manter a segurança, o cartão conta com meios para identificar quando o terminal age maliciosamente e entrega resultados incorretos.

Considerando a importância da segurança nas comunicações móveis e o fato de que os celulares usualmente possuem um cartão inteligente, foi apresentado um protocolo de delegação de emparelhamentos que detalha as interações entre um cartão e um celular e que pode ser usado num criptossistema de cifragem baseado em identidades.

Tomando como base o protocolo de delegação apresentado, foram propostas mudanças que diminuem o número de operações executadas pelo cartão sem comprometer a segurança do criptossistema.

Finalmente, foi discutido ainda que as vantagens de aplicar a técnica de delegação de emparelhamentos dependem de uma série de fatores, como o tempo de comunicação entre o cartão e o terminal, o tempo das operações básicas, como inversão, exponenciação e multiplicação escalar, e do tempo de cálculo do emparelhamento completo.

Capítulo 6

Conclusões

O objetivo deste trabalho foi avaliar a implementação de emparelhamentos bilineares sobre curvas elípticas em cartões inteligentes de oito bits comercialmente disponíveis. Neste sentido, foram apresentados três resultados principais.

Primeiramente, foi feita uma implementação de emparelhamentos bilineares em software de dois tipos de cartões inteligentes: o Java Card JCOP41 e o cartão MULTOS, ambos com arquitetura de oito bits e quantidade de memória RAM reduzida a pouco mais de uma centena de bytes. Apesar de terem sido testados diferentes tipos de emparelhamentos na implementação, em todos os casos notou-se que os tempos obtidos foram extremamente elevados. Mesmo rotinas básicas de aritmética do corpo binário, como redução modular, multiplicação e quadrado, apresentaram uma execução muito demorada nos dois cartões considerados, o que mostrou a inviabilidade prática de uma implementação totalmente em software nesses dois cartões comercialmente disponíveis. Foram identificadas algumas causas que levaram ao baixo desempenho das implementações em software nos cartões MULTOS e Java Card. A primeira delas foram as condições restritivas do hardware, que oferece uma arquitetura de apenas oito bits, pouca memória RAM e baixa frequência de *clock*. A segunda causa que pode ser apontada é o uso de máquina virtual nos dois modelos de cartões. Se por um lado a máquina virtual apresenta a vantagem de permitir uma programação em alto nível e com independência do fabricante do chip, por outro lado ela introduz uma sobrecarga muito grande na execução dos programas desenvolvidos para o cartão inteligente, uma vez que cada instrução precisa ser interpretada antes de ser executada.

Ainda utilizando o cartão Java Card JCOP41, foram realizadas experimentações com seu coprocessador criptográfico, que suporta operações de criptografia de chaves públicas, entre elas o RSA e, dentro da criptografia de curvas elípticas, a operação de assinatura digital ECDSA e o protocolo de troca de chaves ECDH. Tais experimentações levaram a dois resultados interessantes. O primeiro deles evidenciou que os algoritmos de criptografia de curvas elípticas apresentaram uma performance

superior quando comparados com o RSA, considerando o mesmo nível de segurança. A vantagem dos algoritmos de ECC ficaram ainda mais evidentes à medida que os tamanhos de chaves, e por consequência, o nível de segurança, aumentava. O segundo resultado interessante foi mostrar que o uso de coprocessadores permite uma melhora significativa na performance das operações que apresentaram desempenho baixo quando implementadas em software do mesmo cartão. A geração de uma assinatura ECSA, por exemplo, foi executada em menos de um segundo pelo coprocessador do cartão inteligente. Essa operação exige muitas operações no corpo binário, que se fossem executadas por uma implementação em software, levariam um tempo de alguns minutos.

Os resultados obtidos permitiram concluir que não é viável adotar em cartões de oito bits, os mais difundidos no mercado, protocolos de emparelhamentos bilineares baseados apenas em software. Para viabilizar o uso de tais cartões é necessário que os mesmos possuam circuitos específicos (coprocessadores) para execução eficiente de operações básicas sobre corpos finitos e sobre curvas elípticas, e que seja disponibilizada uma interface de programação que permita o acesso direto a tais operações.

