Implementação Eficiente em Software de Criptossistemas de Curvas Elípticas

Julio César López Hernández

Tese de Doutorado

Instituto de Computação
Universidade Estadual de Campinas

Implementação Eficiente em Software de Criptossistemas de Curvas Elípticas

Julio César López Hernández

28 de Abril de 2000

Banca Examinadora:

- Prof. Dr. Ricardo Dahab
  Universidade Estadual de Campinas (Orientador)
- Prof. Dr. Guido C. S. Araújo
  Universidade Estadual de Campinas
- Prof. Dr. Cláudio L. Lucchesi
  Universidade Estadual de Campinas
- Prof. Dr. Daniel Panario
  Universidade de São Paulo
- Prof. Dr. Routo Terada
  Universidade de Toronto

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

López Hernández, Julio César

L881i Implementação eficiente em software de criptossistemas de curvas elípicas / Julio César López Hernández -- Campinas, [S.P. :s.n.], 2000.

Orientador : Ricardo Dahab

Tese (doutorado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Curvas elípticas. 2. Criptografia. I. Dahab, Ricardo. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Implementação Eficiente em Software de Criptossistemas de Curvas Elípticas

Este exemplar corresponde à redação final da Tese devidamente corrigida e defendida por Julio César López Hernández e aprovada pela Banca Examinadora.

Campinas, 28 de Abril de 2000.

Prof. Dr. Ricardo Dahab
Universidade Estadual de Campinas
(Orientador)

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

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 28 de abril de 2000, pela Banca Examinadora composta pelos Professores Doutores:

Prof. Dr. Claudio Leonardo Lucchesi
IC - UNICAMP

Prof. Dr. Daniel Panario
University of Toronto

Prof. Dr. Routo Terada
IME - USP

Prof. Dr. Guido Costa Souza de Araújo
IC - UNICAMP

Prof. Dr. Ricardo Dahab
IC - UNICAMP

Resumo

A criptografia de chave-pública é, reconhecidamente, uma ferramenta muito útil para prover requisitos de segurança tais como confidencialidade, integridade, autenticidade e não-repudio, parte integrante das comunicações.

A principal vantagem dos criptossistemas de curvas elípticas (CCE) em relação a outras tecnologias de chave-pública concorrentes tais como RSA e DSA, é que parâmetros significativamente menores podem ser usados nos CCE com o mesmo nível de segurança. Essa vantagem é especialmente importante em aplicações em ambientes computacionais limitados como cartões inteligentes, telefones celulares, computadores de bolso e pagers.

De um ponto de vista prático, a implementação dos CCE apresenta vários desafios. Uma aplicação baseada nos CCE precisa que várias escolhas sejam feitas tais como o nível de segurança, algoritmos para implementar a aritmética no corpo finito subjacente, algoritmos para implementar a aritmética na curva elíptica, protocolos de curvas elípticas e a plataforma computacional. Essas escolhas podem ter um grande impacto no desempenho da aplicação resultante.

Esta dissertação trata do desenvolvimento de algoritmos eficientes para implementação em software de criptossistemas de curvas elípticas sobre o corpo finito $\mathbb{F}_{2^m}$. Neste contexto, foram desenvolvidos métodos eficientes para implementar a aritmética no corpo finito $\mathbb{F}_{2^m}$, e para calcular múltiplos de um ponto elíptico, a operação fundamental da criptografia pública baseada em curvas elípticas. Nesta dissertação também foi abordado o problema da implementação eficiente em software dos algoritmos propostos, em diferentes plataformas computacionais tais como PCs, estações de trabalho, e em dispositivos limitados como o pager da RIM.

Abstract

It is widely recognized that public-key cryptography is an important tool for providing security services such as confidentiality, data integrity, authentication and non-repudiation, which are requirements present in almost all communications. The main advantage of elliptic curve cryptography (ECC) over competing public-key technologies such as RSA and DSA, is that significantly smaller parameters can be used in ECC, but with equivalent levels of security. This advantage is especially important for applications on constrained environments such as smart cards, cell phones, personal device assistants, and pagers.

From a practical point of view, the implementation of ECC presents various challenges. An ECC-based application requires that several choices be made including the security level, algorithms for implementing the finite field arithmetic, algorithms for implementing the elliptic group operation, elliptic curve protocols, and the computer platform. These choices may have a significant impact on the performance of the resulting application.

This dissertation focuses on developing efficient algorithms for software implementation of ECC over $\mathbb{F}_{2^m}$. In this framework, we study different ways of efficiently implementing arithmetic in $\mathbb{F}_{2^m}$, and computing an elliptic scalar multiplication, the central operation of public-key cryptography based on elliptic curves. We also concentrate on the software implementation of these algorithms for different platforms including PCs, workstations, and constrained devices such as the RIM interactive pager.

This dissertation is a collection of five papers written in English, with an introduction and conclusions written in Portuguese.

Dedicatória

Aos meus pais Pablo Julio López e Deyanira Hernández, por terem sempre me incentivado e apoiado para adquirir o gosto pelos estudos e pelo fascinante mundo dos livros e dos números.

Aos meus irmaõs e irmãs com quem partilhei alegrias e sofrimentos e a quem devo as mais importantes lições de fraternidade; especialmente à minha irmã gêmea Lyda Cristina que sempre me acompanhou com entusiasmo.

À minha esposa e companheira Lucila, por todo o amor, dedicação e paciência. Também por me trazer de volta do mundo dos números!.

Agradecimentos

No Brasil:

- Ao professor Cláudio Lucchesi, que me recebeu e orientou no primeiro semestre e posteriormente fez parte integrante da Banca Examinadora.
- Ao professor Ricardo Dahab, meu orientador, pela confiança, amizade e trabalho conjunto.
- Aos professores da Banca Examinadora: Routo Terada, Cláudio Lucchesi, Daniel Panario, Guido Araújo, pelas valiosas sugestoẽs.
- Aos professores, funcionários e colegas do Instituto de Computação, pela gentileza com que sempre me receberam.
- À coordenadoria de aperfeiçoamento de pessoal de nível superior (CAPES), pela bolsa de doutorado.
- Aos amigos de doutorado Ana, Bruno, Maria Emilia, Luiz, José Roberto, Luiz Mariano, Jerônimo, por tornarem o ambiente de trabalho mais acolhedor.
- Aos meus sogros, cunhado e família pela gentileza e por terem me recebido com tanto carinho.
- À comunidade de estudantes colombianos na Unicamp, por compartilharem das novas experiências culturais vivenciadas.

No Canadá:

- Ao professor Alfred Menezes, pela oportunidade de visitar o centro de criptografia aplicadada (CACR) da Universidade de Waterloo e pela acolhida e orientação de meu trabalho de pesquisa.
- Aos colegas no centro CACR Andreas Stein, Edlyin Teske, Mike Jacobson, Berit Skjernaa, pelas sugestões e solidariedade.

- Aos colegas do grupo de pesquisa Michael Brown, Donny Cheung, Darrel Hankerson, Michael Kirkup, William Lewis, por terem tido importante papel no desenvolvimento de minha pesquisa.

Na Colômbia:

- Aos colegas do departamento de Ciência da Computação da Universidade do Valle, pela amizade e apoio recebido.
- À Universidade do Valle, pelo suporte institucional e apoio financeiro.
- À amiga Ruby Grisales, por toda sua colaboração.
- Ao professor Jürgen Tischer, por sua orientação e estimulo para continuar estudos avançados em Computação.

Finalizando, um especial agradecimento, ao povo brasileiro que contribui para a formação dos pesquisadores nas Universidades Públicas.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

A criptografia tem sido utilizada há séculos em contextos militares e diplomáticos para prover sigilo de informações. Na era moderna das comunicações eletrônicas, os requisitos de segurança tais como confidencialidade, integridade, autenticação e não-repúdio tem assumido um papel muito importante. Também para o provimento desses requisitos, a criptografia tem se mostrado muito útil.

O conceito revolucionário de criptografia de chave pública foi apresentado por Diffie e Hellman, em 1976, no artigo "New directions in cryptography" [26]. Embora os autores não tenham apresentado uma implementação prática para essa idéia, o novo conceito gerou uma intensa atividade de pesquisa na procura de sistemas criptográficos práticos de chave pública. Ainda nesse artigo, é apresentado um engenhoso protocolo para troca de chaves, cuja segurança está baseada na suposta intratabilidade do *problema do logaritmo discreto módulo um número primo* (PLD).

Pouco tempo depois, Ron Rivest, Adi Shamir, e Len Adleman [85], descobriram o primeiro esquema de chave pública para assinatura e ciframento, denominado RSA. O sistema RSA está baseado em outro problema supostamente difícil, a *fatoração de números inteiros muito grandes* (FNI). Atualmente, o melhor algoritmo conhecido para resolver esse problema é o "number field sieve" [79], que tem tempo de execução sub-exponencial.

Em 1984, ElGamal [27] apresentou um outro criptossistema de chave pública baseado no PLD. Esse criptossistema tem sido refinado e incorporado em vários protocolos e uma de suas extensões forma a base do algoritmo de assinatura digital americano (DSA).

A descoberta de vários algoritmos eficientes para resolver o problema do logaritmo discreto nos grupos multiplicativos $\mathbb{Z}_p^*$, e $\mathbb{F}_{2^m}{}^*$, durante os anos de 1978 a 1984, forçou um aumento no tamanho das chaves utilizadas no protocolo Diffie-Hellman, tornando-o mais caro e, em conseqüência, menos atraente. Esta situação levou vários pesquisadores à observação de que tanto o protocolo de troca de chaves de Diffie-Hellman como os sistemas do tipo ElGamal, podem ser estendidos a grupos abelianos arbitrários [51]. Assim, os

esforços de pesquisa foram orientados para a investigação de grupos abelianos onde o problema do logaritmo discreto parece ser intratável e as operações no grupo possam ser implementadas eficientemente em software ou em hardware.

Em 1985, N. Koblitz [47] e V. Miller [68], de forma independente, propuseram utilizar o grupo de pontos de uma curva elíptica sobre um corpo finito para implementar criptossistemas de chave pública. Esses sistemas, denominados *criptossistemas de curvas elípticas* (CCE), têm sua segurança baseada na suposta intratabilidade do *problema do logaritmo discreto no grupo de pontos de uma curva elíptica* (PLDCE).

Nos últimos anos, muitos avanços foram feitos na área dos CCE. O melhor algoritmo conhecido para o problema do logaritmo discreto em curvas elípticas é de tempo exponencial [78]. Embora existam alguns ataques (algoritmos de tempo sub-exponencial [67] e polinomial [87, 93, 95]) para certos tipos de curvas elípticas, esses ataques podem ser evitados facilmente por meio de testes simples, descritos em vários padrões industriais [5, 42].

O fato de não se conhecer um algoritmo geral de tempo sub-exponencial para o PLDCE, possibilita que parâmetros menores sejam usados nos CCE, relativos aos sistemas baseados no PLD. Por exemplo, NIST [72] recomenda o uso de chaves de 3072 bits nos sistemas baseados no PLD e RSA para obter-se um nível de segurança comparável ao fornecido por um algoritmo de chave simétrica de 128 bits. Entretanto, nos CCE são suficientes chaves de 256 bits para obter-se o mesmo nível de segurança.

Algumas vantagens que resultam do fato de usar-se pequenos parâmetros nos CCE incluem velocidade, chaves e certificados pequenos. Para certas aplicações, onde a capacidade de processamento, a potência computacional, o espaço de armazenamento e a banda-passante estejam limitados, os CCE superam outros sistemas de chave pública. Por todas estas razões, os CCE tem tido crescente aceitação, nos setores industriais, como alternativa aos já estabelecidos RSA, protocolo de troca de chaves Diffie-Hellman e DSA.

A criptografia de chave pública, nos últimos anos, tem-se convertido numa das tecnologias básicas para a construção de aplicações muito sensíveis à segurança, tais como correio eletrônico, eleições eletrônicas e comércio eletrônico.

A implementação eficiente da criptografia baseada em curvas elípticas depende de vários fatores como o nível de segurança desejado, a plataforma computacional (software, hardware, ou firmware), restrições no ambiente computacional (velocidade do processador, tamanho do código, memória, banda-passante), métodos eficientes para a aritmética no corpo (soma, multiplicação, cálculo de quadrados e inversos, solução de equações quadráticas) e algoritmos para implementar a aritmética na curva elíptica (soma de pontos e multiplicação escalar).

Nesta tese, nos concentramos na implementação em software de curvas elípticas sobre

o corpo finito $\mathbb{F}_{2^m}$. Vários algoritmos foram desenvolvidos para acelerar a computação da operação central dos CCE, a multiplicação de um ponto elíptico por um número inteiro grande. Nosso trabalho também inclui uma implementação prática das curvas NIST [72] para diferentes plataformas computacionais como PCs, estações de trabalho SPARC e o pager RIM bidirecional.

1.1 Contribuições da Tese

As principais contribuições desta dissertação são:

- Desenvolvimento de um algoritmo eficiente para multiplicação no corpo finito $\mathbb{F}_{2^m}$, cujos elementos são representados usando uma base polinomial. Esse algoritmo é orientado para implementações em software de curvas elípticas sobre $\mathbb{F}_{2^m}$. (Capítulo 3.)

- Melhoramento de um algoritmo desenvolvido por J. Guajardo e C. Paar [37]. Apresentamos fórmulas mais eficientes para calcular duplicações consecutivas de um ponto elíptico em curvas elípticas definidas sobre $\mathbb{F}_{2^m}$. (Capítulo 4)

- Desenvolvimento de uma fórmula nova para duplicar pontos elípticos em curvas definidas sobre $\mathbb{F}_{2^m}$. Baseado nessa fórmula, propusemos um sistema de coordenadas projetivas para a aritmética de uma curva elíptica sobre $\mathbb{F}_{2^m}$. Essa formulação é mais eficiente do que a dos métodos conhecidos. (Capítulo 4)

- Desenvolvimento de um algoritmo eficiente para multiplicação escalar sobre curvas elípticas definidas sobre $\mathbb{F}_{2^m}$. Esse método é atraente tanto para implementações em software como em hardware de curvas aleatórias sobre $\mathbb{F}_{2^m}$. (Capítulo 5).

- Projeto e implementação de uma biblioteca escrita em linguagem C, para suporte de curvas elípticas sobre os corpos $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$ e $\mathbb{F}_{2^{283}}$ (recomendados por NIST [72]), em diferentes plataformas computacionais que incluem PCs, PCs de bolso (PalmPilot) e um pager RIM bidirecional. (Capítulo 6).

- Incorporação da biblioteca de curvas elípticas sobre $\mathbb{F}_{2^m}$ numa implementação do sistema criptográfico PGP no pager bidirecional RIM. (Capítulo 6)

1.2 Estrutura da Tese

Esta dissertação é uma coletânea de artigos científicos obtidos durante o desenvolvimento do projeto de pesquisa. O restante deste texto está organizado da seguinte forma:

O Capítulo 2 contém uma breve introdução ao estudo dos criptossistemas de curvas elípticas com ênfase na implementação em software de curvas elípticas definidas sobre o corpo finito $\mathbb{F}_{2^m}$; vários algoritmos eficientes são apresentados para calcular múltiplos de um ponto elíptico e para implementar a aritmética do corpo finito $\mathbb{F}_{2^m}$, usando uma base polinomial.

O Capítulo 3 apresenta um algoritmo rápido para multiplicação no corpo finito $\mathbb{F}_{2^m}$, onde os elementos são representados usando uma base polinomial. O novo algoritmo é crucial para obter-se uma implementação eficiente em software dos criptossistemas de curvas elípticas definidos sobre os corpos finitos de característica 2. Esse algoritmo é a base da implementação em software das curvas NIST, apresentada no Capítulo 6.

O Capítulo 4 apresenta alguns algoritmos eficientes para a implementação da aritmética no grupo de uma curva elíptica definida sobre $\mathbb{F}_{2^m}$. Em particular, um esquema novo de coordenadas projetivas é apresentado.

O Capítulo 5 traz um método eficiente para multiplicar pontos de uma curva elíptica. Esse algoritmo possui algumas características que o tornam atraente para implementações em hardware ou software de curvas elípticas aleatórias definidas sobre $\mathbb{F}_{2^m}$.

O Capítulo 6 descreve uma implementação prática de curvas elípticas definidas sobre $\mathbb{F}_{2^m}$. O objetivo central foi projetar uma aplicação prática de correio eletrônico com segurança, baseada no sistema criptográfico PGP, para ser executada numa plataforma computacional (com recursos limitados) como o pager RIM bidirecional. Substituímos os algoritmos de chave pública do PGP pelos algoritmos de curvas elípticas, tais como ECDSA (algoritmo análogo ao DSA) e ECAES (algoritmo para ciframento baseado no ElGamal). A aplicação está baseada nas curvas NIST sobre o corpo finito $\mathbb{F}_{2^m}$.

O Capítulo 7 contêm conclusões e alguns comentários para futuros trabalhos.

Capítulo 2

Introdução a Criptossistemas de Curvas Elípticas

Neste capítulo estudamos os conceitos fundamentais em curvas elípticas e a construção de criptossistemas baseados em curvas elípticas (CCE). Abordamos os principais problemas associados à implementação eficiente dos CCE, e apresentamos um resumo dos algoritmos básicos para implementação em software da aritmética no corpo finito $\mathbb{F}_{2^m}$ e a aritmética no grupos de pontos de uma curva elíptica definida sobre $\mathbb{F}_{2^m}$.

O trabalho apresentado neste capítulo foi publicado como relatório técnico No. IC-00-10 no Instituto de Computação, UNICAMP, e submetido ao Journal of Universal Computer Science.

An Overview of Elliptic Curve Cryptography

Julio López and Ricardo Dahab
State University of Campinas
Campinas, SP, Brazil
{julioher,dahab}@dcc.unicamp.br

Abstract

Elliptic curve cryptography (ECC) was introduced by Victor Miller and Neal Koblitz in 1985. ECC proposed as an alternative to established public-key systems such as DSA and RSA, have recently gained a lot attention in industry and academia. The main reason for the attractiveness of ECC is the fact that there is no sub-exponential algorithm known to solve the discrete logarithm problem on a properly chosen elliptic curve. This means that significantly smaller parameters can be used in ECC than in other competitive systems such RSA and DSA, but with equivalent levels of security. Some benefits of having smaller key sizes include faster computations, and reductions in processing power, storage space and bandwidth. This makes ECC ideal for constrained environments such as pagers, PDAs, cellular phones and smart cards. The implementation of ECC, on the other hand, requires several choices such as the type of the underlying finite field, algorithms for implementing the finite field arithmetic, the type of elliptic curve, algorithms for implementing the elliptic group operation, and elliptic curve protocols. Many of these selections may have a major impact on the overall performance. In this paper we present a selective overview of the main methods and techniques used for practical implementations of elliptic curve cryptosystems. We also present a summary of the most recent reported software implementations of ECC.

Key words. Elliptic curve cryptography, finite fields, elliptic scalar multiplication.

2.1 Introduction

In 1985, Victor Miller [68] and N. Koblitz [47], independently, proposed a public-key cryptosystem analogue of the ElGamal schemes [27] in which the group $\mathbb{Z}_p^*$ is replaced by

the group of points on an elliptic curve defined over a finite field. The main attraction of elliptic curve cryptography (ECC) over competing technologies such as RSA and DSA is that the best algorithm known for solving the underlying hard mathematical problem in ECC (the elliptic curve discrete logarithm problem (ECDLP)) takes fully exponential time. On the other hand, the best algorithms known for solving the underlying hard mathematical problems in RSA and DSA (the integer factorization problem, and the discrete logarithm problem, respectively) take sub-exponential time. This means that significantly smaller parameters can be used in ECC than in other systems such as RSA and DSA, but with equivalent levels of security. A typical example of the size in bits of the keys used in different public-key systems, with a comparable level of security (against known attacks), is that a 160-bit ECC key is equivalent to RSA and DSA with a modulus of 1024 bits.

The lack of a sub-exponential attack on ECC offers potential reductions in processing power, storage space, bandwidth and electrical power. These advantages are specially important in applications on constrained devices such as smart cards, pagers, and cellular phones.

From a practical point of view, the performance of ECC depends mainly on the efficiency of finite field computations and fast algorithms for elliptic scalar multiplications. In addition to the numerous known algorithms for these computations, the performance of ECC can be sped up by selecting particular underlying finite fields and/or elliptic curves. Examples of finite fields are $\mathbb{F}_{2^m}$ (for hardware and software implementations) and $\mathbb{F}_p$, where p is a special prime (e.g., a Mersenne prime or a generalized Mersenne prime, see [98]). Examples of families of curves that offer computational advantages for computing a scalar multiplication include Koblitz curves over $\mathbb{F}_{2^m}$. Thus, a fast implementation of a security application based on ECC requires several choices, any of which can have a major impact on the overall performance.

The remainder of this paper is organized as follows. A short introduction to finite field arithmetic is provided in Section 2.2. A brief introduction to elliptic curves is presented in Section 2.3. A list of the main known attacks on the elliptic curve discrete logarithm problem (ECDLP) is provided in Section 2.4. In Section 2.5, we describe several algorithms for computing a scalar multiplication which is the central operation of ECC. Finally, some implementation issues are considered in Section 2.6.

2.2 Finite fields

In this section we present the definition of groups and finite fields. These mathematical structures are fundamental for the construction of an elliptic curve cryptosystem.

A *group* is an algebraic system consisting of a set G together with a binary operation

$\diamond$ defined on G satisfying the following axioms:

- closure: for all x, y in G we have $x \diamond y \in G$;
- associativity: for all x, y and z in G we have $(x \diamond y) \diamond z = x \diamond (y \diamond z)$;
- identity: there exists an e in G such that $x \diamond e = e \diamond x = x$ for all x in G;
- inverse: for all x in G there exists y in G such that $x \diamond y = y \diamond x = e$.

If in addition, the binary operation $\diamond$ satisfies the abelian property:

- abelian: for all x, y in G we have $x \diamond y = y \diamond x$,

then we say that the group G is abelian.

A *finite field* is an algebraic system consisting of a finite set F together with two binary operations $+$ and $\times$, defined on F, satisfying the following axioms:

- F is an abelian group with respect to "+";
- $F \setminus \{0\}$ is an abelian group with respect to "$\times$";
- distributive: for all x, y and z in F we have:

$$\begin{aligned} x \times (y + z) &= (x \times y) + (x \times z) \\ (x + y) \times z &= (x \times z) + (y \times z). \end{aligned}$$

The *order* of a finite field is the number of elements in the field. A fundamental result on the theory of finite fields (see [63]), characterizes the existence of finite fields: there exists a finite field of order q if and only if q is a prime power. In addition, if q is a prime power, then there is essentially only one finite field of order q; this field is denoted by $\mathbb{F}_q$ or $GF(q)$. There are, however, many ways of representing the elements of $\mathbb{F}_q$, and some representations may lead to more efficient implementations of the field arithmetic in hardware or in software.

If $q = p^m$, where p is a prime and m is a positive integer, then p is called the *characteristic* of $\mathbb{F}_q$ and m is called the *extension degree* of $\mathbb{F}_q$. Most standards which specify ECC restrict the order of the underlying finite field to be an odd prime ($q = p$) or a power of 2 ($q = 2^m$).

2.2.1 The finite field $\mathbb{F}_p$

Let p be a prime number. The finite field $\mathbb{F}_p$, called a *prime field*, consists of the set of integers

$$\{0, 1, 2, \ldots, p-1\}$$

with the following arithmetic operations:

- *Addition:* If $a, b \in \mathbb{F}_p$, then $a + b = r$, where r is the remainder of the division of $a + b$ by p and $0 \leq r \leq p - 1$. This operation is called *addition modulo p.*
- *Multiplication:* If $a, b \in \mathbb{F}_p$, then $a \cdot b = s$, where s is the remainder of the division of $a \cdot b$ by p and $0 \leq s \leq p - 1$. This operation is called *multiplication modulo p.*

There are certain primes p for which the modular reduction can be computed very efficiently. For example, let p be the prime $2^{192} - 2^{64} - 1$. To reduce a positive integer $n < p^2$, write

$$n = \sum_{j=0}^{5} A_j \cdot 2^{64j}.$$

Then

$$n \equiv T + S_1 + S_2 + S_3 \pmod{p},$$

where

$$\begin{array}{lcl} T &=& A_2 \cdot 2^{128} + A_1 \cdot 2^{64} + A_0 \\ S_1 &=& A_3 \cdot 2^{64} + A_3 \\ S_2 &=& A_4 \cdot 2^{128} + A_4 \cdot 2^{64} \\ S_3 &=& A_5 \cdot 2^{128} + A_5 \cdot 2^{64} + A_5. \end{array}$$

Thus, the integer reduction by p can be replaced by three additions (mod p), which are much faster. The prime number p is an example of a family of primes called *generalized Mersene numbers*, recently introduced by Solinas [98]. For more examples of primes that are well suited for machine implementation, see [98] and [72]. Several techniques for implementing the finite field arithmetic in $\mathbb{F}_p$ are described in [46, 66, 14, 43, 25, 41].

2.2.2 The finite field $\mathbb{F}_{2^m}$

The finite field $\mathbb{F}_{2^m}$, called a *binary finite field*, can be viewed as a vector space of dimension m over $\mathbb{F}_2$. That is, there exists a set of m elements $\{\alpha_0, \alpha_1, \ldots, \alpha_{m-1}\}$ in $\mathbb{F}_{2^m}$ such that each $a \in \mathbb{F}_{2^m}$ can be written uniquely in the form

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

The set $\{\alpha_0, \alpha_1, \ldots, \alpha_{m-1}\}$ is called a *basis* of $\mathbb{F}_{2^m}$ over F_2. We can then represent a as a binary vector $(a_0, a_1, \ldots, a_{m-1})$. We now introduce two of the most common bases of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$: *polynomial bases* and *normal bases.*

Polynomial basis. Let $f(x) = x^m + \sum_{i=0}^{m-1} f_i x^i$ (where $f_i \in \{0,1\}$, for $i = 0, 1 \ldots, m-1$) be an irreducible polynomial of degree m over $\mathbb{F}_2$; $f(x)$ is called the *reduction polynomial*. For each reduction polynomial, there exists a polynomial basis representation. In such a representation, each element of $\mathbb{F}_{2^m}$ corresponds to a binary polynomial of degree less than m. That is, for $a \in \mathbb{F}_{2^m}$ there exist m numbers $a_i \in \{0,1\}$ such that

$$a = a_{m-1}x^{m-1} + \cdots + a_1 x + a_0.$$

The field element $a \in \mathbb{F}_{2^m}$ is usually denoted by the bit string $(a_{m-1} \ldots a_1 a_0)$ of length m. The following operations are defined on the elements of $\mathbb{F}_{2^m}$ when using a polynomial representation with reduction polynomial $f(x)$. Assume that $a = (a_{m-1} \ldots a_1 a_0)$ and $b = (b_{m-1} \ldots b_1 b_0)$.

- *Addition:* $a + b = c = (c_{m-1} \ldots c_1 c_0)$, where $c_i = (a_i + b_i) \bmod 2$. That is, addition corresponds to bitwise exclusive-or.
- *Multiplication:* $a \cdot b = c = (c_{m-1} \ldots c_1 c_0)$, where $c(x) = \sum_{i=0}^{m-1} c_i x^i$ is the remainder of the division of the polynomial $(\sum_{i=0}^{m-1} a_i x^i)(\sum_{i=0}^{m-1} b_i x^i)$ by $f(x)$.

The following procedure is commonly used to choose a reduction polynomial: if an irreducible *trinomial* $x^m + x^k + 1$ exists over $\mathbb{F}_2$, then the reduction polynomial $f(x)$ is chosen to be the irreducible trinomial with the lowest-degree middle term x^k.[1] If no irreducible trinomial exists, then select instead a *pentanomial* $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, such that k_1 has the minimal value; the value of k_2 is minimal for the given k_1; and k_3 is minimal for the given k_1 and k_2.

Normal basis. A *normal basis* of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$ is a basis of the form $\{\beta, \beta^2, \ldots, \beta^{2^{m-1}}\}$, where $\beta \in \mathbb{F}_{2^m}$. It is well known (see [63]) that such a basis always exists. Therefore, every element $a \in \mathbb{F}_{2^m}$ can be written as $a = \sum_{i=0}^{m-1} a_i \beta^{2^i}$, where $a_i \in \{0,1\}$. The field element a is usually denoted by the bit string $(a_0 a_1 \ldots a_{m-1})$ of length m. A normal basis representation of $\mathbb{F}_{2^m}$ has the computational advantage that squaring an element is a simple cyclic shift of the vector representation, an operation that is efficiently implemented in hardware. Multiplication of different elements, on the other hand, is in general a more complicated operation. Fortunately, for the particular class of normal bases called *Gaussian normal bases* (GNB), the field arithmetic operations can be implemented very efficiently [42]. The *type* T of a GNB is a positive integer measuring the complexity of the multiplication operation with respect to that basis; the smaller the type, the faster the multiplication.

[1] Although this selection may affect the speed of the almost inverse algorithm (see [25]), it allows for faster reduction modulo $f(x)$.

The existence of a Gaussian normal basis has been characterized in [71] and [8]. In particular, a GNB exists whenever m is not divisible by 8. In addition, if m is divisible by 8 and T is a positive integer, then a type T GNB for $\mathbb{F}_{2^m}$ exists if and only if $p = Tm+1$ is prime and $\gcd(Tm/k, m) = 1$, where k is the multiplicative order of 2 modulo p.

The finite field operations in $\mathbb{F}_{2^m}$, using a Gaussian normal basis of type T, are defined as follows. Assume that $a = (a_0 a_1 \dots a_{m-1})$ and $b = (b_0 b_1 \dots b_{m-1})$. Then:

- *Addition:* $a + b = c = (c_0 c_1 \dots c_{m-1})$, where $c_i = (a_i + b_i) \bmod 2$. That is, field addition is performed bitwise.

