IJIS (2001) 1: 36–63 / Digital Object Identifier (DOI) 10.1007/s102070100002

The Elliptic Curve Digital Signature Algorithm (ECDSA)

Don Johnson[1], **Alfred Menezes**[1,2], **Scott Vanstone**[1,2]

[1] Certicom Research, Canada
[2] Department of Combinatorics and Optimization, University of Waterloo, Canada
E-mails: {djohnson,amenezes,svanstone}@certicom.com

Published online: 27 July 2001 –

Abstract. The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analogue of the Digital Signature Algorithm (DSA). It was accepted in 1999 as an ANSI standard and in 2000 as IEEE and NIST standards. It was also accepted in 1998 as an ISO standard and is under consideration for inclusion in some other ISO standards. Unlike the ordinary discrete logarithm problem and the integer factorization problem, no subexponential-time algorithm is known for the elliptic curve discrete logarithm problem. For this reason, the strength-per-key-bit is substantially greater in an algorithm that uses elliptic curves. This paper describes the ANSI X9.62 ECDSA, and discusses related security, implementation, and interoperability issues.

Keywords: Signature schemes – Elliptic curve cryptography – DSA – ECDSA

1 Introduction

The Digital Signature Algorithm (DSA) was specified in a U.S. Government Federal Information Processing Standard (FIPS) called the Digital Signature Standard (DSS [70]). Its security is based on the computational intractability of the discrete logarithm problem (DLP) in prime-order subgroups of $\mathbb{Z}_p^*$.

Elliptic curve cryptosystems (ECC) were invented by Neal Koblitz [49] and Victor Miller [67] in 1985. They can be viewed as elliptic curve analogues of the older discrete logarithm (DL) cryptosystems in which the subgroup of $\mathbb{Z}_p^*$ is replaced by the group of points on an elliptic curve over a finite field. The mathematical basis for the security of elliptic curve cryptosystems is the computational intractability of the elliptic curve discrete logarithm problem (ECDLP).

Since the ECDLP appears to be significantly harder than the DLP, the strength-per-key-bit is substantially greater in elliptic curve systems than in conventional discrete logarithm systems. Thus, smaller parameters, but with equivalent levels of security, can be used with ECC than with DL systems. The advantages that can be gained from smaller parameters include speed (faster computations) and smaller keys and certificates. These advantages are especially important in environments where processing power, storage space, bandwidth, or power consumption is constrained.

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analogue of the DSA. ECDSA was first proposed in 1992 by Scott Vanstone [108] in response to NIST's (National Institute of Standards and Technology) request for public comments on their first proposal for DSS. It was accepted in 1998 as an ISO (International Standards Organization) standard (ISO 14888-3), accepted in 1999 as an ANSI (American National Standards Institute) standard (ANSI X9.62), and accepted in 2000 as an IEEE (Institute of Electrical and Electronics Engineers) standard (IEEE 1363-2000) and a FIPS standard (FIPS 186-2). It is also under consideration for inclusion in some other ISO standards. In this paper, we describe the ANSI X9.62 ECDSA, present rationale for some of the design decisions, and discuss related security, implementation, and interoperability issues.

The remainder of this paper is organized as follows. In Sect. 2, we review digital signature schemes and the DSA. A brief tutorial on finite fields and elliptic curves is provided in Sects. 3 and 4, respectively. In Sect. 5, methods for domain parameter generation and validation are considered, while Sect. 6 discusses methods for key pair generation and public key validation. The ECDSA signature and verification algorithms are presented in Sect. 7. The security of ECDSA is studied in Sect. 8. Finally, some im-

plementation and interoperability issues are considered in Sects. 9 and 10.

2 Digital signature schemes

2.1 Background

Digital signature schemes are designed to provide the digital counterpart to handwritten signatures (and more). A digital signature is a number dependent on some secret known only to the signer (the signer's private key), and, additionally, on the contents of the message being signed. Signatures must be verifiable – if a dispute arises as to whether an entity signed a document, an unbiased third party should be able to resolve the matter equitably, without requiring access to the signer's private key. Disputes may arise when a signer tries to *repudiate* a signature it did create, or when a *forger* makes a fraudulent claim.

This paper is concerned with asymmetric digital signatures schemes with an appendix. "Asymmetric" means that each entity selects a key pair consisting of a private key and a related public key. The entity maintains the secrecy of the private key that it uses for signing messages, and makes authentic copies of its public key available to other entities which use it to verify signatures. "Appendix" means that a cryptographic hash function is used to create a message digest of the message, and the signing transformation is applied to the message digest rather than to the message itself.

Security. Ideally, a digital signature scheme should be existentially unforgeable under chosen-message attack. This notion of security was introduced by Goldwasser et al. [33]. Informally, it asserts that an adversary who is able to obtain entity A's signatures for any message of its choice is unable to successfully forge A's signature on a single other message.

Applications. Digital signature schemes can be used to provide the following basic cryptographic services: data integrity (the assurance that data has not been altered by unauthorized or unknown means), data origin authentication (the assurance that the source of data is as claimed), and non-repudiation (the assurance that an entity cannot deny previous actions or commitments). Digital signature schemes are commonly used as primitives in cryptographic protocols that provide other services including entity authentication (e.g., FIPS 196 [72], ISO/IEC 9798-3 [40], and Blake-Wilson and Menezes [10]), authenticated key transport (e.g., Blake-Wilson and Menezes [10], ANSI X9.63 [4], and ISO/IEC 11770-3 [41]), and authenticated key agreement (e.g., ISO/IEC 11770-3 [41], Diffie et al. [21], and Bellare et al. [8]).

Classification. The digital signature schemes in use today can be classified according to the hard underlying mathematical problem which provides the basis for their security:

1. Integer factorization (IF) schemes, which base their security on the intractability of the integer factorization problem. Examples of these include the RSA [85] and Rabin [84] signature schemes.
2. Discrete logarithm (DL) schemes, which base their security on the intractability of the (ordinary) discrete logarithm problem in a finite field. Examples of these include the ElGamal [23], Schnorr [90], DSA [70], and Nyberg-Rueppel [78, 79] signature schemes.
3. Elliptic curve (EC) schemes, which base their security on the intractability of the elliptic curve discrete logarithm problem.

2.2 The Digital Signature Algorithm (DSA)

The DSA was proposed in August 1991 by the U.S. National Institute of Standards and Technology (NIST) and was specified in a U.S. Government Federal Information Processing Standard (FIPS 186 [70]) called the Digital Signature Standard (DSS). The DSA can be viewed as a variant of the ElGamal signature scheme [23]. Its security is based on the intractability of the discrete logarithm problem in prime-order subgroups of $\mathbb{Z}_p^*$.

DSA domain parameter generation. Domain parameters are generated for each entity in a particular security domain. (See also the note below on the secure generation of parameters.)

1. Select a 160-bit prime q and a 1024-bit prime p with the property that $q \mid p-1$.
2. (Select a generator g of the unique cyclic group of order q in $\mathbb{Z}_p^*$.)
   Select an element $h \in \mathbb{Z}_p^*$ and compute $g = h^{(p-1)/q} \bmod p$. (Repeat until $g \neq 1$.)
3. Domain parameters are p, q, and g.

DSA key pair generation. Each entity A in the domain with domain parameters (p, q, g) does the following:

1. Select a random or pseudorandom integer x such that $1 \leq x \leq q-1$.
2. Compute $y = g^x \bmod p$.
3. A's public key is y; A's private key is x.

DSA signature generation. To sign a message m, A does the following:

1. Select a random or pseudorandom integer k, $1 \leq k \leq q-1$.
2. Compute $X = g^k \bmod p$ and $r = X \bmod q$. If $r = 0$ then go to step 1.
3. Compute $k^{-1} \bmod q$.
4. Compute $e = \text{SHA-1}(m)$.
5. Compute $s = k^{-1}\{e + xr\} \bmod q$. If $s = 0$ then go to step 1.
6. A's signature for the message m is (r, s).

DSA signature verification. To verify A's signature (r, s) on m, B obtains authentic copies of A's domain parameters (p, q, g) and public key y and does the following:

1. Verify that r and s are integers in the interval $[1, q-1]$.
2. Compute $e = \text{SHA-1}(m)$.
3. Compute $w = s^{-1} \bmod q$.
4. Compute $u_1 = ew \bmod q$ and $u_2 = rw \bmod q$.
5. Compute $X = g^{u_1} y^{u_2} \bmod p$ and $v = X \bmod q$.
6. Accept the signature if and only if $v = r$.

Security analysis. Since r and s are each integers less than q, DSA signatures are 320 bits in size. The security of the DSA relies on two distinct but related discrete logarithm problems. One is the discrete logarithm problem in $\mathbb{Z}_p^*$ where the number field sieve algorithm (see Gordon [35] and Schirokauer [89]) applies; this algorithm has a subexponential running time. More precisely, the expected running time of the algorithm is

$$O\left(\exp\left((c+o(1))(\ln p)^{1/3}(\ln\ln p)^{2/3}\right)\right), \tag{1}$$

where $c \approx 1.923$, and $\ln n$ denotes the natural logarithm function. If p is a 1024-bit prime, then the expression (1) represents an infeasible amount of computation; thus the DSA using a 1024-bit prime p is currently not vulnerable to this attack. The second discrete logarithm problem works to the base g in the subgroup of order q in $\mathbb{Z}_p^*$: given p, q, g, and y, find x such that $y \equiv g^x \pmod p$. For large p (e.g., 1024 bits), the best algorithm known for this problem is Pollard's rho method [83], and takes about

$$\sqrt{\pi q/2} \tag{2}$$

steps. If $q \approx 2^{160}$, then the expression (2) represents an infeasible amount of computation; thus the DSA is not vulnerable to this attack. However, note that there are two primary security parameters for DSA: the size of p and the size of q. Increasing one without a corresponding increase in the other will not result in an effective increase in security. Furthermore, an advance in algorithms for either one of the two discrete logarithm problems could weaken DSA.

Secure generation of parameters. In response to some criticisms received on the first draft (see Rueppel et al. [86] and Smid and Branstad [99]), FIPS 186 specified a method for generating primes p and q "verifiably at random". This feature prevents an entity (e.g., a central authority generating domain parameters to be shared by a network of entities) from intentionally constructing "weak" primes p and q for which the discrete logarithm problem is relatively easy. For a further discussion of this issue, see Gordon [34]. FIPS 186 also specifies two methods, based on DES and SHA-1, for pseudorandomly generating private keys x and per-message secrets k. FIPS 186 mandates the use of these algorithms or any other FIPS-approved security methods.