A segunda contribuição deste trabalho ocorreu durante o trabalho de implementação de emparelhamento em software nos cartões inteligentes, quando foi possível desenvolver equações específicas para adaptar o método conhecido como *Montgomery's Ladder* para efetuar multiplicação escalar de pontos de curvas elípticas supersingulares em corpos binários. O método não usa pré-computação, o que pode ser considerado uma vantagem para dispositivos com baixa capacidade de armazenamento, como os cartões inteligentes. Na comparação com outros métodos de multiplicação escalar, como o *Double-and-Add*, que também não usa pré-computação, o método proposto se mostrou mais eficiente quando se empregam as chamadas coordenadas projetivas. A vantagem para o caso de coordenadas afins depende do custo relativo entre as operações de multiplicação e inversão no corpo base. Além disso, o método proposto, herda do algoritmo *Montgomery's Ladder* a característica de ser robusto contra ataques do tipo *side-channel*.

Por fim, a terceira contribuição deste trabalho se deu por meio da modificação no protocolo de delegação de emparelhamentos. Essa técnica tem o objetivo de reduzir a carga de operações efetuadas pelo cartão que faz parte de algum criptossistema baseado em emparelhamentos bilineares. Foi proposta uma modificação no protocolo de delegação que se aplica para o caso em que o primeiro ponto do emparelhamento ser privado e o segundo ponto ser público, como ocorre no criptosssistema de cifragem de Boneh-Franklin. Com a modificação proposta, foi possível obter uma redução no número de operações executadas pelo cartão em relação a outro protocolo similar proposto na literatura, sem reduzir, no entanto, a segurança da delegação de emparelhamentos.

6.1 Trabalhos futuros

Ao longo deste trabalho foram apresentados resultados descritos na literatura de implementação de emparelhamentos bilineares em dispositivos computacionalmente restritos, como redes de sensores e cartões inteligentes de 32 bits. Uma possibilidade de trabalho a ser explorada é verificar a viabilidade de implementar emparelhamentos em outros dispositivos, como aqueles que empregam a tecnologia de *Near Field Communication* (NFC), que pode ser aplicada a celulares e combina mobilidade, a segurança proporcionada pelos cartões inteligentes e a possibilidade de interação com dispositivos próximos por meio de ondas de alta frequência.

Com relação ao método de multiplicação escalar de pontos de curvas elípticas supersingulares em corpos binários, foram apresentados resultados para coordenadas projetivas simples. Sabe-se, no entanto, que existem outras coordenadas que poderiam ser exploradas, como as coordenadas López-Dahab e as coordenadas Jacobianas [29], por exemplo. Essas diferentes coordenadas devem ser investigadas para verificar se elas podem evidenciar ainda mais a vantagem do método proposto.

Por fim, com relação à delegação de emparelhamentos, foi proposta uma modificação na técnica para o caso de o primeiro argumento do emparelhamento ser um ponto privado e o segundo, um ponto público. Existem ainda outras configurações envolvendo os argumentos do emparelhamento bilinear, nas quais pode-se investigar a possibilidade de propor modificações que levem a uma redução nas operações necessárias, sem comprometer a segurança. Ainda com relação à delegação de emparelhamentos, outra possibilidade a ser explorada é comparar a delegação de emparelhamentos com a implementação de um emparelhamento completo, a fim de verificar qual enfoque é mais indicado do ponto de vista de consumo de memória e processamento, no caso de se dispor de um cartão que consiga efetuar eficientemente operações no corpo base. Para realizar essa comparação, poderiam ser consideradas diferentes implementações de emparelhamentos bilineares, com diferentes curvas elípticas.

Referências Bibliográficas

[1] Guido M. Bertonib, Luca Breveglieria, Liqun Chenc, Pasqualina Fragnetob, Keith A. Harrisonc e Gerardo Pelosi. A pairing SW implementation for Smartcards. *Journal of Systems and Software*, 81:1240–1247, 2008.

[2] Leonardo B. Oliveira, Michael Scott, Julio López e Ricardo Dahab. TinyPBC: Pairings for Authenticated Identity-Based Non-Interactive Key Distribution in Sensor Networks. Cryptology ePrint Archive, Report 2007/482, 2007.