- *Squaring:* Since squaring is a linear operation in $\mathbb{F}_{2^m}$,

$$a^2 = (\sum_{i=0}^{m-1} a_i \beta^{2^i})^2 = \sum_{i=0}^{m-1} a_i \beta^{2^{i+1}} = \sum_{i=0}^{m-1} a_{i-1 \bmod m} \beta^{2^i} = (a_{m-1} a_0 a_1 \dots a_{m-2}).$$

  Hence squaring a finite field element is a simple rotation of the vector representation.

- *Multiplication:* Let $p = Tm+1$ and let $u \in \mathbb{F}_p$ be an element of order T. Define the sequence $F(1), F(2), \dots, F(p-1)$ by

$$F(2^i u^j \bmod p) = i \text{ for } 0 \le i \le m-1, 0 \le j \le T-1.$$

  For each l, $0 \le l \le m-1$, define A_l and B_l by

$$\begin{aligned} A_l &= \sum_{k=1}^{p-2} a_{F(k+1)+l}\, b_{F(p-k)+l}, \text{ and} \\ B_l &= \sum_{k=1}^{m/2} (a_{k+l-1}\, b_{m/2+k+l-1} + a_{m/2+k+l-1}\, b_{k+l-1}) + A_l. \end{aligned}$$

  Then $a \cdot b = c = (c_0 c_1 \dots c_{m-1})$, where

$$c_l = \begin{cases} A_l & \text{if } T \text{ is even,} \\ B_l & \text{if } T \text{ is odd,} \end{cases}$$

  for each $l, 0 \le l \le m-1$, where indices are reduced modulo m.

See [42] for a good survey on finite field algorithms using a normal basis in $\mathbb{F}_{2^m}$. Consult Agnew, Mullin and Vanstone [2] and Rosing [86] for a hardware and software implementation, respectively, of a normal basis in $\mathbb{F}_{2^m}$.

2.2.3 Finite field arithmetic in $\mathbb{F}_{2^m}$ using a polynomial basis

In this section we describe various bit-level algorithms for performing computations in the finite field $\mathbb{F}_{2^m}$ using a polynomial basis representation. These algorithms can be easily modified to obtain word-level algorithms, which are well suited for software implementations.

Addition. Addition in $\mathbb{F}_{2^m}$ is the usual addition of vectors over $\mathbb{F}_2$. That is, add the corresponding bits modulo 2.

Algorithm 1: bit-level method for addition in $\mathbb{F}_{2^m}$

INPUT: $a = (a_{m-1} \ldots a_1 a_0) \in \mathbb{F}_{2^m}$ and $b = (b_{m-1} \ldots b_1 b_0) \in \mathbb{F}_{2^m}$
OUTPUT: $c = a + b = (c_{m-1} \ldots c_1 c_0)$

1. **for** j **from** 0 **to** $m-1$ **do**
   Set $c_j \leftarrow (a_j + b_j) \bmod 2$
2. **return**(c).

Modular reduction. By the definition of multiplication in $\mathbb{F}_{2^m}$, the result of a polynomial multiplication or squaring has to be reduced modulo an irreducible polynomial of degree m. This reduction operation is particularly efficient when the irreducible polynomial $f(x)$ is a trinomial or a pentanomial. The following algorithm for computing $a(x) \bmod f(x)$ works by reducing the degree of $a(x)$ until it is less than m.

Algorithm 2: bit-level method for modular reduction in $\mathbb{F}_{2^m}$

INPUT: $a = (a_{2m-2} \ldots a_1 a_0)$ and $f = (f_m f_{m-1} \ldots f_1 f_0)$
OUTPUT: $c = a \bmod f$

1. **for** i **from** $2m-2$ **to** m **do**
   for j **from** 0 **to** $m-1$ **do**
   if $f_j \neq 0$ **then** $a_{i-m+j} \leftarrow a_{i-m+j} + a_i$
2. **return**$(c \leftarrow (a_{m-1} \ldots a_1 a_0))$.

Squaring. This operation can be calculated in an efficient way by observing that the square of a polynomial a is given by

$$a(x)^2 = (\sum_{i=0}^{m-1} a_i x^i)^2 = \sum_{i=0}^{m-1} a_i^2 x^{2i}.$$

This equation yields a simple algorithm:

Algorithm 3: bit-level method for squaring in $\mathbb{F}_{2^m}$

INPUT: $a = (a_{m-1} \ldots a_1 a_0)$ and $f = (f_m f_{m-1} \ldots f_1 f_0)$
OUTPUT: $c = a^2 \bmod f$

1. Set $t \leftarrow \sum_{i=0}^{m-1} a_i^2 x^{2i}$
2. Set $c \leftarrow t \bmod f$ //Use Algorithm 2
3. **return**(c).

A known technique for speeding up the computation in step 1 is to use a table lookup (see Schroeppel *et al* [89] for details).

Multiplication. The basic method for performing a multiplication in $\mathbb{F}_{2^m}$ is the "shift-and-add" method. It is analogous to the binary method for exponentiation, with the square and multiplication operations being replaced by the multiplication of a field element by x and field addition operations, respectively. Given $a \in \mathbb{F}_{2^m}$, the shift-left operation $xa(x) \bmod f(x)$ can be performed as follows

$$xa(x) \bmod f(x) = \begin{cases} \sum_{j=1}^{m-1} a_{j-1} x^j & \text{if } a_{m-1} = 0, \\ \sum_{j=1}^{m-1} (a_{j-1} + f_j) x^j + f_0 & \text{if } a_{m-1} \neq 0. \end{cases}$$

Then the steps of the "shift-and-add" method are given below.

Algorithm 4: "shift-and-add" method

INPUT: $a \in \mathbb{F}_{2^m}, b \in \mathbb{F}_{2^m}$ and $f = (f_m f_{m-1} \ldots f_1 f_0)$
OUTPUT: $c = ab \bmod f$

1. Set $c(x) \leftarrow 0$
2. **for** j **from** $m-1$ **to** 0 **do**
   Set $c(x) \leftarrow xc(x) \bmod f(x)$
   if $a_j \neq 0$ **then** Set $c(x) \leftarrow c(x) + b(x)$
3. **return**(c).

This method requires $m-1$ shift-left operations and m field additions on average. The speed of this method can be improved by using programming tricks such as *separated name variables* and *loop-unrolled code*. In [62] we have proposed a fast algorithm for multiplication that is significantly faster than the "shift-and-add" method, but requires some temporary storage.

Inversion. The basic algorithm for computing multiplicative inverses is the extended Euclidean algorithm. A high level description of this method is the following:

Algorithm 5: Extended Euclidean algorithm

INPUT: $a \in \mathbb{F}_{2^m}$ $(a \neq 0)$ and $f = (f_m f_{m-1} \cdots f_1 f_0)$
OUTPUT: $c = a^{-1} \bmod f$

1. Set $b_1(x) \leftarrow 1$, $b_2(x) \leftarrow 0$
   Set $p_1(x) \leftarrow a(x)$, $p_2(x) \leftarrow f(x)$
2. **while** degree$(p_1) \neq 0$ **do**
   if degree(p_1) < degree(p_2) **then**
   exchange p_1, p_2 and b_1, b_2
   Set $j \leftarrow$ degree(p_1)-degree(p_2)
   Set $p_1(x) \leftarrow p_1(x) + x^j p_2(x)$, $b_1(x) \leftarrow b_1(x) + x^j b_2(x)$
3. **return**$(c(x) \leftarrow b_1(x))$.

An alternative method for computing inverses, called the *almost inverse algorithm*, was proposed by Schroeppel *et al* [89]. This method works quite well when the reduction polynomial is a trinomial of the form $x^m + x^k + 1$ with $k > w$ and $m - k > w$, where w is the word size of the computer used. The authors suggested a number of implementation tricks that can be used for improving the speed of this method; many of these tricks also work for the extended Euclidean algorithm. Note that in the context of elliptic curve computations over $\mathbb{F}_{2^m}$, most of the inversions required can be avoided by using a *projective scheme* [59]. In this case, we trade inversions for multiplications and other finite field operations.

2.3 Elliptic curves over finite fields

In this section we give a short introduction to the theory of elliptic curves defined over finite fields. Additional information on elliptic curves and its applications to cryptography can be found in Blake *et al* [14], Menezes [64], Chapter 6 of Koblitz's book [49], and [92].

There are several ways of defining equations for elliptic curves, which depend on whether the field is a prime finite field or a characteristic two finite field. The *Weierstrass* equations for both finite fields $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$ are described in the next two sections.

2.3.1 Elliptic curves over $\mathbb{F}_p$

Let $p > 3$ be an odd prime and let $a, b \in \mathbb{F}_p$ satisfy $4a^3 + 27b^2 \neq 0 \pmod{p}$. Then an *elliptic curve* $E(\mathbb{F}_p)$ over $\mathbb{F}_p$ defined by the parameters $a, b \in \mathbb{F}_p$ consists of the set of solutions or points $P = (x, y)$ for $x, y \in \mathbb{F}_p$ to the equation:

$$y^2 = x^3 + ax + b \tag{2.1}$$

together with a special point $\mathcal{O}$ called the *point at infinity*. For a given point $P = (x_P, y_P)$, x_P is called the x-coordinate of P, and y_P is called the y-coordinate of P.

An addition operation $+$ can be defined on the set $E(\mathbb{F}_p)$ such that $(E(\mathbb{F}_p), +)$ forms an abelian group with $\mathcal{O}$ acting as its identity. It is this algebraic group that is used to construct elliptic curve cryptosystems. The addition operation in $E(\mathbb{F}_p)$ is specified as follows:

1. $P + \mathcal{O} = \mathcal{O} + P = P$ for all $P \in E(\mathbb{F}_p)$.
2. If $P = (x, y) \in E(\mathbb{F}_p)$, then $(x, y) + (x, -y) = \mathcal{O}$. (The point $(x, -y) \in E(\mathbb{F}_p)$ is denoted $-P$, and is called the *negative* of P.)
3. Let $P = (x_1, y_1) \in E(\mathbb{F}_p)$ and $Q = (x_2, y_2) \in E(\mathbb{F}_p)$, where $P \neq \pm Q$. Then $P + Q = (x_3, y_3)$, where
$$x_3 = \lambda^2 - x_1 - x_2, \ y_3 = \lambda(x_1 - x_3) - y_1, \text{ and } \lambda = \frac{y_2 - y_1}{x_2 - x_1}.$$
4. Let $P = (x_1, y_1) \in E(\mathbb{F}_p)$. Then $P + P = 2P = (x_3, y_3)$, where
$$x_3 = \lambda^2 - 2x_1, \ y_3 = \lambda(x_1 - x_3) - y_1 \text{ and } \lambda = \frac{3x_1^2 + a}{2y_1}.$$
This operation is called the *doubling* of a point.

Notice that the addition of two different elliptic curve points in $E(\mathbb{F}_p)$ requires the following arithmetic operations in $\mathbb{F}_p$: one inversion, two multiplications, one squaring and six additions. Similarly, doubling an elliptic curve point in $E(\mathbb{F}_p)$ requires one inversion, two multiplications, two squarings and eight additions. Since inversion in $\mathbb{F}_p$ is, in general, an expensive operation, an alternative method to compute the sum of two elliptic points is to use projective coordinates. In this case, the inversion operation is traded for more multiplications and other less expensive finite field operations. See [20] for several proposed projective schemes.

The following algorithm implements the addition of two points on $E(\mathbb{F}_p)$ in terms of affine coordinates.

Algorithm 6: Addition on $E(\mathbb{F}_p)$

INPUT: An elliptic curve $E(\mathbb{F}_p)$ with parameters $a, b \in \mathbb{F}_p$, and points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$.
OUTPUT: $Q = P_1 + P_2$.

1. **if** $P_1 = \mathcal{O}$, **then** **return**$(Q \leftarrow P_2)$
2. **if** $P_2 = \mathcal{O}$, **then** **return**$(Q \leftarrow P_1)$
3. **if** $x_1 = x_2$ **then**
   if $y_1 = y_2$ **then** $\lambda \leftarrow (3x_1^2 + a)/(2y_1) \bmod p$
   else return$(Q \leftarrow \mathcal{O})$ // $y_1 = -y_2$ //
   else $\lambda \leftarrow (y_2 - y_1)/(x_2 - x_1) \bmod p$
4. Set $x_3 \leftarrow \lambda^2 - x_1 - x_2 \bmod p$
5. Set $y_3 \leftarrow \lambda(x_1 - x_3) - y_1 \bmod p$
6. **return**$(Q \leftarrow (x_3, y_3))$.

2.3.2 Elliptic curves over $\mathbb{F}_{2^m}$

A (non-supersingular) *elliptic curve* $E(\mathbb{F}_{2^m})$ over $\mathbb{F}_{2^m}$ defined by the parameters $a, b \in \mathbb{F}_{2^m}, b \neq 0$, consists of the set of solutions or points $P = (x, y)$ for $x, y \in \mathbb{F}_{2^m}$ to the equation:

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

together with a special point $\mathcal{O}$ called the *point at infinity*.

As in the case of elliptic curves over $\mathbb{F}_p$, the set of points on $E(\mathbb{F}_{2^m})$ can be equipped with an abelian group structure. This addition operation is specified as follows:

1. $P + \mathcal{O} = \mathcal{O} + P = P$ for all $P \in E(\mathbb{F}_{2^m})$.
2. If $P = (x, y) \in E(\mathbb{F}_{2^m})$, then $(x, y) + (x, -y) = \mathcal{O}$. (The point $(x, -y) \in E(\mathbb{F}_{2^m})$ is denoted $-P$, and is called the *negative* of P.)
3. Let $P = (x_1, y_1) \in E(\mathbb{F}_{2^m})$ and $Q = (x_2, y_2) \in E(\mathbb{F}_{2^m})$, where $P \neq \pm Q$. Then $P + Q = (x_3, y_3)$, where

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a, \ y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \text{ and } \lambda = \frac{y_2 + y_1}{x_2 + x_1}.$$

4. Let $P = (x_1, y_1) \in E(\mathbb{F}_{2^m})$. Then $P + P = 2P = (x_3, y_3)$, where

$$x_3 = \lambda^2 + \lambda + a, \ y_3 = \lambda(x_1 + x_3) + x_3 + y_1 \text{ and } \lambda = x_1 + \frac{x_1}{y_1}.$$

Notice that the addition of two different elliptic curve points in $E(\mathbb{F}_{2^m})$ requires one inversion, two multiplications, one squaring and eight additions in $\mathbb{F}_{2^m}$. Doubling[2] a point in $E(\mathbb{F}_{2^m})$ requires one inversion, two multiplications, one squaring and six additions. For situations[3] where the computation of an inversion operation is relatively expensive compared to a multiplication, projective schemes offer computational advantages. Fast algorithms for the arithmetic of elliptic curves over $\mathbb{F}_{2^m}$ in projective coordinates are described in [59].

The following algorithm implements the addition of two points on $E(\mathbb{F}_{2^m})$ in terms of affine coordinates.

Algorithm 7: Addition on $E(\mathbb{F}_{2^m})$

INPUT: An elliptic curve $E(\mathbb{F}_{2^m})$ with parameters $a, b \in \mathbb{F}_{2^m}$, and points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$.
OUTPUT: $Q = P_1 + P_2$.

1. **if** $P_1 = \mathcal{O}$, **then** **return**$(Q \leftarrow P_2)$
2. **if** $P_2 = \mathcal{O}$, **then** **return**$(Q \leftarrow P_1)$
3. **if** $x_1 = x_2$ **then**
   if $y_1 = y_2$ **then** $\lambda \leftarrow x_1 + y_1/x_1$, $x_3 \leftarrow \lambda^2 + \lambda + a$
   else return$(Q \leftarrow \mathcal{O})$ // $y_2 = y_1 + x_1$ //
   else $\lambda \leftarrow (y_2 + y_1)/(x_2 + x_1)$, $x_3 \leftarrow \lambda^2 + \lambda + x_1 + x_2 + a$
4. Set $y_3 \leftarrow \lambda(x_1 + x_3) + x_3 + y_1$
5. **return**$(Q \leftarrow (x_3, y_3))$.

2.3.3 Definitions and basic results

Scalar multiplication. The central operation of cryptographic schemes based on ECC is the elliptic scalar multiplication (operation analogue of the exponentiation in multiplicative groups). Given an integer k and a point $P \in E(\mathbb{F}_q)$, the *elliptic scalar multiplication* kP is the result of adding P to itself k times. In Section 2.5, we will describe some efficient algorithms for calculating kP.

Orders. The *order* of a point P on an elliptic curve is the smallest positive integer r such that $rP = \mathcal{O}$. If k and l are integers, then $kP = lP$ if and only if $k \equiv l \pmod{r}$.

[2] An alternative method for computing $2P$ is described in [59].

[3] See [2] for a hardware implementation and [40] for a software implementation of $\mathbb{F}_{2^m}$ where an inversion costs about 24 and 10 multiplications, respectively.

Curve order. The number of points of $E(\mathbb{F}_q)$, denoted by $\#E(\mathbb{F}_q)$, is called the *curve order* of the curve. This number can be computed in polynomial time by Schoof's algorithm [88]. This algorithm is required for setting up an elliptic curve system based on random curves. In this case, one selects parameters a and b with the property that the curve order of the resulting curve be divisible by a large prime (see Section 2.4 for an explanation of this condition).

Basic facts. Let E be an elliptic curve over a finite field $\mathbb{F}_q$. Then:

- Hasse's theorem states that $\#E(\mathbb{F}_q) = q + 1 - t$, where $|t| \leq 2\sqrt{q}$. That is, the number of points in $E(\mathbb{F}_q)$ is approximately q.
- If q is a power of 2, then $\#E(\mathbb{F}_q)$ is even. More specifically, $\#E(\mathbb{F}_q) = 0 \pmod 4$ if $Tr(a) = 0$,[4] and $\#E(\mathbb{F}_q) = 2 \pmod 4$ if $Tr(a) = 1$.
- $E(\mathbb{F}_q)$ is an abelian group of rank 1 or 2. That is, $E(\mathbb{F}_q)$ is isomorphic to $Z_{n_1} \times Z_{n_2}$, where n_2 divides n_1 and $q - 1$.
- If q is a power of two and $P = (x, y) \in E(\mathbb{F}_q)$ is a point of odd order, then the trace of the x-coordinate of all multiples of P is equal to the trace of the parameter a. That is, $Tr(x(kP)) = Tr(a)$ for each integer k. This result, due to Seroussi [94], is the basis of an efficient algorithm for a compact representation of points on elliptic curves over $\mathbb{F}_{2^m}$. Knudsen's method [45] for computing elliptic scalar multiplications is also based on this result.

2.3.4 ECC domain parameters

The operation of public-key cryptographic schemes involves arithmetic operations on an elliptic curve over a finite field determined by some elliptic curve domain parameters. In this section, we describe the elliptic curve parameters over the finite fields $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$. ECC domain parameters over $\mathbb{F}_q$ are a septuple:

$$T = (q, FR, a, b, G, n, h)$$

consisting of a number q specifying a prime power ($q = p$ or $q = 2^m$), an indication FR (*field representation*) of the method used for representing field elements $\in \mathbb{F}_q$, two field elements a and $b \in \mathbb{F}_q$ that specify the equation of the elliptic curve E over $\mathbb{F}_q$ (i.e., $y^2 = x^3 + ax + b$ in the case $p > 3$, and $y^2 + xy = x^3 + ax^2 + b$ when $p = 2$), a base point $G = (x_G, y_G)$ on $E(\mathbb{F}_q)$, a prime n which is the order of G, and an integer h which is the cofactor $h = \#E(\mathbb{F}_q)/n$.

Several algorithms for the generation and validation of elliptic curve domain parameters have been proposed (see for example [72] and [33]). Since the primary security

[4] The *trace* $Tr(\cdot)$ is a linear map from $\mathbb{F}_{2^m}$ to $\mathbb{F}_2$ defined by $Tr(a) = \sum_{i=0}^{m-1} a^{2^i}$.

parameter is n, the ECC key length is thus defined to be the bit-length of n. For example, NIST curves [72] are described by parameters which avoid all known attacks. The security level provided by these curves is at least as much as symmetric-key ciphers with key lengths 80 to 256 bits. In Table 2.1 we compare key sizes of different cryptosystems with a comparable level of security (against known attacks).

Symmetric cipher key length	Example algorithm	ECC key length for equivalent security	DSA/RSA key length for equivalent security
80	SKIPJACK	160	1024
112	Triple-DES	224	2048
128	128-bit AES	256	3072
192	192-bit AES	384	7680
256	256-bit AES	512	15360

Table 2.1: ECC, DSA and RSA key length comparisons.

2.3.5 Elliptic curve protocols: ECDH, ECDSA, ECAES

In this section, we give a short description of three fundamental protocols based on elliptic curves: the Elliptic Curve Diffie-Hellman (ECDH), the Elliptic Curve Digital Signature Algorithm (ECDSA) and the Elliptic Curve Authenticated Encryption Scheme (ECAES). The ECDH is the elliptic version of the well-known Diffie-Hellman key agreement method; the ECDSA is the elliptic curve analogue of the DSA, proposed by Scott Vanstone [100] in 1992; and the ECAES is a variant of the ElGamal public-key encryption scheme, proposed by Abdalla, Bellare and Rogaway [1] in 1999.

Key generation. An entity A's public and private key pair is associated with a particular set of elliptic curve domain parameters (q, FR, a, b, G, n, h)[5].

To generate a key pair, entity A does the following:

1. Select a random or pseudo-random integer d in the interval $[1,n-1]$.
2. Compute $Q = dG$.
3. A's public key is Q; A's private key is d.

Public key validation. This process ensures that a public key satisfies the arithmetic requirements of elliptic curve public key (see [92]). A public key $Q = (x_Q, y_Q)$ associated

[5]This association can be assured cryptographically (i.e., with certificates) or by context (e.g., all entities use the same domain parameters)

with a domain parameter $(q, FR, a, b.G, n, h)$ is validated using the following procedure (called explicit validation):

1. Check that $Q \neq \mathcal{O}$.
2. Check that x_Q and y_Q are properly represented elements of $\mathbb{F}_q$.
3. Check that Q lies on the elliptic curve defined by a and b.
4. Check that $nQ = \mathcal{O}$.

Public key validation with step 4 omitted is called *partial* public-key validation.

ECDH. The basic idea of this primitive is to generate a shared secret value from a private key owned by one entity A and a public key owned by another entity B so if both entities execute the primitive simultaneously with corresponding keys as input, they will recover the same shared secret value. We assume that entity A has domain parameters $D = (q, FR, a, b, G, n, h)$ and a private key d_A. We also suppose that entity B has a public key Q_B associated with D. The public key Q_B should at least be partially valid.

Entity A uses the following procedure to calculate a shared secret value with B:

1. Compute $P = d_A Q_B = (x_P, y_P)$.
2. Check that $P \neq \mathcal{O}$.
3. The shared secret value is $z = x_P$.

If step 1 is computed as $P = hd_A Q_B = (x_P, y_P)$, then we call this primitive *elliptic curve cofactor Diffie-Hellman*. The incorporation of the cofactor h into the calculation of the secret value is to provide efficient resistance to attacks such as small subgroup attacks (see [92]).

ECAES. The setup for encryption and decryption is the following. We suppose that receiver B has domain parameters $D = (q, FR, a, b, G, n, h)$ and public key Q_B. We also suppose that sender A has authentic copies of D and Q_B. In the following, MAC denotes a message authentication code (MAC) algorithm such as HMAC [55], ENC a symmetric encryption scheme such as Triple-DES, and KDF a key derivation function which derives cryptographic keys from a shared secret point.

To encrypt a message m for B, A performs:

1. Select a random integer r from $[1, n-1]$.
2. Compute $R = rG$.
3. Compute $K = hrQ_B = (K_x, K_y)$. Check that $K \neq \mathcal{O}$.

4. Compute $k_1||k_2 = \mathrm{KDF}(K_x)$.
5. Compute $c = \mathrm{ENC}_{k_1}(m)$.
6. Compute $t = \mathrm{MAC}_{k_2}(c)$.
7. Send (R, c, t) to B.

To decrypt a ciphertext (R, c, t), B does:

8. Perform a partial key validation on R.
9. Compute $K = hd_BR = (K_x, K_y)$. Check that $K \neq \mathcal{O}$.
10. Compute $k_1||k_2 = \mathrm{KDF}(K_x)$.
11. Verify that $t = \mathrm{MAC}_{k_2}(c)$.
12. Compute $m = \mathrm{ENC}_{k_1}^{-1}(c)$.

The time consuming operations in encryption and decryption are the scalar multiplications in steps 3 and 9.

ECDSA. The setup for generating and verifying signatures using the ECDSA is the following. We suppose that signer A has domain parameters $D = (q, FR, a, b, G, n, h)$ and public key Q_A. We also suppose that B has authentic copies of D and Q_A. In the following SHA-1 denotes the 160-bit hash function [73].

To sign a message m, A does the following:

1. Select a random integer k from $[1,n-1]$.
2. Compute $kG = (x_1, y_1)$ and $r = x_1 \bmod n$.
   If $r = 0$ then go to step 1.
3. Compute $k^{-1} \bmod n$.
4. Compute $e = \text{SHA-1}(m)$.
5. Compute $s = k^{-1}\{e + d_A \cdot r\} \bmod n$.
   If $s = 0$ then go to step 1.
6. A's signature for the message m is (r, s).

To verify A's signature (r, s) on m, B performs the following steps:

7. Verify that r and s are integers in $[1,n-1]$.
8. Compute $e =$SHA-1(m).
9. Compute $w = s^{-1} \bmod n$.
10. Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
11. Compute $u_1G + u_2Q_A = (x_1, y_1)$.

12. Compute $v = x_1 \bmod n$.
13. Accept the signature if and only if $v = r$.

The time consuming operations in signature generation and signature verification are the scalar multiplications in steps 2 and 11.

2.4 Discrete logarithm problem

The security of ECC is based on the apparent intractability of the following *elliptic curve discrete logarithm problem* (ECDLP): given an elliptic curve $E(\mathbb{F}_q)$, a point $P \in E(\mathbb{F}_q)$ of order n, and a point $Q \in E(\mathbb{F}_q)$, determine the integer k, $0 \leq k \leq n-1$, such that $Q = kP$, provided that such an integer exists.

The Pohlig and Hellman algorithm [76] reduces the computation of l to the problem of computing l modulo each of the prime factors of n. Therefore, n should be selected prime to obtain the maximum level of security. In practice, one must select an elliptic curve $E(\mathbb{F}_q)$ such that $\#E(\mathbb{F}_q) = h \cdot n$ where n is a prime and h is a small integer.

The most efficient general algorithm known to date is the Pollard-ρ method [78], and its recent modifications by Gallant, Lambert, and Vanstone [30], and Wiener and Zuccherato [105], which requires about $\sqrt{\pi n}/2$ elliptic group operations. Van Oorschot and Wiener [80] showed that the Pollard-ρ method can be parallelized, and that the expected running time of this algorithm, using r processors, is roughly $\sqrt{\pi n}/(2r)$ groups operations. This runtime is exponential in n.

Although no general subexponential algorithms to solve the ECDLP are known, there are fast algorithms for solving the ECDLP on special curves (e.g., curves for which the number of points has special properties). We list next some of these known attacks and explain how they can be avoided in practice.

- *Supersingular elliptic curves.* Menezes, Okamato and Vanstone [67] and Frey and Rück [28] showed that, under mild assumptions, the ECDLP can be reduced to the traditional discrete logarithm problem in some extension field F_{q^k}, for some integer k. This reduction algorithm is only practical if k is small. For the class of supersingular[6] elliptic curves it is known that $k \leq 6$. Hence, this reduction algorithm gives a sub-exponential time algorithm for the ECDLP. However, Balasubramanian and Koblitz [10] have shown that for most randomly generated elliptic curves we have $k > \log^2 q$. To avoid this attack in a particular curve, one needs to check that

[6] An elliptic curve over $\mathbb{F}_q$ is said to be *supersingular* if the trace of E, $t(E) = q + 1 - \#E(\mathbb{F}_q)$, is divisible by the characteristic of $\mathbb{F}_q$.

n, the largest prime factor of the curve order, does not divide $q^k - 1$ for all small k for which the ordinary logarithm problem in $\mathbb{F}_{q^k}$ is tractable. In practice this checking is done for all k, $1 \leq k \leq 30$.

- *Prime-field anomalous curves.* An elliptic curve E over $\mathbb{F}_p$ is said to be *prime-field-anomalous* if $\#E(\mathbb{F}_p) = p$. Semaev [93], Smart [95] and Satoh and Araki [87] independently proposed a polynomial-time algorithm for the ECDLP in $E(\mathbb{F}_p)$. This attack does not appear to extend to any other class of elliptic curves. In practice this attack is avoided by verifying that the curve order does not equal the cardinality of the underlying finite field.
- *Binary composite finite fields.* Suppose that E is an elliptic curve defined over the composite finite field $\mathbb{F}_{2^m}$, where $m = r \cdot s$. Recently, Galbraith and Smart [29], and Gaundry, Hess and Smart [32] have showed that the complexity of the discrete logarithm problem on a significant portion of elliptic curves defined over $\mathbb{F}_{2^{4s}}$ is smaller than the Pollard-rho method. The authors concluded that this attack does not appear to be a threat to elliptic curves defined over $\mathbb{F}_{2^m}$, for m prime, but that only curves that satisfy an additional condition (see [14, pp. 18]), should be used for setting up an elliptic curve cryptosystem.

Additional information on other attacks for the ECDLP as well for attacks on elliptic curve protocols can be found in ANSI X9.62 [5], ANSI X9.63 [6], Blake, Seroussi and Smart [14], Johnson and Menezes [44], Koblitz, Menezes and Vanstone [51], Araki, Satoh and Miura [7], and Certicom's ECC challenge [19].