3 Finite fields

We provide a brief introduction to finite fields. For further information, see Chapt. 3 of Koblitz [52], or the books by McEliece [61] and Lidl and Niederreitter [59].

A *finite field* consists of a finite set of elements F together with two binary operations on F, called addition and multiplication, that satisfy certain arithmetic properties. The *order* of a finite field is the number of elements in the field. There exists a finite field of order q if and only if q is a prime power. If q is a prime power, then there is essentially only one finite field of order q; this field is denoted by $\mathbb{F}_q$. There are, however, many ways of representing the elements of $\mathbb{F}_q$. 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 the elliptic curve cryptographic techniques restrict the order of the underlying finite field to be an odd prime ($q = p$) or a power of 2 ($q = 2^m$). In Sect. 3.1, we describe the elements and operations of the finite field $\mathbb{F}_p$. In Sect. 3.2, elements and the operations of the finite field $\mathbb{F}_{2^m}$ are described, together with two methods for representing the field elements: *polynomial basis representations* and *normal basis representations.*

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

Let p be a prime number. The finite field $\mathbb{F}_p$, called a *prime field*, is comprised 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 when $a + b$ is divided by p and $0 \le r \le p-1$. This is known as *addition modulo p.*
- *Multiplication:* If $a, b \in \mathbb{F}_p$, then $a \cdot b = s$, where s is the remainder when $a \cdot b$ is divided by p and $0 \le s \le p-1$. This is known as *multiplication modulo p.*
- *Inversion:* If a is a non-zero element in $\mathbb{F}_p$, the *inverse* of a modulo p, denoted a^{-1}, is the unique integer $c \in \mathbb{F}_p$ for which $a \cdot c = 1$.

Example 1. (*The finite field $\mathbb{F}_{23}$*) The elements of $\mathbb{F}_{23}$ are $\{0, 1, 2, \ldots, 22\}$. Examples of the arithmetic operations in $\mathbb{F}_{23}$ are: (1) $12 + 20 = 9$; (2) $8 \cdot 9 = 3$; and (3) $8^{-1} = 3$.

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

The field $\mathbb{F}_{2^m}$, called a *characteristic two finite field* or a *binary finite field*, can be viewed as a vector space of dimension m over the field $\mathbb{F}_2$, which consists of the two elements 0 and 1. That is, there exist m elements

$\alpha_0, \alpha_1, \ldots, \alpha_{m-1}$ in $\mathbb{F}_{2^m}$ such that each element $\alpha \in \mathbb{F}_{2^m}$ can be uniquely written in the form:

$$\alpha = a_0\alpha_0 + a_1\alpha_1 + \cdots + a_{m-1}\alpha_{m-1}, \text{ where } a_i \in \{0,1\}.$$

Such a set $\{\alpha_0, \alpha_1, \ldots, \alpha_{m-1}\}$ is called a *basis* of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$. Given such a basis, a field element α can be represented as the bit string $(a_0a_1 \ldots a_{m-1})$. Addition of field elements is performed by bitwise XOR-ing the vector representations. The multiplication rule depends on the basis selected.

There are many different bases of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$. Some bases lead to more efficient software or hardware implementations of the arithmetic in $\mathbb{F}_{2^m}$ than other bases. ANSI X9.62 permits two kinds of bases: polynomial bases and normal bases.

3.2.1 Polynomial basis representations

Let $f(x) = x^m + f_{m-1}x^{m-1} + \cdots + f_2x^2 + f_1x + f_0$ (where $f_i \in \{0,1\}$ for $i = 0, 1, \ldots, m-1$) be an irreducible polynomial of degree m over $\mathbb{F}_2$. That is, $f(x)$ cannot be factored as a product of two polynomials over $\mathbb{F}_2$, each of degree less than m. Each such polynomial $f(x)$ defines a polynomial basis representation of $\mathbb{F}_{2^m}$, which is described next. $f(x)$ is called the *reduction polynomial*.

Field elements. The finite field $\mathbb{F}_{2^m}$ is comprised of all polynomials over $\mathbb{F}_2$ of degree less than m:

$$\mathbb{F}_{2^m} = \{a_{m-1}x^{m-1} + \cdots + a_1x + a_0 \,:\, a_i \in \{0,1\}\}.$$

The field element $a_{m-1}x^{m-1} + \cdots + a_1x + a_0$ is usually denoted by the bit string $(a_{m-1} \ldots a_1a_0)$ of length m, so that

$$\mathbb{F}_{2^m} = \{(a_{m-1} \ldots a_1a_0) \,:\, a_i \in \{0,1\}\}.$$

Thus the elements of $\mathbb{F}_{2^m}$ can be represented by the set of all binary strings of length m. The multiplicative identity element (1) is represented by the bit string $(00 \ldots 01)$, while the additive identity element (0) is represented by the bit string of all 0's.

Field operations. The following arithmetic operations are defined on the elements of $\mathbb{F}_{2^m}$ when using a polynomial basis representation with reduction polynomial $f(x)$:

- *Addition:* If $a = (a_{m-1} \ldots a_1a_0)$ and $b = (b_{m-1} \ldots b_1b_0)$ are elements of $\mathbb{F}_{2^m}$, then $a + b = c = (c_{m-1} \ldots c_1c_0)$, where $c_i = (a_i + b_i) \bmod 2$. That is, field addition is performed bitwise.
- *Multiplication:* If $a = (a_{m-1} \ldots a_1a_0)$ and $b = (b_{m-1} \ldots b_1b_0)$ are elements of $\mathbb{F}_{2^m}$, then $a \cdot b = r = (r_{m-1} \ldots r_1r_0)$, where the polynomial $r_{m-1}x^{m-1} + \cdots + r_1x + r_0$ is the remainder when the polynomial

$$(a_{m-1}x^{m-1} + \cdots + a_1x + a_0) \cdot (b_{m-1}x^{m-1} + \cdots + b_1x + b_0)$$

  is divided by $f(x)$ over $\mathbb{F}_2$.
- *Inversion:* If a is a non-zero element in $\mathbb{F}_{2^m}$, the *inverse* of a, denoted a^{-1}, is the unique element $c \in \mathbb{F}_{2^m}$ for which $a \cdot c = 1$.

Example 2. (*A polynomial basis representation of the finite field* $\mathbb{F}_{2^4}$) Let $f(x) = x^4 + x + 1$ be the reduction polynomial. Then the 16 elements of $\mathbb{F}_{2^4}$ are:

$0(0000)$	$x^3(1000)$
$1(0001)$	$x^3+1(1001)$
$x(0010)$	$x^3+x(1010)$
$x+1(0011)$	$x^3+x+1(1011)$
$x^2(0100)$	$x^3+x^2(1100)$
$x^2+1(0101)$	$x^3+x^2+1(1101)$
$x^2+x(0110)$	$x^3+x^2+x(1110)$
$x^2+x+1(0111)$	$x^3+x^2+x+1(1111)$.

Examples of the arithmetic operations in $\mathbb{F}_{2^4}$ are:

- $(1101) + (1001) = (0100)$.
- $(1101) \cdot (1001) = (1111)$ since $(x^3+x^2+1) \cdot (x^3+1) = x^6+x^5+x^2+1$ and $(x^6+x^5+x^2+1) \bmod (x^4+x+1) = x^3+x^2+x+1$.
- $(1101)^{-1} = (0100)$.

The element $\alpha = x = (0010)$ is a generator of $\mathbb{F}_{2^4}^*$ since its order is 15 as the following calculations show:

$\alpha^1 = (0010)$	$\alpha^2 = (0100)$	$\alpha^3 = (1000)$
$\alpha^4 = (0011)$	$\alpha^5 = (0110)$	$\alpha^6 = (1100)$
$\alpha^7 = (1011)$	$\alpha^8 = (0101)$	$\alpha^9 = (1010)$
$\alpha^{10} = (0111)$	$\alpha^{11} = (1110)$	$\alpha^{12} = (1111)$
$\alpha^{13} = (1101)$	$\alpha^{14} = (1001)$	$\alpha^{15} = (0001)$.

Selecting a reduction polynomial. A *trinomial* over $\mathbb{F}_2$ is a polynomial of the form $x^m + x^k + 1$, where $1 \le k \le m-1$. A *pentanomial* over $\mathbb{F}_2$ is a polynomial of the form $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, where $1 \le k_1 < k_2 < k_3 \le m-1$. ANSI X9.62 specifies the following rules for selecting the reduction polynomial for representing the elements of $\mathbb{F}_{2^m}$.

1. If there exists an irreducible trinomial of degree m over $\mathbb{F}_2$, then the reduction polynomial $f(x)$ must be an irreducible trinomial of degree m over $\mathbb{F}_2$. To maximize the chances for interoperability, ANSI X9.62 recommends that the trinomial used should be x^m+x^k+1 for the smallest possible k.
2. If there does not exist an irreducible trinomial of degree m over $\mathbb{F}_2$, then the reduction polynomial $f(x)$ must be an irreducible pentanomial of degree m over $\mathbb{F}_2$. To maximize the chances for interoperability, ANSI X9.62 recommends that the pentanomial used should be $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$ chosen according to the following criteria:[1] (1) k_3 is as small as possible;

[1] Actually, ANSI X9.62 recommends the following criteria for selecting the pentanomial: (1) k_1 is as small as possible; (2) for this particular value of k_1, k_2 is as small as possible; and (3) for these particular values of k_1 and k_2, k_3 is as small as possible. However, the ANSI X9F1 committee agreed in April 1999 to change this rec-

(2) for this particular value of k_3, k_2 is a small as possible; and (3) for these particular values of k_3 and k_2, k_1 is as small as possible.

3.2.2 Normal basis representations

A *normal basis* of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$ is a basis of the form $\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}\}$, where $\beta \in \mathbb{F}_{2^m}$. Such a basis always exists. Any element $a \in \mathbb{F}_{2^m}$ can be written as $a = \sum_{i=0}^{m-1} a_i \beta^{2^i}$, where $a_i \in \{0,1\}$. Normal basis representations have the computational advantage that squaring an element can be done very efficiently (see Field Operations below). Multiplying distinct elements, on the other hand, can be cumbersome in general. For this reason, ANSI X9.62 specifies that *Gaussian* normal bases be used, for which multiplication is both simpler and more efficient.

Gaussian normal bases (GNB). The *type* of a GNB is a positive integer measuring the complexity of the multiplication operation with respect to that basis. Generally speaking the smaller the type, the more efficient the multiplication. For a given m and T, the field $\mathbb{F}_{2^m}$ can have at most one GNB of type T. Thus it is proper to speak of *the* type T GNB of $\mathbb{F}_{2^m}$. See Mullin et al. [69] and Ash et al. [5] for further information on GNBs.