[3] Jean-Luc Beuchat, Hiroshi Doi, Kaoru Fujita, Atsuo Inomata, Piseth Ith, Akira Kanaoka, Masayoshi Katouno, Masahiro Mambo, Eiji Okamoto, Takeshi Okamoto, Takaaki Shiga, Masaaki Shirase, Ryuji Soga, Tsuyoshi Takagi, Ananda Vithanage e Hiroyasu Yamamoto. FPGA and ASIC implementations of the η_T pairing in characteristic three. *Comput. Electr. Eng.*, 36(1):73–87, 2010.

[4] Tommi Elo. Lessons learned on implementing ECDSA on a Java smart card. Disponível em `http://www.tml.hut.fi/Research/TeSSA/Papers/Elo/Elo_Nordsec2000.pdf`, 2000. Último acesso em agosto de 2010.

[5] István Zsolt Berta e Zoltán Ádám Mann. Implementing Elliptic Curve Cryptography on PC and Smart Card. Disponível em `http://www.pp.bme.hu/ee/2002_1/pdf/ee2002_1_04.pdf`, 2002. Último acesso em agosto de 2010.

[6] Danival Taffarel Calegari. Uma implantação de criptografia de curvas elípticas no Java Card. Tese de Mestrado. Universidade Estadual de Campinas. Instituto de Computação, 2002.

[7] Gemplus. Gemplus develops the world's first Identity-Based Encryption for smart cards. Disponível em `http://www.gemalto.com/press/gemplus/2004/id_security/02-11-2004-Identity-Based_Encryption.htm`, 2004.

[8] Michael Scott, Neil Costigan e Wesam Abdulwahab. Implementing Cryptographic Pairings on Smartcards. Cryptology ePrint Archive, Report 2006/144, 2006.

[9] Augusto J. Devegili, Michael Scott e Ricardo Dahab. Implementing Cryptographic Pairings over Barreto-Naehrig Curves. Cryptology ePrint Archive, Report 2007/390, 2007.

[10] Paulo S.L.M. Barreto, Hae Y. Kim, Ben Lynn e Michael Scott. Efficient Algorithms for Pairing-Based Cryptosystems. Cryptology ePrint Archive, Report 2002/008, 2002.

[11] Piotr Szczechowiak, Leonardo B. Oliveira, Michael Scott, Martin Collier e Ricardo Dahab. NanoECC: testing the limits of elliptic curve cryptography in sensor networks. In *EWSN'08: Proceedings of the 5th European conference on Wireless sensor networks*, pages 305–320, Berlin, Heidelberg, 2008. Springer-Verlag.

[12] Diego F. Aranha, Leonardo B. Oliveira, Julio López e Ricardo Dahab. Nanopbc: Implementing cryptographic pairings on an 8-bit platform. CHiLE 2009. Disponível em `http://inst-mat.utalca.cl/chile2009/Slides/Diego_Aranha_2.pdf`, 2009. Último acesso em agosto de 2010.

[13] Piotr Szczechowiak, Anton Kargl, Michael Scott e Martin Collier. On the application of pairing based cryptography to wireless sensor networks. In *WiSec '09: Proceedings of the second ACM conference on Wireless network security*, pages 1–12, New York, NY, USA, 2009. ACM.

[14] Darrel Hankerson, Alfred J. Menezes e Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[15] Alfred J. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, Norwell, MA, USA, 1994. Foreword By-Koblitz, Neal.

[16] RSA Laboratories. Frequently Asked Questions about Today's Cryptography. Disponível em `http://www.rsasecurity.com/rsalabs/faq/files/rsalabs_faq41.pdf`, 2000.

[17] Joan Daemen e Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer Verlag, Berlin, Heidelberg, New York, 2002.

[18] Whitfield Diffie e Martin E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory.*, Vol. 22:644–654, 1976.

[19] Ron L. Rivest, Adi Shamir e Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[20] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[21] Victor S. Miller. Use of elliptic curves in cryptography. In *Lecture notes in computer sciences; 218 on Advances in cryptology—CRYPTO 85*, pages 417–426, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

[22] Michael Scott e Paulo S. L. M. Barreto. Generating More MNT Elliptic Curves. *Designs, Codes and Cryptography*, 38(2):209–217, 2006.

