An Overview of Elliptic Curve Cryptography

Julio López *Ricardo Dahab*

Relatório Técnico IC–00-10

Maio de 2000

An Overview of Elliptic Curve Cryptography

Julio López[*] Ricardo Dahab[†]

Institute of Computing
State University of Campinas
Campinas, 13081-970 São Paulo, Brazil
{julioher,rdahab}@dcc.unicamp.br

May 22, 2000

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.

1 Introduction

In 1985, Victor Miller [56] and N. Koblitz [36], independently, proposed a public-key cryptosystem analogue of the ElGamal schemes [21] 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

[*]Institute of Computing, State University of Campinas, 13081-970 Campinas, SP, Brazil, and Dept. of Computer Science, University of Valle, Cali, Colombia.

[†]Institute of Computing, State University of Campinas, 13081-970 Campinas, SP, Brazil. Research partially supported by a Pronex-Finep grant 107/97.

(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 [79]). 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. A brief introduction to elliptic curves is presented in Section 3. A list of the main known attacks on the elliptic curve discrete logarithm problem (ECDLP) is provided in Section 4. In Section 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 6.

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 [51]), 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 ($q = 2^m$).

2.1 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 $0 \le r \le p-1$. 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 $0 \le s \le p-1$. 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

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

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 [79]. For more examples of primes that are well suited for machine implementation, see [79] and [59]. Several techniques for implementing the finite field arithmetic in $\mathbb{F}_p$ are described in [35, 54, 12, 32, 19, 30].

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 [51]) 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 [31]. 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.

The existence of a Gaussian normal basis has been characterized in [58] and [6]. 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 $a = (a_0 a_1 \ldots a_{m-1})$ and $b = (b_0 b_1 \ldots b_{m-1})$. Then:

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

- *Squaring:* Since squaring is a linear operation in $\mathbb{F}_{2^m}$,
$$a^2 = (\sum_{i=0}^{m-1} a_i \beta^{2^i})^2 = \sum_{i=0}^{m-1} a_i \beta^{2^{i+1}} = \sum_{i=0}^{m-1} a_{i-1 \bmod m} \beta^{2^i} = (a_{m-1} a_0 a_1 \ldots a_{m-2}).$$
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 $F(1), F(2), \ldots, F(p-1)$ 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
$$\begin{aligned} A_l &= \sum_{k=1}^{p-2} a_{F(k+1)+l}\, b_{F(p-k)+l}, \text{ and} \\ B_l &= \sum_{k=1}^{m/2} (a_{k+l-1}\, b_{m/2+k+l-1} + a_{m/2+k+l-1}\, b_{k+l-1}) + A_l. \end{aligned}$$

[1]Although this selection may affect the speed of the almost inverse algorithm (see [19]), it allows for faster reduction modulo $f(x)$.

Then $a \cdot b = c = (c_0 c_1 \ldots c_{m-1})$, 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 $l, 0 \leq l \leq m-1$, where indices are reduced modulo m.

See [31] 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 [67] for a hardware and software implementation, respectively, of a normal basis in $\mathbb{F}_{2^m}$.

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: $a = (a_{m-1} \dots a_1 a_0)$ and $f = (f_m f_{m-1} \dots f_1 f_0)$
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* [70] 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 $f = (f_m f_{m-1} \dots f_1 f_0)$
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 [50] 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* [70]. 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* [47]. In this case, we trade inversions for multiplications and other finite field operations.

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* [12], Menezes [52], Chapter 6 of Koblitz's book [38], and [73].

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.

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:

$$y^2 = x^3 + ax + b \tag{1}$$

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 [16] 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($Q \leftarrow (x_3, y_3)$).
```

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:

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

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 [47].

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

[2]An alternative method for computing $2P$ is described in [47].

[3]See [2] for a hardware implementation and [29] for a software implementation of $\mathbb{F}_{2^m}$ where an inversion costs about 24 and 10 multiplications, respectively.

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($Q \leftarrow (x_3, y_3)$).
```

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 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 $k \equiv l \pmod r$.

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 [69]. 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 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 $|t| \le 2\sqrt{q}$. 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 [75], is the basis of an efficient algorithm for a compact representation of points on elliptic curves over

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

$\mathbb{F}_{2^m}$. Knudsen's method [34] for computing elliptic scalar multiplications is also based on this result.

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 [59] and [26]). Since the primary security parameter is n, the ECC key length is thus defined to be the bit-length of n. For example, NIST curves [59] 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 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 1: ECC, DSA and RSA key length comparisons.

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 [81] 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 [73]). A public key $Q = (x_Q, y_Q)$ associated 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 [73]).

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

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

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 [43], 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 [60].

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 $e = \text{SHA-1}(m)$.

9. Compute $w = s^{-1} \bmod n$.
10. Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
11. Compute $u_1 G + u_2 Q_A = (x_1, y_1)$.
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.