2.5 Algorithms for elliptic scalar multiplication

The implementation of public key protocols of ECC such as ECDH, ECDSA and ECAES, requires elliptic scalar multiplications. That is, calculations of the form

$$Q = kP = \underbrace{P + \cdots + P}_{k \text{ times}}$$

where P is a curve point, and k is an integer in the range $1 \leq k \leq \text{order}(P)$. Depending on the protocol, the point P is either a fixed point that generates a large, prime order subgroup of $E(\mathbb{F}_q)$, or P is an arbitrary point in such a subgroup.

Many authors have discussed methods for exponentiation in a multiplicative group, which can, therefore, be extended to computing elliptic scalar multiplication [36, 66, 53, 54]. However, elliptic curve groups have special properties that allow for some extra optimizations. In this section we will describe some efficient algorithms for computing kP. These algorithms, depending on the elliptic curve and the characteristic of the finite

field, can be further optimized. Finally, we summarize recent techniques suitable for hardware or software implementation of ECC.

2.5.1 Basic methods

Binary method. The simplest (and oldest) method for computing kP is based on the binary representation of k. If $k = \sum_{i=0}^{l-1} k_j 2^j$, where each $k_j \in \{0,1\}$, then kP can be computed as

$$kP = \sum_{j=0}^{l-1} k_j 2^j P = 2(\cdots 2(2k_{l-1}P + k_{l-2}P) + \cdots) + k_0 P.$$

This method requires l doublings and $w_k - 1$ additions, where w_k is the weight (the number of ones) of the binary representation of k.

An improved method for computing kP can be obtained from the following facts:

- Every integer k has a unique representation of the form $k = \sum_{j=0}^{l-1} k_j 2^j$, where each $k_j \in \{-1, 0, 1\}$, such that no two consecutive digits are nonzero. This representation, known as *non-adjacent form* (NAF), was first described by Reitwiesner [83] (see also [14]).
- The expected weight of a NAF of length l is $l/3$, see [14].
- The computation of the negation of a point $P = (x, y) \in E(\mathbb{F}_q)$ ($-P = (x, -y)$ or $-P = (x, x+y)$) is virtually free, so the cost of addition or subtraction is practically the same.

There are, however, several algorithms for computing the NAF of k from its binary representation (see for example [66]). The following method, from Solinas [97], computes the NAF of an integer k.

Algorithm 8: Computation of NAF(k)

INPUT: An integer k
OUTPUT: The non-adjacent form of k, NAF(k)= $(u_{l-1} \dots u_1 u_0)$

1. Set $c \leftarrow k$, $l \leftarrow 0$
2. **while** $c > 0$ **do**
   if c odd **then**
   Set $u_l \leftarrow 2 - (c \bmod 4)$
   Set $c \leftarrow c - u_l$
   else Set $u_l \leftarrow 0$
   Set $c \leftarrow c/2$, $l \leftarrow l + 1$
3. **return**(NAF(k) $\leftarrow (u_{l-1} \dots u_1 u_0)$).

Addition-Subtraction method. This algorithm, analogue of the binary method, performs an addition or subtraction depending on the sign of each digit of k, scanned from left to right.[7] The details are given in Algorithm 9. This algorithm requires l doublings and $l/3$ additions on average. This implies, for example, that for elliptic curves over $\mathbb{F}_p$, using the projective coordinates given in [42], we obtain an improvement of about 14% over the binary method.

Algorithm 9: Addition-Subtraction method

INPUT: An integer k and a point $P = (x, y) \in E(\mathbb{F}_q)$
OUTPUT: The point $Q = kP \in E(\mathbb{F}_q)$

1. Compute NAF$(k) = (u_{l-1} \dots u_1 u_0)$
2. Set $Q \leftarrow \mathcal{O}$
3. **for** j **from** $l-1$ **downto** 0 **do**
   Set $Q \leftarrow 2Q$
   if $u_j = 1$ **then** Set $Q \leftarrow Q + P$
   if $u_j = -1$ **then** Set $Q \leftarrow Q - P$
4. **return**(Q).

Window method. Several generalizations of the binary method such as the m-ary method, sliding method, etc., work by processing simultaneously a block of digits. In these methods, depending on the size of the blocks (or windows) a number of precomputed points are required. We describe a typical window method called the *width-w window method* (see [97]).

Let w be an integer greater than 1. Then every positive number k has a unique *width-w nonadjacent form* $k = \sum_{j=0}^{l-1} u_j 2^j$ where:

- each nonzero u_j is odd and less than 2^{w-1} in absolute value;
- among any w consecutive coefficients, at most one is nonzero.

The width-w NAF is written $\mathrm{NAF}_w(k) = (u_{l-1} \dots u_1 u_0)$. A generalization of Algorithm 8 for computing $\mathrm{NAF}_w(k)$ is described in Algorithm 10. Given the width-w NAF of an integer k, and a point $P \in E(\mathbb{F}_q)$, the calculation of kP can be carried out by Algorithm 11.

[7] This algorithm can be modified to obtain a *right-to-left* version, which does not need storage for the NAF(k), see [97] for more details.

Algorithm 10: Computation of $\mathrm{NAF}_w(k)$

INPUT: An integer k
OUTPUT: $\mathrm{NAF}_w(k) = (u_{l-1} \ldots u_1 u_0)$

1. Set $c \leftarrow k$, $l \leftarrow 0$
2. **while** $c > 0$ **do**
   if c odd **then**
   Set $u_l \leftarrow 2 - (c \bmod 2^w)$
   if $u_l > 2^{w-1}$ **then** Set $u_l \leftarrow u_l - 2^w$
   Set $c \leftarrow c - u_l$
   else Set $u_l \leftarrow 0$
   Set $c \leftarrow c/2$, $l \leftarrow l + 1$
3. **return**$(\mathrm{NAF}_w(k) \leftarrow (u_{l-1} \ldots u_1 u_0))$.

Algorithm 11: The width-w window method

INPUT: Integers k and w, and a point $P = (x, y) \in E(\mathbb{F}_q)$
OUTPUT: The point $Q = kP \in E(\mathbb{F}_q)$

// Precomputation:
// Compute uP for u odd and $2 < u < 2^{w-1}$
1. Set $P_0 \leftarrow P$, $T \leftarrow 2P$
2. **for** i **from** 1 **to** $2^{w-2} - 1$ **do**
   Set $P_i \leftarrow P_{i-1} + T$
// Main Computation:
3. Compute $\mathrm{NAF}_w(k) = (u_{l-1} \ldots u_1 u_0)$
4. Set $Q \leftarrow \mathcal{O}$
5. **for** j **from** $l - 1$ **downto** 0 **do**
   Set $Q \leftarrow 2Q$
   if $u_j \neq 0$ **then**
   Set $i \leftarrow (|u_j| - 1)/2$
   if $u_j > 0$ **then** Set $Q \leftarrow Q + P_i$
   else Set $Q \leftarrow Q - P_i$
6. **return**(Q).

The number of nonzero digits in the $\mathrm{NAF}_w(k)$ is on average $l/(w+1)$ [99]. Therefore, Algorithm 11 requires $2^{w-2} - 1$ additions and one doubling for the precomputation step, and $l/(w+1)$ additions and $l-1$ doublings for the main computation. Note that although the number of additions can be reduced by selecting an apropriate width w, the number

of doublings is the same as in the previous methods. The total number of finite field operations required for computing kP depends mainly on the algorithms used for the elliptic operations (affine or projective coordinates), the cost-ratio of inversion to multiplication, and the width w.

Comb method. This method, developed by Lim and Lee [58], can be used for computing kP when P is a fixed point, known in advance of the computation. In order to compute kP, the l-bit integer k is divided into h blocks K_r, each one of length $a = \lceil l/h \rceil$. In addition, each block K_r is subdivided into v blocks of size $b = \lceil a/v \rceil$. Thus, k can be written as

$$k = \sum_{r=0}^{h-1} \sum_{s=0}^{v-1} \sum_{t=0}^{b-1} k_{vbr+bs+t} 2^{vbr+bs+t}.$$

Then, Lim/Lee's method uses the following expression for computing kP:

$$kP = \sum_{t=0}^{b-1} 2^t \Big(\sum_{s=0}^{v-1} G[s][I_{s,t}] \Big),$$

where the precomputation array $G[s][u]$ for $0 \leq s < v$, $0 \leq u < 2^h$, and $u = (u_{h-1} \ldots u_0)_2$, is defined by the following equations:

$$\begin{aligned} G[0][u] &= \sum_{r=0}^{h-1} u_r 2^{rvb} P, \\ G[s][u] &= 2^{sb} G[0][u], \end{aligned}$$

and the number $I_{s,t}$, for $0 \leq s < v-1$ and $0 \leq t < b$ is defined by

$$I_{s,t} = \sum_{r=0}^{h-1} k_{vbr+bs+t} 2^r.$$

A detailed description of Lim/Lee's method is given in Algorithm 12. This algorithm requires $v(2^h - 1)$ elliptic points of storage, and the average number of operations to perform a scalar multiplication is $b - 1$ doublings and $(2^h - 1)/2^h vb - 1$ additions on average, but $vb - 1$ additions in the worst case. The selection of both parameters h and v presents a trade-off between precomputation (memory) and online computations (speed). Some improvements to this algorithm are discussed in [21]. For other algorithms for computing kP when P is a known point, see [66].

Algorithm 12: Lim/Lee method

INPUT: Integers k, h, v and an array of points $G[s][u]$, with $0 \leq s < v$ and $1 \leq u < 2^h$.

// The array G is computed as:

for u **from** 1 **to** $2^h - 1$ **do**

for s **from** 0 **to** $v - 1$ **do**

Set $u \leftarrow (u_{h-1} \ldots u_1 u_0)_2$

Set $G[s][u] \leftarrow 2^{sb} \sum_{i=0}^{h-1} u_i 2^{vbi} P$.

OUTPUT: The point $Q = kP \in E(\mathbb{F}_q)$.

// Main Computation:

1. Set $Q \leftarrow \mathcal{O}$
2. **for** t **from** $b - 1$ **downto** 0 **do**

Set $Q \leftarrow 2Q$

for s **from** $v - 1$ **downto** 0 **do**

Set $I_{s,t} \leftarrow \sum_{i=0}^{h-1} 2^i k_{vbi+bs+t}$

if $I_{s,t} \neq 0$ **then** $Q \leftarrow Q + G[s][I_{s,t}]$

3. **return**(Q).

2.5.2 Faster methods

In recent years, the study of fast methods for computing a scalar multiplication has been an active research area. In this section we summarize some of these recent methods.

- An algorithm for computing repeated doublings (i.e., $2^i P$), for elliptic curves defined over $\mathbb{F}_{2^m}$ was proposed by López and Dahab [59]. This algorithm, an improvement over the formulas presented by Guajardo and Paar [37], computes $2^i P$ with only one inversion, and it is faster than the usual method for computing $2^i P$ (i consecutive doublings) if the cost-ratio of inversion to multiplication is at least 2.5. This method can be used to speed up window methods such as the one described in the previous section.
- Another algorithm for computing repeated doublings, for elliptic curves over $\mathbb{F}_{2^m}$, was proposed by Schroeppel [91]. This algorithm is useful for situations where the computation of an inverse is relatively fast compared to a multiplication. A slightly improved version of this method is the following:

Algorithm 13: Repeated doublings on $E(\mathbb{F}_{2^m})$

INPUT: An integer i and a point $P = (x, y) \in E(\mathbb{F}_{2^m})$
OUTPUT: The point $Q = 2^i P$

1. Set $\lambda \leftarrow x + y/x$
2. **for** j **from** 1 **to** i-1 **do**
   Set $x_2 \leftarrow \lambda^2 + \lambda + a$
   Set $\lambda_2 \leftarrow \lambda^2 + a + \dfrac{b}{x^4 + b}$
   Set $x \leftarrow x_2, \ \lambda \leftarrow \lambda_2$
3. Set $x_2 \leftarrow \lambda^2 + \lambda + a, \ y_2 \leftarrow x^2 + (\lambda + 1) \cdot x_2$
4. **return**$(Q \leftarrow (x_2, y_2))$.

This method is based on the observation that doubling a point using the representation (x, λ)[8] is faster than using the affine representation (x, y). Thus, we save one field multiplication in each iteration of Algorithm 13. A further optimization is to use a fast routine to multiply by the constant b. This method can be used for speeding up window methods in affine coordinates.

- For elliptic curves over $\mathbb{F}_p$, Itoh *et al* [43] proposed fast formulas for computing repeated doublings in projective coordinates, which reduce both the number of field multiplications and the number of field additions. This technique works in combination with window methods.
- An optimized version of an algorithm developed by Montgomery [69], was proposed by Lopez and Dahab [60]. This algorithm works for every elliptic curve defined over $\mathbb{F}_{2^m}$, is faster than the addition-subtraction method, and it is suitable for both hardware and software implementations. In addition, this algorithm has the property that in each iteration the same amount of computation (an addition followed by a doubling) is performed. This may help to prevent timing attacks [50].
- An algorithm for computing elliptic scalar multiplications which replaces the doubling operation by the halving operation (i.e., the computation of Q such that $2Q = P$) was proposed by Knudsen [45]. This algorithm works for half of the elliptic curves defined over $\mathbb{F}_{2^m}$ (i.e., curves whose elliptic curve parameter a satisfies $Tr(a) = 1$). The implementation of this method requires fast routines for the following operations in $\mathbb{F}_{2^m}$: the square root of a field element, the trace of a field element, and the solution of quadratic equations of the form $x^2 + x = s$, for $s \in \mathbb{F}_{2^m}$. Since these operations can be carried out very efficiently using a normal basis, this

[8] Every point $P = (x, y) \in E(\mathbb{F}_{2^m}), x \neq 0$, can be represented as the pair $(x, \lambda), \lambda = x + y/x$, but (x, λ) is not a point on $E(\mathbb{F}_{2^m})$.

approach is suitable for hardware implementations. The implementation of Knudsen's method, using a polynomial basis, presents a trade off between memory and speed for both implementations hardware and software.

2.5.3 Koblitz curves

These curves, also known as binary anomalous curves, were first proposed for cryptographic use by Koblitz [48]. They are elliptic curves over $\mathbb{F}_{2^m}$ with coefficients a and b either 0 or 1. Since it is required that $b \neq 0$, then the curves must be defined by the equations:

$$E_0 : y^2 + xy = x^3 + 1 \text{ and } E_1 : y^2 + xy = x^3 + x^2 + 1.$$

Koblitz curves have the following interesting property: if (x, y) is a point on $E_a, a = 0$ or $a = 1$, so is the point (x^2, y^2). Moreover, every point $P = (x, y) \in E_a$ satisfies the relation

$$(x^4, y^4) + 2P = \mu \cdot (x^2, y^2). \tag{2.3}$$

where

$$\mu = (-1)^{1-a}.$$

By using the Frobenius map over $\mathbb{F}_2$: $\tau(x, y) = (x^2, y^2)$, equation (2.3) can be written as

$$\tau(\tau P) + 2P = \mu\tau P, \text{ for all } P \in E_a.$$

Then the Frobenius map τP can be regarded as a multiplication by the complex number $\tau = \frac{\mu+\sqrt{-7}}{2}$ satisfying $\tau^2 + 2 = \mu\tau$.

Several methods have been proposed to take advantage of the Frobenius map, starting with the observation of Koblitz [48], that four consecutive doublings of a point $P = (x, y) \in E_1$ can be computed efficiently via the formula

$$16P = \tau^2 P - \tau^4 P = (x^4, y^4) - (x^{16}, y^{16}).$$

The fastest method known for computing kP on Koblitz curves is due to Solinas [97]. This method uses an expansion for kP of the form

$$kP = \sum_{i=0}^{l-1} k_i \tau^i P, \; k_i \in \{-1, 0, 1\} \text{ and } l \approx \log k.$$

Then, the calculation of kP can be carried out by a similar method to Algorithm 9 where the doublings are replaced by evaluations of the Frobenius map. Before we describe Solinas' method, the following sequences $\rho_a(n)$ and $\sigma_a(n)$ are defined:

- $\rho_a(0) = 0,\ \rho_a(1) = a - 1,\ \rho_a(n+1) = \mu\rho_a(n) - 2\rho_a(n-1) + a - 2.$
- $\sigma_a(0) = 0,\ \sigma_a(1) = a - 1,\ \sigma_a(n+1) = \mu\sigma_a(n) - 2\sigma_a(n-1).$

Algorithm 14 describe Solinas' method for computing an elliptic scalar multiplication on the Koblitz curve $E_a(\mathbb{F}_{2^m})$.

Algorithm 14: τ- adic NAF method for Koblitz curves

INPUT: An integer k and a point $P = (x, y) \in E_a(\mathbb{F}_{2^m})$.
OUTPUT: The point $Q = kP \in E_a(\mathbb{F}_{2^m})$

// Reduction modulo $(\tau^m - 1)/(\tau - 1)$
1. Set $r \leftarrow \lfloor \rho_a(m) \cdot k/2^{m-1} \rfloor,\quad s \leftarrow \lfloor \sigma_a(m) \cdot k/2^m \rfloor$
2. Set $t \leftarrow 2\rho_a(m) + \mu\sigma_a(m),\quad v \leftarrow \sigma_a(m) \cdot s$
3. Set $c \leftarrow k - t \cdot r - 2v,\ d \leftarrow \sigma_a(m) \cdot r - 2\rho_a(m) \cdot s$
// Main computation
4. Set $Q \leftarrow \mathcal{O},\ D \leftarrow P$
5. **while** $c \neq 0$ or $d \neq 0$ **do**
   if c odd **then** Set $u \leftarrow (c - 2d \pmod 4)$
   else Set $u \leftarrow 0$
   Set $c \leftarrow c - u$
   if $u = 1$ **then** Set $Q \leftarrow Q + D$
   if $u = -1$ **then** Set $Q \leftarrow Q - D$
   Set $D \leftarrow \tau D$
   Set $e \leftarrow c/2,\ c \leftarrow d + \mu e,\ d \leftarrow -e$
6. **return**(Q).

This algorithm requires, on average, $m/3$ elliptic additions and m evaluations of the Frobenius map. For comparison, if we implement Koblitz curves over $\mathbb{F}_{2^{163}}$, using a normal basis[9] with the projective coordinates given in [59], Algorithm 9 takes 972 multiplications, while Solinas' algorithm requires 486 multiplications, obtaining a theoretical improvement of about 50%. Further speedups can be obtained by using window techniques; see Solinas [97][10] for the "width-w τ-*addic* NAF method" analogous to Algorithm 11.

2.6 Implementation issues

When implementing ECC, there are many factors that may guide the choices required in the implementation of a particular application. The factors include: security considera-

[9] For hardware implementations, the squarings are much faster than multiplications.
[10] Routine 6 from [97] fails when $a = 0$ and $w = 6$. A new version of this routine was given in [99].

tions (the ECDLP and security of the protocols), methods for implementing the finite field arithmetic, methods for computing elliptic scalar multiplications, the application platform (hardware or software), constraints of the computing environment (processor speed, code size, power consumption), and constraints of the communication environment (bandwidth, response time). Since these factors can have a major impact on the overall performance of the application, it is recommended that they all be taken together for better results.

2.6.1 System setup

Setting up an elliptic curve cryptosystem requires several basic choices including:

- An underlying finite field $\mathbb{F}_q$
  (e.g., $q = p$, $q = 2^m$ or $q = p^m$, $p > 3$)
- A representation of the finite field elements
  (e.g., Montgomery residue for $\mathbb{F}_p$, polynomial or normal basis for $\mathbb{F}_{2^m}$)
- Algorithms for implementing the finite field operations
  (e.g., Montgomery multiplication in $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$, the extended Euclidean algorithm and the almost inverse algorithm for computing multiplicative inverses)
- An *appropriate* elliptic curve over $\mathbb{F}_q$
  (e.g., the NIST curves)
- Algorithms for implementing the elliptic curve operations
  (e.g., windows methods in affine or projective coordinates)
- Elliptic curve protocols
  (e.g., ECDSA, ECDH)

By an appropriate elliptic curve, we mean an elliptic curve defined over the finite field $\mathbb{F}_q$ that resists all known attacks on the ECDLP. Specifically:

1. The number of points, $\#E(\mathbb{F}_q)$, is divisible by a prime n that is sufficiently large to resist the parallelized Pollard ρ-attack [80] againts general curves, and its improvements [30, 105] which apply to Koblitz curves.
2. $\#E(\mathbb{F}_q) \neq q$, to resist the following attacks: Semaev [93], Smart [95], and Satoh-Araki [87].
3. n does not divide $q^k - 1$ for all $1 \leq k \leq 30$, to resist the Weil paring attack [67] and the Tate paring attack [28].
4. All binary fields $\mathbb{F}_{2^m}$ chosen have the property that m is prime, to resist recent attacks [29, 32] on elliptic curves defined over $\mathbb{F}_{2^m}$ where m is composite.

Examples of appropriate curves to be used in real world cryptosystems are given in [72] and [33].

2.6.2 Previous software implementations of ECC

In the last five years, there have been many reported software implementations of elliptic curves over finite fields. Most of these implementations focus on a single cryptographic application, such as designing a fast implementation of ECDSA for one particular finite field. Typical examples of finite fields used in these implementations are $\mathbb{F}_{2^{155}}$[89], $\mathbb{F}_{2^{167}}$ [15], $\mathbb{F}_{2^{176}}$ [37, 9], $\mathbb{F}_{2^{191}}$ [25], $\mathbb{F}_p$ (p a 160-bit prime) [41], $\mathbb{F}_p$ (p a 192-bit prime) [25], and $\mathbb{F}_{(2^{63}-25)^3}$ [11]. In [61], we have compiled timing results of several reported software implementations of ECC. In this section, we summarize three examples of software implementations of ECC on general purpose computers.

- Schroppel *et al.* [89] reported an implementation of an elliptic curve analogue of Diffie-Hellman key exchange algorithm over $\mathbb{F}_{2^{155}}$ with a trinomial basis representation. A detailed description of the finite field arithmetic in $\mathbb{F}_{2^{155}}$ is provided, including a fast method for computing reciprocals, called the almost inverse algorithm. An improved method for doubling an elliptic curve point is also presented. Two computer architectures were used to measure performance, a Sun Sparc-IPC (25 MHz), with 32 bit word size, and a DEC Alpha 3000 (175 MHz), with a 64-bit size word. The implementation was written in C with several programming tricks. The performance results are given in Table 2.2.

Field and Curve Operations over $\mathbb{F}_{2^{155}}$	Sparc IPC	Alpha
Squaring	11.9	0.64
Multiplication	116.4	7.59
Inversion	280.1	25.21
ECDH key exchange	137,000	11,500
DH key exchange (512 bits)	2,670,000	185,000

Table 2.2: Timings (in microseconds) for finite field and elliptic curve operations.

- De Win *et al.* [25] described an implementation of ECDSA, for both $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$, and made comparisons with other signature algorithms such as RSA and DSA. The platform used was a Pentium-Pro 200 MHz running Windows NT 4.0 and using MSVC 4.2 and maximal optimization. The code for RSA and DSA was written in C, using macros in assembly language. The elliptic curve code was mainly written in C++ and for $\mathbb{F}_p$ the same multi-precision routines in C were called as for RSA and DSA. The modulus for both RSA and DSA was 1024 bits long. For the elliptic curves, the field sizes for $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$ were approximately 191 bits. Table 2.3 summarizes the results of their implementation.

	ECDSA $\mathbb{F}_{2^m}$	ECDSA $\mathbb{F}_p$	RSA	DSA
Key generation	11.7	5.5	1 sec.	22.7
Signature	11.3	6.3	43.3	23.6
Verification	60	26	0.65	28.3
Scalar multiplication	50	21.1	-	-

Table 2.3: Timing comparison of ECDSA , DSA, and RSA signature operations. All timings in milliseconds, unless otherwise indicated.

- Bailey and Paar [11] introduced a new type of finite fields which can be used to achieve a fast software implementation of elliptic curve cryptosystems. This class of finite fields called Optimal Extension Field (OEF), is of the form $\mathbb{F}_{p^m}$, where p is a prime of special form and m a positive integer. The OEFs take advantage of the fast integer arithmetic found on modern RISC workstation processors. The authors provided a list of OEFs suitable for processors with 8, 16, 32 and 64 bit word sizes. In [12], the same authors presented further improved algorithms for the finite field arithmetic, and timing results of their elliptic curve implementation on several platforms. Two Alpha workstations DEC 21064 and 21164A, and a 233 MHz Intel Pentium/MMx PC were used to measure performance. The implementation for the workstations was written in optimized C, resorting to assembly to perform polynomial multiplications; the implementation for the PC was written entirely in C. The sizes of chosen finite fields were approximately 183 bits. Table 2.4 presents the timings to perform an elliptic scalar multiplication of an arbitrary point.

Operation	Alpha 21064 150 MHz	Alpha 21164A 600 MHz	Pentium/MMX 233 MHz
kP	7.0	1.09	13.1

Table 2.4: Timings (in milliseconds) for an elliptic scalar multiplication.

2.6.3 An example of a software implementation of ECC

In this section we present some details of the ECC software implementation reported in [16]. This paper describes an experience with porting PGP to the Research in Motion (RIM) two-way pager, and incorporating ECC into PGP.

- *Finite fields:* $\mathbb{F}_{2^m}$, $m = 163, 233, 283$.

- *Representation:* A polynomial basis was used for each finite field, with the following reduction polynomials: $x^{163}+x^7+x^6+x^3+1$ for $\mathbb{F}_{2^{163}}$, $x^{233}+x^{74}+1$ for $\mathbb{F}_{2^{233}}$ and $x^{283}+x^{12}+x^7+x^6+1$ for $\mathbb{F}_{2^{283}}$.
- *Algorithms for the finite field arithmetic:* The squaring operation was sped up by using a table lookup of 512 bytes. The multiplication operation was carried out by the algorithm described in [62]. The inverse operation was carried out by the extended Euclidean algorithm.
- *Curves:* The Koblitz and random curves over $\mathbb{F}_{2^{163}}, \mathbb{F}_{2^{233}}$ and $\mathbb{F}_{2^{283}}$ were selected from the list of NIST recommended curves [72].
- *Algorithms for the elliptic curve group:* For random curves, the method given in [60] was implemented for computing scalar multiplications when P is an arbitrary point. Lim/Lee's method [66], with 16 points of precomputation, was implemented using the projective coordinates given in [59] for computing scalar multiplications when P is a known point (e.g., for signing). For a Koblitz curve, Solinas' methods [97] were implememented using projective coordinates, with width $w = 5$ for random points, and $w = 6$ for a known point (in this case, 16 points of precomputation are required).
- *EC protocols:* The protocols implemented were: ECDSA and ECAES.
- *Multi-precision library:* The library *bc* from OpenSSL [81], written entirely in C, was used to perform the modular arithmetic operations required in the elliptic curve protocols as well in Solinas' methods.
- *Platforms:* A Pentium II 400 MHz and a RIM pager 10 MHz.
- *Language:* The implementation was written entirely in C.
- *RSA:* The RSA code, written entirely in C, was taken from the OpenSSL library.
- *Timings:* The performance results provided are only for the case $m = 163$ (see [16] for more timings). Table 2.5 shows the timings for finite field operations in $\mathbb{F}_{2^{163}}$.

Operations in $\mathbb{F}_{2^{163}}$	Pentium II 400 MHz	RIM pager 10 MHz
Squaring	0.41	100
Multiplication	2.97	1,515
Inversion	31.23	12,500

Table 2.5: Timings (in microseconds) for finite field operations in $\mathbb{F}_{2^{163}}$.

The performance results for the ECC operations using Koblitz and random curves over $\mathbb{F}_{2^{163}}$ are summarize in Table 2.6. Timings for RSA operations, with a modulus of 1024 bits, are given in Table 2.7.

	Koblitz curve over $\mathbb{F}_{2^{163}}$		Random curve over $\mathbb{F}_{2^{163}}$	
	RIM pager	P II	RIM pager	P II
Key Generation	751	1.47	1,085	2.12
ECAES encrypt	1,759	4.37	3,132	6.67
ECAES decrypt	1,065	2.85	2,114	4.69
ECDSA signing	1,011	2.11	1,335	2.64
ECDSA verifying	1,826	4.09	3,243	6.46

Table 2.6: Timings (in milliseconds) for ECC operations over$\mathbb{F}_{2^{163}}$.

	1024-bit modulus	
	RIM Pager	Pentium II
RSA key generation	580,405	2,740.87
RSA encrypt ($e = 3$)	533	2.70
RSA encrypt ($e = 2^{16} + 1$)	1,241	5.34
RSA decrypt	15,901	67.32
RSA signing	15,889	66.56
RSA verifying ($e = 3$)	301	1.23
RSA verifying ($e = 2^{16} + 1$)	1,008	3.86

Table 2.7: Timings (in milliseconds) for 1024-bit RSA operations.

- *Conclusions:* Since the two systems RSA-1024 and ECC-163 have a comparable level of security, the following conclusions can be drawn from the timings:
  - RSA public-key operations (encryption and signature) are faster than ECC public-key operations.
  - ECC private key operations (decryption and signature generation) are faster than RSA private-key operations.
  - Koblitz curves perform better than random curves, especially for encrypting and verifying.
  - With respect to the the PGP operations Signing-and-encrypting and Verifying-and-decryting, the performance of ECC (Koblitz curves) is about five times the performance of RSA on the RIM pager.

2.7 Conclusions

In this paper, we have presented an overview of the main ideas behind the public-key technology based on elliptic curves. We have focused on algorithms for software implementation of elliptic curves defined over the binary field $\mathbb{F}_{2^m}$. We have also presented a summary of the fastest software implementations of ECC reported on general purpose computers.