Existence of Gaussian normal bases. A GNB exists whenever m is not divisible by 8. Let m be a positive integer not divisible by 8, and let T be 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.

Field elements. If $\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}\}$ is a normal basis of $\mathbb{F}_{2^m}$ over $\mathbb{F}_2$, then the field element $a = \sum_{i=0}^{m-1} a_i \beta^{2^i}$ is represented by the binary string $(a_0 a_1 \ldots a_{m-1})$ of length m, so that

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

The multiplicative identity element (1) is represented by the bit string of all 1's, while the additive identity element (0) is represented by the bit string of all 0's.

Field operations. The following arithmetic operations are defined on the elements of $\mathbb{F}_{2^m}$ when using a GNB of type T:

- *Addition*: If $a = (a_0 a_1 \ldots a_{m-1})$ and $b = (b_0 b_1 \ldots b_{m-1})$ are elements of $\mathbb{F}_{2^m}$, then $a + b = c = (c_0 c_1 \ldots c_{m-1})$, where $c_i = (a_i + b_i) \bmod 2$. That is, field addition is performed bitwise.
- *Squaring*: Let $a = (a_0 a_1 \ldots a_{m-1}) \in \mathbb{F}_{2^m}$. Since squaring is a linear operation in $\mathbb{F}_{2^m}$,

$$a^2 = \left(\sum_{i=0}^{m-1} a_i \beta^{2^i}\right)^2 = \sum_{i=0}^{m-1} a_i \beta^{2^{i+1}} = \sum_{i=0}^{m-1} a_{i-1} \beta^{2^i} = (a_{m-1} a_0 a_1 \ldots a_{m-2}),$$

  with indices reduced modulo m. Hence squaring a field element can be accomplished by 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

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

  If $a = (a_0 a_1 \ldots a_{m-1})$ and $b = (b_0 b_1 \ldots b_{m-1})$ are elements of $\mathbb{F}_{2^m}$, then $a \cdot b = c = (c_0 c_1 \ldots c_{m-1})$, where

$$c_l = \begin{cases} \sum_{k=1}^{p-2} a_{F(k+1)+l} b_{F(p-k)+l} & \text{if } T \text{ is even,} \\ \sum_{k=1}^{m/2} (a_{k+l-1} b_{m/2+k+l-1} & \\ \quad + a_{m/2+k+l-1} b_{k+l-1}) & \\ \quad + \sum_{k=1}^{p-2} a_{F(k+1)+l} b_{F(p-k)+l} & \text{if } T \text{ is odd,} \end{cases}$$

  for each l, $0 \le l \le m-1$, where indices are reduced modulo m.
- *Inversion:* If a is a non-zero element in $\mathbb{F}_{2^m}$, the inverse of a in $\mathbb{F}_{2^m}$, denoted a^{-1}, is the unique element $c \in \mathbb{F}_{2^m}$ for which $a \cdot c = 1$.

Example 3. (*A Gaussian normal basis representation of the finite field* $\mathbb{F}_{2^4}$) For the type $T = 3$ GNB for $\mathbb{F}_{2^4}$, let $u = 9 \in \mathbb{F}_{13}$ be an element of order 3. The sequence of $F(i)$'s is:

$$F(1) = 0 F(2) = 1 F(3) = 0\ F(4) = 2\ \ F(5) = 1\ \ F(6) = 1$$
$$F(7) = 3 F(8) = 3 F(9) = 0 F(10) = 2 F(11) = 3 F(12) = 2.$$

The formulas for the product terms c_l are:

$$c_0 = a_0(b_1 + b_2 + b_3) + a_1(b_0 + b_2) + a_2(b_0 + b_1) + a_3(b_0 + b_3)$$
$$c_1 = a_1(b_2 + b_3 + b_0) + a_2(b_1 + b_3) + a_3(b_1 + b_2) + a_0(b_1 + b_0)$$
$$c_2 = a_2(b_3 + b_0 + b_1) + a_3(b_2 + b_0) + a_0(b_2 + b_3) + a_1(b_2 + b_1)$$
$$c_3 = a_3(b_0 + b_1 + b_2) + a_0(b_3 + b_1) + a_1(b_3 + b_0) + a_2(b_3 + b_2).$$

For example, if $a = (1000)$ and $b = (1101)$, then $c = a \cdot b = (0010)$.

Selecting a Gaussian normal basis. ANSI X9.62 specifies the following rules for selecting a GNB for representing the elements of $\mathbb{F}_{2^m}$ (when m is not divisible by 8).

1. If there exists a type 2 GNB of $\mathbb{F}_{2^m}$, then this basis must be used.
2. If there does not exist a type 2 GNB of $\mathbb{F}_{2^m}$ but there does exist a type 1 GNB, then the type 1 GNB must be used.

ommendation in a forthcoming revision of ANSI X9.62 to the one given above in order to be consistent with the IEEE 1363-2000 and FIPS 186-2 recommendations.

3. If neither a type 1 nor a type 2 GNB of $\mathbb{F}_{2^m}$ exists, then the GNB of the smallest type must be used.

The selection of type 2 GNBs over type 1 GNBs was somewhat arbitrary – both types of GNBs admit efficient implementation of field arithmetic. This is not a practical concern since finite fields which have both type 1 and type 2 GNBs are relatively scarce – the only such fields $\mathbb{F}_{2^m}$ with m between 160 and 600 are $\mathbb{F}_{2^{210}}$ and $\mathbb{F}_{2^{378}}$. Neither of these two fields are among those recommended by NIST (see Sect. 10.2).

4 Elliptic curves over finite fields

We give a quick introduction to the theory of elliptic curves. Chapter 6 of Koblitz's book [52] provides an introduction to elliptic curves and elliptic curve systems. For a more detailed account, consult Menezes [63] or Blake et al. [9]. Some advanced books on elliptic curves are Enge [24] and Silverman [94].

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

Let $p > 3$ be an odd prime. An *elliptic curve* E over $\mathbb{F}_p$ is defined by an equation of the form

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

where $a, b \in \mathbb{F}_p$, and $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$. The set $E(\mathbb{F}_p)$ consists of all points (x, y), $x \in \mathbb{F}_p$, $y \in \mathbb{F}_p$ that satisfy the defining equation (3), together with a special point $\mathcal{O}$ called the *point at infinity*.

Example 4. (*Elliptic curve over* $\mathbb{F}_{23}$) Let $p = 23$ and consider the elliptic curve $E : y^2 = x^3 + x + 4$ defined over $\mathbb{F}_{23}$. (In the notation of (3), we have $a = 1$ and $b = 4$.) Note that $4a^3 + 27b^2 = 4 + 432 = 436 \equiv 22 \pmod{23}$, so E is indeed an elliptic curve. The points in $E(\mathbb{F}_{23})$ are $\mathcal{O}$ and the following:

$(0, 2)$	$(0, 21)$	$(1, 11)$	$(1, 12)$	$(4, 7)$
$(4, 16)$	$(7, 3)$	$(7, 20)$	$(8, 8)$	$(8, 15)$
$(9, 11)$	$(9, 12)$	$(10, 5)$	$(10, 18)$	$(11, 9)$
$(11, 14)$	$(13, 11)$	$(13, 12)$	$(14, 5)$	$(14, 18)$
$(15, 6)$	$(15, 17)$	$(17, 9)$	$(17, 14)$	$(18, 9)$
$(18, 14)$	$(22, 5)$	$(22, 19)$.		

Addition formula. There is a rule, called the *chord-and-tangent rule*, for adding two points on an elliptic curve $E(\mathbb{F}_p)$ to give a third elliptic curve point. Together with this addition operation, the set of points $E(\mathbb{F}_p)$ forms a group with $\mathcal{O}$ serving as its identity. It is this group that is used in the construction of elliptic curve cryptosystems.

The addition rule is best explained geometrically. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two distinct points on an elliptic curve E. Then the *sum* of P and Q, denoted $R = (x_3, y_3)$, is defined as follows. First draw a line through P and Q; this line intersects the elliptic curve at a third point. Then R is the reflection of this point in the x axis. This is depicted in Fig. 1. The elliptic curve in the figure consists of two parts, the ellipse-like figure and the infinite curve.

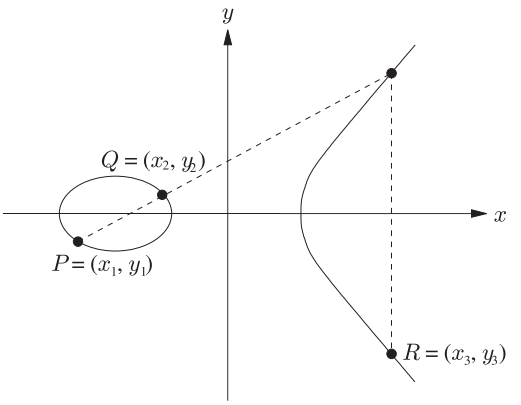

Fig. 1. Geometric description of the addition of two distinct elliptic curve points: $P + Q = R$

If $P = (x_1, y_1)$, then the *double* of P, denoted $R = (x_3, y_3)$, is defined as follows. First draw a tangent line to the elliptic curve at P. This line intersects the elliptic curve at a second point. Then R is the reflection of this point in the x axis. This is depicted in Fig. 2.

The following algebraic formulas for the sum of two points and the double of a point can now be derived from the geometric description.

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 $(x, -y)$ is denoted by $-P$, and is called the *negative* of P; observe that $-P$ is indeed a point on the curve.)
3. (Point addition) 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

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \quad \text{and}$$

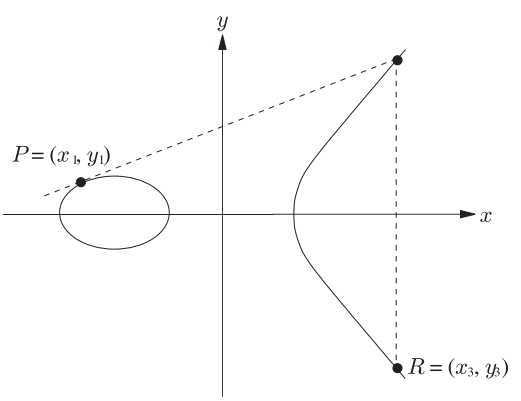

Fig. 2. Geometric description of the doubling of an elliptic curve point: $P + P = R$

$$y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1.$$

4. (Point doubling) Let $P = (x_1, y_1) \in E(\mathbb{F}_p)$, where $P \neq -P$. Then $2P = (x_3, y_3)$, where

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \quad \text{and}$$
$$y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1.$$

Observe that the addition of two elliptic curve points in $E(\mathbb{F}_p)$ requires a few arithmetic operations (addition, subtraction, multiplication, and inversion) in the underlying field $\mathbb{F}_p$.