[23] Paulo S. L. M. Barreto e Michael Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. volume 3897, pages 319–331. Springer-Verlag, 2006.

[24] David Freeman. Constructing Pairing-Friendly Elliptic Curves with Embedding Degree 10. In *10th Workshop on Elliptic Curves in Cryptography (ECC 2006*, pages 452–465. Springer-Verlag, 2006.

[25] Ezekiel J. Kachisa, Edward F. Schaefer e Michael Scott. Constructing Brezing-Weng Pairing-Friendly Elliptic Curves Using Elements in the Cyclotomic Field. In *Pairing '08: Proceedings of the 2nd international conference on Pairing-Based Cryptography*, pages 126–135, Berlin, Heidelberg, 2008. Springer-Verlag.

[26] Daniel J. Bernstein, Peter Birkner, Tanja Lange e Christiane Peters. Optimizing double-base elliptic-curve single-scalar multiplication. In *INDOCRYPT'07: Proceedings of the cryptology 8th international conference on Progress in cryptology*, pages 167–182, Berlin, Heidelberg, 2007. Springer-Verlag.

[27] Nigel P. Smart. The Hessian Form of an Elliptic Curve. In *CHES '01: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, pages 118–125, London, UK, 2001. Springer-Verlag.

[28] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest e Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, second edition, 2001.

[29] Henri Cohen e Gerhard Frey, editors. *Handbook of elliptic and hyperelliptic curve cryptography*. CRC Press, 2005.

[30] Neal Koblitz. Hyperelliptic cryptosystems. *Journal of Cryptology*, 1(3):139–150, 1989.

[31] Alfred J. Menezes, Yi-Hong Wu e Robert J. Zuccerato. An Elementary Introduction to Hyperelliptic Curves. Technical report, University of Waterloo, 1996.

[32] Tanja Lange. Efficient arithmetic on genus 2 hyperelliptic curves over finite fields via explicit formulae. Cryptology ePrint Archive, Report 2002/121, 2002.

[33] Steven D. Galbraith, Michael Harrison e David J. Mireles Morales. Efficient Hyperelliptic Arithmetic using Balanced Representation for Divisors. Cryptology ePrint Archive, Report 2008/265, 2008.

[34] Howon Kim, Thomas Wollinger, Doo H. Choi, Dong G. Han e Mun K. Lee. Hyperelliptic Curve Crypto-Coprocessor over Affine and Projective Coordinates. *ETRI Journal*, 30:365–376, 2008.

[35] Victor S. Miller. Short Programs for functions on Curves. In *IBM Thomas J. Watson Research Center*, 1986.

[36] Steven D. Galbraith, Keith Harrison e David Soldera. Implementing the Tate Pairing. In *ANTS-V: Proceedings of the 5th International Symposium on Algorithmic Number Theory*, pages 324–337, London, UK, 2002. Springer-Verlag.

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

[38] Paulo S. L. M. Barreto, Steven Galbraith, Colm O'hEigeartaigh e Michael Scott. Efficient Pairing Computation on Supersingular Abelian Varieties. Cryptology ePrint Archive, Report 2004/375, 2004.

[39] Florian Hess, Nigel Smart P. Smart e Frederik Vercauteren. The Eta Pairing Revisited. *IEEE Transactions on Information Theory*, 52:4595–4602, 2006.

[40] Eunjeong Lee, Hyang-Sook e Cheol-Min Park. Efficient and generalized pairing computation on Abelian varieties. *IEEE Trans. Inf. Theor.*, 55(4):1793–1803, 2009.

[41] Antoine Joux e Kim Nguyen. Separating Decision Diffie-Hellman from Diffie-Hellman in cryptographic groups. Cryptology ePrint Archive, Report 2001/003, 2001.

[42] Paulo Barreto. The Pairing-Based Crypto Lounge. Disponível em `http://www.larc.usp.br/~pbarreto/pblounge.html`. Último acesso em agosto de 2010.