Capítulo 3

Um Algoritmo para Multiplicação em $\mathbb{F}_{2^m}$

Este capítulo descreve um algoritmo eficiente para multiplicação em $\mathbb{F}_{2^m}$, cujos elementos são representados usando uma base polinomial. O método proposto pode ser utilizado para implementação em software de curvas elípticas definidas sobre $\mathbb{F}_{2^m}$. Os tempos de execução deste algoritmo, em diferentes plataformas computacionais, indicam que o novo algoritmo é significativamente mais rápido do que o método padrão de multiplicação em $\mathbb{F}_{2^m}$.

O trabalho apresentado neste capítulo foi publicado como relatório técnico No. IC-00-09 no Instituto de Computação, UNICAMP, e submetido à conferência Indocrypt 2000.

High-Speed Software Multiplication in $\mathbb{F}_{2^m}$

Julio López and Ricardo Dahab
Institute of Computing
State University of Campinas,
Campinas, C.P. 6176, 13083-970, SP, Brazil
{julioher,dahab}@dcc.unicamp.br

Abstract

In this paper we describe an efficient algorithm for multiplication in $\mathbb{F}_{2^m}$, where the field elements of $\mathbb{F}_{2^m}$ are represented in standard polynomial basis. The proposed algorithm can be used in practical software implementations of elliptic curve cryptography. Our timing results, on several platforms, show that the new method is significantly faster than the "shift-and-add" method.

Key words. Multiplication in $\mathbb{F}_{2^m}$, Polynomial Basis, Elliptic Curve Cryptography.

3.1 Introduction

Efficient algorithms for multiplication in $\mathbb{F}_{2^m}$ are required to implement cryptosystems such as the Diffie-Hellman and elliptic curve cryptosystems defined over $\mathbb{F}_{2^m}$. Efficient implementation of the field arithmetic in $\mathbb{F}_{2^m}$ depends greatly on the particular basis used for the finite field. Two common choices of bases for $\mathbb{F}_{2^m}$ are normal and polynomial. Normal bases seem more suitable for hardware implementations (see [2]).

In this paper we describe a technique for multiplication in the finite field $\mathbb{F}_{2^m}$, where the field elements are represented as binary polynomials modulo an irreducible binary polynomial of degree m. The proposed method is about 2-5 times faster than the standard multiplication, and is particularly useful for software implementation of elliptic curve cryptosystems over $\mathbb{F}_{2^m}$. It is based on the observation that Lim/Lee's method [58] (or comb method [66]), designed for exponentiation, can be modified to work in $\mathbb{F}_{2^m}$.

The remainder of this paper is organized as follows. In Section 3.2 we describe the finite field $\mathbb{F}_{2^m}$ using a polynomial basis, along with a description of the standard algorithm for multiplication in $\mathbb{F}_{2^m}$. A description of a simple version of Lee/Lim's method and two versions of the proposed method are described in Section 3.3. In Section 3.4, we present timing results on different computational platforms.

3.2 The finite field $\mathbb{F}_{2^m}$

3.2.1 Polynomial basis representation

In this section we describe the finite field $\mathbb{F}_{2^m}$, called a *characteristic two finite field* or a *binary finite field*, in terms of a *polynomial basis representation*. Let $f(x) = x^m + \sum_{i=0}^{m-1} f_i x^i$ (where $f_i \in \{0,1\}$, for $i = 0, \ldots, m-1$) be an irreducible polynomial of degree m over $\mathbb{F}_2$; polynomial $f(x)$ is called the *reduction polynomial*. A *polynomial basis* is specified by a reduction polynomial. In such a representation, the bit string $(a_{m-1} \ldots a_1 a_0)$ is taken to represent the polynomial

$$a_{m-1}x^{m-1} + \ldots + a_1 x^1 + a_0$$

over $\mathbb{F}_2$. Thus, the finite field $\mathbb{F}_{2^m}$ can be represented by the set of all polynomials of degree less than m over $\mathbb{F}_2$. That is,

$$\mathbb{F}_{2^m} = \{(a_{m-1} \ldots a_1 a_0) \mid a_i \in \{0,1\}\}.$$

The field arithmetic is implemented as polynomial arithmetic modulo $f(x)$. In this representation, addition and multiplication of $a = (a_{m-1} \ldots a_1 a_0)$ and $b = (b_{m-1} \ldots b_1 b_0)$ are performed as follows:

- *Addition:* $a + b = (c_{m-1} \ldots c_1 c_0)$, where $c_i = (a_i + b_i) \bmod 2$.
- *Multiplication:* $c = a \cdot b = (c_{m-1} \ldots c_1 c_0)$, where the polynomial $c(x) = \sum_{i=0}^{m-1} c_i x^i$ is the remainder of the division of polynomial $(\sum_{i=0}^{m-1} a_i x^i) \cdot (\sum_{i=0}^{m-1} b_i x^i)$ by $f(x)$. That is, $c = ab \bmod f$.

For efficiency reasons, the reduction polynomial can be selected as a *trinomial* $x^m + x^k + 1$, where $1 \leq k \leq m-1$ or a *pentanomial* $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, where $1 < k_1 < k_2 < k_3 < m-1$. ANSI X9.62 [5] specifies several rules for choosing the reduction polynomial.

In software implementations, we partition the bit representation of a field element $a = (a_{m-1} \ldots a_1 a_0)$ into blocks of the same size. Let w be the word size of a computer (typical values are $w = 8, 16, 32, 64$), and s be the number of words required to pack a into words. That is, $s = \lceil m/w \rceil$. Then, we can write a as an sw-bit number consisting of s words, where each word is of length w. Thus, we can write

$$a = (A_{s-1} \ldots A_1 A_0),$$

where each A_i is of length w and

$$A_i = (a_{iw+w-1} \ldots a_{iw+1} a_{iw}) \in \mathbb{F}_{2^w}.$$

In polynomials terms,

$$a(x) = \sum_{i=0}^{s-1} A_i(x)x^{iw} = \sum_{i=0}^{s-1}\sum_{j=0}^{w-1} a_{iw+j}x^{iw+j}. \tag{3.1}$$

3.2.2 Recent methods for multiplication in $\mathbb{F}_{2^m}$

In recent years, several algorithms for software multiplication in $\mathbb{F}_{2^m}$ have been reported; however, we are interested in techniques that can be used when m is prime.[1] In Schroeppel *et al.* [90] various programming tricks are discussed for implementing the "shift-and-add" method, a basic algorithm for multiplication in $\mathbb{F}_{2^m}$. A slight variant of this method is described by De Win *et al.* [24]. In Koç [52], a word-level Montgomery multiplication algorithm in $\mathbb{F}_{2^m}$ is proposed. This method is significantly faster than the standard method whenever the multiplication of two words of size w, each one representing a polynomial in $\mathbb{F}_{2^w}$ can be performed in few cycles. Since this operation is not available in most general purpose processors, the alternative is to use table lookup. This approach requires, for example, 128 Kbytes for $w = 8$ and 16 Gbytes for $w = 16$, making it less attractive for practical applications. Another well known method for multiplication in $\mathbb{F}_{2^m}$ is that of Karatsuba (see for example [14]).

3.2.3 The "shift-and-add" method

In this section we describe the basic method for computing $c(x) = a(x) \cdot b(x) \bmod f(x)$ in $\mathbb{F}_{2^m}$. It is analogous to the binary method for exponentiation, with the square and multiplication operations being replaced by the SHIFT (multiplication of a field element by x) and field addition operations, respectively. Thus, the "shift-and-add" method processes the bits of polynomial $a(x)$ from left to right, and uses the following equation to perform $c = ab \bmod f$:

$$c(x) = x(\cdots x(xa_{m-1}b(x) + a_{m-2}b(x) \bmod f(x)) + \cdots) + a_0 b(x) \bmod f(x).$$

Assume that $a(x) = \sum_{i=0}^{s-1} A_i x^{wi}$, $b(x) = \sum_{i=0}^{s-1} B_i x^{wi}$, and $f(x) = \sum_{i=0}^{s-1} F_i x^{wi}$. Then the steps of the "shift-and-add" method are given below.

[1] Many standards that include elliptic curves defined over $\mathbb{F}_{2^m}$ recommend for security reasons, the use of binary finite fields with the property that m be prime.

Algorithm 1: the "shift-and-add" method.

INPUT: $a = (A_{s-1} \dots A_0)$, $b = (B_{s-1} \dots B_0)$, and $f = (F_{s-1} \dots F_0)$.
OUTPUT: $c = (C_{s-1} \dots C_0) = a \cdot b \bmod f$.

1. Set $k \leftarrow m - 1 - w(s-1)$, $c \leftarrow 0$
2. **for** i **from** $s-1$ **downto** 0 **do**
   for j **from** k **downto** 0 **do**
   Set $c \leftarrow \mathtt{SHIFT}(c)$
   if $a_{iw+j} = 1$ **then** $c \leftarrow c \oplus b$
   if $c_m = 1$ **then** $c \leftarrow c \oplus f$
   Set $k \leftarrow w - 1$
3. **return** (c).

This algorithm requires $m-1$ shift operations and m field additions on average, but the number of field additions can be reduced by selecting the reduction polynomial $f(x)$ as a trinomial or a pentanomial. Observe that in this algorithm, the multiplication step (the computation of $d(x) = a(x) \cdot b(x)$) and the reduction step (the computation of $c(x) = d(x) \bmod f(x)$) are integrated. Since for the proposed algorithm these steps are separated, we include Algorithm 2 for performing the reduction step. Assume that $f(x) = x^m + g(x)$, where the degree of polynomial $g(x)$ is less than $m - w$.

Algorithm 2: modular reduction.

INPUT: $a = (A_{n-1} \dots A_{s-1} \dots A_0)$, and $f = (F_{s-1} \dots F_0)$.
OUTPUT: $c = (C_{s-1} \dots C_0) = a \bmod f$

1. **for** i **from** $n-1$ **downto** s **do**
   Set $d \leftarrow iw - m$
   Set $t \leftarrow A_i(x)x^d \cdot f(x) = \sum_{j=0}^{w-1} a_{iw+j} x^{d+j} \cdot f(x)$
   // $t = (T_i \dots T_{i-s} 0 \dots 0)$, where $T_i = A_i$ //
   for j **from** i **downto** $i-s$ **do**
   Set $A_j \leftarrow A_j \oplus T_j$
2. Set $t \leftarrow \sum_{j=0}^{sw-1-m} a_{m+j} x^j \cdot f(x)$
   // $t = (T_{s-1} \dots T_0)$ //
3. **for** j **from** $s-1$ **downto** 0 **do**
   Set $A_j \leftarrow A_j \oplus T_j$
4. **return** $(c \leftarrow (A_{s-1} \dots A_0))$.

Algorithm 2 works by zeroing out the most significant word of $a(x)$ in each iteration of step 1. A chosen multiple of the reduction polynomial $f(x)$ is added to $a(x)$ which lowers

the degree of $a(x)$ by w. This is possible because the degree of $g(x)$ is less than $m - w$. Finally, the leading $sw - m$ bits of A_{s-1} are cancelled in step 3 obtaining a polynomial of degree less than m. The number of XOR operations will depend on the weight of the reduction polynomial $f(x)$. For example, if $f(x)$ is a pentanomial then Algorithm 2 requires at most $8n$ XOR operations.

Remark 1. The use of standard programming tricks such as *separated name variables*, and *loop-unrolled code*, can be used to improve the performance of both Algorithms 1 and 2. See [90] for some suggested programming optimizations.

3.3 Proposed method

In this section we describe two versions of the new algorithm for multiplication in $\mathbb{F}_{2^m}$. The first version is a straightforward extension of Lim/Lee's method, which does not require extra temporary memory. The second version is based on a window technique. Before we describe the proposed algorithms, we discuss a simple version of Lim/Lee's method for exponentiation, using the terminology of additive groups; this will help us to understand the extension to $\mathbb{F}_{2^m}$.

In order to compute the "multiplication" $a \cdot g$ (the addition of g to itself a times) where a is an integer and g is an element of an additive group, the number a is divided into s words of size w. Then a can be written as

$$a = (A_{s-1} \dots A_1 A_0) = \sum_{i=0}^{s-1} A_i 2^{wi},$$

where each $A_i, 0 \leq i < s$, has the binary representation $(a_{iw+w-1} \dots a_{iw+1} a_{iw})_2$. Based on the binary representation $(u_{s-1} \dots u_1 u_0)_2$ of u, $1 \leq u < 2^s$, and the group elements $2^{wi} \cdot g, 0 \leq i < s-1$, define the vector $P[u]$ of precomputations by the following equation:

$$P[u] = u_{s-1} 2^{w(s-1)} \cdot g + u_{s-2} 2^{w(s-2)} \cdot g + \dots + u_1 2^w \cdot g + u_0 \cdot g.$$

Then the multiplication $a \cdot g = \sum_{i=0}^{s-1} A_i 2^{wi} \cdot g$, can be computed as

$$a \cdot g = \sum_{j=0}^{w-1} 2^j \Big(\sum_{i=0}^{s-1} a_{iw+j} 2^{wi} \cdot g \Big) = \sum_{j=0}^{w-1} 2^j P[I_j], \tag{3.2}$$

where $I_j = (a_{(s-1)w+j} \dots a_{w+j} a_j)_2$. A detailed algorithm for computing $a \cdot g$ using the Lim/Lee's precomputation technique is given in Algorithm 3.

Algorithm 3: Lim/Lee's algorithm.

INPUT: $a = \sum_{i=0}^{s-1} A_i 2^{wi}, A_i = (a_{iw+w-1} \dots a_{iw})_2, 0 \leq i < s$, and g.
OUTPUT: $r = a \cdot g$

// Precomputation //
1. **for** u **from** 0 **downto** $2^s - 1$ **do**
   Set $u \leftarrow (u_{s-1} \dots u_1 u_0)_2$
   Set $P[u] \leftarrow \sum_{i=0}^{s-1} u_i 2^{wi} \cdot g$
// Main Computation //
2. Set $r \leftarrow 0$
3. **for** j **from** $w-1$ **downto** 0 **do**
   Set $r \leftarrow r + r$
   Set $u \leftarrow (a_{(s-1)w+j} \dots a_{w+j} a_j)_2$
   Set $r \leftarrow r + P[u]$
4. **return** (r).

Algorithm 3 performs well in situations where the group element g is known in advance, since the calculation of the precomputation step can be made off-line. A faster version of this algorithm, with more precomputations, is discussed in [58].

Next we explain the extension of Algorithm 3 to the finite field $\mathbb{F}_{2^m}$. Let a and b be two polynomials in $\mathbb{F}_{2^m}$. Assume that a can be represented as $a = (A_{s-1} \dots A_0)$. By replacing 2 by x and $2^w \cdot g$ by $x^w b(x)$ in (3.2), we obtain the following formal expression for the product $a(x)b(x)$:

$$a(x)b(x) = \sum_{j=0}^{w-1} x^j \Big(\sum_{i=0}^{s-1} a_{iw+j} x^{wi}\Big) b(x).$$

It is easy to verify that indeed the above formula for $a(x)b(x)$ is correct. Then an algorithm, analogue of Algorithm 3, can be derived for computing $ab \bmod f$ when b is a polynomial known in advance. By observing that the operation $x^{wi}b(x)$ is virtually free (it consists of an arrangement of the words representing b), the precomputation of the $2^s - 1$ polynomials: $P[u] = \sum_{i=0}^{s-1} u_i x^{wi}, 1 < u < 2^s, u = (u_{s-1} \dots u_0)_2$, can be made online. This eliminates the need of storing $2^s - 1$ polynomials, and the resulting algorithm is faster than Algorithm 1, even when b is not a fixed polynomial. The details of this method are given in Algorithm 4.

Algorithm 4: basic proposed method.

INPUT: $a = (A_{s-1} \dots A_0)$, $b = (B_{s-1} \dots B_0)$, and $f = (F_{s-1} \dots F_0)$.
OUTPUT: $c = (C_{s-1} \dots C_0) = ab \bmod f$

1. Set $T_i \leftarrow 0$; $i = 0, \dots, 2s-1$
2. **for** j **from** $w-1$ **downto** 0 **do**
   for i **from** 0 **to** $s-1$ **do**
   if $a_{iw+j} \neq 0$ **then**
   for k **from** 0 **to** $s-1$ **do**
   Set $T_{k+i} \leftarrow T_{k+i} \oplus B_k$
   if $j \neq 0$ **then** $T \leftarrow xT$ // shift T//
3. Set $c \leftarrow T$ **mod** f // Use Algorithm 2 //
4. **return** (c).

The idea of window methods [14, pp. 66] for exponentiation can be extended to Algorithm 4 to obtain a more efficient algorithm, provided that extra temporary memory is available. For example, if we define the precomputed vector $P_{16}[u]$ for $0 \leq u < 16$, using the equation

$$P_{16}[u](x) = (u_3x^3 + u_2x^2 + u_1x + u_0)b(x),$$

where $u = (u_3 \dots u_0)_2$, then the product $a(x)b(x)$ can be computed as

$$\begin{aligned}
a(x)b(x) &= \sum_{i=0}^{s-1}\sum_{j=0}^{w-1} a_{iw+j}x^{iw+j}b(x) \\
&= \sum_{j=0}^{w-1} x^j \sum_{i=0}^{s-1} a_{iw+j}x^{iw}b(x) \\
&= \sum_{j=0}^{w/4-1} x^{4j} \sum_{i=0}^{s-1} (a_{iw+j+3}x^3 + \cdots + a_{iw+j+1}x + a_{iw+j})x^{iw}b(x) \\
&= \sum_{j=0}^{w/4-1} x^{4j} (\sum_{i=0}^{s-1} x^{wi} P_{16}[u_{i,j}](x)), \text{ where } u_{i,j} = (a_{iw+j+3} \dots a_{iw+j})_2.
\end{aligned}$$

Based on the above formula for ab, we derived an algorithm that processes simultaneously four bits of each word of a and trades in each iteration four multiplications by x for one multiplication by x^4. This method is described in Algorithm 5.

Algorithm 5: fast proposed method.

INPUT: $a = (A_{s-1} \dots A_0)$, $b = (B_{s-1} \dots B_0)$, and $f = (F_{s-1} \dots F_0)$.
OUTPUT: $c = (C_{s-1} \dots C_0) = ab \bmod f$.

1. **for** j **from** 0 **to** 15 **do**
   Set $P_{16}[j] \leftarrow (j_3 x^3 + \cdots + j_0) b(x), j = (j_3 j_2 j_1 j_0)_2$
2. Set $T_i \leftarrow 0$; $i = 0, \dots, 2s-1$
3. **for** j **from** $w/4 - 1$ **downto** 0 **do**
   for i **from** 0 **to** $s-1$ **do**
   Set $u_{i,j} \leftarrow A_i / 2^{4j} \bmod 16$
   for k **from** 0 **to** $s-1$ **do**
   Set $T_{k+i} \leftarrow T_{k+i} \oplus P_{16}[u_{i,j}][k]$
   if $j \neq 0$ **then** $T \leftarrow x^4 T$
4. Set $c \leftarrow T$ **mod** f // Use Algorithm 2 //
5. **return** (c).

Remark 2. When b is known in advance, Algorithm 5 can be modified to work with a larger window size. If we process eight bits at the same time, then we need 256 field elements of precomputations. By observing that $\sum_{i=0}^{7} a_i x^i b(x) = \sum_{j=0}^{3} a_j x^j b(x) + \sum_{j=0}^{3} a_{4+j} x^j x^4 b(x)$, we reduce the precomputation to 32 field elements at the expense of doing more XOR operations.

3.3.1 Performance comparison

Let us compare the performance of Algorithms 4 and 5. We calculate the number of XOR operations and SHIFT operations required in each algorithm. We assume that the reduction polynomial is a pentanomial, so the total number of XOR operations required by Algorithm 2 is at most $8(2s-1)$. Therefore, Algorithm 4 requires $2(w-1)$ SHIFT operations and $sm/2 + 8(2s-1)$ XOR operations on average. Similarly, Algorithm 5 requires $3 + 2(w/4 - 1)$ SHIFT[2] operations and $s(11 + m/4) + 8(2s-1)$ XOR operations on average. Thus, the time saved in Algorithm 5 is at the expense of using 16 field elements of temporary memory. In Table 3.1 we compared the number of operations required by Algorithms 1, 4 and 5, for the particular case $m = 163$, $w = 32$, $s = 6$, and the pentanomial $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$.

[2]We are assuming that multiplying a polynomial by x^4 is comparable in speed to multiplying a polynomial by x.

Algorithms	XOR	SHIFT
Algorithm 1	81*6+ 81*2 = 648	162
Algorithm 4	81*6 + 42 = 528	62
Algorithm 5	52*6 + 42 = 354	17

Table 3.1: Number of operations for Algorithms 1, 4 and 5.

3.4 Timing results

This section presents running timings for the proposed algorithms and the "shift-and-add" method on the following platforms: a 233 MHz Pentium MMX, a 400 MHz Pentium II, a 450 MHz Sun UltraSparc workstation and a 10 MHz Intel 386 processor (RIM interactive pager [13]). The implementation was written entirely in C, and the compilers used were gcc for the workstation Sun and the Pentium MMX, and Microsoft Visual C++ (version 6.0) for the other architectures. All algorithms were implemented with a comparable level of programming optimizations.

Tables 3.2 and 3.3 show timings to perform a multiplication in $\mathbb{F}_{2^{163}}$ using Algorithms 1, 4 and 5.[3] From Table 3.2, Algorithm 4 performs 45% to 49% faster than Algorithm 1, and the best speed up was obtained on the UltraSparc machine. In Table 3.3 the performances of the fast version of the proposed algorithm (Algorithm 5) and the standard method are compared. We observed a significant improvement: Algorithm 5 is about 3.0 to 5.5 times faster than the standard method.

	Pentium 233 MHz	UltraSparc 450 MHz
Algorithm 1	31.27	10.97
Algorithm 4	17.07	5.55

Table 3.2: Timings (in microseconds) of the "shift-and-add" method and Algorithm 4 for multiplication in $\mathbb{F}_{2^{163}}$.

3.4.1 Applications

The most important application of this work is in software implementations of elliptic curve cryptography over $\mathbb{F}_{2^m}$. Our timings on different architectures have shown that Algorithm 5 is significantly faster than the standard method in modern workstations

[3]Recently, NIST has recommended elliptic curves over $\mathbb{F}_{2^{163}}$ for US federal government use [72].

	RIM 10 MHz	Pentium 233 MHz	Pentium II 400 MHz	UltraSparc 450 MHz
Algorithm 1	4,848	31.27	16.48	10.97
Algorithm 5	1,515	10.20	2.97	2.52

Table 3.3: Timings (in microseconds) of the "shift-and-add" method and Algorithm 5 for multiplication in $\mathbb{F}_{2^{163}}$.

as well as in wireless devices such as the RIM pager (a hand-held device with an Intel processor running at 10 MHz [13]).

3.5 Conclusions

There are several techniques that can be used for speeding up the computation of $c = ab \bmod f$ in $\mathbb{F}_{2^m}$. In this paper we have shown a technique based on Lim/Lee's method for exponentiations. It turns out that our software implementation of the optimized version (Algorithm 5), on different platforms, proved to be significantly faster than the "shift-and-add" method, making it useful for software implementations of elliptic curve cryptography in different computational environments.

Capítulo 4

Algoritmos Eficientes para a Aritmética em Curvas Elípticas sobre $\mathbb{F}_{2^m}$

Este capítulo descreve três contribuições para a implementação eficiente dos criptossistemas de curvas elípticas sobre $\mathbb{F}_{2^m}$. A primeira é um método novo para duplicar um ponto elíptico, o qual é mais simples de implementar do que o melhor método conhecido, desenvolvido por Schroeppel, e que favorece coeficientes elípticos dispersos. A segunda é uma versão generalizada e melhorada das fórmulas de Guajardo e Paar para calcular duplicações consecutivas de um ponto elíptico. A terceira contribuição consiste em um sistema novo de coordenadas projetivas. Os algoritmos resultantes desta formulação levam a um ganho de 17% na computação de uma multiplicação escalar, comparado com métodos anteriores baseados em coordenadas projetivas.

Este capítulo é uma versão revisada do artigo apresentado no workshop: fifth annual workshop on Selected Areas in Cryptography, SAC'98, Kingston, Canadá. Publicado em Lecture Notes in Computer Science, **1556**, pp. 201-212, Springer-Verlag, 1998.

Improved Algorithms for Elliptic Curve Arithmetic in $\mathbb{F}_{2^m}$*

Julio López and Ricardo Dahab
State University of Campinas
Campinas, SP, Brazil
{julioher,dahab}@dcc.unicamp.br

Abstract

This paper describes three contributions for efficient implementation of elliptic curve cryptosystems in $\mathbb{F}_{2^m}$. The first is a new method for doubling an elliptic curve point, which is simpler to implement than the fastest known method, due to Schroeppel, and which favors sparse elliptic curve coefficients. The second is a generalized and improved version of the Guajardo and Paar's formulas for computing repeated doubling points. The third contribution consists of a new kind of projective coordinates that provides the fastest known arithmetic on elliptic curves. The algorithms resulting from this new formulation lead to a running time improvement for computing a scalar multiplication of about 17% over previous projective coordinate methods.

4.1 Introduction

Elliptic curves defined over finite fields of characteristic two have been proposed for Diffie-Hellman type cryptosystems [26]. The calculation of $Q = kP$, for P a point on the elliptic curve and k an integer, is the core operation of elliptic curve public-key cryptosystems. Therefore, reducing the number of field operations required to perform the scalar multiplication kP is crucial for efficient implementation of these cryptosystems.

In this paper we discuss efficient methods for implementing elliptic curve arithmetic. We present better results than those reported in [96, 42, 37]; our basic technique is to rewrite the elliptic operations (doubling and addition) with less costly field operations (inversions and multiplications), and replace general field multiplications by multiplications by fixed elliptic coefficients.

The first method is a new formula for doubling a point, i.e., for calculating the sum of equal points. This method is simpler to implement than Schroeppel's method [96] since it does not require a quadratic solver. If the elliptic curve coefficient b is sparse, i.e., with few

*This paper is a revised version of the paper appearing in the Proceedings of SAC'98.

1's in its representation, thus making the multiplication by the constant b more efficient than a general field multiplication, then our new formula should lead to an improvement of up to 12% compared to Schroeppel's method [96]. We also note that our formula can be applied to composite finite fields as well.

In [37], a new approach is introduced for accelerating the computation of repeated doubling points. This method can be viewed as computing consecutive doublings using fractional field arithmetic. We have generalized and improved the formulas presented in that paper. The new formulas can be used to speed-up variants of the sliding-window method. For field implementations where the cost-ratio of inversion to multiplication varies from 2.5 to 4 (typical values of practical software field implementations), we expect a speed-up of 7% to 22% in performing a scalar multiplication.

In [91], Schroeppel proposes an algorithm for computing repeated doubling points removing most of the general field multiplications, and favoring elliptic curves with sparse coefficients. Using his method, the computation of $2^iP, i \geq 2$ requires i field inversions, i multiplications by a fixed constant, one general field multiplication, and a quadratic solver. Since inversion is the most expensive field operation, this method is suitable for finite fields where field inversion is relatively fast. If the cost-ratio of inversion to multiplication is less than 3, this algorithm may be faster than our repeated doubling algorithm.

When field inversion is costly (e.g., for normal basis representation, the cost-ratio of inversion to multiplication is at least 7 [37, 96]), projective coordinates offer an alternative method for efficiently implementing the elliptic curve arithmetic. Based on our doubling formula, we have developed a new kind of projective coordinates which should lead to an improvement of 38% over the traditional projective arithmetic coordinates [64] and 17% on the recent projective coordinates presented in [42], for calculating a multiple of a point.

The remainder of the paper is organized as follows. Section 4.2 presents a brief summary of elliptic curves defined over finite fields of characteristic two. In Section 4.3, we present our doubling point algorithm. Based on this method, we describe an algorithm for repeated doubling points in Section 4.4. In Section 4.5, we describe the new projective coordinates. An implementation of the doubling and adding projective algorithms is given in the appendix.

4.2 Elliptic curves over $\mathbb{F}_{2^m}$

A non-supersingular elliptic curve E over $\mathbb{F}_{2^m}$ is defined to be the set of solutions $(x, y) \in \mathbb{F}_{2^m} \times \mathbb{F}_{2^m}$ to the equation,

$$y^2 + xy = x^3 + ax^2 + b \ ,$$

where a and $b \in \mathbb{F}_{2^m}, b \neq 0$, together with the point at infinity denoted by $\mathcal{O}$.

It is well known that E forms a commutative finite group, with $\mathcal{O}$ as the group identity, under the addition operation known as the "tangent and chord method". Explicit rational formulas for the addition rule involve several arithmetic operations (adding, squaring, multiplication and inversion) in the underlying finite field. In what follows, we will only be concerned with formulas for doubling a point P in affine coordinates; formulas for adding two different points in affine or projective coordinates can be found in [64, 42].

Let $P = (x_1, y_1)$ be a point of E. The doubling point formula [64] to compute $2P = (x_2, y_2)$ is given by

$$\begin{cases} x_2 &= x_1^2 + \frac{b}{x_1^2} \ , \\ y_2 &= x_1^2 + (x_1 + \frac{y_1}{x_1}) \cdot x_2 + x_2 \ . \end{cases} \tag{4.1}$$

Note that the x-coordinate of doubling point formula $2P$ depends only on the x-coordinate of P and the coefficient b, but doubling a point requires two general field multiplications, one multiplication by the constant b and one field inversion.