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 [61] 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 [62], and its recent modifications by Gallant, Lambert, and Vanstone [24], and Wiener and Zuccherato [82], which requires about $\sqrt{\pi n}/2$ elliptic group operations. Van Oorschot and Wiener [63] 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 [55] and Frey and Rück [22] 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 subexponential time algorithm for the ECDLP. However, Balasubramanian and Koblitz [8] 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 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, $1 \leq k \leq 30$.

[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 $\mathbb{F}_q$.

- *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 [74], Smart [76] and Satoh and Araki [68] 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 [23], and Gaundry, Hess and Smart [25] 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 [12, 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 [3], ANSI X9.63 [4], Blake, Seroussi and Smart [12], Johnson and Menezes [33], Koblitz, Menezes and Vanstone [40], Araki, Satoh and Miura [5], and Certicom's ECC challenge [15].

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 [27, 54, 41, 42]. 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.

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 [65] (see also [12]).
- The expected weight of a NAF of length l is $l/3$, see [12].
- 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 [54]). The following method, from Solinas [78], 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 [31], we obtain an improvement of about 14% over the binary method.

[7]This algorithm can be modified to obtain a *right-to-left* version, which does not need storage for the NAF(k), see [78] for more details.

```
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($Q$).
```

<u>*Window method.*</u> 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 [78]).

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.

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

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

1.  Set $c \leftarrow k$,  $l \leftarrow 0$
2.  while $c > 0$ do
        if $c$ odd then
            Set $u_l \leftarrow 2 - (c \bmod 2^w)$
            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($\mathrm{NAF}_w(k) \leftarrow (u_{l-1} \dots u_1 u_0)$).
```

```
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 $\mathrm{NAF}_w(k)$ = $(u_{l-1} \dots u_1 u_0)$
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 $l/(w + 1)$ [80]. 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 [46], 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:

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

where the precomputation array $G[s][u]$ for $0 \leq s < v$, $0 \leq u < 2^h$, and $u = (u_{h-1} \dots u_0)_2$, 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 [17]. For other algorithms for computing kP when P is a known point, see [54].

```
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 $u \leftarrow (u_{h-1} \dots u_1 u_0)_2$
        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($Q$).
```