[43] Antoine Joux. A One Round Protocol for Tripartite Diffie-Hellman. In *ANTS-IV: Proceedings of the 4th International Symposium on Algorithmic Number Theory*, pages 385–394, London, UK, 2000. Springer-Verlag.

[44] Dan Boneh e Xavier Boyen. Short Signatures without Random Oracles. In *Eurocrypt 2004*, pages 56–73. Springer-Verlag, 2004.

[45] Dan Boneh e Matthew K. Franklin. Identity-Based Encryption from the Weil Pairing. In *CRYPTO '01: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, pages 213–229, London, UK, 2001. Springer-Verlag.

[46] Clifford Cocks. An Identity Based Encryption Scheme Based on Quadratic Residues. In *Proceedings of the 8th IMA International Conference on Cryptography and Coding*, pages 360–363, London, UK, 2001. Springer-Verlag.

[47] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 47–53, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[48] Sattam S. Al-Riyami e Kenneth G. Paterson. Certificateless Public Key Cryptography. Cryptology ePrint Archive, Report 2003/126, 2003.

[49] Tsz Hon Yuen, Willy Susilo e Yi Mu. How to Construct Identity-Based Signatures without the Key Escrow Problem. Cryptology ePrint Archive, Report 2009/421, 2009.

[50] Routo Terada. *Segurança de Dados- Criptografia em Redes de Computador*. Edgard Blücher, São Paulo, 2000.

[51] William Stallings. *Cryptography and Network Security Principles and Practices*. Prentice Hall, Upper Saddle River, NJ 07458, USA, 2005.

[52] Waldyr Dias Benits Júnior. Sistemas Criptográficos Baseados em Identidades Pessoais. Tese de Mestrado. Universidade de São Paulo, 2003.

[53] Dan Boneh e Xavier Boyen. Efficient Selective-ID Secure Identity Based Encryption without Random Oracles. In *Advances in Cryptology - EUROCRYPT 2004*, volume 3027, pages 223–238, 2004.

[54] Mugino Saeki. Elliptic curve cryptosystems. Tese de Mestrado. McGill University, Montreal, 1997.

[55] Alfred Menezes. Supersingular Elliptic Curves in Cryptography. Disponível em `http://www.pairing-conference.org/2007/invited/Menezes_slide.pdf`. Último acesso em agosto de 2010.

[56] X. Boyen e L. Martin. RFC 5091: Identity-Based Cryptography Standard (IBCS) #1: Supersingular Curve Implementations of the BF and BB1 Cryptosystems. Disponível em `http://www.rfc-editor.org/rfc/rfc5091.txt`. Último acesso em agosto de 2010.

[57] Julio López e Ricardo Dahab. Fast Multiplication on Elliptic Curves over GF(2m) without Precomputation. In *CHES '99: Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, pages 316–327, London, 1999. Springer-Verlag.

[58] Katsuyuki Okeya e Kouichi Sakurai. Efficient Elliptic Curve Cryptosystems from a scalar multiplication algorithm with recovery of the y-coordinate on a Montgomery-form elliptic curve. In *Cryptographic Hardware and Embedded Systems - CHES 2001*, pages 126–141, Berlim, 2001. Springer-Verlag.

[59] Tetsuya Izu e Tsuyoshi Takagi. A Fast Parallel Elliptic Curve Multiplication Resistant against Side Channel Attacks. pages 280–296. Springer-Verlag, 2002.

[60] Wieland Fischer, Christophe Giraud, Erik Woodward Knudsen e Jean-Pierre Seifert. Parallel scalar multiplication on general elliptic curves over $\mathbb{F}_p$ hedged against Non-Differential Side-Channel Attacks. Cryptology ePrint Archive, Report 2002/007, 2002.

[61] Bruce Schneier. *Applied cryptography: protocols, algorithms, and source code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[62] Peter L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. In *Mathematics of Computation*, volume 48, pages 243–264. American Mathematical Society, 1987.

[63] Wolfgang Rankl e Wolfgang Effing. *Smart Card Handbook*. John Wiley & Sons, 3rd edition, 2004.