Example 5. (*Elliptic curve addition*) Consider the elliptic curve defined in Example 4.

1. Let $P = (4, 7)$ and $Q = (13, 11)$. Then $P + Q = (x_3, y_3)$ is computed as follows:

$$x_3 = \left(\frac{11 - 7}{13 - 4}\right)^2 - 4 - 13 = 3^2 - 4 - 13$$
$$= -8 \equiv 15 \pmod{23},$$

and

$$y_3 = 3(4 - 15) - 7 = -40 \equiv 6 \pmod{23}.$$

Hence $P + Q = (15, 6)$.

2. Let $P = (4, 7)$. Then $2P = P + P = (x_3, y_3)$ is computed as follows:

$$x_3 = \left(\frac{3(4^2) + 1}{14}\right)^2 - 8 = 15^2 - 8$$
$$= 217 \equiv 10 \pmod{23},$$

and

$$y_3 = 15(4 - 10) - 7 = -97 \equiv 18 \pmod{23}.$$

Hence $2P = (10, 18)$.

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

An *elliptic curve* E over $\mathbb{F}_{2^m}$ is defined by an equation of the form

$$y^2 + xy = x^3 + ax^2 + b, \tag{4}$$

where $a, b \in \mathbb{F}_{2^m}$, and $b \neq 0$. The set $E(\mathbb{F}_{2^m})$ consists of all points (x, y), $x \in \mathbb{F}_{2^m}$, $y \in \mathbb{F}_{2^m}$ that satisfy the defining equation (4), together with a special point $\mathcal{O}$ called the *point at infinity.*

Example 6. (*Elliptic curve over* $\mathbb{F}_{2^4}$) Consider $\mathbb{F}_{2^4}$ as represented by the irreducible trinomial $f(x) = x^4 + x + 1$ (see Example 2 of Sect. 3). Consider the elliptic curve $E : y^2 + xy = x^3 + \alpha^4 x^2 + 1$ over $\mathbb{F}_{2^4}$. (In the notation of (4), we have $a = \alpha^4$ and $b = 1$.) Note that $b \neq 0$, so E is indeed an elliptic curve. The points in $E(\mathbb{F}_{2^4})$ are $\mathcal{O}$ and the following:

$(0, 1)$	$(1, \alpha^6)$	$(1, \alpha^{13})$	(α^3, α^8)	(α^3, α^{13})
(α^5, α^3)	(α^5, α^{11})	(α^6, α^8)	(α^6, α^{14})	(α^9, α^{10})
(α^9, α^{13})	(α^{10}, α)	(α^{10}, α^8)	$(\alpha^{12}, 0)$	$(\alpha^{12}, \alpha^{12})$.

Addition formula. As with elliptic curves over $\mathbb{F}_p$, there is a chord-and-tangent rule for adding points on an elliptic curve $E(\mathbb{F}_{2^m})$ to give a third elliptic curve point. Together with this addition operation, the set of points $E(\mathbb{F}_{2^m})$ forms a group with $\mathcal{O}$ serving as its identity.

The algebraic formula for the sum of two points and the double of a point are the following.

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 $(x, y) + (x, x + y) = \mathcal{O}$. (The point $(x, x + y)$ is denoted by $-P$, and is called the *negative* of P; observe that $-P$ is indeed a point on the curve.)
3. (Point addition) 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

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

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

$$x_3 = x_1^2 + \frac{b}{x_1^2} \quad \text{and} \quad y_3 = x_1^2 + \left(x_1 + \frac{y_1}{x_1}\right)x_3 + x_3.$$

Example 7. (*Elliptic curve addition*)
Consider the elliptic curve defined in Example 6.

1. Let $P = (\alpha^6, \alpha^8)$ and $Q = (\alpha^3, \alpha^{13})$. Then $P + Q = (x_3, y_3)$ is computed as follows:

$$x_3 = \left(\frac{\alpha^8 + \alpha^{13}}{\alpha^6 + \alpha^3}\right)^2 + \frac{\alpha^8 + \alpha^{13}}{\alpha^6 + \alpha^3} + \alpha^6 + \alpha^3 + \alpha^4$$
$$= \left(\frac{\alpha^3}{\alpha^2}\right)^2 + \frac{\alpha^3}{\alpha^2} + \alpha^6 + \alpha^3 + \alpha^4 = 1$$

and

$$y_3 = \left(\frac{\alpha^8 + \alpha^{13}}{\alpha^6 + \alpha^3}\right)(\alpha^6 + 1) + 1 + \alpha^8$$
$$= \left(\frac{\alpha^3}{\alpha^2}\right)(\alpha^{13}) + \alpha^2 = \alpha^{13}.$$

Hence $P + Q = (1, \alpha^{13})$.

2. Let $P = (\alpha^6, \alpha^8)$. Then $2P = P + P = (x_3, y_3)$ is computed as follows:

$$x_3 = (\alpha^6)^2 + \frac{1}{(\alpha^6)^2} = \alpha^{12} + \alpha^3 = \alpha^{10}$$

and

$$y_3 = (\alpha^6)^2 + \left(\alpha^6 + \frac{\alpha^8}{\alpha^6}\right)\alpha^{10} + \alpha^{10}$$
$$= \alpha^{12} + \alpha^{13} + \alpha^{10} = \alpha^8.$$

Hence $2P = (\alpha^{10}, \alpha^8)$.

4.3 Basic facts

Group order. Let E be an elliptic curve over a finite field $\mathbb{F}_q$. Hasse's theorem states that the number of points on an elliptic curve (including the point at infinity) is $\#E(\mathbb{F}_q) = q + 1 - t$ where $|t| \leq 2\sqrt{q}$; $\#E(\mathbb{F}_q)$ is called the *order* of E and t is called the *trace* of E. In other words, the order of an elliptic curve $E(\mathbb{F}_q)$ is roughly equal to the size q of the underlying field.

Group structure. $E(\mathbb{F}_q)$ is an abelian group of rank 1 or 2. That is, $E(\mathbb{F}_q)$ is isomorphic to $\mathbb{Z}_{n_1} \times \mathbb{Z}_{n_2}$, where n_2 divides n_1, for unique positive integers n_1 and n_2. Here, $\mathbb{Z}_n$ denotes the cyclic group of order n. Moreover, n_2 divides $q-1$. If $n_2 = 1$, then $E(\mathbb{F}_q)$ is said to be *cyclic*. In this case $E(\mathbb{F}_q)$ is isomorphic to $\mathbb{Z}_{n_1}$, and there exists a point $P \in E(\mathbb{F}_q)$ such that $E(\mathbb{F}_q) = \{kP : 0 \leq k \leq n_1 - 1\}$; such a point is called a *generator* of $E(\mathbb{F}_q)$.

Example 8. (*Cyclic elliptic curve*)
Consider the elliptic curve $E(\mathbb{F}_{23})$ defined in Example 4. Since $\#E(\mathbb{F}_{23}) = 29$, which is prime, $E(\mathbb{F}_{23})$ is cyclic and any point other than $\mathcal{O}$ is a generator of $E(\mathbb{F}_{23})$. For example, $P = (0, 2)$ is a generator, as the following shows:

$1P = (0, 2)$	$2P = (13, 12)$	$3P = (11, 9)$
$4P = (1, 12)$	$5P = (7, 20)$	$6P = (9, 11)$
$7P = (15, 6)$	$8P = (14, 5)$	$9P = (4, 7)$
$10P = (22, 5)$	$11P = (10, 5)$	$12P = (17, 9)$
$13P = (8, 15)$	$14P = (18, 9)$	$15P = (18, 14)$
$16P = (8, 8)$	$17P = (17, 14)$	$18P = (10, 18)$
$19P = (22, 18)$	$20P = (4, 16)$	$21P = (14, 18)$
$22P = (15, 17)$	$23P = (9, 12)$	$24P = (7, 3)$
$25P = (1, 11)$	$26P = (11, 14)$	$27P = (13, 11)$
$28P = (0, 21)$	$29P = \mathcal{O}$.	

5 ECDSA domain parameters

The domain parameters for ECDSA consist of a suitably chosen elliptic curve E defined over a finite field $\mathbb{F}_q$ of characteristic p, and a *base point* $G \in E(\mathbb{F}_q)$. Domain parameters may either be shared by a group of entities, or specific to a single user.

Section 5.1 describes the requirements for what constitutes "suitable" domain parameters. In Sect. 5.2, a procedure is specified for generating elliptic curves verifiably at random. Section 5.3 outlines a method for generating domain parameters, while Sect. 5.4 presents a procedure for verifying that a given set of domain parameters meets all requirements.

5.1 Domain parameters

In order to facilitate interoperability, some restrictions are placed on the underlying field size q and the representation used for the elements of $\mathbb{F}_q$. Moreover, to avoid some specific known attacks, restrictions are placed on the elliptic curve and the order of the base point.

Field requirements. The order of the underlying finite field is either $q = p$, an odd prime, or $q = 2^m$, a power of 2. In the case $q = p$, the underlying finite field is $\mathbb{F}_p$, the integers modulo p. In the case $q = 2^m$, the underlying finite field is $\mathbb{F}_{2^m}$ whose elements are represented with respect to a polynomial or a normal basis as described in Sect. 3.

Elliptic curve requirements. In order to avoid Pollard's rho [83] and the Pohlig-Hellman [81] attacks on the elliptic curve discrete logarithm problem (see Sect. 8.1), it is necessary that the number of $\mathbb{F}_q$-rational points on E be divisible by a sufficiently large prime n. ANSI X9.62 mandates that $n > 2^{160}$. Having fixed an underlying field $\mathbb{F}_q$, n should be selected to be as large as possible, i.e., one should have $n \approx q$, so $\#E(\mathbb{F}_q)$ is *almost prime*. In the remainder of this paper, we shall assume that $n > 2^{160}$ and that $n > 4\sqrt{q}$. The *co-factor* is defined to be $h = \#E(\mathbb{F}_q)/n$.

Some further precautions should be exercised when selecting the elliptic curve. To avoid the reduction algorithms of Menezes et al. [64] and Frey and Rück [29], the curve should be non-supersingular (i.e., p should not divide $(q + 1 - \#E(\mathbb{F}_q))$). More generally, one should verify that n does not divide $q^k - 1$ for all $1 \leq k \leq C$, where C is large enough so that it is computationally infeasible to find discrete logarithms in $\mathbb{F}_{q^C}$ ($C = 20$ suffices in practice [3]). Finally, to avoid the attack of Semaev [93], Smart [98], and Satoh and Araki [88] on $\mathbb{F}_q$-anomalous curves, the curve should not be $\mathbb{F}_q$-anomalous (i.e., $\#E(\mathbb{F}_q) \neq q$).