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^iP), for elliptic curves defined over $\mathbb{F}_{2^m}$ was proposed by López and Dahab [47]. This algorithm, an improvement over the formulas presented by Guajardo and Paar [28], computes 2^iP with only one inversion, and it is faster than the usual method for computing 2^iP (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 [72]. 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 $Q = 2^iP$

```
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 $\lambda_2 \leftarrow \lambda^2 + a + \frac{b}{x^4 + b}$
        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* [32] 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 [57], was proposed by Lopez and Dahab [48]. 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 [39].
- An algorithm for computing elliptic scalar multiplications which replaces the doubling operation by the halving operation (i.e., the computation of Q such that $2Q = P$)

[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 $E(\mathbb{F}_{2^m})$.

was proposed by Knudsen [34]. 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 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.

5.3 Koblitz curves

These curves, also known as binary anomalous curves, were first proposed for cryptographic use by Koblitz [37]. 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

$$(x^4, y^4) + 2P = \mu \cdot (x^2, y^2). \tag{3}$$

where

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

By using the Frobenius map over $\mathbb{F}_2$: $\tau(x, y) = (x^2, y^2)$, equation (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 [37], 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 [78]. 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: $\tau$- 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 $r \leftarrow \lfloor \rho_a(m) \cdot k/2^{m-1} \rfloor$,   $s \leftarrow \lfloor \sigma_a(m) \cdot k/2^m \rfloor$
2.  Set $t \leftarrow 2\rho_a(m) + \mu\sigma_a(m)$,   $v \leftarrow \sigma_a(m) \cdot s$
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 [47], 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 [78][10] for the "width-w τ-*addic* NAF method" analogous to Algorithm 11.

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 considerations (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.

[9]For hardware implementations, the squarings are much faster than multiplications.
[10]Routine 6 from [78] fails when $a = 0$ and $w = 6$. A new version of this routine was given in [80].

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 [63] againts general curves, and its improvements [24, 82] which apply to Koblitz curves.
2. $\#E(\mathbb{F}_q) \neq q$, to resist the following attacks: Semaev [74], Smart [76], and Satoh-Araki [68].
3. n does not divide $q^k - 1$ for all $1 \leq k \leq 30$, to resist the Weil paring attack [55] and the Tate paring attack [22].
4. All binary fields $\mathbb{F}_{2^m}$ chosen have the property that m is prime, to resist recent attacks [23, 25] 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 [59] and [26].

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 $\mathbb{F}_{2^{155}}$[70], $\mathbb{F}_{2^{167}}$ [13], $\mathbb{F}_{2^{176}}$ [28, 7], $\mathbb{F}_{2^{191}}$ [19], $\mathbb{F}_p$ (p a 160-bit prime) [30], $\mathbb{F}_p$ (p a 192-bit prime) [19], and $\mathbb{F}_{(2^{63}-25)^3}$ [9]. In [49], 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.* [70] 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.

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

- De Win *et al.* [19] 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 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 3: Timing comparison of ECDSA , DSA, and RSA signature operations. All timings in milliseconds, unless otherwise indicated.

- Bailey and Paar [9] 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 [10], 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 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 4: Timings (in milliseconds) for an elliptic scalar multiplication.

6.3 An example of a software implementation of ECC

In this section we present some details of the ECC software implementation reported in [14]. 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 [50]. 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 [59].
- *Algorithms for the elliptic curve group:* For random curves, the method given in [48] was implemented for computing scalar multiplications when P is an arbitrary point. Lim/Lee's method [54], with 16 points of precomputation, was implemented using the projective coordinates given in [47] for computing scalar multiplications when P is a known point (e.g., for signing). For a Koblitz curve, Solinas' methods [78] were implemenemented 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 [64], 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 [14] for more timings). Table 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 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 6. Timings for RSA operations, with a modulus of 1024 bits, are given in Table 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 6: Timings (in milliseconds) for ECC operations over$\mathbb{F}_{2^{163}}$.

- *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.

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

– 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.

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.

8 Acknowledgments

The authors wish to thank Guido Araújo, Cláudio Lucchesi, Alfred Menezes, Daniel Panario and Routo Terada for many helpful comments and suggestions.

References

[1] M. Abdalla, M. Bellare and P. Rogaway. “DHAES: An encryption scheme on the Diffie-Hellman problem”, preprint 1999. `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 X9.62, “The elliptic curve digital signature algorithm (ECDSA)”, American Bankers Association, 1999.

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

[5] 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.

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

[7] 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.

[8] 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).

[9] 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.

[10] 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.

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

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

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

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

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

[16] 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.

[17] Biljana Cubaleska, Andreas Rieke, and Thomas Hermann, "Improving and extending the Lim/Lee exponentiation algorithm", *Proceeding of SAC'99*, LNCS, to appear.

[18] 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.

[19] 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.

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

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

[22] 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.

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

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

[25] 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`

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

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

[28] 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.

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

[30] 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.

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

[32] 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.

[33] 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/`

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

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

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

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

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

[39] 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.

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

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

[42] 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.

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

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

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

[46] 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.

[47] 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.

[48] J. Lopez and R. Dahab, "Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation", *Proceedings of the First Workshop on Cryptographic Hardware and Embedded Systems (CHES'99)*, LNCS **1717**, pp. 316-327, Springer-Verlag, 1999.

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

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

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

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

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

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

[55] 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.

[56] 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.

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

[58] 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).

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

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

[61] 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.

[62] J. Pollard, "Monte Carlo methods for index computation mod p", *Mathematics of Computation*, **32**, pp. 918-924, 1978.

[63] P. Van Oorschot and M. Wiener, "Parallel collision search with cryptanalytic applications", *Journal of Cryptology*, **12**, pp. 1-28, 1999.

[64] OpenSSL, `http://www.openssl.org`

[65] G. Reitwiesner, "Binary arithmetic", *Advances in Computers*, **1**, pp. 231-308, 1960.

[66] 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.

[67] M. Rosing, *Implementing Elliptic Curve Cryptography*, Manning Publications Greenwich, CT (1999).

[68] 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.

[69] R. Schoof, "Elliptic curves over finite fields and the computation of square roots mod p", *Math. Comp.,* **44**, pp. 483-494, 1985.

[70] 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.

[71] 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.

[72] R. Schroeppel, "Faster elliptic calculations in $GF(2^n)$," *preprint*, March 6, 1998.

[73] SEC 1, "Elliptic curve cryptography", Standards for Efficiency Cryptography Group, September, 1999. Working Draft. Available at `http://www.secg.org`

[74] 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.

[75] G. Seroussi, "Compact representation of elliptic curve points over $\mathbb{F}_{2^m}$". Hewlett-Packard Laboratories, Technical report No. HPL-98-135, August 1998.

[76] N. Smart, "The discrete logarithm problem on elliptic curves of trace one", *Journal of Cryptology*, **12**,pp. 193-196, 1999.

[77] J. Solinas, "An improved algorithm for arithmetic on a family of elliptic curves," *Advances in Cryptology, Proc. Crypto'97*, LNCS **1294** B. Kaliski, Ed., Spring-Verlag, pp. 357-371, 1997.

[78] 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/`

[79] 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/`

[80] J. Solinas, "Efficient arithmetic on Koblitz curves", *Designs, Codes and Cryptography*, **19**, pp. 195-249, 2000.

[81] S. Vanstone, "Responses to NIST's proposal", *Communications of the ACM*, **35**, pp. 50-52, (communicated by John Anderson), July 1992.

[82] M. Wiener and R. Zuccherato, "Faster attacks on elliptic curve cryptosystems", *Selected Areas in Cryptography'98*, LNCS **1556**, pp. 190-200, Springer-Verlag, 1998.