[64] Nicolas T. Courtois. The Dark Side of Security by Obscurity and Cloning MiFare Classic Rail and Building Passes Anywhere, Anytime. Cryptology ePrint Archive, Report 2009/137, 2009.

[65] Sun Developer Network. Java Card Development Kit. Disponível em `http://java.sun.com/javacard/devkit/`. Último acesso em agosto de 2010.

[66] MULTOS Web site. Disponível em `http://www.multos.com`. Último acesso em agosto de 2010.

[67] Dan Boneh, Richard A. Demillo e Richard J. Lipton. On the Importance of Eliminating Errors in Cryptographic Computations. *Journal of Cryptology*, 14:101–119, 2001.

[68] USA Smart Card Web site. Disponível em `http://www.usasmartcard.com`, 2007. Último acesso em agosto de 2010.

[69] Ashling Development Tools for Philips HiPerSmart™Smart Cards. Disponível em `http://www.testech-elect.com/ashling/hps_tools.htm`. Último acesso em agosto de 2010.

[70] DigitalIDNews. Disponível em `http://www.digitalidnews.com/2010/03/17/arm-unveils-new-32-bit-processor-for-smart-cards`. Último acesso em agosto de 2010.

[71] Eclipse.org Web site. Disponível em `http://www.eclipse.org`. Último acesso em agosto de 2010.

[72] NXP Semiconductors Web site. Disponível em `http://www.nxp.com`. Último acesso em agosto de 2010.

[73] IEEE P1363 / D13 (Draft Version 13). Standard Specifications for Public Key Cryptography. Disponível em `http://grouper.ieee.org/groups/1363/P1363/draft.html`, 1999. Último acesso em agosto de 2010.

[74] Certicom Research. Sec 2: Recommended Elliptic Curve Domain Parameters. Disponível em `http://www.secg.org/collateral/sec2_final.pdf`, 2000. Standards for Efficient Cryptography Version 1.0.

[75] Michael Scott. Optimal Irreducible Polynomials for $GF(2^m)$ Arithmetic. Cryptology ePrint Archive, Report 2007/192, 2007.

[76] A. Karatsuba e Yu. Ofman. Multiplication of Many-Digital Numbers by Automatic Computers. In *Proceedings of the USSR Academy of Sciences*, pages 293–294, 1962.

[77] Michael Scott. Optimal Irreducible Polynomials for GF(2^m) Arithmetic. Cryptology ePrint Archive, Report 2007/192, 2007.

[78] Diego Aranha, Julio López e Darrel Hankerson. High-speed parallel software implementation of the η_T pairing. *Topics in Cryptology - CT-RSA 2010*, LNCS 5985:89–105, 2010.

[79] Miracl Multiprecision Integer e Rational Arithmetic C/C++ Library. Disponível em `http://www.shamus.ie`. Último acesso em agosto de 2010.

[80] Benoit Chevallier-Mames, Jean-Sebastien Coron, Noel McCullagh, David Naccache e Michael Scott. Secure Delegation of Elliptic-Curve Pairing. Cryptology ePrint Archive, Report 2005/150, 2005.

[81] Yuto Kawahara, Tsuyoshi Takagi e Eiji Okamoto. Efficient Implementation of Tate Pairing on a Mobile Phone using Java. Cryptology ePrint Archive, Report 2006/299, 2006.

[82] Bo Gyeong Kang, Moon S. Lee e Je H. Park. Efficient Delegation of Pairing Computation. Cryptology ePrint Archive, Report 2005/259, 2005.

[83] Hong-Bin Tsai. Secure delegation of pairing computation. Disponível em `http://fractal.ee.ntu.edu.tw/~hbtsai/Secure_Delegation_of_Pairing_Computation.pdf`, 2005. Não publicado. Último acesso em agosto de 2010.

[84] Patrick P. Tsang, Sherman S. M. Chow e Sean W. Smith. Batch pairing delegation. In *IWSEC'07: Proceedings of the Security 2nd international conference on Advances in information and computer security*, pages 74–90, Berlin, Heidelberg, 2007. Springer-Verlag.

[85] American National Standards Institute. ANSI X9.62. Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA), 1999.