Schroeppel [89] improved the doubling point formula saving the multiplication by the constant b. His improved doubling point formula is :

$$\begin{cases} x_2 &= M^2 + M + a \ , \\ y_2 &= x_1^2 + M \cdot x_2 + x_2 \ , \\ M &= x_1 + \frac{x_1}{y_1} \ . \end{cases} \tag{4.2}$$

Observe that the x-coordinates of the previous doubling point formula lead to the quadratic equation for M:

$$M^2 + M + a \ = \ x_1^2 + \frac{b}{x_1^2} \ . \tag{4.3}$$

If we assume that the cost of multiplying by a sparse fixed constant is comparable in speed to field addition, and that solving the previous quadratic equation is faster, then we obtain another method for doubling a point with an effective cost of one general multiplication and one field inversion. A description of this method, developed by Schroeppel, can be found in [96, pp. 370-371] and [42].

In the next section, we introduce a new doubling point formula which requires also a general field multiplication, one field inversion, but does not depend on a quadratic solver.

4.3 A New doubling point formula

Given an elliptic curve point $P = (x_1, y_1)$, the coordinates of the doubling point $2P = (x_2, y_2)$ can be calculated by the following new doubling point formula:

$$\begin{cases} x_2 &= x_1^2 + \frac{b}{x_1^2} \ , \\ y_2 &= \frac{b}{x_1^2} + ax_2 + (y_1^2 + b) \cdot (1 + \frac{b}{x_1^4}) \ . \end{cases} \tag{4.4}$$

To derive the above formula we transform the y-coordinate of the doubling point formula (4.2):

$$\begin{aligned} y_2 &= x_1^2 + (x_1 + \frac{y_1}{x_1}) \cdot x_2 + x_2 = \frac{b}{x_1^2} + (\frac{y_1^2 + b + ax_1^2}{x_1^2}) \cdot x_2 \\ &= \frac{b}{x_1^2} + ax_2 + \frac{y_1^2 + b}{x_1^2} \cdot (\frac{x_1^4 + b}{x_1^2}) = \frac{b}{x_1^2} + ax_2 + (y_1^2 + b) \cdot (1 + \frac{b}{x_1^4}) \ . \end{aligned}$$

4.3.1 Performance analysis

We begin with the observation that our doubling formula eliminates the need for computing the field element M from formula (4.2), which requires either one general multiplication or a quadratic solver. The calculation of $2P$ requires one general field multiplication, two field multiplications by the fixed constant b, and one field multiplication by the constant a. This last multiplication can be avoided by choosing the coefficient a to be 0 or 1.[1] Thus, our formula favors elliptic curves with sparse coefficients, i.e., those having relatively few 1's in their representation.

In order to compare the running time of our formula with Schroeppel's method [96] for computing a scalar multiplication, we made the following assumptions:

- Adding and squaring field elements is fast compared to a multiplication.
- Multiplying a field element by a sparse constant is comparable to adding.
- The cost of solving the quadratic equation (4.3) and determining the right solution is about half of that of a field multiplication (this is true for the finite field implementation given in [89], but no efficient method is known for tower fields [91]).

The fastest methods for computing a scalar multiplication [89, 54] perform five point doublings for every point-addition, on average. Table 4.1 compares our formula, in performing a scalar multiplication, for different values of the cost-ratio r of inversion to multiplication.

[1] E is isomorphic to E_1: $y^2 + xy = x^3 + \alpha x^2 + b$, where $Tr(\alpha) = Tr(a)$, $\alpha = 0$ or γ and $Tr(\gamma) = 1$ (if n is odd, we can take $\gamma = 1$), see [64, pp. 39].

Table 4.1: The number of field multiplications for computing $2^5P + Q$.

Cost-Ratio	New Formula #Mult.	Schroeppel [96] #Mult.	Improv. %
$r = 2$	19	21.5	12
$r = 2.5$	22	24.5	10
$r = 3$	25	27.5	9
$r = 4$	31	33.5	7

Therefore, for practical field implementations as those given in [89, 25, 37], our formula should lead to a running time improvement of up to 12% in computing a scalar multiplication. However, for elliptic curves selected at random (where the coefficient b is not necessarily sparse), both our and Schroeppel's method may not give a computational advantage. A better algorithm for computing 2^5P is presented in the next section.

4.4 Repeated doubling algorithm

We present a method for computing repeated doublings, $2^iP, i \geq 2$, which is based on fractional field arithmetic and the doubling formula. The idea is to successively compute the elliptic points $2^jP = (x_j, y_j)$, $j = 2, 3, \ldots, i$, as triples $(\nu_j, \omega_j, \delta_j)$ of field elements, where $x_i = \frac{\nu_i}{\delta_i}$ and $y_i = \frac{\omega_i}{\delta_i^2}$. The exact formulation is given in the following theorem.

Theorem 1 *Let $P = (x, y)$ be a point on the elliptic curve E. Then the coordinates of the point $2^iP = (x_i, y_i), i \geq 2$, are given by*

$$x_i = \frac{\nu_i}{\delta_i} , \tag{4.5}$$

$$y_i = \frac{\omega_i}{\delta_i^2} , \tag{4.6}$$

where

$$\begin{aligned} \nu_{k+1} &= \nu_k^4 + b\delta_k^4 \ , \quad \nu_0 = x \\ \delta_{k+1} &= (\delta_k \cdot \nu_k)^2 \ , \quad \delta_0 = 1 \\ \omega_{k+1} &= b\delta_k^4 \cdot \delta_{k+1} + \nu_{k+1} \cdot (a\delta_{k+1} + \omega_k^2 + b\delta_k^4) \ , \quad \omega_0 = y, \ 0 \leq k < i. \end{aligned}$$

Proof. We will prove by induction on i that $x_i = \frac{\nu_i}{\delta_i}$ and $y_i = \frac{\omega_i}{\delta_i^2}$. This is easily true for $i = 2$. Now assume that the statement is true for $i = n$; we prove it for $i = n + 1$:

$$\begin{aligned} x_{n+1} &= \frac{b}{x_n^2} + x_n^2 = \frac{b\delta_n^2}{\nu_n^2} + \frac{\nu_n^2}{\delta_n^2} \\ &= \frac{b\delta_n^4 + \nu_n^4}{\nu_n^2 \cdot \delta_n^2} = \frac{\nu_{n+1}}{\delta_{n+1}} \ ; \end{aligned}$$

similarly, for y_{n+1} we obtain:

$$\begin{aligned} y_{n+1} &= \frac{b}{x_n^2} + ax_{n+1} + (y_n^2 + b) \cdot (1 + \frac{b}{x_n^4}) \\ &= \frac{b\delta_n^2}{\nu_n^2} + a\frac{\nu_{n+1}}{\delta_{n+1}} + (\frac{\omega_n^2}{\delta_n^4} + b) \cdot (1 + \frac{b\delta_n^4}{\nu_n^4}) \\ &= \frac{b\delta_n^4}{\delta_{n+1}} + a\frac{\nu_{n+1}}{\delta_{n+1}} + \frac{(\omega_n^2 + b\delta_n^4) \cdot \nu_{n+1}}{\delta_{n+1}^2} \\ &= \frac{\omega_{n+1}}{\delta_{n+1}^2} \ . \end{aligned}$$

The following algorithm, based on Theorem 1, implements repeated doublings in terms of the affine coordinates of $P = (x, y)$.

Figure 4.1: Algorithm 1: Repeated doubling points.

INPUT: $P = (x, y) \in E \quad i \geq 2$.
OUTPUT: $Q = 2^i P$.

Set $V \leftarrow x^2$, $D \leftarrow V$, $W \leftarrow y$, $T \leftarrow b$.
for $k = 1$ **to** $i - 1$ **do**
  Set $V \leftarrow V^2 + T$.
  Set $W \leftarrow D \cdot T + V \cdot (aD + W^2 + T)$.
  if $k \neq i - 1$ **then**
    $V \leftarrow V^2$, $D \leftarrow D^2$, $T \leftarrow bD^2$, $D \leftarrow D \cdot V$.
  fi
od
Set $D \leftarrow D \cdot V$.
Set $M \leftarrow D^{-1} \cdot (V^2 + W)$.
Set $x \leftarrow D^{-1} \cdot V^2$.
Set $x_i \leftarrow M^2 + M + a, \quad y_i \leftarrow x^2 + M \cdot x_i + x_i$.
return$(Q \leftarrow (x_i, y_i))$.

Note that the correctness of this algorithm follows directly from the proof of Theorem 1 and formula (4.2).

Corollary 1 *Assume that P is an elliptic point of order larger than 2^i. Then Algorithm 1 performs $3i-1$ general field multiplications, $i-1$ multiplications by the fixed constant b, and $5i-4$ field squarings.*

4.4.1 Complexity comparison

Since Algorithm 1 cuts down the number of field inversions at the expense of more field multiplications, the computational advantage of Algorithm 1 over repeated doubling (using the standard point doubling formula (4.2)) depends on r, the cost-ratio of inversion to multiplication. Assuming that adding and squaring is fast, we conclude, from Corollary 1, that Algorithm 1 outperforms the computation of five consecutive doublings when $r > 2$. Table 4.2 shows the number of field multiplications needed for computing $2^5P + Q$ for several methods and for different values of r. Note that the standard algorithm and Guajardo and Paar's formulas do not use the elliptic curve coefficient b, whereas Algorithm 1 does.

Table 4.2: Comparison of Algorithm 1 with other algorithms.

Ratio	Algorithm 1		Schroeppel [91]		G.P. [37]	Standard (4.2)
r	b sparse	b random	b sparse	b random	b random	b random
2.5	21	25	18.5	22.5	27	27
3	22	26	21.5	25.5	28	30
3.5	23	27	24.5	28.5	29	33
4	24	28	27.5	31.5	30	36

Algorithm 1 obtains its best performance for field implementations when r is at least three. If the elliptic curve is selected at random, then we expect Algorithm 1 to be up to 22% faster than the standard algorithm. For field implementations where $r < 3$, (for example [89, 25]), Schroeppel's method [91] outperforms Algorithm 1.

4.5 A new kind of projective coordinates

When field inversion in $\mathbb{F}_{2^m}$ is relatively expensive, then it may be of computational advantage to use fractional field arithmetic to perform elliptic curve additions, as well as, doublings. This is done with the use of projective coordinates.

4.5.1 Basic facts

A *projective plane* P^2 is defined to be the set of equivalence classes of triples (X, Y, Z), not all zero, where (X_1, Y_1, Z_1) and (X_2, Y_2, Z_2) are said to be equivalent if there exists $\lambda \in \mathbb{F}_{2^m}$, $\lambda \neq 0$ such that $X_1 = \lambda X_2, Y_1 = \lambda^2 Y_2$ and $Z_1 = \lambda Z_2$. Each equivalence class is called a *projective point.* Note that if a projective point $P = (X, Y, Z)$ has nonzero Z, then P can be represented by the projective point $(x, y, 1)$, where $x = X/Z$ and $y = Y/Z^2$. Therefore, the projective plane can be identified with all points (x, y) of the ordinary (affine) plane plus the points for which $Z = 0$.

Any equation $f(x, y) = 0$ of a curve in the affine plane corresponds to an equation $F(X, Y, Z) = 0$, where F is obtained by replacing $x = X/Z$, $y = Y/Z^2$, and multiplying by a power of Z to clear the denominators. In particular, the *projective equation* of the affine equation $y^2 + xy = x^3 + ax^2 + b$ is given by

$$Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4 \ .$$

If $Z = 0$ in this equation, then $Y^2 = 0$, i.e., $Y = 0$. Therefore, $(1, 0, 0)$ is the only projective point that satisfies the equation for which $Z = 0$. This point is called *the point at infinity* (denoted $\mathcal{O}$).

The resulting projective elliptic equation is

$$E = \{(x, y, z) \in P^2, y^2 + xyz = x^3z + ax^2z^2 + bz^4\} \ .$$

To convert an affine point (x, y) to a projective point, one sets $X = x$, $Y = y$, $Z = 1$. Similarly, to convert a projective point (X, Y, Z) to an affine point, we compute $x = X/Z$, $y = Y/Z^2$. The projective coordinates of the point $-P(X, Y, Z)$ are given by $-P(X, Y, Z) = (X, XZ + Y, Z)$. The algorithms for adding two projective points are given below.

4.5.2 Projective elliptic arithmetic

In this section we present new formulas for adding elliptic curve points in projective coordinates. These formulas can be derived directly from the formulas for adding points in affine coordinates (see [64]).

Projective elliptic doubling

The projective form of the doubling formula is

$$2(X_1, Y_1, Z_1) = (X_2, Y_2, Z_2) \ ,$$

where

$$\begin{aligned} Z_2 &= Z_1^2 \cdot X_1^2 \ , \\ X_2 &= X_1^4 + b \cdot Z_1^4 \ , \\ Y_2 &= bZ_1^4 \cdot Z_2 + X_2 \cdot (aZ_2 + Y_1^2 + bZ_1^4) \ . \end{aligned}$$

Projective elliptic addition

The projective form of the adding formula is

$$(X_0, Y_0, Z_0) + (X_1, Y_1, Z_1) = (X_2, Y_2, Z_2) \ ,$$

where

$$\begin{array}{lll} A_0 = Y_1 \cdot Z_0^2 \ , & D = B_0 + B_1 \ , & H = C \cdot F \ , \\ A_1 = Y_0 \cdot Z_1^2 \ , & E = Z_0 \cdot Z_1 \ , & X_2 = C^2 + H + G \ , \\ B_0 = X_1 \cdot Z_0 \ , & F = D \cdot E \ , & I = D^2 \cdot B_0 \cdot E + X_2 \ , \\ B_1 = X_0 \cdot Z_1 \ , & Z_2 = F^2 \ , & J = D^2 \cdot A_0 + X_2 \ , \\ C = A_0 + A_1 \ , & G = D^2 \cdot (F + aE^2) \ , & Y_2 = H \cdot I + Z_2 \cdot J \ . \end{array}$$

These formulas can be improved for the special case $Z_1 = 1$:

$$(X_0, Y_0, Z_0) + (X_1, Y_1, 1) = (X_2, Y_2, Z_2),$$

where

$$\begin{array}{ll} A = Y_1 \cdot Z_0^2 + Y_0 \ , & E = A \cdot C \ , \\ B = X_1 \cdot Z_0 + X_0 \ , & X_2 = A^2 + D + E \ , \\ C = Z_0 \cdot B \ , & F = X_2 + X_1 \cdot Z_2 \ , \\ D = B^2 \cdot (C + aZ_0^2) \ , & G = X_2 + Y_1 \cdot Z_2 \ , \\ Z_2 = C^2 \ , & Y_2 = E \cdot F + Z_2 \cdot G \ . \end{array}$$

4.5.3 Performance analysis

The new projective doubling algorithm requires three general field multiplications, two multiplications by a fixed constant, and five squarings. Since doubling a point takes one general field multiplication less than the previous projective doubling algorithm given in [42], we obtain an improvement of about 20% for doubling a point, in general. For sparse coefficients b, we may obtain an improvement of up to a 25%.

The new projective adding algorithm requires 13 general multiplications, one multiplication by a fixed constant and six squarings. If $a = 0$ (or $a = 1$) and $Z_1 = 1$, then only

nine general field multiplications and four squarings are required. Thus, we obtain one field multiplication less than the previous projective addition algorithm presented in [42]. The number of field operations required to perform an elliptic addition for various kinds of projective coordinates is listed in Table 4.3.

Now we can estimate the improvement of a scalar multiplication using the new projective coordinates. We will consider only the case $a = 0$ (or $a = 1$) and $Z_1 = 1$, since for this situation we obtain the best improvement. The number of field operations for computing $2^5P + Q$ is given in Table 4.3. Using these values we can conclude that the computation of a scalar multiplication, based on the new projective coordinates, is on average 17% and 38% faster than the previous projective coordinates [64, 42].

Table 4.3: The number of field operations for $2^5P + Q$ ($a = 0$ or 1, $Z_1 = 1$)

Projective coordinates	Doubling		Adding		Cost of $2^5P + Q$	
	#Mult.	#Sqr.	#Mult.	#Sqr.	#Mult.	#Sqr.
$(x/z, y/z^2)$	4	5	9	4	29	29
$(x/z^2, y/z^3)$	5	5	10	4	35	29
$(x/z, y/z)$	7	5	12	1	47	26

4.6 Conclusions

We have presented improved methods for faster implementation of the arithmetic of an elliptic curve defined over $\mathbb{F}_{2^m}$. Our methods are easy to implement and can be applied to all elliptic curves defined over fields of characteristic two, independently of the specific field representation. They favor sparse elliptic coefficients but also perform well for elliptic curves selected at random. In general, they should lead to an improvement of up to 20% in the computation of a scalar multiplication.

4.7 Appendix

Algorithm 2: Projective elliptic doubling algorithm

Input: the finite field $\mathbb{F}_{2^m}$; the field elements a and $c = b^{2^{m-1}} (c^2 = b)$ defining a curve E over $\mathbb{F}_{2^m}$; projective coordinates (X_1, Y_1, Z_1) for a point P_1 on E.
Output: projective coordinates (X_2, Y_2, Z_2) for the point $P_2 = 2P_1$.

1. $T_1 \leftarrow X_1$
2. $T_2 \leftarrow Y_1$
3. $T_3 \leftarrow Z_1$
4. $T_4 \leftarrow c$
5. if $T_1 = 0$ or $T_3 = 0$ then
   output $(1, 0, 0)$ and stop.
6. $T_3 \leftarrow T_3^2$
7. $T_4 \leftarrow T_3 \times T_4$
8. $T_4 \leftarrow T_4^2$
9. $T_1 \leftarrow T_1^2$
10. $T_3 \leftarrow T_1 \times T_3$ $= Z_2$
11. $T_1 \leftarrow T_1^2$
12. $T_1 \leftarrow T_4 + T_1$ $= X_2$
13. $T_2 \leftarrow T_2^2$
14. if $a \neq 0$ then
    $T_5 \leftarrow a$
    $T_5 \leftarrow T_3 \times T_5$
    $T_2 \leftarrow T_5 + T_2$
15. $T_2 \leftarrow T_4 + T_2$
16. $T_2 \leftarrow T_1 \times T_2$
17. $T_4 \leftarrow T_3 \times T_4$
18. $T_2 \leftarrow T_4 + T_2$ $= Y_2$
19. $X_2 \leftarrow T_1$
20. $Y_2 \leftarrow T_2$
21. $Z_2 \leftarrow T_3$

This algorithm requires 3 general field multiplications, 5 field squarings and 5 temporary variables. If also $a = 0$, then only 4 temporary variables are required.

Algorithm 3: Projective elliptic adding algorithm

Input: the finite field $\mathbb{F}_{2^m}$; the field elements a and b defining a curve E over $\mathbb{F}_{2^m}$; projective coordinates (X_0, Y_0, Z_0) and $(X_1, Y_1, 1)$ for points P_0 and P_1 on E.
Output: projective coordinates (X_2, Y_2, Z_2) for the point $P_2 = P_0 + P_1$, unless $P_0 = P_1$. In this case, the triple $(0, 0, 0)$ is returned. (The triple (0,0,0) is not a valid projective point on the curve, but rather a marker indicating that the Doubling Algorithm should be used, see [42].)

1. $T_1 \leftarrow X_0$
2. $T_2 \leftarrow Y_0$
3. $T_3 \leftarrow Z_0$
4. $T_4 \leftarrow X_1$
5. $T_5 \leftarrow Y_1$
6. $T_6 \leftarrow T_4 \times T_3$
7. $T_1 \leftarrow T_6 + T_1$ $= B$
8. $T_6 \leftarrow T_3^2$
9. if $a \neq 0$ the
   $T_7 \leftarrow a$
   $T_7 \leftarrow T_6 \times T_7$
10. $T_6 \leftarrow T_5 \times T_6$
11. $T_2 \leftarrow T_6 + T_2$ $= A$
12. if $T_1 = 0$ then
    if $T_2 = 0$ then output $(0, 0, 0)$ and stop.
    else output (1,0,0) and stop.
13. $T_6 \leftarrow T_1 \times T_3$ $= C$
14. $T_1 \leftarrow T_1^2$
15. if $a \neq 0$ then
    $T_7 \leftarrow T_6 + T_7$
    $T_1 \leftarrow T_7 \times T_1$ $= D$
    else $T_1 \leftarrow T_6 \times T_1$ $= D$
16. $T_3 \leftarrow T_6^2$ $= Z_2$
17. $T_6 \leftarrow T_2 \times T_6$ $= E$
18. $T_1 \leftarrow T_6 + T_1$
19. $T_2 \leftarrow T_2^2$
20. $T_1 \leftarrow T_2 + T_1$ $= X_2$
21. $T_4 \leftarrow T_3 \times T_4$
22. $T_5 \leftarrow T_3 \times T_5$
23. $T_4 \leftarrow T_1 + T_4$ $= F$
24. $T_5 \leftarrow T_1 + T_5$ $= G$

25. $T_4 \leftarrow T_6 \times T_4$
26. $T_5 \leftarrow T_3 \times T_5$
27. $T_2 \leftarrow T_4 + T_5$ $= Y_2$
28. $X_2 \leftarrow T_1$
29. $Y_2 \leftarrow T_2$
30. $Z_2 \leftarrow T_3$

This algorithm requires 9 general field multiplications, 4 field squarings and 7 temporary variables. If also $a = 0$, then only 6 temporary variables are required.

Capítulo 5

Um Algoritmo para Multiplicação Escalar em Curvas Elípticas sobre $\mathbb{F}_{2^m}$ sem Pré-computação

Neste capítulo é apresentado um algoritmo para multiplicação escalar em curvas elípticas definidas sobre $\mathbb{F}_{2^m}$. O algoritmo é uma versão otimizada de um método desenvolvido por Montgomery [69]. Nosso algoritmo é fácil de implementar tanto em hardware como em software, funciona em qualquer curva elíptica sobre $\mathbb{F}_{2^m}$, não requer pontos pré-calculados, e é em média mais rápido do que o método "soma-subtração" descrito no standard P1363 [42]. Além disso, o método requer menos registros que nos esquemas projetivos, e a quantidade de computação necessária para uma multiplicação escalar é fixa para todos os multiplicadores do mesmo tamanho em bits (isto pode ajudar a prevenir ataques baseados em medidas de tempo de execução [50]). Portanto, o método melhorado têm muitas características almejadas para implementar curvas elípticas em ambientes com recursos limitados.

Este capítulo é uma versão revisada do artigo apresentado no workshop: Cryptographic Hardware Embedded Systems, CHES'99, Worcester, USA. Publicado em Lecture Notes in Computer Science, **1717**, pp. 316-327, Springer-Verlag, 1999.

Fast Multiplication on Elliptic Curves over $\mathbb{F}_{2^m}$ without Precomputation*

Julio López and Ricardo Dahab
Institute of Computing
State University of Campinas,
Campinas, C.P. 6176, 13083-970, SP, Brazil
{juliohr,dahab}@dcc.unicamp.br

Abstract

This paper describes an algorithm for computing elliptic scalar multiplications on non-supersingular elliptic curves defined over $\mathbb{F}_{2^m}$. The algorithm is an optimized version of a method described in [2], which is based on Montgomery's method [69]. Our algorithm is easy to implement in both hardware and software, works for any elliptic curve over $\mathbb{F}_{2^m}$, requires no precomputed multiples of a point, and is faster on average than the addition-subtraction method described in draft standard IEEE P1363. In addition, the method requires less memory than projective schemes and the amount of computation needed for a scalar multiplication is fixed for all multipliers of the same binary length. Therefore, the improved method possesses many desirable features for implementing elliptic curves in restricted environments.

Key words. Elliptic curves over $\mathbb{F}_{2^m}$, Point multiplication.

5.1 Introduction

Elliptic curve cryptography first suggested by Koblitz [47] and Miller [68] is becoming increasingly common for implementing public-key protocols as the Diffie-Hellman key agreement. The security of these cryptosystems relies on the presumed intractability of the discrete logarithm problem on elliptic curves. Since there is no known sub-exponential type algorithm for elliptic curves over finite fields, the sizes of the fields, keys, and other parameters can be considered shorter than other public key cryptosystems such as RSA with the same level of security. This can be especially an advantage for applications where resources such as memory and/or computing power are limited.

*This paper is a revised version of the paper appearing in the Proceedings of CHES'99.

Elliptic curves over $\mathbb{F}_{2^m}$ are particularly attractive because the finite field operations can be implemented very efficiently in hardware and software. See for example [2] for a hardware implementation of $\mathbb{F}_{2^{155}}$, and [25] for a software implementation of $\mathbb{F}_{2^{191}}$.

Given an elliptic point P and a large integer k of about the size of the underlying field, the operation *elliptic scalar multiplication*, kP, is defined to be the elliptic point resulting from adding P to itself k times. This operation, analogous to exponentiation in multiplicative groups, is the most time consuming operation of the elliptic curve cryptosystems.

In this paper, the calculation of kP for a random integer k and a random point P is considered. An efficient scalar multiplication algorithm, which is an optimized version of an algorithm described in [2], is presented. The proposed algorithm is suitable for hardware and software implementation of random elliptic curves over $\mathbb{F}_{2^m}$.

5.2 Previous work

The basic method for computing kP is the addition-subtraction method described in draft standard IEEE P1363 [42]. This method is an improved version over the well known "add-and-double" (or binary) method, which requires no precomputations. For a random multiplier k, this algorithm performs on average $\frac{8}{3}\log_2 k$ field multiplications and $\frac{4}{3}\log_2 k$ field inversions in affine coordinates, and $8\frac{1}{3}\log_2 k$ field multiplications in projective coordinates.

Several proposed generalizations of the binary method (for exponentiation in a multiplicative group), such as the k-ary method, the signed window method, can be extended to compute elliptic scalar multiplications over a finite field [66]. These algorithms are based on the use of precomputation and methods for recoding the multiplier. In [36], several algorithms are analyzed under various conditions. However, most of the proposed optimizations may not be worthwhile when memory is at a premium.

Some special classes of elliptic curves defined over $\mathbb{F}_{2^m}$ allow efficient implementations. For anomalous curves, the fastest known algorithm to compute kP is given in [96]; for curves defined over small subfields, efficient algorithms are presented in [70].

In [37, 91, 59] some techniques are presented for accelerating methods such as k-ary and window based methods. These methods are suitable for software implementation of random elliptic curves over $\mathbb{F}_{2^m}$.

A different approach for computing kP was introduced by Montgomery [69]. This approach is based on the binary method and the observation that the x-coordinate of the sum of two points whose difference is known can be computed in terms of the x-coordinates of the involved points. This method uses the following variant of the binary method:

Figure 5.1: Algorithm 1: Binary Method

INPUT: An integer $k > 0$ and a point P.
OUTPUT: $Q = kP$.

1. Set $k \leftarrow (k_{l-1} \ldots k_1 k_0)_2$.
2. Set $P_1 \leftarrow P$, $P_2 \leftarrow 2P$.
3. **for** i **from** $l-2$ **downto** 0 **do**
   if $k_i = 1$ **then**
   Set $P_1 \leftarrow P_1 + P_2$, $P_2 \leftarrow 2P_2$.
   else
   Set $P_2 \leftarrow P_2 + P_1$, $P_1 \leftarrow 2P_1$.
4. **return**$(Q \leftarrow P_1)$.

Note that this method maintains the invariant relationship $P_2 - P_1 = P$, and performs an addition and a doubling in each iteration. In [65], Montgomery's method was applied for reducing the number of registers needed to add points in supersingular curves over $\mathbb{F}_{2^m}$. However, the authors observed that the benefits in storage provided by Montgomery's method is at a considerable expense of speed.

From the point of view of hardware implementation of elliptic curves over $\mathbb{F}_{2^m}$, few papers have discussed efficient methods for computing kP. In [2], Montgomery's method was adapted for non-supersingular elliptic curves over $\mathbb{F}_{2^m}$. However, the formulas given for implementing each iteration are not efficient in terms of field multiplications.

In this paper we will present an efficient implementation of Montgomery's method for computing kP on non-supersingular elliptic curves over $\mathbb{F}_{2^m}$.

The remainder of the paper is organized as follows. In Section 5.3 we present a short introduction to elliptic curves over $\mathbb{F}_{2^m}$. The proposed algorithm is described and analyzed in Section 5.4. Some running times of the proposed algorithm based on LiDIA are presented in Section 5.5. An implementation of the proposed algorithm is given in the appendix.

5.3 Elliptic curves over $\mathbb{F}_{2^m}$

Here we present a brief introduction to elliptic curves; more information on elliptic curves over finite fields of characteristic two can be found in [64, 42]. Let $\mathbb{F}_{2^m}$ be a finite field of characteristic two. A non-supersingular elliptic curve E over $\mathbb{F}_{2^m}$ is defined to be the set of solutions $(x, y) \in \mathbb{F}_{2^m} \times \mathbb{F}_{2^m}$ to the equation,

$$y^2 + xy = x^3 + ax^2 + b \ ,$$

where a and $b \in \mathbb{F}_{2^m}, b \neq 0$, together with the point at infinity denoted by $\mathcal{O}$.

It is well known that E forms a commutative finite group, with $\mathcal{O}$ as the group identity, under the addition operation known as the "tangent and chord method". Explicit rational formulas for the addition rule involve several arithmetic operations (addition, squaring, multiplication and inversion) in the underlying finite field. Formulas for adding two points in projective coordinates can be found in [64, 59]. In affine coordinates, the elliptic group operation is given by the following. Let $P = (x_1, y_1) \in E$; then $-P = (x_1, x_1 + y_1)$. For all $P \in E$, $\mathcal{O} + P = P + \mathcal{O} = P$. If $Q = (x_2, y_2) \in E$ and $Q \neq -P$, then $P + Q = (x_3, y_3)$, where

$$x_3 = \begin{cases} (\frac{y_1 + y_2}{x_1 + x_2})^2 + \frac{y_1 + y_2}{x_1 + x_2} + x_1 + x_2 + a \ , & P \neq Q \\ x_1^2 + \frac{b}{x_1^2} \ , & P = Q \end{cases} \tag{5.1}$$

and

$$y_3 = \begin{cases} (\frac{y_1 + y_2}{x_1 + x_2})(x_1 + x_3) + x_3 + y_1 \ , & P \neq Q \\ x_1^2 + (x_1 + \frac{y_1}{x_1})x_3 + x_3 \ , & P = Q. \end{cases} \tag{5.2}$$

Notice that the x-coordinate of $2P$ does not involve the y-coordinate of P. This observation will be used in the derivation of the improved method.