A prudent way to guard against these attacks, and similar attacks against special classes of curves that may be discovered in the future, is to select the elliptic curve E *at random* subject to the condition that $\#E(\mathbb{F}_q)$ is divisible by a large prime – the probability that a random curve succumbs to these special-purpose attacks is negligible. A curve can be selected *verifiably at random* by choosing the coefficients of the defining elliptic curve equation as the outputs of a one-way function such as SHA-1 according to some pre-specified procedure. A procedure for accomplishing this, similar in spirit to the method given in FIPS 186 [70] for selecting DSA primes verifiably at random, is described in Sect. 5.2.

Summary. To summarize, domain parameters are comprised of:

1. a field size q, where either $q = p$, an odd prime, or $q = 2^m$;
2. an indication FR (*field representation*) of the representation used for the elements of $\mathbb{F}_q$;
3. (optional) a bit string `seedE` of length at least 160 bits, if the elliptic curve was generated in accordance with the method described in Sect. 5.2;
4. two field elements a and b in $\mathbb{F}_q$ which define 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 $y^2 + xy = x^3 + ax^2 + b$ in the case $p = 2$);
5. two field elements x_G and y_G in $\mathbb{F}_q$ which define a finite point $G = (x_G, y_G)$ of prime order in $E(\mathbb{F}_q)$;
6. the order n of the point G, with $n > 2^{160}$ and $n > 4\sqrt{q}$; and
7. the cofactor $h = \#E(\mathbb{F}_q)/n$.

5.2 Generating an elliptic curve verifiably at random

This subsection describes the method that is used for generating an elliptic curve *verifiably* at random. The defining parameters of the elliptic curve are defined to be outputs of the one-way hash function SHA-1 (as specified in FIPS 180-1 [71]). The input seed to SHA-1 then serves as proof (under the assumption that SHA-1 cannot be inverted) that the elliptic curve was indeed generated at random. This provides some assurance to the user of the elliptic curve that the entity who generated the elliptic curve did not intentionally construct a "weak" curve which the entity could subsequently exploit to recover the user's private keys. Use of this generation method can also help mitigate concerns regarding the possible future discovery of new and rare classes of weak elliptic curves, as such rare curves would essentially never be generated.

5.2.1 The case $q = p$

The following notation is used: $t = \lceil \log_2 p \rceil$, $s = \lfloor (t-1)/160 \rfloor$, and $v = t - 160 \cdot s$.

ALGORITHM 1: GENERATING A RANDOM ELLIPTIC CURVE OVER $\mathbb{F}_p$.
INPUT: A field size p, where p is an odd prime.
OUTPUT: A bit string `seedE` of length at least 160 bits and field elements $a, b \in \mathbb{F}_p$ that define an elliptic curve E over $\mathbb{F}_p$.

1. Choose an arbitrary bit string `seedE` of length $g \geq 160$ bits.
2. Compute $H = \text{SHA-1}(\texttt{seedE})$, and let c_0 denote the bit string of length v bits obtained by taking the v rightmost bits of H.
3. Let W_0 denote the bit string of length v bits obtained by setting the leftmost bit of c_0 to 0. (This ensures that $r < p$.)
4. Let z be the integer whose binary expansion is given by the g-bit string `seedE`.
5. For i from 1 to s do:
   4.1. Let s_i be the g-bit string which is the binary expansion of the integer $(z + i) \bmod 2^g$.
   4.2. Compute $W_i = \text{SHA-1}(s_i)$.
6. Let W be the bit string obtained by concatenating $W_0, W_1, \ldots, W_s$ as follows: $W = W_0 \| W_1 \| \cdots \| W_s$.
7. Let r be the integer whose binary expansion is given by W.
8. If $r = 0$ or if $4r + 27 \equiv 0 \pmod p$ then go to step 1.
9. Choose arbitrary integers $a, b \in \mathbb{F}_p$, not both 0, such that $r \cdot b^2 \equiv a^3 \bmod p$. (For example, one may take $a = r$ and $b = r$.)
10. The elliptic curve chosen over $\mathbb{F}_p$ is $E : y^2 = x^3 + ax + b$.
11. Output(`seedE`, a, b).

Isomorphism classes of elliptic curves over $\mathbb{F}_p$. Two elliptic curves $E_1 : y^2 = x^3 + a_1 x + b_1$ and $E_2 : y^2 = x^3 + a_2 x + b_2$ defined over $\mathbb{F}_p$ are *isomorphic* over $\mathbb{F}_p$ if and only if there exists $u \in \mathbb{F}_p$, $u \neq 0$, such that $a_1 = u^4 a_2$ and $b_1 = u^6 b_2$. (Isomorphic elliptic curves are essentially the same. In particular, if E_1 is isomorphic to E_2, then the groups $E_1(\mathbb{F}_p)$ and $E_2(\mathbb{F}_p)$ are isomorphic as abelian groups.) Observe that if E_1 and E_2 are isomorphic and $b_1 \neq 0$ (so $b_2 \neq 0$), then $\frac{a_1^3}{b_1^2} = \frac{a_2^3}{b_2^2}$. The singular elliptic curves, i.e., the curves $E : y^2 = x^3 + ax + b$ for which $4a^3 + 27b^2 \equiv 0 \pmod p$ are precisely those which either have $a = 0$ and $b = 0$, or $\frac{a^3}{b^2} = -\frac{27}{4}$. If $r \in \mathbb{F}_p$, $r \neq 0$, $r \neq -\frac{27}{4}$, then there are precisely two isomorphism classes of curves $E : y^2 = x^3 + ax + b$ with $\frac{a^3}{b^2} \equiv r \pmod p$. Hence, there are essentially only two choices for (a, b) in step 9 of Algorithm 1. The conditions $r \neq 0$ and $r \neq -\frac{27}{4}$ imposed in step 8 ensure the exclusion of singular elliptic curves. Finally, we mention that this method of generating curves will never produce the elliptic curves with $a = 0$, $b \neq 0$, nor the elliptic curves with $a \neq 0$, $b = 0$. This is not a concern because such curves constitute a negligible fraction of all elliptic curves, and therefore are unlikely to ever be generated by any method which selects an elliptic curve uniformly at random.

The twist of an elliptic curve over $\mathbb{F}_p$. The non-isomorphic elliptic curves $E_1 : y^2 = x^3 + ax + b$ and $E_2 : y^2 = x^3 + ac^2 x^2 + bc^3$, where $c \in \mathbb{F}_p$ is a quadratic non-residue modulo p, are said to be *twists* of each other. Note that both these curves have the same r value. Their orders are related by the equation $\#E_1(\mathbb{F}_p) + \#E_2(\mathbb{F}_p) = 2p + 2$. Thus, if one is able to compute $\#E_1(\mathbb{F}_p)$, then one can easily deduce $\#E_2(\mathbb{F}_p)$.

ALGORITHM 2: VERIFYING THAT AN ELLIPTIC CURVE WAS RANDOMLY GENERATED OVER $\mathbb{F}_p$.
INPUT: A field size p (a prime), a bit string `seedE` of length $g \geq 160$ bits, and field elements $a, b \in \mathbb{F}_p$ that define an elliptic curve $E : y^2 = x^3 + ax + b$ over $\mathbb{F}_p$.
OUTPUT: Acceptance or rejection that E was randomly generated using Algorithm 1.

1. Compute $H = \text{SHA-1}(\texttt{seedE})$, and let c_0 denote the bit string of length v bits obtained by taking the v rightmost bits of H.
2. Let W_0 denote the bit string of length v bits obtained by setting the leftmost bit of c_0 to 0.
3. Let z be the integer whose binary expansion is given by the g-bit string seedE.
4. For i from 1 to s do:
   4.1. Let s_i be the g-bit string which is the binary expansion of the integer $(z+i) \bmod 2^g$.
   4.2. Compute $W_i = \text{SHA-1}(s_i)$.
5. Let W be the bit string obtained by concatenating $W_0, W_1, \ldots, W_s$ as follows: $W' = W_0 \,\|\, W_1 \,\|\, \cdots \,\|\, W_s$.
6. Let r' be the integer whose binary expansion is given by W'.
7. If $r' \cdot b^2 \equiv a^3 \pmod p$ then accept; otherwise reject.

5.2.2 The case $q = 2^m$

The following notation is used: $s = \lfloor (m-1)/160 \rfloor$ and $v = m - 160 \cdot s$.

ALGORITHM 3: GENERATING A RANDOM ELLIPTIC CURVE OVER $\mathbb{F}_{2^m}$.

INPUT: A field size $q = 2^m$.

OUTPUT: A bit string seedE with a length of at least 160 bits and field elements $a, b \in \mathbb{F}_{2^m}$, which define an elliptic curve E over $\mathbb{F}_{2^m}$.

1. Choose an arbitrary bit string seedE of length $g \geq 160$ bits.
2. Compute $H = \text{SHA-1}(\texttt{seedE})$, and let b_0 denote the bit string of length v bits obtained by taking the v rightmost bits of H.
3. Let z be the integer whose binary expansion is given by the g-bit string seedE.
4. For i from 1 to s do:
   4.1. Let s_i be the g-bit string which is the binary expansion of the integer $(z+i) \bmod 2^g$.
   4.2. Compute $b_i = \text{SHA-1}(s_i)$.
5. Let b be the field element obtained by concatenating $b_0, b_1, \ldots, b_s$ as follows: $b = b_0 \,\|\, b_1 \,\|\, \cdots \,\|\, b_s$.
6. If $b = 0$ then go to step 1.
7. Let a be an arbitrary element of $\mathbb{F}_{2^m}$.
8. The elliptic curve chosen over $\mathbb{F}_{2^m}$ is $E : y^2 + xy = x^3 + ax^2 + b$.
9. Output(seedE, a, b).

Isomorphism classes of elliptic curves over $\mathbb{F}_{2^m}$. Two elliptic curves $E_1 : y^2 + xy = x^3 + a_1x^2 + b_1$ and $E_2 : y^2 + xy = x^3 + a_2x^2 + b_2$ defined over $\mathbb{F}_{2^m}$ are *isomorphic* over $\mathbb{F}_{2^m}$ if and only if $b_1 = b_2$ and $\text{Tr}(a_1) = \text{Tr}(a_2)$, where Tr is the trace function $\text{Tr} : \mathbb{F}_{2^m} \longrightarrow \mathbb{F}_2$ defined by $\text{Tr}(\alpha) = \alpha + \alpha^2 + \alpha^{2^2} + \cdots + \alpha^{2^{m-1}}$. (Isomorphic elliptic curves are essentially the same. In particular, if E_1 is isomorphic to E_2, then the groups $E_1(\mathbb{F}_{2^m})$ and $E_2(\mathbb{F}_{2^m})$ are isomorphic as abelian groups.) It follows that a set of representatives of the isomorphism classes of elliptic curves over $\mathbb{F}_{2^m}$ is $\{y^2 + xy = x^3 + ax^2 + b \mid b \in \mathbb{F}_{2^m}, b \neq 0, a \in \{0, \gamma\}\}$, where $\gamma \in \mathbb{F}_{2^m}$ is a fixed element with $\text{Tr}(\gamma) = 1$ (if m is odd, we can take $\gamma = 1$). Hence, having selected b, there are essentially only two choices for a in step 7 of Algorithm 3.