[86] National Institute of Standards e Technology. FIPS 186-2 Digital Signature Standard (DSS). Federal Information Processing Standards Publication 186-2, 2000.

[87] Alfred J Menezes, Scott A. Vanstone e Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996.

Apêndice A

Algoritmos de criptografia sobre curvas elípticas

A.1 Assinatura digital - Elliptic Curve Digital Signature Algorithm (ECDSA)

O esquema de assinaturas digitais conhecido como *Elliptic Curve Digital Signature Algoritm* (ECDSA) é padronizado pelo ANSI X9.62 [85], FIPS 186-2 [86] and IEEE 1363-2000 [73].

O primeiro passo no ECDSA é o estabelecimento dos parâmetros públicos, representados por $D = (m, f(x), a, b, P(x_{_P}, y_{_P}), n, h)$ para corpos binários, onde m denota a ordem do corpo, $f(x)$ é um polinômio irredutível, a e b são coeficientes que definem a equação da curva elíptica, $P\,(x_{_P}, y_{_P})$ é chamado de ponto base de ordem n e cofator h [14]. Cada entidade nesse esquema possui um par de chaves dado pela chave privada $d_i \in [1, n-1]$ e pela chave pública $Q_i(x_{_{Q_i}}, y_{_{Q_i}}) = d_i P(x_{_P}, y_{_P})$. A geração de assinatura ocorre de acordo com o Algoritmo A.1, onde $H(\cdot)$ denota uma operação de hash criptográfico [87], cujo resultado possui n bits.

A verificação da assinatura no ECDSA é efetuada de acordo com o Algoritmo A.2.

A.2 Protocolo de troca de chaves - Elliptic Curve Diffie-Hellman (ECDH)

Uma das maneiras de aproveitar as vantagens dos algoritmos de criptografia simétrica e assimétrica é através dos protocolos de troca de chaves, como o protocolo Diffie-Hellman [18], que permite que dois participantes estabeleçam uma chave secreta em um canal inseguro. Essa chave

Algoritmo A.1 ECDSA - Geração de assinatura [14]

Entrada: Chave privada d_i, mensagem $\mathcal{M}$ e parâmetros públicos $D = (m, f(x), a, b, P(x_P, y_P), n, h)$.
Saída: Assinatura (r, s).

1: Gere um número aleatório $k \in [1, n-1]$.
2: $T(x_T, y_T) \leftarrow kP(x_P, y_P)$.
3: $r \leftarrow x_T \mod n$.
4: **Se** $r = 0$ **então**
5: Vá para passo 1.
6: $e \leftarrow H(\mathcal{M})$.
7: $s \leftarrow k^{-1}(e + d_i r) \mod n$.
8: **Se** $s = 0$ **então**
9: Vá para passo 1.
10: **Retorne** (r, s).

Algoritmo A.2 ECDSA - Verificação de assinatura [14]

Entrada: Chave pública Q_i, mensagem $\mathcal{M}$, assinatura (r, s) e parâmetros públicos $D = (m, f(x), a, b, P(x_P, y_P), n, h)$.
Saída: Assinatura válida ou inválida.

1: **Se** r and s are not integers $\in [1, n-1]$ **então**
2: **Retorne** "Assinatura inválida".
3: $e \leftarrow H(\mathcal{M})$.
4: $w \leftarrow s^{-1} \mod n$.
5: $u_1 \leftarrow ew \mod n$.
6: $u_2 \leftarrow rw \mod n$.
7: $T(x_T, y_T) \leftarrow u_1 P(x_P, y_P) + u_2 Q_i(x_{Q_i}, y_{Q_i})$.
8: **Se** $T = \mathcal{O}$ **então**
9: **Retorne** "Assinatura inválida".
10: $v \leftarrow x_T \mod n$.
11: **Se** $v = r$ **então**
12: **Retorne** "Assinatura válida".
13: **Senão**
14: **Retorne** "Assinatura inválida".

secreta é então utilizada para cifrar a informação subsequente utilizando algoritmos de criptografia simétrica.