5.4 Improved method

This section describes the improved method for computing kP. We first develop an algorithm in affine coordinates which requires two field inversions in each iteration. Next a "projective" version is presented with more field multiplications, but with only one field inversion at the end of the computation.

5.4.1 Affine version

The extension of Montgomery's method [69] to elliptic curves over $\mathbb{F}_{2^m}$ requires formulas for implementing Step 3 of Algorithm 1. In what follows we give efficient formulas that use only the x-coordinates of P_1, P_2 and P for performing the arithmetic operations needed in Algorithm 1. At the end of the lth iteration of Algorithm 1, we obtain the x-coordinates of kP and $(k+1)P$. We also provide a simple formula for recovering the y-coordinate of kP.

The following lemma gives another formula for computing the x-coordinate of the addition of two different points.

Lemma 1 *Let $P_1 = (x_1, y_1)$, and $P_2 = (x_2, y_2)$ be elliptic points. Then the x-coordinate of $P_1 + P_2$, x_3, can be computed as follows.*

$$x_3 = \frac{x_1 y_2 + x_2 y_1 + x_1 x_2^2 + x_2 x_1^2}{(x_1 + x_2)^2} \quad . \tag{5.3}$$

Proof. Since P_1 and P_2 are elliptic points, it follows that $y_1^2 + y_2^2 + x_1 y_1 + x_2 y_2 + x_1^3 + x_2^3 = 0$. The result then follows easily from formula (5.1).

The following lemma shows how to compute the x-coordinate for the addition of two points whose difference is known.

Lemma 2 *Let $P = (x, y), P_1 = (x_1, y_1)$, and $P_2 = (x_2, y_2)$ be elliptic points. Assume that $P_2 = P_1 + P$. Then the x-coordinate of $P_1 + P_2$, x_3, can be computed in terms of the x-coordinates of P, P_1 and P_2 as follows.*

$$x_3 = \begin{cases} x + (\frac{x_1}{x_1 + x_2})^2 + \frac{x_1}{x_1 + x_2} \quad , & P_1 \neq P_2 \\ x_1^2 + \frac{b}{x_1^2} \quad , & P_1 = P_2. \end{cases} \tag{5.4}$$

Proof. The case $P = \mathcal{O}$ follows directly from (5.1). Applying formula (5.3), we obtain that the x-coordinate of $P_2 + P_1$ can be rewritten as

$$x_3 = \frac{x_1 y_2 + x_2 y_1 + x_1 x_2^2 + x_2 x_1^2}{(x_1 + x_2)^2} \quad . \tag{5.5}$$

Similarly, the x-coordinate of $P_2 - P_1$ satisfies

$$x = \frac{x_1 y_2 + x_2 (x_1 + y_1) + x_1 x_2^2 + x_2 x_1^2}{(x_1 + x_2)^2} \quad . \tag{5.6}$$

The result follows from adding (5.5) and (5.6).

The next lemma allows one to compute the y-coordinate of P_1 when P and the x-coordinates of P_1 and $P_1 + P$ are known.

Lemma 3 *Let $P = (x, y), P_1 = (x_1, y_1)$, and $P_2 = (x_2, y_2)$ be elliptic points. Assume that $P_2 = P_1 + P$ and $x \neq 0$. Then the y-coordinate of P_1 can be expressed in terms of P, and the x-coordinates of P_1 and P_2 as follows.*

$$y_1 = \ (x_1 + x)\{(x_1 + x)(x_2 + x) + x^2 + y\}/x + y \quad . \tag{5.7}$$

Proof. Since $P_2 = P_1 + P$, we obtain from (5.3) that y_1 satisfies the following equation:

$$x_2 (x_1 + x)^2 = x_1 y + x y_1 + x_1 x^2 + x x_1^2 \quad .$$

Therefore,

$$\begin{aligned} xy_1 &= x_2x_1^2 + x_2x^2 + x_1y + x_1x^2 + xx_1^2 \\ &= x_1\{x_1x_2 + x_1x + x^2 + y\} + x\{xx_2\} \\ &= x_1\{x_1x_2 + x_1x + x^2 + xx_2 + x^2 + y\} \\ &\quad + x\{x_1x_2 + x_1x + xx_2 + y\} + xy \\ &= (x_1 + x)\{(x_1 + x)(x_2 + x) + x^2 + y\} + xy. \end{aligned}$$

The following algorithm, based on Lemmas 2 and 3, implements Montgomery's method in affine coordinates.

Figure 5.2: Algorithm 2A: Montgomery Scalar Multiplication

INPUT: An integer $k \geq 0$ and a point $P = (x, y) \in E$.
OUTPUT: $Q = kP$.

1. if $k = 0$ or $x = 0$ then output$(0, 0)$ and stop.
2. Set $k \leftarrow (k_{l-1} \ldots k_1 k_0)_2$.
3. Set $x_1 \leftarrow x$, $x_2 \leftarrow x^2 + b/x^2$.
4. **for** i **from** $l - 2$ **downto** 0 **do**
   Set $t \leftarrow \frac{x_1}{x_1 + x_2}$.
   if $k_i = 1$ **then**
   Set $x_1 \leftarrow x + t^2 + t$, $x_2 \leftarrow x_2^2 + b/x_2^2$.
   else
   Set $x_2 \leftarrow x + t^2 + t$, $x_1 \leftarrow x_1^2 + b/x_1^2$.
5. Set $r_1 \leftarrow x_1 + x$, $r_2 \leftarrow x_2 + x$.
6. Set $y_1 \leftarrow r_1(r_1r_2 + x^2 + y)/x + y$
7. **return**$(Q \leftarrow (x_1, y_1))$.

Observe that Algorithm 2A, in each iteration of Step 4, performs two field inversions, one general field multiplication, one multiplication by the constant b, two squarings, and four additions; it follows that the total number of field operations to compute kP is given in the following lemma:

Lemma 4 *For computing kP, Algorithm 2A takes exactly the following number of field operations in $\mathbb{F}_{2^m}$:*

$$\begin{aligned} \#INV. &= 2\lfloor \log_2 k \rfloor + 1 \ , \quad & \#MULT. &= 2\lfloor \log_2 k \rfloor + 4 \ , \\ \#ADD. &= 4\lfloor \log_2 k \rfloor + 6 \ , \quad & \#SQR. &= 2\lfloor \log_2 k \rfloor + 2. \end{aligned}$$

Remark. A further improvement to Algorithm 2A is to use an optimized routine to multiply by the constant b. Another potential improvement is to compute in parallel x_1 and x_2 from Step 4, since these calculations are independent of each other.

5.4.2 Projective version

When field inversion in $\mathbb{F}_{2^m}$ is relatively expensive (e.g., inversion based on Fermat's theorem requires at least 7 multiplications in $\mathbb{F}_{2^m}$ if $m \geq 128$), then it may be of computational advantage to use fractional field arithmetic to perform elliptic curve calculations.

Let P, P_1 and P_2 be points on the curve E such that $P_2 = P_1 + P$. Let the x-coordinate of P_i be represented by X_i/Z_i, for $i \in \{1, 2\}$. From Lemma 2, when the x-coordinate of $2P_i$ is converted to projective coordinates it becomes

$$\begin{cases} x(2P_i) = X_i^4 + b \cdot Z_i^4 \ , \\ z(2P_i) = Z_i^2 \cdot X_i^2. \end{cases} \tag{5.8}$$

Similarly, the x-coordinate of $P_1 + P_2$ in projective coordinates can be computed as the fraction X_3/Z_3, where

$$\begin{cases} Z_3 = (X_1 \cdot Z_2 + X_2 \cdot Z_1)^2 \ , \\ X_3 = x \cdot Z_3 + (X_1 \cdot Z_2) \cdot (X_2 \cdot Z_1). \end{cases} \tag{5.9}$$

The addition formula requires three general field multiplications, one multiplication by x (i.e., the x-coordinate of P, which is fixed during the computation of kP), one squaring and two additions; doubling requires one general field multiplication, one multiplication by the constant b, four squarings, and one addition. A method based on these formulas is described in the next algorithm.

Figure 5.3: Algorithm 2P: Montgomery Scalar Multiplication

INPUT: An integer $k \geq 0$ and a point $P = (x, y) \in E$.
OUTPUT: $Q = kP$.

1. if $k = 0$ or $x = 0$ then output$(0, 0)$ and stop.
2. Set $k \leftarrow (k_{l-1} \ldots k_1 k_0)_2$.
3. Set $X_1 \leftarrow x$, $Z_1 \leftarrow 1$, $X_2 \leftarrow x^4 + b$, $Z_2 \leftarrow x^2$.
4. **for** i **from** $l - 2$ **downto** 0 **do**
   if $k_i = 1$ **then**
   Madd(X_1, Z_1, X_2, Z_2), Mdouble(X_2, Z_2).
   else
   Madd(X_2, Z_2, X_1, Z_1), Mdouble(X_1, Z_1).
5. **return**$(Q \leftarrow$ Mxy$(X_1, Z_1, X_2, Z_2))$.

An implementation of the procedures `Madd`, `Mdouble` and `Mxy` is given in the appendix.

Lemma 5 *Algorithm 2P performs exactly the following number of field operations in* $\mathbb{F}_{2^m}$*:*

$$\#INV. = 1 \ , \qquad \#MULT. = 6\lfloor \log_2 k \rfloor + 10 \ ,$$
$$\#ADD. = 3\lfloor \log_2 k \rfloor + 7 \ , \qquad \#SQR. = 5\lfloor \log_2 k \rfloor + 3.$$

Remark. Since the complexity of both versions of Algorithm 2 does not depend on the number of 1's (or 0's) in the binary representation of k, this may help to prevent timing attacks. On the other hand, the use of restricted multipliers (e.g., with small Hamming weight) does not speedup directly Algorithms 2A and 2P, and this is a disadvantage compared to methods such as the binary method. However, from a practical point of view, most protocols in cryptographic applications use random multipliers.

5.4.3 Complexity comparison

In the sequel, we assume that adding and squaring in $\mathbb{F}_{2^m}$ is relatively fast. Now we compare the complexities of the addition-subtraction method to the complexity of the proposed method. This is a fair comparison since both methods do not use precomputation. For a random multiplier k, the addition-subtraction method in projective coordinates, given in [42], performs $8.3\log_2 k$ field multiplications; it follows that we expect Algorithm 2P to be about 28% faster on average. However, if we use the formulas given in [59] for implementing the group operation in projective schemes, Algorithm 2P is about 14% faster than the addition-subtraction method. In the following table we summarize the complexities of these methods.

Table 5.1: Complexity Comparison of Algorithm 2P with other algorithms ($a = 0, 1$).

Method	Projective Coordinates
Binary [64]	$13\log_2 k$
Add-Sub [42]	$8.3\log_2 k$
Add-sub[59]	$7\log_2 k$
Algorithm 2P	$6\log_2 k$

Now we derive the cost of the addition-subtraction method (using affine coordinates) in terms of field multiplications. As mentioned in Section 5.2, this method performs on average $\frac{8}{3}\log_2 k$ field multiplications and $\frac{4}{3}\log_2 k$ field inversions. Thus, the total cost is $\frac{1}{3}(4r + 8)$ multiplications, where r is the cost-ratio of inversion to multiplication. This shows that for implementations of the finite field $\mathbb{F}_{2^m}$ where $r > 2.5$ (see for example [2, 25, 37]), Algorithm 2P gives a computational advantage over the addition-subtraction method.

5.5 Running times

In this section we present some running times we obtained in our software implementation of the proposed algorithm over the finite fields $\mathbb{F}_{2^m}$, where $m = 163, 191$ and 239. To represent the finite fields we used LiDIA [57], a C++ based library. This finite field implementation uses a polynomial basis representation and the irreducible modulus is chosen as sparse as possible. We used a Sun UltraSPARC 300MHz machine. For comparison, we list in Table 5.2 the timings for the basic arithmetic operations in $\mathbb{F}_{2^m}$.

Table 5.2: Average running times (in microseconds) for $\mathbb{F}_{2^m}$ using LiDIA.

Extension m	Add.	Sqr.	Mult.	Inv.
163	0.6	2.3	10.5	96.2
191	0.7	2.0	10.9	118.1
239	0.8	2.6	14.6	162.8

Notice that one field inverse costs more than 9 field multiplications; therefore, the use of LiDIA may illustrate the performance of the proposed algorithm in situations where a field inverse is relatively expensive compared to field multiplication.

Table 5.3: Average running times (in milliseconds) for computing mP.

Extension m	Binary[64]	Add-Sub.[42]	Algorithm 2P
163	27.5	19.1	13.5
191	33.1	22.4	16.0
239	52.3	35.1	25.6

In Table 5.3 we present average running times for computing a scalar multiplication using several methods. These values were obtained using the following test: we select 10 random elliptic curves ($a = 0$) over $\mathbb{F}_{2^m}$, then we multiply a random point P in each curve with 100 randomly chosen integers of size $< 2^m$. We implemented the binary method in projective coordinates (see [64]), the addition-subtraction method [42] and Algorithm 2P. From Table 5.3 we conclude that the proposed method on average is 27-29% faster than the addition-subtraction method and 51% faster than the binary method. These timings show that the theoretical improvement of Algorithm 2P, given in Table 5.1, is observed in a actual implementation.

5.6 Conclusion

In this paper, we have presented an efficient method for computing elliptic scalar multiplications, which is an optimized version of an algorithm presented in [2]. The method performs exactly $6\lfloor \log_2 k \rfloor + 10$ field multiplication for computing kP on elliptic curves selected at random, is easy to implement in both hardware and software, requires no precomputations, works for any implementation of $GF(2^n)$, is faster than the addition-subtraction method on average, and uses fewer registers than methods based on projective schemes. Therefore, the method appears useful for applications of elliptic curves in constraint environments such as mobile devices and smart cards.

5.7 Appendix

`Mdouble` (Doubling algorithm)

Input: the finite field $\mathbb{F}_{2^m}$; the field elements a and $c = b^{2^{m-1}} (c^2 = b)$ defining a curve E over $\mathbb{F}_{2^m}$; the x-coordinate X/Z for a point P.
Output: the x-coordinate X/Z for the point $2P$.

1. $T_1 \leftarrow c$
2. $X \leftarrow X^2$
3. $Z \leftarrow Z^2$
4. $T_1 \leftarrow Z \times T_1$
5. $Z \leftarrow Z \times X$
6. $T_1 \leftarrow T_1^2$
7. $X \leftarrow X^2$
8. $X \leftarrow X + T_1$

This algorithm requires one general field multiplication, one field multiplication by the constant c, four field squarings and one temporary variable.

`Madd` (Adding algorithm)

Input: the finite field $\mathbb{F}_{2^m}$; the field elements a and b defining a curve E over $\mathbb{F}_{2^m}$; the x-coordinate of the point P; the x-coordinates X_1/Z_1 and X_2/Z_2 for the points P_1 and P_2 on E.
Output: The x-coordinate X_1/Z_1 for the point $P_1 + P_2$.

1. $T_1 \leftarrow x$
2. $X_1 \leftarrow X_1 \times Z_2$
3. $Z_1 \leftarrow Z_1 \times X_2$
4. $T_2 \leftarrow X_1 \times Z_1$
5. $Z_1 \leftarrow Z_1 + X_1$
6. $Z_1 \leftarrow Z_1^2$
7. $X_1 \leftarrow Z_1 \times T_1$
8. $X_1 \leftarrow X_1 + T_2$

This algorithm requires three general field multiplications, one field multiplication by x, one field squaring and two temporary variables.

Mxy (Affine coordinates)

Input: the finite field $\mathbb{F}_{2^m}$; the affine coordinates of the point $P = (x, y)$; the x-coordinates X_1/Z_1 and X_2/Z_2 for the points P_1 and P_2.
Output: The affine coordinates $(x_k, y_k) = (X_2, Z_2)$ for the point P_1.

1. if $Z_1 = 0$ then output (0,0) and stop.
2. if $Z_2 = 0$ then output $(x, x + y)$ and stop.
3. $T_1 \leftarrow x$
4. $T_2 \leftarrow y$
5. $T_3 \leftarrow Z_1 \times Z_2$
6. $Z_1 \leftarrow Z_1 \times T_1$
7. $Z_1 \leftarrow Z_1 + X_1$
8. $Z_2 \leftarrow Z_2 \times T_1$
9. $X_1 \leftarrow Z_2 \times X_1$
10. $Z_2 \leftarrow Z_2 + X_2$
11. $Z_2 \leftarrow Z_2 \times Z_1$
12. $T_4 \leftarrow T_1^2$
13. $T_4 \leftarrow T_4 + T_2$
14. $T_4 \leftarrow T_4 \times T_3$
15. $T_4 \leftarrow T_4 + Z_2$
16. $T_3 \leftarrow T_3 \times T_1$
17. $T_3 \leftarrow inverse(T_3)$
18. $T_4 \leftarrow T_3 \times T_4$
19. $X_2 \leftarrow X_1 \times T_3$
20. $Z_2 \leftarrow X_2 + T_1$
21. $Z_2 \leftarrow Z_2 \times T_4$
22. $Z_2 \leftarrow Z_2 + T_2$

This algorithm requires one field inversion, ten general field multiplications, one field squaring and four temporary variables.

Capítulo 6

PGP em Dispositivos Limitados sem Fio

Neste capítulo descrevemos um experimento prático, em que a infra-estrutura criptográfica de chave pública de uma implementação do PGP (RSA e ElGamal) no pager bidirecional RIM foi substituída por algoritmos baseados em curvas elípticas sobre $\mathbb{F}_{2^m}$. Os resultados mostram que o desempenho dos criptossistemas de curvas elípticas (CCE) foi melhor do que os outros sistemas de chave pública para o mesmo nível de segurança teórico. A mesma biblioteca dos CCE foi implementada em outras plataformas (estações de trabalho, PCs e PalmPilot) e a comparação de desempenho com outras tecnologias de chave pública (RSA, DSA e ElGamal) também se mostraram favorável aos CCE.

O trabalho apresentado neste capítulo foi aceito para apresentação no 9th USENIX Security Symposium, a realizar-se em agosto de 2000 em Denver, Colorado, EUA.

PGP in Constrained Wireless Devices

Michael Brown* Donny Cheung* Darrel Hankerson†
Julio Lopez Hernandez‡ Michael Kirkup* Alfred Menezes*

Abstract

The market for Personal Digital Assistants (PDAs) is growing at a rapid pace. An increasing number of products, such as the PalmPilot, are adding wireless communications capabilities. PDA users are now able to send and receive email just as they would from their networked desktop machines. Because of the inherent insecurity of wireless environments, a system is needed for secure email communications. The requirements for the security system will likely be influenced by the constraints of the PDA, including limited memory, limited processing power, limited bandwidth, and a limited user interface.

This paper describes our experience with porting PGP to the Research in Motion (RIM) two-way pager, and incorporating elliptic curve cryptography into PGP's suite of public-key ciphers. Our main conclusion is that PGP is a viable solution for providing secure and interoperable email communications between constrained wireless devices and desktop machines.

6.1 Introduction

It is expected that there will be more than 530 million wireless subscribers by the year 2001, and over a billion by 2004 (see [102]). Efforts are underway, most notable among them the Wireless Application Protocol (WAP) [101], to define and standardize the emerging wireless Internet. Users will access wireless services including telephony, email and web browsing, using a variety of wireless devices such as mobile phones, PDAs (such as the PalmPilot), pagers, and laptop computers equipped with wireless modems. Many wireless devices are constrained by limited CPU, memory, battery life, and user interface (e.g., small screen size, or a lack of graphics capabilities). Wireless networks are constrained by

*Dept. of Combinatorics and Optimization, University of Waterloo, Canada.
Emails: {mk3brown,dccheung,mkirkup,ajmeneze}@cacr.math.uwaterloo.ca
†Dept. of Discrete and Statistical Sciences, Auburn University, USA.
Email: hankedr@mail.auburn.edu
‡Institute of Computing, State University of Campinas, Brazil, and Dept. of Computer Science, University of Valle, Colombia. Email: julioher@dcc.unicamp.br

low bandwidth, high latency, and unpredictable availability and stability. The purpose of this paper is to examine the viability of using PGP for providing secure and interoperable email communications between constrained wireless devices and desktop machines.

There are two popular standards for email security: S/MIME and PGP. S/MIME [82] provides confidentiality and authentication services to the MIME (Multipurpose Internet Mail Extensions) Internet email format standard. PGP (Pretty Good Privacy) [17, 31] is an email security standard that has been widely used since it was first introduced by Zimmermann in 1991 [106]. While it appears that S/MIME will emerge as the industry standard for commercial and organizational use, it also appears that PGP will remain the choice for personal email security for many users in the years to come.

The specific goals of this project were three-fold:

1. Port the basic PGP functionality to the RIM pager, and implement a workable key management system and a usable user interface that is appropriate for the RIM pager environment.
2. Achieve interoperability with existing PGP implementations for workstation and PalmPilot platforms.
3. Incorporate standards-based and commercial-strength elliptic curve cryptography into PGP's suite of public-key algorithms.

The remainder of this paper is organized as follows. §6.2 provides a brief history of PGP, and summarizes the security services offered by PGP. A description of the RIM two-way pager including hardware, software, user interface, development tools, and the paging environment, is provided in §6.3. A brief overview of the PalmPilot is presented in §6.4. Elliptic curve cryptography is introduced in §6.5, along with a description of our implementation. We provide timing comparisons of our ECC implementation with RSA and DL implementations on a variety of platforms. Our experience with porting PGP to the RIM pager is described in §6.6. Our implementation, including a description of the user interface and key management facilities, is presented in §6.7. In §6.8, we describe some possible directions for future work. Finally, §6.9 makes concluding remarks.

6.2 Pretty Good Privacy

6.2.1 History of PGP

The history of the Pretty Good Privacy (PGP) application is both interesting and convoluted, and encompasses issues in national security, personal privacy, patents, personalities, and politics; see, for example, [31]. A myriad of PGP releases emerged, in part due to US Government restrictions on exports.

The initial PGP application was released in 1991. According to [31] this was an "emergency release" prompted in part by a proposed anti-crime bill which would require eavesdropping ability for the US Government on all communications systems. An RSA-based public-key scheme was used, along with a symmetric-key algorithm developed by Zimmermann known as Bass-O-Matic.

Security concerns over Bass-O-Matic resulted in its replacement with IDEA in PGP 2. A commercial version of PGP was developed in 1993 with ViaCrypt (which had a license from Public Key Partners for RSA). Although RSA Data Security had released a reference implementation (RSAREF) of RSA that could be used for non-commercial purposes, there were interface and other difficulties preventing its use in PGP. In 1994, RSAREF 2.0 was released and included changes which MIT recognized would solve the interface problems. This eventually led to PGP 2.6, a version which could be used freely for non-commercial purposes, and which quickly leaked out of the US and developed into several international variants.

MIT PGP 2.6.2 increased the ceiling on the maximum size of an RSA modulus (from 1024 to 2048 bits, although ViaCrypt reports a patch correcting certain bugs with the longer moduli). The symmetric-key cipher is IDEA, a 64-bit block cipher with 128-bit keys; MD5 is used as the hash function, having digest length of 128 bits. A dependency tree for various US and international versions and variants may be found via [75].

Work on PGP 3 began in 1994, and was released by PGP Inc (formed by Zimmermann) as PGP 5 in May 1997.[1] New algorithms were present, including DSA [72] for signatures, an ElGamal public-key encryption scheme [27], the Secure Hash Algorithm (SHA-1) [73] with 160-bit message digests, and the symmetric-key ciphers CAST and Triple-DES (64-bit block ciphers with key sizes of 128 and 168 bits, respectively).

In August of 1997, the IETF was approached concerning a proposal to bring PGP to a standards body as a protocol. An OpenPGP working group was formed. Using PGP 5 as the base, a format specification was promoted to a Proposed Standard by the IESG in October 1998. The resulting IETF specification for OpenPGP [18] describes an unencumbered architecture, although compatibility with PGP 2.6 was encouraged. A reference implementation was written by Tom Zerucha and provided in a form suitable for scanning to circumvent US export restrictions [17].

In December 1999, Network Associates (which had acquired PGP Inc in December 1997) was granted a license by the US Government to export PGP. An international PGP project [74], which had been making PGP available world-wide by scanning paper copies that were (legally) exported from the US, announced that the lifting of the ban on strong

[1] Callas [17] notes that ViaCrypt had released several products with a version number of 4 although they were derivatives of PGP 2, and "it was easier to explain why three became five than to explain why three was the new program and four the old one."

encryption "marks the end of the PGPi scanning and OCR project, which started with PGP 5.0i in 1997."

Several OpenPGP-compliant applications have been developed. The reference implementation by Zerucha [17] relies on the OpenSSL library [81], and has been used by Zerucha as the basis for a PalmPilot implementation. The standard does not require the use of patented algorithms, and applications such as GNU Privacy Guard [34], released in 1999 as a replacement for PGP, can be both compliant and distributable without patent restrictions (since it does not include IDEA or RSA).

6.2.2 PGP security services

Key generation and storage. PGP allows a user to generate multiple key pairs (public-key/private-key pairs) for each public scheme supported. Different key pairs are generated for public-key encryption and for digital signatures. The key pairs, together with public keys of other users, are stored in a file called the key ring.

Information stored with a public key includes the user's name, email address, trust and validity indicators, key type, key size, expiry date, fingerprint (e.g., the 160-bit SHA-1 hash of the formatted public key), and a key ID (e.g., the low order 64 bits of the fingerprint).

Private keys are not stored directly in the key ring. Instead, the user selects a passphrase which is salted and hashed to derive a key k for a symmetric encryption scheme. The private key is encrypted using k, the passphrase is discarded, and the encrypted private key is stored. Subsequently, when the user wishes to access a private key (in order to decrypt a message or sign a message), the passphrase must be supplied so that the system can regenerate k and recover the private key.

Cryptographic services. PGP uses a combination of symmetric-key and public-key methods to provide authentication and confidentiality.

A message can be signed using the private key from a suitable public-key signature scheme. The recipient can verify the signature once an authentic copy of the signer's corresponding public key is obtained. The OpenPGP standard requires support for SHA-1 as a hash algorithm and the DSA, and encourages support for the MD5 hash function and RSA as a signature algorithm.

The use of symmetric-key algorithms (such as DES) alone for encryption is supported, although PGP is known more for the confidentiality provided by a combination of public-key and symmetric-key schemes. Since public-key encryption schemes tend to be computationally expensive, a session key is used with a symmetric-key scheme to encrypt a message; the session key is then encrypted using one or more public keys (typically, one for each recipient), and then the encrypted message along with each encrypted session

key is delivered. The standard requires support for an ElGamal public-key encryption scheme and Triple-DES; support for RSA, IDEA, and CAST is encouraged.

Signatures and encryption are often used together, to provide authentication and confidentiality. The message is first signed and then encrypted as described above.

Key management. The OpenPGP standard does not have a trust model. An OpenPGP-compliant PGP implementation could support a hierarchical X.509-based public key infrastructure (PKI). The trust model employed by existing PGP implementations is a combination of direct trust and the web of trust. In the former, user A obtains B's public key directly from B; fingerprints facilitate this process as only the fingerprints have to be authenticated. In the web of trust model, one or more users can attest to the validity of B's public key by signing it with their own signing key. If A possesses an authentic copy of the public key of one of these users, then A can verify that user's signature thereby obtaining a measure of assurance of the authenticity of B's public key. This chaining of trust can be carried out to any depth.

6.3 RIM's Pager

6.3.1 Overview

The RIM wireless handheld device is built around a custom Intel 386 processor running at 10 MHz. Current models carry 2 Mbytes of flash memory and 304 Kbytes of SRAM. There is a fairly conventional (if rather small) keyboard with a 6- or 8-line by 28 character (depending on font) graphical display. A thumb-operated trackwheel takes the place of a conventional mouse (see Figure 6.1).

A set of applications including a calendar and address book are commonly installed; even the occasional game of Tetris (falling blocks) is possible with efficient use of the graphical display. The main attraction is the wireless communication features, in particular, email solutions. The integrated wireless modem is essentially invisible, with no protruding antennae. The device is roughly 3.5in x 2.5in x 1in (89mm x 64mm x 25mm) and weighs 5 ounces (142 g) with the single AA battery (there is also an internal lithium cell). RIM claims that the battery will last roughly three weeks with typical usage patterns.

A docking cradle can be used to directly connect the device to a serial port. Software for Microsoft Windows is provided to download programs and other information, and to synchronize application data. An RS-232 compatible serial port on the pager runs at 19200 bps.

To be slightly more precise, RIM has two hardware devices, the 850 and the 950, which are combined with software to provide communications solutions. We used RIM's

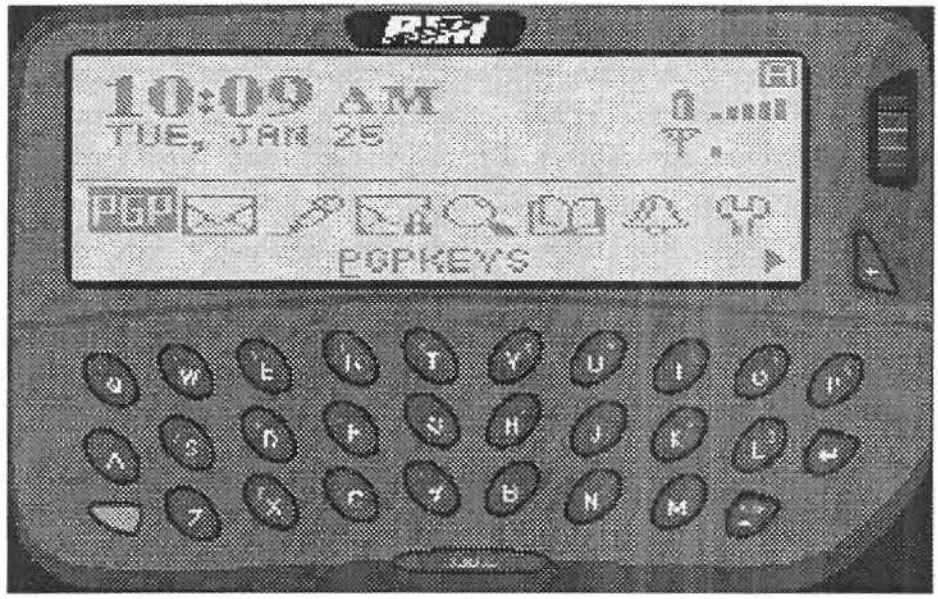

Figure 6.1: The RIM pager.