The twist of an elliptic curve over $\mathbb{F}_{2^m}$. The non-isomorphic elliptic curves $E_1 : y^2 + xy = x^3 + a_1x^2 + b$ and $E_2 : y^2 + xy = x^3 + a_2x^2 + b$ where $\text{Tr}(a_1) \neq \text{Tr}(a_2)$ are said to be *twists* of each other. Their orders are related by the equation $\#E_1(\mathbb{F}_{2^m}) + \#E_2(\mathbb{F}_{2^m}) = 2^{m+1} + 2$. Thus, if one is able to compute $\#E_1(\mathbb{F}_{2^m})$, then one can easily deduce $\#E_2(\mathbb{F}_{2^m})$. The order of an elliptic curve over $\mathbb{F}_{2^m}$ is always even. Furthermore, $\#E_1(\mathbb{F}_{2^m}) \equiv 0 \pmod 4$ if $\text{Tr}(a_1) = 0$, and $\#E_1(\mathbb{F}_{2^m}) \equiv 2 \pmod 4$ if $\text{Tr}(a_1) = 1$.

ALGORITHM 4: VERIFYING THAT AN ELLIPTIC CURVE WAS RANDOMLY GENERATED OVER $\mathbb{F}_{2^m}$.

INPUT: A field size $q = 2^m$, a bit string seedE of length $g \geq 160$ bits, and field elements $a, b \in \mathbb{F}_{2^m}$, which define an elliptic curve $E : y^2 + xy = x^3 + ax^2 + b$ over $\mathbb{F}_{2^m}$.

OUTPUT: Acceptance or rejection that E was randomly generated using Algorithm 3.

1. Compute $H = \text{SHA-1}(\texttt{seedE})$, and let b_0 denote the bit string of length v bits obtained by taking the v rightmost bits of H.
2. Let z be the integer whose binary expansion is given by the g-bit string seedE.
3. For i from 1 to s do:
   4.1. Let s_i be the g-bit string which is the binary expansion of the integer $(z+i) \bmod 2^g$.
   4.2. Compute $b_i = \text{SHA-1}(s_i)$.
4. Let b' be the field element obtained by concatenating $b_0, b_1, \ldots, b_s$ as follows: $b' = b_0 \,\|\, b_1 \,\|\, \cdots \,\|\, b_s$.
5. If $b = b'$ then accept; otherwise reject.

5.3 Domain parameter generation

The following is one way to generate cryptographically secure domain parameters:

1. Select coefficients a and b from $\mathbb{F}_q$ verifiably at random using Algorithm 1 or Algorithm 3. Let E be the curve $y^2 = x^3 + ax + b$ in the case $q = p$, and $y^2 + xy = x^3 + ax^2 + b$ in the case $q = 2^m$.
2. Compute $N = \#E(\mathbb{F}_q)$.
3. Verify that N is divisible by a large prime n ($n > 2^{160}$ and $n > 4\sqrt{q}$). If not, then go to step 1.
4. Verify that n does not divide $q^k - 1$ for each k, $1 \leq k \leq 20$. If not, then go to step 1.
5. Verify that $n \neq q$. If not, then go to step 1.
6. Select an arbitrary point $G' \in E(\mathbb{F}_q)$ and set $G = (N/n)G'$. Repeat until $G \neq \mathcal{O}$.

Point counting. In 1985 Schoof [91] presented a polynomial-time algorithm for computing $\#E(\mathbb{F}_q)$, the number of points on an elliptic curve over $\mathbb{F}_q$ in the case when q is odd; the algorithm was later extended to the case

of $q = 2^m$ by Koblitz [50]. Schoof's algorithm is rather inefficient in practice for the values of q which are of practical interest (i.e., $q > 2^{160}$). In the last few years a lot of work has been done on improving and refining Schoof's algorithm, now called the Schoof–Elkies–Atkin (SEA) algorithm; for example, see Lercier and Morain [58] and Lercier [56]. With these improvements, cryptographically suitable elliptic curves over fields whose orders are as large as 2^{200} can be randomly generated in a few hours on a workstation (see Lercier [57] and Izu et al. [44]). More recently, Satoh [87] (see also M. Fouquet et al. [26]) presented a new algorithm for point counting over binary fields that is superior to the SEA algorithm. With Satoh's algorithm, the number of points on an elliptic curve over $\mathbb{F}_{2^m}$ for $m \approx 200$ can be determined in only a few seconds on a fast PC.

The complex multiplication (CM) method. Another method for generating cryptographically suitable elliptic curves is the CM method. Over $\mathbb{F}_p$ the CM method is also called the Atkin–Morain method [68]; over $\mathbb{F}_{2^m}$ it is also called the Lay–Zimmer method [55]. A detailed description of the CM method can be found in IEEE 1363-2000 [39].

Let E be an elliptic curve over $\mathbb{F}_q$ of order N. Let $Z = 4q - (q+1-N)^2$ and write $Z = DV^2$ where D is a squarefree integer. Then E is said to have *complex multiplication* by D. If one knows D for a given curve, then one can efficiently compute the order of the curve.

The CM method first finds a D for which there exists an elliptic curve E over $\mathbb{F}_q$ with complex multiplication by D and having nearly prime order $N = nh$ (where n is prime), and furthermore where $n \neq q$ and n does not divide $q^k - 1$ for each $1 \leq k \leq 20$. It then constructs the coefficients of E. The CM method is only efficient for small D, in which case it is much faster than Schoof's algorithm. Thus, a potential drawback of the CM method is that it can only be used to generate elliptic curves having complex multiplication by small D.

Koblitz curves. These curves, also known as anomalous binary curves, were first proposed for cryptographic use by Koblitz [51]. They are elliptic curves over $\mathbb{F}_{2^m}$ whose defining equations have coefficients in $\mathbb{F}_2$. Thus, there are two Koblitz curves over $\mathbb{F}_{2^m}$: $y^2 + xy = x^3 + 1$ and $y^2 + xy = x^3 + x^2 + 1$. Solinas [100, 102], building on the earlier work of Meier and Staffelbach [62], showed how one can compute kP very efficiently for arbitrary k where P is a point on a Koblitz curve. Since performing such scalar multiplications is the dominant computational step in ECDSA signature generation and verification (see Sect. 7), Koblitz curves are very attractive for use in the ECDSA.

5.4 Domain parameter validation

Domain parameter validation ensures that the domain parameters have the requisite arithmetical properties. Reasons for performing domain parameter validation in practice include: (1) prevention of malicious insertion of invalid domain parameters which may enable some attacks; and (2) detection of inadvertent coding or transmission errors. Use of an invalid set of domain parameters can void all expected security properties.

An example of a concrete (albeit far-fetched) attack that can be launched if domain parameter validation for a signature scheme is not performed was demonstrated by Blake-Wilson and Menezes [11]. The attack is on a key agreement protocol which employs the ElGamal signature scheme.

Methods for validating domain parameters. The assurance that a set $D = (q, \mathrm{FR}, a, b, G, n, h)$ of EC domain parameters is valid can be provided to an entity using one of the following methods:

1. A performs explicit domain parameter validation using Algorithm 5 (shown below).
2. A generates D itself using a trusted system.
3. A receives assurance from a trusted party T (e.g., a certification authority) that T has performed explicit domain parameter validation of D using Algorithm 5.
4. A receives assurance from a trusted party T that D was generated using a trusted system.

ALGORITHM 5: EXPLICIT VALIDATION OF A SET OF EC DOMAIN PARAMETERS.
INPUT: A set of EC domain parameters $D = (q, \mathrm{FR}, a, b, G, n, h)$.
OUTPUT: Acceptance or rejection of the validity of D.

1. Verify that q is an odd prime ($q = p$) or a power of 2 ($q = 2^m$).
2. Verify that FR is a "valid" representation for $\mathbb{F}_q$.
3. Verify that $G \neq \mathcal{O}$.
4. Verify that a, b, x_G, and y_G are properly represented elements of $\mathbb{F}_q$ (i.e., integers in the interval $[0, p-1]$ in the case $q = p$, and bit strings of length m bits in the case $q = 2^m$).
5. (Optional) If the elliptic curve was randomly generated in accordance with Algorithm 1 or Algorithm 3 of Sect. 5.2, verify that `seedE` is a bit string with a length of at least 160 bits and use Algorithm 2 or Algorithm 4 to verify that a and b were suitably derived from `seedE`.
6. Verify that a and b define an elliptic curve over $\mathbb{F}_q$ (i.e., $4a^3 + 27b^2 \not\equiv 0 \pmod{p}$ if $q = p$; $b \neq 0$ if $q = 2^m$).
7. Verify that G lies on the elliptic curve defined by a and b (i.e., $y_G^2 = x_G^3 + ax_G + b$ in the case $q = p$, and $y_G^2 + x_G y_G = x_G^3 + ax_G^2 + b$ in the case $q = 2^m$).
8. Verify that n is prime.
9. Verify that $n > 2^{160}$ and that $n > 4\sqrt{q}$.
10. Verify that $nG = \mathcal{O}$.
11. Compute $h' = \lfloor (\sqrt{q}+1)^2/n \rfloor$ and verify that $h = h'$.
12. Verify that n does not divide $q^k - 1$ for each k, $1 \leq k \leq 20$.

13. Verify that $n \neq q$.
14. If any verification fails, then D is *invalid*; otherwise D is *valid*.

Verifying the order of an elliptic curve. Recall that by Hasse's theorem, $(\sqrt{q}-1)^2 \leq \#E(\mathbb{F}_q) \leq (\sqrt{q}+1)^2$. Hence $n > 4\sqrt{q}$ implies that n^2 does not divide $\#E(\mathbb{F}_q)$, and thus $E(\mathbb{F}_q)$ has a unique subgroup of order n. Also, since $(\sqrt{q}+1)^2 - (\sqrt{q}-1)^2 = 4\sqrt{q}$, there is a unique integer h such that $q+1-2\sqrt{q} \leq nh \leq q+1+2\sqrt{q}$, namely $h = \lfloor (\sqrt{q}+1)^2/n \rfloor$. Thus steps 9, 10, and 11 of Algorithm 5 verify that $\#E(\mathbb{F}_q)$ is indeed equal to nh.