O protocolo de troca de chaves de Diffie-Hellman pode ser adaptado para utilizar pontos de curvas elípticas, que é o chamado *Elliptic Curve Diffie-Hellman* (ECDH). Para ilustrar esse protocolo, seja (d_A, Q_A) o par de chaves privada e pública, respectivamente, da entidade A. O escalar d_A pertencente ao corpo finito $\mathbb{F}_q$, Q_A é um ponto da curva $E(\mathbb{F}_q)$ e $Q_A = d_A P$, onde P é um ponto público da curva elíptica $E(\mathbb{F}_q)$. De modo análogo, a entidade B também possui seu par de chaves (d_B, Q_B). Se A

e B estão separados por um canal inseguro e desejam compartilhar entre si uma chave secreta, eles devem executar os passos a seguir.

1. A calcula o ponto de curva elíptica $P_{_A} = d_{_A} Q_{_B}$.

2. B calcula o ponto de curva elíptica $P_{_B} = d_{_B} Q_{_A}$.

Ao final desses dois passos, A e B terão obtido o mesmo ponto da curva, já que $P_{_A} = d_{_A} Q_{_B} = d_{_A} d_{_B} P = d_{_B} d_{_A} P = d_{_B} Q_{_A} = P_{_B}$. A chave secreta pode ser dada pela coordenada x do ponto resultante ou então pelo hash desse valor [73]. O protocolo é considerado seguro já que não há exposição de dados, exceto as chaves públicas das entidades participantes. As chaves privadas são mantidas em segredo, já que, pelo Elliptic Curve Discrete Logarithm Problem, não é computacionalmente viável calcular d_i dados o ponto Q_i, que representa a chave pública, e o ponto público P.

Apêndice B

Algoritmo *Montgomery's Ladder*

B.1 Invariante do Algoritmo *Montgomery's Ladder*

Para facilitar o entendimento da invariante do Algoritmo *Montgomery's Ladder*, o mesmo é reproduzido no Algoritmo B.1.

Algoritmo B.1 Multiplicação escalar usando o método Montgomery's Ladder [62]

Entrada: Ponto P_0, escalar positivo $s = (s_{n-1}, s_{n-2}, \ldots, s_0)_2$, $n \geq 2$.
Saída: Ponto sP_0.

1: $R \leftarrow P_0$, $S \leftarrow 2P_0$.
2: **Para** $i \leftarrow n-2$ **até** 0 **faça**
3: **Se** $s_i = 0$ **então**
4: $S \leftarrow R + S$, $R \leftarrow 2R$.
5: **Senão**
6: $R \leftarrow R + S$, $S \leftarrow 2S$.
7: **Retorne** R.

Lema 1. *Em toda iteração do algoritmo* Montgomery's Ladder *para calcular sP_0, onde P_0 é o ponto que está sendo multiplicado e s é um escalar positivo cuja representação binária é dada por $s = (s_{n-1}, s_{n-2}, \ldots, s_0)_2$, a diferença entre as variáveis S e R é constante e igual ao ponto P_0.*

Demonstração. A base da indução é provada pela construção do algoritmo, já que a variável R é inicializada com P_0 e a variável S é inicializada com $2P_0$. Portanto, vale a diferença $S - R = P_0$ no início das operações.

Agora vamos supor que durante uma iteração $0 \leq i \leq n-2$ tenhamos $R = kP_0$ e $S = (k+1)P_0$, onde $1 \leq k \leq s$. Se tal suposição for verdadeira, então teremos $S - R = P_0$.

Para verificar se na iteração $i + 1$ a diferença $S - R = P_0$ se mantém, devemos considerar dois casos:

- se $s_i = 0$: os valores das variáveis temporárias são atualizados de modo que $S = (k + k + 1)P_0 = (2k + 1)P_0$ e $R = 2kP_0$. Vale, portanto, a relação $S - R = P_0$.

- se $s_i = 1$: os valores das variáveis temporárias são atualizados de modo que $R = (k + k + 1)P_0 = (2k + 1)P_0$ e $S = 2(k + 1)P_0 = (2k + 2)P_0$. Vale, portanto, a relação $S - R = P_0$.

Ao final das iterações, tem-se $i = 0$ e, pelas condições apresentadas anteriormente, terá sido mantida a relação $S - R = P_0$.

□