BlackBerry solution [13] which uses the same hardware as the RIM Inter@ctive Pager 950. The 950 is more of a 2-way pager, sold in Canada by Cantel and in the US by BellSouth Wireless Data. The BlackBerry is sold directly by RIM and includes features such as single mailbox integration and PIM synchronization to the device.

The RIM 850 looks very similar to the 950 device, but runs on a different wireless network (ARDIS for the 850 as opposed to Mobitex for the 950). The RIM 850 is resold through American Mobile Satellite Corporation (AMSC) in the US, and is part of the AMSC and SkyTel eLink solution.

6.3.2 Software development

The BlackBerry Software Developer's Kit (SDK) is designed to make use of the features in Microsoft's C++ compiler packages. The SDK is freely available from [84]. A handheld application is built as a Windows DLL, a process which allows use of development and debugging facilities available for Windows. However, only a small subset of the usual library calls may be used, along with calls to SDK-supplied routines. The resulting DLL is then stripped of extraneous information and ported into the handheld operating system.

For simplicity, the multitasking is cooperative. An application is expected to periodically yield control; in fact, failure to yield within 10 seconds can trigger a pager reset. As an example, public-key operations tend to be computationally expensive, and it was necessary to insert explicit task yields in the code developed for this paper.

The SDK includes a simulator which can be used to test applications on the handheld

operating system without having to download to the device (the images in this paper are snapshots of the simulator). A radio device (RAP modem) can be connected via serial port to the host machine so that applications running in the simulator can communicate with the Mobitex network. Alternately, a pager in the cradle can be used to exchange email with the simulator, provided that the pager is in coverage.

The simulator is essential for serious development, although testing on the pager can reveal bugs not found in the simulator. For example, we managed to link applications in such a way that they would work in the simulator but fail on the pager. At one point, we carelessly used some instructions introduced on the Intel 486, which would work in the simulator when running on a 486-or-better, but would fail on a 386.

6.3.3 File system

The pager relies on flash memory to store non-volatile data. Writing to flash is significantly more expensive than reading, primarily because flash is a write-once, bulk-erase device. Rewriting a single word of flash involves saving the contents of the 64K sector, erasing, and rewriting the entire sector. The longest step in this operation is erasing the sector, and takes approximately 5 seconds. A log-structured file system is employed in order to maintain acceptable performance. Periodically, the expensive process of committing the log updates is performed in order to free file system space.

The programming interface to the file system is generally through a relatively small number of high-level database-style calls. Handles are used to read and update databases and variable-length records, a simple but effective method to cooperate with the updating process of the log-structured file system. It is possible to use stream-style I/O operations of the type familiar to C programmers, which we occasionally found useful for testing code fragments developed on more traditional systems.

6.4 The PalmPilot

For comparison, our crypto routines were also run on the PalmPilot, a very popular PDA based on a 16 MHz Motorola 68000-type "Dragonball" processor.[2] Recent models carry 2–4 MB of memory in addition to ROM, although considerable expansion is possible. In 1999, wireless capabilities were introduced on the Palm VII. The communications model differs from the RIM device; in particular, the Palm does not qualify as a pager in the usual sense. There is an antenna which must be physically activated and then the device can request information. A NiCad battery charged from two AAA batteries common in the Palm series is used to power the radio.

[2] According to [77], "Even after two rounds of Microsoft's best Windows CE efforts, PalmPilot OS devices still represent 80% of all palmtop sales."

Ian Goldberg had adapted portions of Eric Young's well-known SSLeay library (now OpenSSL [81]) for use on the PalmPilot [35]. The resulting library was used by Zerucha in building a Palm version of his reference OpenPGP, and by Daswani and Boneh [23] in their paper on electronic commerce.

We used Palm development tools based on the GNU C compiler (gcc-2.7.2.2). Timings were done on a Palm V running PalmOS 3.0. There are code segment and stack restrictions which must be considered in the design of a larger application, and our code had to be divided into several libraries in order to accomodate the Palm.

6.5 Elliptic Curve Cryptography

6.5.1 Introduction

Elliptic curve cryptography (ECC) was proposed independently in 1985 by Neal Koblitz [47] and Victor Miller [68]. For an introduction to ECC, the reader is referred to Chapter 6 of Koblitz's book [49], or the recent book by Blake, Seroussi and Smart [14].

The primary reason for the attractiveness of ECC over RSA and discrete log (DL[3]) public-key systems is that the best algorithm known for solving the underlying hard mathematical problem in ECC (the elliptic curve discrete logarithm problem, ECDLP) takes fully exponential time. On the other hand, the best algorithms known for solving the underlying hard mathematical problems in RSA and DL systems (the integer factorization problem, and the discrete logarithm problem) take subexponential time. This means that the algorithms for solving the ECDLP become infeasible much more rapidly as the problem size increases than those algorithms for the integer factorization and discrete logarithm problems. For this reason, ECC offers security equivalent to that of RSA and DL systems, while using significantly smaller key sizes.

Table 6.1 lists ECC key lengths and very rough estimates of DL and RSA key lengths that provide the same security (against known attacks) as some common symmetric encryption schemes. The ECC key lengths are twice the key lengths of their symmetric cipher counterparts since the best general algorithm known for the ECDLP takes $(\sqrt{\pi 2^k})/2$ steps for k-bit ECC keys, while exhaustive key search on a symmetric cipher with l-bit keys takes 2^l steps. The estimates for DL security were obtained from [3]. The estimates for RSA security are the same as those for DL security because the best algorithms known for the integer factorization and discrete logarithm problems have the same expected running times. These estimates are roughly the same as the estimates provided by Lenstra and Verheul in their very thorough paper [56].

[3]Examples of DL systems are the ElGamal public-key encryption scheme and the DSA signature scheme which is specified in the Digital Signature Standard. PGP documentation refer to these two schemes as Diffie-Hellman/DSS or DH/DSS.

Symmetric cipher key lengths	Example algorithm	ECC key lengths for equivalent security	DL/RSA key lengths for equivalent security
80	SKIPJACK	160	1024
168	Triple-DES	224	2048
128	128-bit AES	256	3072
192	192-bit AES	384	7680
256	256-bit AES	512	15360

Table 6.1: ECC, DL, and RSA key length comparisons.

The advantages that may be gained from smaller ECC parameters include speed (faster computation) and smaller keys and certificates. These advantages are especially important in environments where processing power, storage space, bandwidth, or power consumption are at a premium such as smart cards, pagers, cellular phones, and PDAs.

6.5.2 Selecting ECC parameters

Notation. In the following, $\mathbb{F}_q$ denotes a finite field of order q, and E denotes an elliptic curve defined over $\mathbb{F}_q$. $\#E(\mathbb{F}_q)$ denotes the number of points on the elliptic curve E. The point at infinity is denoted by $\mathcal{O}$. There is a group law for adding any two elliptic curve points. If k is an integer and $P \in E(\mathbb{F}_q)$ is a point, then kP is the point obtained by adding together k copies of P; this process is called scalar multiplication.

Domain parameters. ECC domain parameters consist of the following:

- q — the field size.
- FR — method used for representing field elements.
- a, b — elements of $\mathbb{F}_q$ which determine the equation of an elliptic curve E.
- G — the base point of prime order.
- n — the order of G.
- h — the cofactor: $h = \#E(\mathbb{F}_q)/n$.

The primary security parameter (see §6.5.4) is n. The ECC key length is thus defined to be the bitlength of n. Typical choices for q are an odd prime (in which case $\mathbb{F}_q$ is called a *prime field*) or a power of 2 (in which case $\mathbb{F}_q$ is called a *binary field*).

Curves selected. For this project, we chose binary fields $\mathbb{F}_{2^m}$, for $m = 163$, 233 and 283. Suitably chosen elliptic curves over these fields provide at least as much security as symmetric-key ciphers with key lengths 80, 112 and 128 bits respectively (see Table 6.1). A polynomial basis representation was used to represent field elements. Such a

m	163
$f(x)$	$x^{163} + x^7 + x^6 + x^3 + 1$
E	$Y^2 + XY = X^3 + X^2 + 1$
n	4000000000000000000020108A2E0CC0D99F8A5EF
h	2
m	233
$f(x)$	$x^{233} + x^{74} + 1$
E	$Y^2 + XY = X^3 + 1$
n	8000000000000000000000000000069D5BB915BCD46EFB1AD5F173ABDF
h	4
m	283
$f(x)$	$x^{283} + x^{12} + x^7 + x^5 + 1$
E	$Y^2 + XY = X^3 + 1$
n	1FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFE9AE2ED07577265DF F7F94451E061E163C61
h	4

Table 6.2: Koblitz curves selected.

representation is defined by a reduction polynomial $f(x)$, which is an irreducible binary polynomial of degree m. For each field $\mathbb{F}_{2^m}$, we chose a random curve over $\mathbb{F}_{2^m}$ and a Koblitz curve [48] over $\mathbb{F}_{2^m}$ from the list of elliptic curves recommended by NIST for US federal government use [72]. The salient features of the Koblitz curves are provided in Table 6.2. Koblitz curves have special structure that enable faster elliptic curve arithmetic in some environments (see [96, 97]). The number of points on each of the chosen curves is almost prime; that is, $\#E(\mathbb{F}_{2^m}) = nh$, where n is prime and $h = 2$ or $h = 4$. Since $\#E(\mathbb{F}_{2^m}) \approx 2^m$, it follows that the ECC key length is approximately equal to m. Security implications of these choices are discussed in §6.5.4.

6.5.3 ECC protocols

Key generation. An entity A's public and private key pair is associated with a particular set of EC domain parameters $(q, \mathrm{FR}, a, b, G, n, h)$. This association can be assured cryptographically (e.g., with certificates) or by context (e.g., all entities use the same domain parameters).

To generate a key pair, entity A does the following:

1. Select a random integer d from $[1, n-1]$.
2. Compute $Q = dG$.

3. A's public key is Q; A's private key is d.

Public key validation. This process ensures that a public key has the requisite arithmetic properties. A public key $Q = (x_Q, y_Q)$ associated with domain parameters $(q, \text{FR}, a, b, G, n, h)$ is validated using the following procedure:

1. Check that $Q \neq \mathcal{O}$.
2. Check that x_Q and y_Q are properly represented elements of $\mathbb{F}_q$.
3. Check that Q lies on the elliptic curve defined by a and b.
4. Check that $nQ = \mathcal{O}$.

The computationally expensive operation in public key validation is the scalar multiplication in step 4. This step can sometimes be incorporated into the protocol that uses Q – this is done in the ECAES below. Public key validation with step 4 omitted is called *partial* public key validation.

Elliptic curve authenticated encryption scheme (ECAES). The ECAES, proposed by Abdalla, Bellare and Rogaway [1], is a variant of the ElGamal public-key encryption scheme [27]. It is efficient and provides security against adaptive chosen-ciphertext attacks.

We suppose that receiver B has domain parameters $D = (q, \text{FR}, a, b, G, n, h)$ and public key Q. We also suppose that A has authentic copies of D and Q. In the following, MAC is a message authentication code (MAC) algorithm such as HMAC [55], ENC is a symmetric encryption scheme such as Triple-DES. KDF denotes a key derivation function which derives cryptographic keys from a shared secret point.

To encrypt a message m for B, A does:

1. Select a random integer r from $[1, n-1]$.
2. Compute $R = rG$.
3. Compute $K = hrQ$. Check that $K \neq \mathcal{O}$.
4. Compute $k_1 \,||\, k_2 = \text{KDF}(K)$.
5. Compute $c = \text{ENC}_{k_1}(m)$.
6. Compute $t = \text{MAC}_{k_2}(c)$.
7. Send (R, c, t) to B.

To decrypt ciphertext (R, c, t), B does:

1. Perform a partial key validation on R.
2. Compute $K = hdR$. Check that $K \neq \mathcal{O}$.
3. Compute $k_1 \,||\, k_2 = \text{KDF}(K)$.
4. Verify that $t = \text{MAC}_{k_2}(c)$.

5. Compute $m = \text{ENC}_{k_1}^{-1}(c)$.

The computationally expensive operations in encryption and decryption are the scalar multiplications in steps 2-3 and step 2, respectively.

Elliptic curve digital signature algorithm (ECDSA). The ECDSA is the elliptic curve analogue of the DSA [72]. SHA-1 is the 160-bit hash function [73].

We suppose that signer A has domain parameters $D = (q, \text{FR}, a, b, G, n, h)$ and public key Q. We also suppose that B has authentic copies of D and Q.

To sign a message m, A does the following:

1. Select a random integer k from $[1, n-1]$.
2. Compute $kG = (x_1, y_1)$ and $r = x_1 \bmod n$.
   If $r = 0$ then go to step 1.
3. Compute $k^{-1} \bmod n$.
4. Compute $e = \text{SHA-1}(m)$.
5. Compute $s = k^{-1}\{e + dr\} \bmod n$.
   If $s = 0$ then go to step 1.
6. A's signature for the message m is (r, s).

To verify A's signature (r, s) on m, B should do the following:

1. Verify that r and s are integers in $[1, n-1]$.
2. Compute $e = \text{SHA-1}(m)$.
3. Compute $w = s^{-1} \bmod n$.
4. Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
5. Compute $u_1 G + u_2 Q = (x_1, y_1)$.
6. Compute $v = x_1 \bmod n$.
7. Accept the signature if and only if $v = r$.

The computationally expensive operations in signature generation and signature verification are the scalar multiplications in step 2 and step 5, respectively.

6.5.4 Security issues

Hardness of the ECDLP. It can easily be verified that the elliptic curves $E(\mathbb{F}_q)$ chosen resist all known attacks on the ECDLP. Specifically:

1. The number of points, $\#E(\mathbb{F}_q)$, is divisible by a prime n that is sufficiently large to resist the parallelized Pollard rho attack [80] against general curves, and its improvements [30, 105] which apply to Koblitz curves.

2. n does not divide $q^k - 1$ for all $1 \leq k \leq 30$, confirming resistance to the Weil pairing attack [67] and the Tate pairing attack [28].
3. $\#E(\mathbb{F}_q) \neq q$, confirming resistance to the Semaev attack [93].
4. All binary fields $\mathbb{F}_{2^m}$ chosen have the property that m is prime, thereby circumventing recent attacks [29, 32] on the ECDLP for elliptic curves over binary fields $\mathbb{F}_{2^m}$ where m is composite.

Security of ECAES. The ECAES modifies the ElGamal encryption scheme by using the one-time Diffie-Hellman shared secret, $hrdG$, to derive secret keys k_1 and k_2 The first key k_1 is used to encrypt the message using a symmetric cipher, while the second key k_2 is used to authenticate the resulting ciphertext. The latter provides resistance to chosen-ciphertext attacks. Some formal justification of ECAES security is provided in [1], where it is proven to be semantically secure against adaptive chosen-ciphertext attack on the assumption that the underlying symmetric encryption and MAC schemes are secure, and assuming the hardness of certain variants of the elliptic curve Diffie-Hellman problem.

In order to correctly balance the security of the ECAES cryptographic components, one should ideally employ a $\frac{k}{2}$-bit block cipher and a k-bit hash function for HMAC when using a k-bit elliptic curve (see Table 6.1). Our implementation used the 112-bit block cipher Triple-DES in CBC-mode and the 160-bit hash function SHA-1 for all 3 choices of ECC key lengths (163, 233 and 283). A future version of our implementation should allow for a variable output-length hash function (e.g., the forthcoming SHA-2) and a variable-length block cipher (e.g., the AES).

Security of ECDSA. ECDSA is the straightforward elliptic curve analogue of the DSA, which has been extensively scrutinized since it was proposed in 1991. For a summary of the security properties of the ECDSA, see [44].

Our implementation used the 160-bit hash function SHA-1 for all 3 choices of ECC key lengths (163, 233 and 283). As with the ECAES, a future version of our ECDSA implementation should allow for a variable output-length hash function.

6.5.5 Timings

This section presents timings for the ECC operations on a Pentium II 400 MHz machine, a PalmPilot and the RIM pager, and compares them with timings for RSA and DL operations.

ECC timings. Our ECC code was written entirely in C on a Sun Sparcstation and, in order to ensure portability, no assembler was used. We encountered no problems in porting the code to the Pentium II, RIM pager, and PalmPilot platforms, although some changes were required in order to cooperate with the 16-bit options used in the Palm version of

the "big number" library of OpenSSL. No effort was made to optimize the ECC code for these particular platforms; it is very likely that significant performance improvements could be obtained by optimizing the ECC (and DL and RSA) code for these platforms. Further details of our ECC implementations are reported in [40].

For other ECC implementation reports, see [89] for a C implementation of elliptic curve arithmetic over $\mathbb{F}_{2^{155}}$, [25] for a C/C++ of elliptic curve arithmetic over $\mathbb{F}_{2^{191}}$ and over a 191-bit prime field, and [41] for an assembly language implementation of elliptic curve arithmetic over a 160-bit prime field on a 10 MHz 16-bit microcomputer.

Tables 6.3, 6.4 and 6.5 present timings of our implementation for ECC operations using the Koblitz curves and random curves over $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$ and $\mathbb{F}_{2^{283}}$.

	Koblitz curve over $\mathbb{F}_{2^{163}}$			Random curve over $\mathbb{F}_{2^{163}}$		
	RIM pager	PalmPilot	P II	RIM pager	PalmPilot	P II
Key generation	751	1,334	1.47	1,085	1,891	2.12
ECAES encrypt	1,759	2,928	4.37	3,132	5,458	6.67
ECAES decrypt	1,065	1,610	2.85	2,114	3,564	4.69
ECDSA signing	1,011	1,793	2.11	1,335	2,230	2.64
ECDSA verifying	1,826	3,263	4.09	3,243	5,370	6.46

Table 6.3: Timings (in milliseconds) for ECC operations over $\mathbb{F}_{2^{163}}$ on various platforms.

	Koblitz curve over $\mathbb{F}_{2^{233}}$			Random curve over $\mathbb{F}_{2^{233}}$		
	RIM pager	PalmPilot	P II	RIM pager	PalmPilot	P II
Key generation	1,552	2,573	3.11	2,478	3,948	4.58
ECAES encrypt	3,475	5,563	7.83	6,914	11,373	13.99
ECAES decrypt	2,000	2,969	4.85	4,593	7,551	9.55
ECDSA signing	1,910	3,080	4.03	3,066	4,407	5.52
ECDSA verifying	3,701	5,878	7.87	7,321	11,964	14.08

Table 6.4: Timings (in milliseconds) for ECC operations over $\mathbb{F}_{2^{233}}$ on various platforms.

RSA timings. The RSA code, written entirely in C, was taken from the OpenSSL library [81]. Tables 6.6 and 6.7 present timings for 512, 768, 1024, and 2048-bit RSA operations.

DL timings. The DSA and ElGamal code, also written entirely in C, was obtained from the OpenSSL and OpenPGP libraries. For ElGamal, the prime p was chosen to be a safe prime; that is $p = 2q + 1$ where q is also prime. Table 6.8 presents timings for 512,

	Koblitz curve over $\mathbb{F}_{2^{283}}$			Random curve over $\mathbb{F}_{2^{283}}$		
	RIM pager	PalmPilot	P II	RIM pager	PalmPilot	P II
Key generation	2,369	4,062	4.50	3,857	6,245	6.88
ECAES encrypt	5,227	8,579	11.02	11,264	18,273	20.86
ECAES decrypt	2,932	4,495	6.78	7,498	12,046	13.88
ECDSA signing	2,760	4,716	5.64	4,264	6,816	8.08
ECDSA verifying	5,485	9,059	11.46	11,587	18,753	21.15

Table 6.5: Timings (in milliseconds) for ECC operations over $\mathbb{F}_{2^{283}}$ on various platforms.

	512-bit modulus			768-bit modulus		
	Pager	Pilot	P II	Pager	Pilot	P II
RSA key generation	73,673	189,461	346.77	287,830	496,356	953.01
RSA encrypt ($e = 3$)	213	317	1.13	388	587	1.87
RSA encrypt ($e = 17$)	262	410	1.28	451	753	2.17
RSA encrypt ($e = 2^{16} + 1$)	428	743	1.90	793	1,347	3.32
RSA decrypt	2,475	5,858	11.05	7,905	16,262	28.05
RSA signing	2,466	5,751	10.78	7,889	16,047	27.72
RSA verifying ($e = 3$)	99	200	0.40	214	413	0.78
RSA verifying ($e = 17$)	147	293	0.56	273	577	1.07
RSA verifying ($e = 2^{16} + 1$)	314	623	1.17	616	1,221	2.24

Table 6.6: Timings (in milliseconds) for 512-bit and 768-bit RSA operations on various platforms.

768 and 1024-bit DSA and ElGamal operations. For encryption, the per-message secret key is not of full length (i.e., the bitlength of p), but of bitlength 200 + (bitlength of p)/32; this explains why ElGamal encryption is faster than ElGamal decryption. The ElGamal operations could be sped up significantly if DSA-like parameters were used (i.e., $p = kq + 1$, where q is a 160-bit prime).

Comparison. The performance of all three families of public-key systems (ECC, RSA and DL) are sufficiently fast for PGP implementations on a Pentium machine—it hardly matters whether a user has to wait 10 ms or 100 ms to sign and encrypt a message.

On the pager, RSA public-key operations (encryption and signature verification) are faster than ECC public-key operations, especially when the public exponent is $e = 3$. For example, verifying a 1024-bit RSA signature takes about 300 ms, while verifying a 163-bit ECC signature (using a Koblitz curve) takes about 1,800 ms. On the other hand, RSA private-key operations (decryption and signature generation) are slower than ECC private-

	1024-bit modulus			2048-bit modulus		
	Pager	Pilot	P II	Pager	Pilot	P II
RSA key generation	580,405	1,705,442	2,740.87	—	—	26,442.04
RSA encrypt ($e = 3$)	533	1,023	2.70	1,586	3,431	7.26
RSA encrypt ($e = 17$)	683	1,349	3.23	2,075	4,551	9.09
RSA encrypt ($e = 2^{16} + 1$)	1,241	2,670	5.34	4,142	8,996	16.57
RSA decrypt	15,901	36,284	67.32	112,091	292,041	440.78
RSA signing	15,889	36,130	66.56	111,956	288,236	440.69
RSA verifying ($e = 3$)	301	729	1.23	1,087	2,392	4.20
RSA verifying ($e = 17$)	445	1,058	1.76	1,585	3,510	6.10
RSA verifying ($e = 2^{16} + 1$)	1,008	2,374	3.86	3,608	7,973	13.45

Table 6.7: Timings (in milliseconds) for 1024-bit and 2048-bit RSA operations on various platforms.

	512-bit modulus			768-bit modulus			1024-bit modulus		
	Pager	Pilot	PII	Pager	Pilot	PII	Pager	Pilot	PII
ElGamal k. g.	—	—	51,704	—	—	219,820	—	—	1,200,157
ElGamal enc.	7,341	17,338	19.13	16,078	34,904	35.91	26,588	73,978	67.78
ElGamal dec.	8,704	19,060	22.55	26,958	56,708	59.53	57,248	148,059	144.73
DSA key gen.	—	—	3,431	—	—	14,735	—	—	54,674
DSA signing	2,955	6,329	7.53	6,031	11,875	15.55	9,529	25,525	24.28
DSA verifying	5,531	12,389	14.31	11,594	24,277	26.13	18,566	52,286	47.23

Table 6.8: Timings (in milliseconds) for DL operations on various platforms.

key operations. For example, signing with a 1024-bit RSA key takes about 16,000 ms, while signing with a 163-bit ECC key takes about 1,000 ms. ECC has a clear advantage over RSA for PGP operations that require both private key and public key computations. Signing-and-encrypting together takes 16,400 ms with 1024-bit RSA (using $e = 3$), and 2800 ms with 163-bit ECC (using a Koblitz curve). Verifying-and-decrypting together takes 16,200 ms with 1024-bit RSA, and 2,900 ms with 163-bit ECC.

Similar conclusions are drawn when comparing RSA and ECC performance on the PalmPilot.

Private key operations with 2048-bit RSA are too slow for the pager and the PalmPilot, while 233-bit ECC and 283-bit ECC operations are tolerable for PGP applications on the pager.

Since domain parameters are used in our ECC implementation, ECC key generation only involves a single scalar multiplication and thus is very fast on the pager. RSA,

ElGamal and DSA key generation on the pager is prohibitively slow. However, ElGamal and DSA key generation would be feasible on the pager if precomputed domain parameters (primes p and q, and generator g) were used.

6.5.6 Interoperability

The elliptic curves and protocols were selected to conform with the prevailing ECC standards and draft standards.

The Koblitz and random curves over $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$ and $\mathbb{F}_{2^{283}}$ are from the list of NIST recommended curves [72]. The representations, for both field elements and for elliptic curve points, are compliant with the ANSI X9.62 [5], ANSI X9.63 [6], IEEE P1363 [42] and FIPS 186-2 [72] standards. In addition, the Koblitz curve over $\mathbb{F}_{2^{163}}$ is explicitly listed in the WAP wTLS specification [103].

Our ECDSA implementation conforms to the security and interoperability requirements of ANSI X9.62, IEEE P1363, and FIPS 186-2. Our ECAES implementation conforms to the security and interoperability requirements of ANSI X9.63. The cryptographic components HMAC and Triple-DES (in CBC mode) of ECAES are compliant, respectively, with RFC 2104 [55] and ANSI X9.52 [4].

6.6 Porting PGP to the Pager

There are now a number of cryptographic libraries and PGP applications which have received extensive development and for which source code is available; see, for example, cryptlib by Peter Gutmann [38] and Crypto++ by Wei Dai [22]. Our plan was to adapt existing code, adding public-key schemes based on elliptic curves. For comparisons and development, it was essential that the code run on several platforms in addition to the RIM device.

Our initial work was with GNU Privacy Guard (GnuPG) [34], an OpenPGP-compliant freely distributable replacement for PGP, which was nearing a post-beta release in 1999. Initial tests on the pager with several fragments adapted from GnuPG sources were promising, and the code appeared to be ideal for adding the elliptic curve routines and testing on Unix-based and other systems. However, it appeared that untangling code dependencies for our use on the pager would be unpleasant. (Perhaps a better understanding of GnuPG internals and design decisions would have changed our opinion.)

Jonathan Callas suggested that we look again at the OpenPGP reference implementation [17], which we had put aside after initial testing revealed a few portability and alignment problems in the code. The reference implementation relied on the OpenSSL library [81].

The OpenPGP reference implementation is surprisingly complete for the amount of code, although it is admittedly a little rough on the edges.[4] The code was developed on a Linux/x86 system, and modifications were required for alignment errors which prevented the program from running on systems such as Solaris/SPARC. In addition, some portability changes were required, including code involving the "long long" data type. For the RIM pager, the separation of the PGP code from the well-tested OpenSSL library, along with the small size of the OpenPGP sources, were definite advantages. Finally, it should be noted that the OpenSSL libraries build easily on Unix and Microsoft Windows systems, and are designed so that adding routines such as the elliptic curve code is straightforward.

Although applications for the pager are built as Windows DLLs, the pager is not a Windows-based system. There are significant restrictions on the calls that can be used, extending to those involving memory allocation, time and character handling, and the file system. There is no floating-point processor on the pager. In order to adapt code developed on more traditional systems, we wrote a library of compatibility functions to use with the pager. Some functions were trivial (such as those involving memory allocation, since the SDK included equivalent calls); others, such as the stream I/O calls, were written to speed testing and porting and cannot be recommended as particularly robust or elegant.

We used portions of OpenSSL 0.9.4, along with the library in the OpenPGP reference implementation. Relatively few changes to OpenSSL were required, and could be restricted to header files in many cases. The elliptic curve routines were integrated, including additions to the scripts used to build OpenSSL. For some platforms, OpenSSL can be built using assembly-language versions of certain key routines to improve execution speed. Some of these files for the Intel x86 include instructions (such as bswap) which were introduced for the 486, and cannot be used on the pager.

The OpenPGP sources were modified to correct the alignment bugs and portability problems mentioned above, and necessary changes were made for the elliptic curve schemes (public-key algorithms 18 and 19 in the OpenPGP specification [18]). The compatibility library, along with a few stream-to-memory conversion functions allowed fairly direct use of the OpenPGP sources on the pager.

The only code tested exclusively in the pager environment involved the user interface (see §6.7.1). The SDK provides a fairly powerful and high-level API for working with the display and user input. The difficulties we encountered were mostly due to the lack of support in the API for direct manipulation of messages desired in a PGP framework. In part, this reflects a deliberate design decision by BlackBerry to develop a robust and intuitive communication solution which provides some protection against misbehaving

[4] Zerucha writes that he wasn't "careful about wiping memory and preventing memory leaks and other things to make the code robust" [17].

applications.[5]

The pager DLLs for the interface and PGP library were over 400 KB in combined size. This includes all of the OpenPGP required algorithms and recommended algorithms such as IDEA and RSA, along with the new schemes based on elliptic curves. For a rough comparison, the code size for the main executable from the OpenPGP reference implementation (with the addition of the elliptic curve routines) is 300–400 KB, depending on platform.