As noted in Sect. 5.2, counting the number of points on a randomly generated elliptic curve is a complicated and cumbersome task. In practice, one may buy software from a vendor to perform the point counting. We note that since the alleged order of an elliptic curve can be efficiently verified with 100% certainty, such software does not have to be trusted.

6 ECDSA key pairs

An ECDSA key pair is associated with a particular set of EC domain parameters. The public key is a random multiple of the base point, while the private key is the integer used to generate the multiple. Section 6.1 summarizes the procedure for key pair generation. Section 6.2 presents a procedure for verifying that a given public key meets all requirements. Section 6.3 discusses the importance of proving possession of a private key corresponding to a public key to a certification authority (CA) when the public key is being certified by the CA.

6.1 Key pair generation

An entity A's key pair is associated with a particular set of EC domain parameters $D = (q, \mathrm{FR}, a, b, G, n, h)$. This association can be assured cryptographically (e.g., with certificates) or by context (e.g., all entities use the same domain parameters). The entity A must have the assurance that the domain parameters are valid (see Sect. 5.4) prior to key generation.

ECDSA key pair generation. Each entity A does the following:

1. Select a random or pseudorandom 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.

6.2 Public key validation

Public key validation, as first enunciated by Johnson [46], ensures that a public key has the requisite arithmetical properties. Successful execution of this routine demonstrates that an associated private key logically exists, although it does not demonstrate that someone has actually computed the private key nor that the claimed owner actually possesses the private key. Reasons for performing public key validation in practice include: (1) prevention of malicious insertion of an invalid public key that may enable some attacks; and (2) detection of inadvertent coding or transmission errors. Use of an invalid public key can void all expected security properties.

An example of a concrete attack that can be launched if public key validation is not performed was demonstrated by Lim and Lee [60]. The attack is on a Diffie–Hellman-based key agreement protocol.

Methods for validating public keys. The assurance that a public key Q is valid can be provided to an entity A using one of the following methods:

1. A performs explicit public key validation using Algorithm 6 (shown below).
2. A generates Q itself using a trusted system.
3. A receives assurance from a trusted party T (e.g., a certification authority) that T has performed explicit public key validation of A using Algorithm 6.
4. A receives assurance from a trusted party T that Q was generated using a trusted system.

Algorithm 6: Explicit validation of an ECDSA public key.
Input: A public key $Q = (x_Q, y_Q)$ associated with valid domain parameters $(q, \mathrm{FR}, a, b, G, n, h)$.
Output: Acceptance or rejection of the validity of Q.

1. Check that $Q \neq \mathcal{O}$.
2. Check that x_Q and y_Q are properly represented elements of $\mathbb{F}_q$ (i.e., integers in the interval $[0, p-1]$ in the case $q = p$, and bit strings of length m bits in the case $q = 2^m$).
3. Check that Q lies on the elliptic curve defined by a and b.
4. Check that $nQ = \mathcal{O}$.
5. If any check fails, then Q is *invalid*; otherwise Q is *valid*.

6.3 Proof of possession of a private key

If an entity C is able to certify A's public key Q as its own public key, then C can claim that A's signed messages originated from C. To avoid this, the CA should require all entities A to prove possession of the private keys corresponding to its public keys before the CA certifies the public key as belonging to A. This proof of possession can be accomplished by a variety of means, for example by requiring A to sign a message of the CA's choice, or by using zero-knowledge techniques (see Chaum et al. [19]). Note that proof of possession of a private key provides different assurances than from public key validation. The former demonstrates possession of a private key even though it

may correspond to an invalid public key, while the latter demonstrates validity of a public key but not ownership of the corresponding private key. Doing both provides a high level of assurance.

7 ECDSA signature generation and verification

This section describes the procedures for generating and verifying signatures using the ECDSA.

ECDSA signature generation. To sign a message m, an entity A with domain parameters $D = (q, \mathrm{FR}, a, b, G, n, h)$ and associated key pair (d, Q) does the following:

1. Select a random or pseudorandom integer k, $1 \leq k \leq n-1$.
2. Compute $kG = (x_1, y_1)$ and convert x_1 to an integer $\overline{x}_1$.
3. Compute $r = x_1 \bmod n$. If $r = 0$ then go to step 1.
4. Compute $k^{-1} \bmod n$.
5. Compute SHA-1(m) and convert this bit string to an integer e.
6. Compute $s = k^{-1}(e + dr) \bmod n$. If $s = 0$ then go to step 1.
7. A's signature for the message m is (r, s).

ECDSA signature verification. To verify A's signature (r, s) on m, B obtains an authentic copy of A's domain parameters $D = (q, \mathrm{FR}, a, b, G, n, h)$ and associated public key Q. It is recommended that B also validates D and Q (see Sects. 5.4 and 6.2). B then does the following:

1. Verify that r and s are integers in the interval $[1, n-1]$.
2. Compute SHA-1(m) and convert this bit string to an integer e.
3. Compute $w = s^{-1} \bmod n$.
4. Compute $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$.
5. Compute $X = u_1 G + u_2 Q$.
6. If $X = \mathcal{O}$, then reject the signature. Otherwise, convert the x coordinate x_1 of X to an integer $\overline{x}_1$, and compute $v = \overline{x}_1 \bmod n$.
7. Accept the signature if and only if $v = r$.

Proof that signature verification works. If a signature (r, s) on a message m was indeed generated by A, then $s = k^{-1}(e + dr) \bmod n$. Rearranging gives

$$k \equiv s^{-1}(e + dr) \equiv s^{-1}e + s^{-1}rd \equiv we + wrd \equiv u_1 + u_2 d \pmod{n}.$$

Thus $u_1 G + u_2 Q = (u_1 + u_2 d)G = kG$, and so $v = r$ as required.

Conversion between data types. ANSI X9.62 specifies a method for converting field elements to integers. This is used to convert the field element x_1 to an integer in step 2 of signature generation and step 6 of signature verification prior to computing $x_1 \bmod n$. ANSI X9.62 also specifies a method for converting bit strings to integers. This is used to convert the output e of SHA-1 to an integer prior to its use in the modular computation in step 5 of signature generation and step 2 of signature verification.

Public-key certificates. Before verifying A's signature on a message, B needs to obtain an authentic copy of A's domain parameters D and associated public key Q. ANSI X9.62 does not specify a mechanism for achieving this. In practice, authentic public keys are most commonly distributed via certificates. A's public-key certificate should include a string of information that uniquely identifies A (such as A's name and address), her domain parameters D (if these are not already known from context), her public key Q, and a CA's signature over this information. B can then use his authentic copy of the CA's public key to verify A's certificate, thereby obtaining an authentic copy of A's static public key.

Rationale for checks on r and s in signature verification. Step 1 of signature verification checks that r and s are integers in the interval $[1, n-1]$. These checks can be performed very efficiently and are prudent measures in light of known attacks on related ElGamal signature schemes that do not perform these checks (for examples of such attacks, see Bleichenbacher [12]). The following is a plausible attack on ECDSA if the check $r \neq 0$ (and, more generally, $r \not\equiv 0 \pmod{n}$) is not performed. Suppose that A is using the elliptic curve $y^2 = x^3 + ax + b$ over $\mathbb{F}_p$, where b is a quadratic residue modulo p, and suppose that A uses a base point $G = (0, \sqrt{b})$ of prime order n. (It is plausible that all entities select a base point with 0 x coordinate in order to minimize the size of domain parameters.) An adversary can now forge A's signature on any message m of its choice by computing $e = \text{SHA-1}(m)$. It can easily be checked that $(r = 0, s = e)$ is a valid signature for m.

Comparing DSA and ECDSA. Conceptually, the ECDSA is simply obtained from the DSA by replacing the subgroup of order q of $\mathbb{Z}_p^*$ generated by g with the subgroup of points on an elliptic curve that are generated by G. The only significant difference between ECDSA and DSA is in the generation of r. The DSA does this by taking the random element $X = g^k \bmod p$ and reducing it modulo q, thus obtaining an integer in the interval $[1, q-1]$. The ECDSA generates r in the interval $[1, n-1]$ by taking the x coordinate of the random point kG and reducing it modulo n.

8 Security considerations

The security objective of ECDSA is to be existentially unforgeable against a chosen-message attack. The goal of an adversary who launches such an attack against a legitimate entity A is to obtain a valid signature on a single

message m, after having obtained A's signature on a collection of messages (not including m) of the adversary's choice.

Some progress has been made in proving the security of ECDSA, albeit mostly in strong theoretical models. Slight variants of DSA and ECDSA (but not ECDSA itself) have been proven to be existentially unforgeable against chosen-message attack by Pointcheval and Stern [82] (see also [14]) under the assumptions that the discrete logarithm problem is hard and that the hash function employed is a random function. ECDSA itself has been proven secure by Brown [15] under the assumption that the underlying group is a generic group and that the hash function employed is collision resistant.

The possible attacks on ECDSA can be classified as follows:

1. Attacks on the elliptic curve discrete logarithm problem.
2. Attacks on the hash function employed.
3. Other attacks.

This section summarizes the current knowledge of these attacks and how they can be avoided in practice.

8.1 The elliptic curve discrete logarithm problem

One way in which an adversary can succeed is to compute A's private key d from A's domain parameters $(q, \mathrm{FR}, a, b, G, n, h)$ and public key Q. The adversary can subsequently forge A's signature on any message of its choice.

Problem definition. The *elliptic curve discrete logarithm problem (ECDLP)* is: given an elliptic curve E defined over a finite field $\mathbb{F}_q$, a point $P \in E(\mathbb{F}_q)$ of order n, and a point $Q = lP$ where $0 \le l \le n-1$, determine l.

8.1.1 Known attacks

This subsection overviews the algorithms known for solving the ECDLP and discusses how they can be avoided in practice.