6.7 Implementation

6.7.1 User interface

PGP in any form has not been an easy application for novices to manage properly, in part due to the sophistication required, but also because of poor interface design [104]. The goals for our user interface design were rather modest: that a user who is familiar with using PGP on a workstation, and is comfortable operating the RIM device, should, without having to refer to a manual or help pages, be easily able to figure out how to use PGP on the pager and avoid dangerous errors (such as those described in [104]). As mentioned in §6.3.1, the graphics capabilities and screen size of the RIM device are very limited. This forced us to keep our PGP implementation simple and only offer the user the essential features.

A glimpse of our user interface is provided in Figures 6.1–6.5. Clicking on the PGP icon (see Figure 6.1) displays the list of users whose keys are in the public key ring (see Figure 6.2). Selecting a user name displays the menu shown in Figure 6.3, which allows

Figure 6.2: Listing of PGP keys.

the user to view the key's attributes, compose a new key, delete a key, or send a key.

[5] During our work on this project, BlackBerry modified the API to provide some of the access needed to smoothly integrate PGP into their mail application.

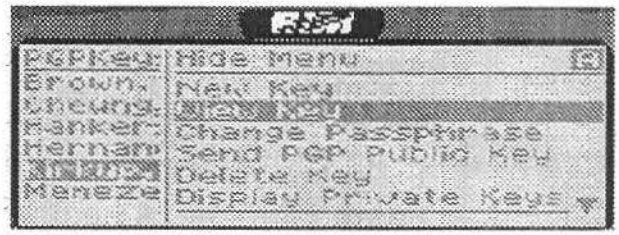

Figure 6.3: The main menu.

6.7.2 Key generation and storage

The main PGP menu (Figure 6.3) has an option "New Key" for creating a key pair. Users can enter their name, email address, pager PIN, and select a key type and key length (see Figure 6.4). The key types and key sizes presently available are ECC (random

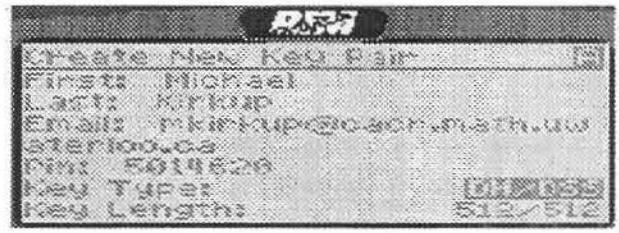

Figure 6.4: Screen for creating a new key pair.

curve or Koblitz curve; over $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$ or $\mathbb{F}_{2^{283}}$), DH/DSS (512/512, 768/768, 1024/1024, 1536/1024 or 2048/1024 bits), and RSA (512, 768, 1024, 1536 or 2048 bits). The DH/DSS and RSA key sizes are the ones available in many existing PGP implementations. For the DSA, the maximum bitsize of the prime p is 1024 bits in conformance with the DSS [72]. For ECC, separate key pairs are generated for public-key encryption and digital signatures.

Public keys and private keys are stored in separate key rings. Public key attributes (see Figure 6.5) can be viewed using the "View Key" function available on the main menu. As required by OpenPGP, private keys are encrypted under a user-selected passphrase,

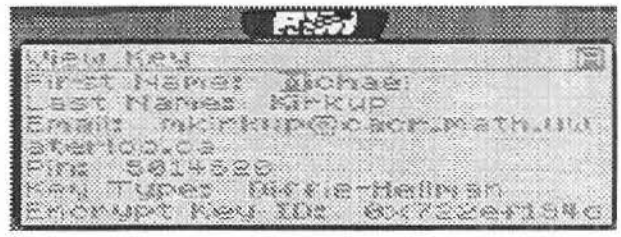

Figure 6.5: Screen for viewing a (portion of the) public key's attributes.

and the encrypted private key is stored. The passphrase has to be entered whenever a private key is required to sign or decrypt a message.

6.7.3 Cryptographic services

The three basic PGP services are available: sign only, encrypt only, or sign-and-encrypt. Users can decide to sign an email, or to encrypt an email, after composing the message. The user is prompted for the passphrase to unlock the private signing key, and to select the public encryption key of the intended recipient. In addition to the times given in Tables 6.3–6.8 for the main operations, there is additional overhead which can be apparent to the user. Verifying the passphrase, for example, may require 20 seconds if the default iteration count is used when hashing the salted passphrase; our implementation used a smaller default iteration count. A small amount of time is added for interaction with the database filesystem for large memory transfers.

6.7.4 Key management

The key management system we implemented was the simplest one possible—the direct trust model (see §6.2.2). A menu item is available (see Figure 6.3) for emailing one's public key to another user. A function is also available for extracting and storing a public key received in an email message. If desired, a public key can be authenticated by verifying its fingerprint by some direct means (e.g., communicating it over the telephone—authenticity is provided by voice recognition).

6.8 Future Work

The following are some directions for future work.

Random number generation. Many systems implement a "random gathering device" whi ch attempts to use environmental noise (keyboard data, system timers, disk characteristics, etc.) to build a cryptographically secure source of random bits [39]. Our pager application used only a rather simple (and most likely not sufficiently secure) seeding process involving the clock and a few other sources. A more sophisticated solution is essential, perhaps tapping into the radio apparatus as a source.

Code size. No serious effort was made to minimize the size of the programs loaded to the pager. There is some code linked from the OpenSSL cryptographic library which could easily be removed (in fact, we were somewhat surprised that the library with the added elliptic curve routines could be used with relatively few modifications for the pager). The library routines adapted from OpenSSL and OpenPGP along with various glue needed

to adapt to the pager accounts for approximately 3/4 of the 370 KB loaded on the device (with the remainder attributed to code involving the screen and user-interface). If some interoperability can be sacrificed, then the code size can also be reduced by removing routines such as CAST or some of the hash algorithms.

Making the OpenPGP code more robust. The OpenPGP reference implementation provides minimal diagnostics and can easily break on bad data. The occasional segmentation fault triggered by bad user data may be merely unpleasant when an application is used on a workstation; such errors on the pager are completely unacceptable. Our application corrects some of the most troublesome shortcomings, but better error-handling is needed.

Key management. We would like to implement an X.509-based PKI or the web of trust model. In either case, we would implement a key server for retrieving and storing keys in a key repository. This would involve setting up a proxy wireless server with which the pager would communicate directly. The proxy server in turn would communicate with existing key servers on the Internet.

6.9 Conclusions

Implementing PGP on the RIM pager. The 32-bit architecture, relatively sophisticated operating system and development environment, and relatively large memory size means that development for the pager is closer to that done for more traditional systems than the small size might suggest. The user interface must be customized for the device, but "generic code" which does not involve file I/O moves fairly easily to the pager.

On the other hand, it appears likely that such devices will continue to have processors which run much more slowly than their desktop counterparts. Long delays in handling encrypted messages or signatures will be a considerable annoyance for users of this type of device. While we used a significant amount of the available memory on the pager, it would be desirable to reduce the resource consumption in a production version of PGP. Battery life will continue to be a major concern, and the overhead of authentication and confidentiality competes with the need to minimize transmissions from the device.

Interoperability. The goal of interoperability was met. All of the required algorithms from RFC 2440 are included, along with several listed as recommended and the elliptic curve routines. Our PGP implementation interoperated with existing implementations for the PalmPilot and workstations.

Elliptic curve cryptography. Elliptic curve solutions fit particularly well into the constrained environment. 1024-bit and 2048-bit RSA private-key operations are too slow for PGP applications, while the performance of 163-bit, 233-bit and 283-bit ECC operations

is tolerable for PGP applications. If PGP (or other email security solutions) is to be used for securing email communications between constrained wireless devices and desktop machines, then our timings show that ECC is preferable to RSA since the performance of the latter on some wireless devices is too slow, while both systems perform sufficiently well on workstations.

General. This paper concentrated on PGP, although the results are more widely applicable. Many of the services targeted at the growing wireless market will require security solutions involving the cryptographic mechanisms used by PGP. The constraints on small wireless devices are likely to be with us for some time, and will require a balance of usability, computational requirements, security, and battery life.

Acknowledgements

The authors would like to thank Jonathan Callas for some enlightening discussions about PGP, and Herb Little for answering our numerous questions about the RIM pager.

Capítulo 7

Conclusões

Nesta dissertação estudamos os criptossistemas de curvas elípticas (CCE) e sua implementação eficiente em software, enfatizando curvas elípticas definidas sobre o corpo finito $\mathbb{F}_{2^m}$.

Os resultados deste trabalho mostram que os CCE podem ser eficientemente implementados em diferentes plataformas tais como PCs, estações de trabalho, computadores de bolso e pagers. No desenvolvimento desta pesquisa foram propostos vários algoritmos para calcular eficientemente múltiplos de um ponto elíptico, a operação central dos CCE. Além disso, foi desenvolvido um algoritmo para multiplicação em $\mathbb{F}_{2^m}$. Assim, nos dois níveis (corpo finito e grupo elíptico) de operações fundamentais para o desempenho dos CCE, foram obtidos algoritmos eficientes. Outra contribuição, não menos importante, foi a implementação de uma biblioteca, baseada em nossos algoritmos, de suporte para curvas elípticas definidas sobre $\mathbb{F}_{2^m}$.

A biblioteca foi projetada para arquiteturas de 32 bits e contêm as seguintes implementações: as curvas NIST (aleatórias e Koblitz) sobre os corpos finitos $\mathbb{F}_{2^m}$, para $m = 163, 233$ e 283, e os algoritmos ECAES e ECDSA para ciframento e assinatura digital, respectivamente. A biblioteca foi escrita na linguagem C. Na nossa opinião, o desempenho da biblioteca em diferentes plataformas é muito bom quando comparado com outras implementações já documentadas. Esta biblioteca estará disponível em breve, sendo o primeiro software público a oferecer serviços criptográficos baseados em curvas elípticas sobre $\mathbb{F}_{2^m}$.

Nossa experiência nos permite elaborar as seguintes observações:

- A implementação em software dos algoritmos para operações no corpo finito é muito sensível ao hardware; este fato é especialmente notável quando comparamos os tempos das operações de cálculo de inversos multiplicativos e de multiplicação. A razão entre esses tempos pode influenciar a escolha do sistema de coordenadas dos pontos da curva elíptica. Por exemplo, se um cálculo de um inverso custa mais de 8

multiplicações, então os algoritmos em coordenadas projetivas oferecem vantagens computacionais sobre os algoritmos em coordenadas afins. Na nossa implementação do algoritmo de Euclides estendido e o algoritmo de multiplicação proposto, a razão inverso/multiplicação observada em diferentes plataformas, variou de 8 a 12. Nessa situação, nossos algoritmos para multiplicações escalares são os melhores candidatos para implementações em software.

- A decisão de escrever uma implementação da aritmética do corpo finito $\mathbb{F}_{2^m}$ foi muito importante nos resultados obtidos. Primeiro, as poucas bibliotecas públicas para corpos finitos de característica 2 não são suficientemente otimizadas, não estão escritas completamente em C, e não são fáceis de adaptar para um corpo finito particular. Segundo, nos permitiu avaliar melhor os algoritmos de multiplicação escalar, já que tínhamos a possibilidade de experimentar diferentes algoritmos para as operações no corpo finito. Finalmente, os progressos nos tempos de execução nos motivaram a testar diferentes técnicas matemáticas ou de programação para melhorar o desempenho da implementação.

Trabalhos futuros

Vislumbramos os seguintes desdobramentos e possibilidades de trabalhos futuros nesta linha de pesquisa:

- Projeto e implementação de uma biblioteca de suporte para operações no corpo $\mathbb{F}_{2^m}$, orientada para processadores de 8, 16 e 64 bits.
- Projeto e implementação de uma biblioteca de suporte das operações aritméticas módulo p, p primo, que tirem proveito de arquiteturas específicas e explorem a estrutura de $\mathbb{F}_p$, para valores particulares de p.
- Implementação e comparação de criptossistemas de curvas elípticas sobre corpos finitos $(\mathbb{F}_p, \mathbb{F}_{2^m}, \mathbb{F}_{p^m})$, em diferentes dispositivos limitados.
- Melhoramento dos algoritmos existentes ou desenvolvimento de novos algoritmos para implementação eficiente em software da aritmética no grupo de pontos de uma curva elíptica.
- Implementação e comparação entre o algoritmo de Euclides estendido e o algoritmo de Schroeppel, para os corpos finitos $\mathbb{F}_{2^m}$, $m = 163, 233$ e 283, em diferentes arquiteturas.

Bibliografia

[1] M. Abdalla, M. Bellare and P. Rogaway. "DHAES: An encryption scheme on the Diffie-Hellman problem", preprint 1999. Available at `http://www-cse.ucsd.edu/users/mihir/`

[2] G. B. Agnew, R. C. Mullin and S. A. Vanstone, "An implementation of elliptic curve cryptosystems over $\mathbb{F}_{2^{155}}$", *IEEE journal on selected areas in communications*, Vol **11**, No. 5, pp. 804-813, 1993.

[3] ANSI X9.30-1, "The digital signature algorithm (DSA) (revised)", American Bankers Association, working draft, July 1999.

[4] ANSI X9.52, "Triple data encryption algorithm modes of operation", American Bankers Association, 1998.

[5] ANSI X9.62, "The elliptic curve digital signature algorithm (ECDSA)", American Bankers Association, 1999.

[6] ANSI X9.63, "Elliptic curve key agreement and key transport protocols", American Bankers Association, working draft, August 1999.

[7] K. Araki, T. Satoh and S. Miura, "Overview of elliptic curve cryptography". In *Proceeding of Public-key Cryptography*, LNCS **1431**, pp. 29-49, Springer-Verlag, 1999.

[8] D. Ash, I. Blake and S. Vanstone, "Low complexity normal bases", *Discrete Applied Mathematics*, **25**, pp. 191-210, 1989.

[9] M. Aydos, E. Savas, and Ç. K. Koç, "Implementing network security protocols based on elliptic curve cryptography", *Proceedings of the Fourth Symposium on Computer Networks*, pp. 130-139, Istanbul, Turkey, May 20-21, 1999.

[10] R. Balasubramanian and N. Koblitz, "The improbability that an elliptic curve has a sub-exponential discrete log problem under the Menezes-Okamoto-Vanstone algorithm", Journal of Cryptology, **11**, pp. 141-145, (1998).

[11] Daniel Bailey and Christof Paar, "Optimal extension fields for fast arithmetic in public-key algorithms". In *Crypto'98*, LNCS **1462**, pp. 472-485, Springer-Verlag, 1998.

[12] Daniel Bailey and Christof Paar, "Inversion in optimal extension fields", *Proceedings of the Conference on The Mathematics of Public Key Cryptography*, Toronto, Canada, June 12-17, 1999.

[13] Blackberry, `http://www.blackberry.net`

[14] I. Blake, G. Seroussi, and N. Smart, *Elliptic Curves in Cryptography*, Cambridge University Press, 1999.

[15] Bogdan Antonescu, *Elliptic Curve Cryptosystems on Embedded Microprocessors*, Master's thesis, ECE Dept., Worcester Polytechnic Institute, Worcester, USA, May 1999.

[16] M. Brown, D. Cheung, D. Hankerson, J. Lopez, M. Kirkup and A. Menezes, "PGP in constrained wireless devices", *Proceedings of the 9th USENIX'2000 Security Symposium*, to appear.

[17] J. Callas, *OpenPGP Specification and Sample Code*, Printers Inc. Bookstore, Palo Alto, March 1999.

[18] J. Callas, L. Donnerhacke, H. Finney and R. Thayer, "OpenPGP message format", Internet RFC 2440, November 1998.

[19] Certicom, "ECC Challenge", Details available at `htpp://www.certicom.com/chal/`

[20] H. Cohen, A. Miyaji, and T. Ono, "Efficient elliptic curve exponentiation using mixed coordinates", In *Asiacrypt'98*, LNCS **1514**, pp. 51-65, Springer-Verlag, 1998.

[21] Biljana Cubaleska, Andreas Rieke, and Thomas Hermann, "Improving and Extending the Lim/Lee Exponentiation Algorithm", *Proceedings of SAC'99*, LNCS, to appear.

[22] W. Dai, Crypto++. `http://www.eskimo.com/ ~weidai/cryptlib.html`

[23] N. Daswani and D. Boneh, "Experimenting with electronic commerce on the Palm-Pilot", *Financial Cryptography '99*, LNCS **1648**, pp. 1-16, Springer-Verlag, 1999.

[24] E. De Win, A. Bosselaers, S. Vanderberghe, P. De Gersem and J. Vandewalle, "A fast software implementation for arithmetic operations in $GF(2^n)$," *Advances in Cryptology, Proc. Asiacrypt'96*, LNCS **1163**, pp. 65-76, Springer-Verlag, 1996.

[25] E. De Win, S. Mister, B. Prennel and M. Wiener, "On the performance of signature based on elliptic curves". In *Algorithmic Number Theory, Proceedings Third Intern. Symp.*, ANTS-III, LNCS **1423**, pp. 252-266, Springer-Verlag, 1998.

[26] W. Diffie and M. Hellman, "New directions in cryptography". IEEE *Transactions on Information Theory*, **22**, pp. 644-654, 1976.

[27] T. ElGamal, "A public key cryptosystems and a signature scheme based on discrete logarithms". IEEE *Transations on Informatio Theory*, **31**, pp. 469-472, 1985.

[28] G. Frey and H. Rück, "A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves", *Mathematics of Computation*, **62**, pp. 865-874, 1994.

[29] S. Galbraith and N. Smart, "A cryptographic application of Weil descent", *Codes and Cryptography*, LNCS **1746**, pp. 191-200, Springer-Verlag, 1999.

[30] R. Gallant, R. Lambert and S. Vanstone, "Improving the parallelized Pollard lambda search on binary anomalous curves", to appear in *Mathematics of Computation*.

[31] S. Garfinkel, *PGP: Pretty Good Privacy*, O'Reilly & Associates, 1995.

[32] P. Gaudry, F. Hess and N. Smart, "Constructive and destructive facets of Weil descent on elliptic curves", preprint, January 2000. Available at `http://www.hpl.hp.com/techreports/2000/HPL-2000-10.html`

[33] GEC 1. "Recommended elliptic curve domain parameters". Standards for Efficient Cryptography Group, September, 1999. Working draft. Available at `http://www.secg.org/`

[34] GNU Privacy Guard, `http://www.gnupg.org`

[35] I. Goldberg, "Pilot stuff from the ISAAC Group", `http://www.isaac.cs.berkeley.edu/pilot/`

[36] D. M. Gordon, "A survey of fast exponentiation methods", *Journal of Algorithms*, **27**, pp. 129-146, 1998.

[37] J. Guajardo and C. Paar, "Efficient algorithms for elliptic curve cryptosystems", *Advances in Cryptology, Proc. Crypto'97*, LNCS **1294**, pp. 342-356, Springer-Verlag, 1997.

[38] P. Gutmann, "Cryptolib". `http://www.cs. auckland.ac.nz/~pgut001/cryptlib`

[39] P. Gutmann, "Software generation of practically strong random numbers", *Proceedings of the Seventh USENIX Security Symposium*, pp. 243-257, 1998.

[40] D. Hankerson, J. Lopez and A. Menezes, "Software implementations of elliptic curve cryptography over fields of characteristic two", draft, 2000.

[41] T. Hasegawa, J. Nakajima and M. Matsui, "A practical implementation of elliptic curve cryptosystems over $GF(p)$ on a 16-bit microcomputer", *Public Key Cryptography - Proceedings of* PKC'98, LNCS **1431**, pp. 182-194, Springer-Verlag, 1998.

[42] IEEE P1363, "Standard specifications for public-key cryptography", ballot draft, 1999. Drafts available at `http://grouper.ieee.org/groups/1363`

[43] K. Itoh, M. Takenaka, N. Torii, S. Temma, and Y. Kurihara, "Fast implementation of public-key cryptography on a DSP TMS320C6201", In *Proceedings of the First Workshop on Cryptographic Hardware and Embedded Systems (CHES'99)*, LNCS **1717**, pp. 61-72, Springer-Verlag, 1999.

[44] D. Johnson and A. Menezes, "The elliptic curve digital signature algorithm (ECDSA)", Technical report CORR 99-06, Department of Combinatorics & Optimization, University of Waterloo, 1999. Available at `http://www.cacr.math.uwaterloo.ca/`

[45] E. W. Knudsen, "Elliptic scalar multiplication using point halving", In *Asiacrypt'99*, LNCS **1716**, pp. 135-149, Springer-Verlag, 1999.

[46] D.E. Knuth, *The Art of Computer Programming*, **2**-Semi-numerical Algorithms. Addison-Wesly, 2nd edition, 1981.

[47] N. Koblitz, "Elliptic curve cryptosystems", *Mathematics of Computation*, **48**, pp. 203-209, 1987.

[48] N. Koblitz, "CM-curves with good cryptographic properties". In *Advances in Cryptology:Crypto'91*, LNCS **576**, pp. 279-287, Springer-Verlag, 1992.

[49] N. Koblitz, *A Course in Number Theory and Cryptography*, 2nd edition, Springer-Verlag, 1994

[50] P. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems", *Advances in Cryptology-CRYPTO'96*, LNCS **1109**, pp. 104-113, Springer-Verlag, 1996.

[51] N. Koblitz, A.J. Menezes, and S. Vanstone, "The state of elliptic curve cryptography", *Designs, Codes, and Cryptography*, **19**, pp. 173-193, 2000.

[52] C. K. Koç and T. Acar, "Montgomery multiplication in $GF(2^k)$", *Designs, Codes and Cryptography*, **14**, pp. 57-69, 1998.

[53] C. K. Koç, "High-Speed RSA implementation", TR 201, RSA Laboratories, 73 pages, November 1994.

[54] K. Koyama and Y. Tsuruoka, "Speeding up elliptic cryptosystems by using a signed binary window method", In *Advances in Cryptography-CRYPTO'92*, LNCS **740**, pp. 345-357, Springer-Verlag, 1992.

[55] H. Krawczyk, M. Bellare and R. Cannetti, "HMAC:Keyed-hashing for message authentication", Internet RFC 2104, February 1997.

[56] A. Lenstra and E. Verheul, "Selecting cryptographic key sizes", *Proceedings of PKC 2000*, LNCS **1751**, pp. 446-465, Springer-Verlag, 2000.

[57] LiDIA Group **LiDIA v1.3**- A library for computational number theory. TH-Darmstadt, 1998.

[58] C. H. Lim and P. J. Lee, "More flexible exponentiation with precomputation", In *Advances in Cryptography-CRYPTO'94*, LNCS **839**, pp. 95-107, Springer-Verlag, 1994.

[59] J. Lopez and R. Dahab, "Improved algorithms for elliptic curve arithmetic in $GF(2^n)$", *SAC'98*, LNCS **1556**, pp. 201-212, Springer-Verlag, 1998.

[60] J. Lopez and R. Dahab, "Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation", *CHES'99*, LNCS **1717**, pp. 316-327, Springer-Verlag, 1999.

[61] J. Lopez and R. Dahab, "Performance of elliptic curve cryptosystems", Technical report, IC-00-08, 2000. Available at
`http://www.dcc.unicamp.br/ic-main/publications-e.html`

[62] J. Lopez and R. Dahab, "High-Speed software multiplication in $\mathbb{F}_{2^m}$", Technical report, IC-00-09, 2000. Available at
`http://www.dcc.unicamp.br/ic-main/publications-e.html`

[63] R. J. McEliece, *Finite Fields for Computer Scientists and Engineers*, Kluwer Academic Publishers, 1987.

[64] A. Menezes, *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, 1993.

[65] A. Menezes and S. Vanstone, "Elliptic curve cryptosystems and their implementation", *Journal of Cryptology*, **6**, pp. 209-224, 1993.

[66] A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

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

[68] V. Miller, "Uses of elliptic curves in cryptography", *Advances in Cryptology: proceedings of Crypto'85*, LNCS **218**, pp. 417-426, New York: Springer-Verlag, 1986.

[69] P. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization", *Mathematics of Computation*, vol **48**, pp. 243-264, 1987.

[70] V. Müller, "Fast multiplication on elliptic curves over small fields of characteristic two", *Journal of Cryptology*, **11**, pp. 219-234, 1998.

[71] R. Mullin, I. Onyszchuk, S. Vanstone and R. Wilson, "Optimal normal bases in $GF(p^n)$", *Discrete Applied Mathematics*, **22**, pp. 149-161, (1988/89).

[72] National Institute of Standards and Technology, "Digital Signature Standard", FIPS Publication 186-2, February 2000. Available at `http://csrc.nist.gov/fips`

[73] National Institute of Standards and Technology, "Secure Hash Standard (SHS)", FIPS Publication 180-1, April 1995. Available at `http://csrc.nist.gov/fips`

[74] The International PGP Home Page, `http://www.pgpi.org`

[75] PGP versions, `http://www.paranoia.com/ ~vax/pgp_versions.html`

[76] S.C. Pohlig and M.E. Hellman, "An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance., *IEEE Transactions on Information Theory*, **24**, pp. 106-110, 1978.

[77] David Poguet, *PalmPilot: The Ultimate Guide*, 2nd edition, O'Reilly & Associates, 1999.

[78] J. Pollard, "Monte Carlo methods for index computation mod p", *Mathematics of Computation*, **32**, pp. 918-924, 1978.

[79] J. Pollard, "Factoring with cubic integers", A. K. Lenstra and H. W. Lenstra Jr. editors, *The Development of the Number Field Sieve*, **1554**, Lecture Notes in Mathematics, pp. 4-10, Springer-Verlag, 1993.

[80] P. Van Oorschot and M. Wiener, "Parallel collision search with cryptanalytic applications", *Journal of Cryptology*, **12**, pp. 1-28, 1999.

[81] OpenSSL, `http://www.openssl.org`

[82] B. Ramsdell, "S/MIME version 3 message specification", Internet RFC 2633, June 1999.

[83] G. Reitwiesner, "Binary arithmetic", *Advances in Computers*, **1**, pp. 231-308, 1960.

[84] RIM Software Developer's Kit (SDK),
`http://developers.rim.net/handhelds/sdk`

[85] R. Rivest, A. Schamir and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, **21**, pp. 120-126, February 1978.

[86] M. Rosing, *Implementing Elliptic Curve Cryptography*, Manning Publications Greenwich, CT (1999).

[87] T. Satoh and K. Araki, "Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves", *Commentarii Mathematici Universitatis Sancti Pauli*, **47**, pp. 81-92, 1998.

[88] R. Schoof, "Elliptic curves over finite fields and the computation of square roots mod p", *Math. Comp.*, **44**, pp. 483-494, 1985.

[89] R. Schroeppel, H. Orman, S. O'Malley and O. Spatscheck, "Fast key exchange with elliptic curve systems," *Advances in Cryptology, Proc. Crypto'95*, LNCS **963**, pp. 43-56, Springer-Verlag, 1995.

[90] R. Schroeppel, H. Orman, S. O'Malley and O. Spatscheck, "Fast key exchange with elliptic curve systems", University of Arizona, C. S., Tech. report 95-03, 1995.

[91] R. Schroeppel, "Faster elliptic calculations in $GF(2^n)$," *preprint*, March 6, 1998.

[92] SEC 1, "Elliptic curve cryptography", Standards for Efficiency Cryptography Group, September, 1999. Working Draft. Available at `http://www.secg.org`

[93] I. Semaev, "Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p", *Mathematics of Computation*, **67**, pp. 353-356, 1998.

[94] G. Seroussi, "Compact representation of elliptic curve points over $\mathbb{F}_{2^m}$". Hewlett-Packard Laboratories, Technical report No. HPL-98-135, August 1998.

[95] N. Smart, "The discrete logarithm problem on elliptic curves of trace one", *Journal of Cryptology*, **12**,pp. 193-196, 1999.

[96] J. Solinas, "An improved algorithm for arithmetic on a family of elliptic curves," *Advances in Cryptology, Proc. Crypto'97*, LNCS **1294**, pp. 357-371, Spring-Verlag, 1997.

[97] J. Solinas, "An improved algorithm for arithmetic on a family of elliptic curves (revised)" Technical report CORR 99-06, Department of Combinatorics & Optimization, University of Waterloo, 1999. Available at `http://www.cacr.math.uwaterloo.ca/`

[98] J. Solinas, "Generalized Mersenne numbers", Technical report CORR 99-06, Department of Combinatorics & Optimization, University of Waterloo, 1999. Available at `http://www.cacr.math.uwaterloo.ca/`

[99] J. Solinas, "Efficient arithmetic on Koblitz curves", *Designs, Codes and Cryptography*, **19**, pp. 195-249, 2000.

[100] S. Vanstone, "Responses to NIST's Proposal", *Communications of the ACM*, **35**, pp. 50-52, (communicated by John Anderson), July 1992.

[101] Wireless Application Protocol Forum, "Wireless Application Protocol", John Wiley & Sons, Inc., 1999. see also `http://www. wapforum.org/`

[102] WAP white paper, 1999, `http://www.wap forum.org/what/whitepapers.htm`

[103] Wireless Application Protocol Forum, "Wireless Application Protocol Wireless Transport Layer Security Specification", Chapter 16 of [101], 1999. Drafts available at `http://www.wapforum.org`

[104] A. Whitten and J. Tygar, "Why Johnny can't encrypt: A usability evaluation of PGP 5.0", *Proceedings of the Eighth USENIX Security Symposium*, 1999.

[105] M. Wiener and R. Zuccherato, "Faster attacks on elliptic curve cryptosystems", *Selected Areas in Cryptography'98*, LNCS **1556**, pp. 190-200, Springer-Verlag, 1998.

[106] P. Zimmermann, *The Official PGP User's Guide*, MIT Press, 1995.