1. *Naive exhaustive search.* In this method, one simply computes successive multiples of P: $P, 2P, 3P, 4P, \ldots$ until Q is obtained. This method can take up to n steps in the worst case.
2. *Pohlig–Hellman algorithm.* This algorithm, due to Pohlig and Hellman [81], exploits the factorization of n, the order of the point P. The algorithm reduces the problem of recovering l to the problem of recovering l modulo each of the prime factors of n; the desired number l can then be recovered by using the Chinese remainder theorem.
   The implications of this algorithm are the following. To construct the most difficult instance of the ECDLP, one must select an elliptic curve whose order is divisible by a large prime n. Preferably, this order should be a prime or almost a prime (i.e., a large prime n times a small integer h). For the remainder of this section, we shall assume that the order n of P is prime.
3. *Baby-step giant-step algorithm.* This algorithm is a time-memory trade-off of the method of exhaustive search. It requires storage for about $\sqrt{n}$ points, and its running time is roughly $\sqrt{n}$ steps in the worst case.
4. *Pollard's rho algorithm.* This algorithm, due to Pollard [83], is a randomized version of the baby-step giant-step algorithm. It has roughly the same expected running time ($\sqrt{\pi n/2}$ steps) as the baby-step giant-step algorithm, but is superior in that it requires a negligible amount of storage.
   Gallant et al. [31] and Wiener and Zuccherato [111] showed how Pollard's rho algorithm can be sped up by a factor of $\sqrt{2}$. Thus the expected running time of Pollard's rho method with this speedup is $(\sqrt{\pi n})/2$ steps.
5. *Parallelized Pollard's rho algorithm.* Van Oorschot and Wiener [80] showed how Pollard's rho algorithm can be parallelized so that when the algorithm is run in parallel on r processors, the expected running time of the algorithm is roughly $(\sqrt{\pi n})/(2r)$ steps. That is, using r processors results in an r-fold speedup.
6. *Pollard's lambda method.* This is another randomized algorithm due to Pollard [83]. Like Pollard's rho method, the lambda method can also be parallelized with a linear speedup. The parallelized lambda method is slightly slower than the parallelized rho method [80]. The lambda method is, however, faster in situations when the logarithm being sought is known to lie in a subinterval $[0, b]$ of $[0, n-1]$, where $b < 0.39n$ [80].
7. *Multiple logarithms.* R. Silverman and Stapleton [97] observed that if a single instance of the ECDLP (for a given elliptic curve E and base point P) is solved using (parallelized) Pollard's rho method, then the work done in solving this instance can be used to speed up the solution of other instances of the ECDLP (for the same curve E and base point P). More precisely, if the first instance takes an expected time t, then the second instance takes an expected time $(\sqrt{2}-1)t \approx 0.41t$. Having solved these two instances, the third instance takes an expected time $(\sqrt{3}-\sqrt{2})t \approx 0.32t$. Having solved these three instances, the fourth instance takes an expected time $(\sqrt{4}-\sqrt{3})t \approx 0.27t$, and so on. Thus subsequent instances of the ECDLP for a particular elliptic curve become progressively easier. Another way of looking at this is that solving k instances of the ECDLP (for the same curve E and base point P) takes only $\sqrt{k}$ as much work as it does to solve one instance of the ECDLP. This analysis does not take into account storage requirements.
   Concerns that successive logarithms become easier can be addressed by ensuring that the elliptic param-

eters are chosen so that the first instance is infeasible to solve.

8. *Supersingular elliptic curves.* Menezes et al. [63, 64] and Frey and Rück [29] showed how, under mild assumptions, the ECDLP in an elliptic curve E defined over a finite field $\mathbb{F}_q$ can be reduced to the ordinary DLP in the multiplicative group of some extension field $\mathbb{F}_{q^k}$ for some $k \geq 1$, where the number field sieve algorithm applies. The reduction algorithm is only practical if k is small – this is not the case for most elliptic curves, as shown by Balasubramanian and Koblitz [6]. To ensure that the reduction algorithm does not apply to a particular curve, one only needs to check that n, the order of the point P, does not divide $q^k - 1$ for all small k for which the DLP in $\mathbb{F}_{q^k}$ is tractable – in practice, when $n > 2^{160}$ then $1 \leq k \leq 20$ suffices [3].
   An elliptic curve E over $\mathbb{F}_q$ is said to be *supersingular* if the trace t of E is divisible by the characteristic p of $\mathbb{F}_q$. For this very special class of elliptic curves, it is known that $k \leq 6$. It follows that the reduction algorithm yields a subexponential-time algorithm for the ECDLP in supersingular curves. For this reason, supersingular curves are explicitly excluded from use in the ECDSA by the above divisibility check.
   More generally, the divisibility check rules out all elliptic curves for which the ECDLP can be efficiently reduced to the DLP in some small extension of $\mathbb{F}_q$. These include the supersingular elliptic curves and elliptic curves of trace 2 (elliptic curves E over $\mathbb{F}_q$ for which $\#E(\mathbb{F}_q) = q - 1$).
9. *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 [98], and Satoh and Araki [88] showed how to efficiently solve the ECDLP for these curves. The attack does not extend to any other classes of elliptic curves. Consequently, by verifying that the number of points on an elliptic curve is not equal to the cardinality of the underlying field, one can easily ensure that the Semaev–Smart–Satoh–Araki attack does not apply.
10. *Curves defined over a small field.* Suppose that E is an elliptic curve defined over the finite field $\mathbb{F}_{2^e}$. Gallant et al. [31], and Wiener and Zuccherato [111] showed how Pollard's rho algorithm for computing elliptic curve logarithms in $E(\mathbb{F}_{2^{ed}})$ can be further sped up by a factor of $\sqrt{d}$ – thus the expected running time of Pollard's rho method for these curves is $(\sqrt{\pi n/d})/2$ steps. For example, if E is a Koblitz curve (see Sect. 5.3), then Pollard's rho algorithm for computing elliptic curve logarithms in $E(\mathbb{F}_{2^m})$ can be sped up by a factor of $\sqrt{m}$. This speedup should be considered when doing a security analysis of elliptic curves whose coefficients lie in a small subfield.
11. *Curves defined over $\mathbb{F}_{2^m}$, m composite.* Galbraith and Smart [30], expanding on earlier work of Frey [27, 28], discuss how the Weil descent might be used to solve the ECDLP for elliptic curves defined over $\mathbb{F}_{2^m}$ where m is composite (such fields are sometimes called *composite* fields). More recently, Gaudry et al. [32] refined these ideas to provide some evidence that when m has a small divisor l, e.g., $l = 4$, the ECDLP for elliptic curves defined over $\mathbb{F}_{2^m}$ can be solved faster than with Pollard's rho algorithm. See also Menezes and Qu [66] for an analysis of the Weil descent attack. In light of these results, it seems prudent to not use elliptic curves over composite fields.
    It should be noted that some ECC standards, including the draft ANSI X9.63 [4], explicitly exclude the use of elliptic curves over composite fields. The ANSI X9F1 committee also agreed in January 1999 to exclude the use of such curves in a forthcoming revision of ANSI X9.62.
12. *Non-applicability of index-calculus methods.* Whether or not there exists a general subexponential-time algorithm for the ECDLP is an important unsettled question, and one of great relevance to the security of ECDSA. It is extremely unlikely that anyone will ever be able to *prove* that no subexponential-time algorithm exists for the ECDLP. However, much work has been done on the DLP over the past 24 years, and more specifically on the ECDLP over the past 16 years, and no subexponential-time algorithm has been discovered for the ECDLP. Miller [67] and J. Silverman and Suzuki [96] have given convincing arguments for why the most natural way in which the index-calculus algorithms can be applied to the ECDLP is most likely to fail.
13. *Xedni-calculus attacks.* A very interesting line of attack on the ECDLP, called the *xedni-calculus attack* was recently proposed by J. Silverman [95]. One intriguing aspect of the xedni-calculus attack is that it can be adapted to solve both the ordinary discrete logarithm and the integer factorization problems. However, it was subsequently shown by a team of researchers including J. Silverman (see Jacobson et al. [45]) that the attack is virtually certain to fail in practice.
14. *Hyperelliptic curves.* Hyperelliptic curves are a family of algebraic curves of arbitrary genus that includes elliptic curves. Hence, an elliptic curve can be viewed as a hyperelliptic curve of genus 1. Adleman et al. [1] (see also Stein et al. [106]) presented a subexponential-time algorithm for the discrete logarithm problem in the jacobian of a large genus hyperelliptic curve over a finite field. However, in the case of elliptic curves, the algorithm is worse than naive exhaustive search.
15. *Equivalence to other discrete logarithm problems.* Stein [105] and Zuccherato [113] showed that the dis-

crete logarithm problem in real quadratic congruence function fields of genus 1 is equivalent to the ECDLP. Since no subexponential-time algorithm is known for the former problem, this may provide further evidence for the hardness of the ECDLP.

8.1.2 Experimental results

The best general-purpose algorithm known for the ECDLP is the parallelized version of Pollard's rho algorithm, which has an expected running time of $(\sqrt{\pi n})/(2r)$ steps, where n is the (prime) order of the base point P and r is the number of processors utilized.

Certicom's ECC challenge. Certicom initiated an ECC challenge [18] in November 1997 in order to encourage and stimulate research on the ECDLP. Their challenges consist of instances of the ECDLP on a selection of elliptic curves. The challenge curves are divided into three categories listed below. In the following, ECCp-k denotes a random curve over a field $\mathbb{F}_p$, ECC2-k denotes a random curve over a field $\mathbb{F}_{2^m}$, and ECC2K-k denotes a Koblitz curve (see Sect. 5.3) over $\mathbb{F}_{2^m}$; k is the bitlength of n. In all cases, the bitsize of the order of the underlying finite field is equal or slightly greater than k (so curves have either prime order or almost prime order).

1. Randomly generated curves over $\mathbb{F}_p$, where p is prime: ECCp-79, ECCp-89, ECCp-97, ECCp-109, ECCp-131, ECCp-163, ECCp-191, ECCp-239, and ECCp-359.
2. Randomly generated curves over $\mathbb{F}_{2^m}$, where m is prime: ECC2-79, ECC2-89, ECC2-97, ECC2-109, ECC2-131, ECC2-163, ECC2-191, ECC2-238, and ECC2-353.
3. Koblitz curves over $\mathbb{F}_{2^m}$, where m is prime: ECC2K-95, ECC2-108, ECC2-130, ECC2-163, ECC2-238, and ECC2-358.

Results of the challenge. Escott et al. [25] report on their 1998 implementation of the parallelized Pollard's rho algorithm which incorporates some improvements of Teske [107]. The hardest instance of the ECDLP they solved was the Certicom ECCp-97 challenge. For this task they utilized over 1200 machines from at least 16 countries, and found the answer in 53 days. The total number of steps executed was about 2×10^{14} elliptic curve additions, which is close to the expected time $((\sqrt{\pi n})/2 \approx 3.5 \times 10^{14}$, where $n \approx 2^{97})$. Escott et al. [25] conclude that the running time of Pollard's rho algorithm in practice fits well with the theoretical predictions. They estimate that the ECCp-109 challenge could be solved by a network of 50 000 Pentium Pro 200 MHz machines in about 3 months.

8.1.3 Hardware attacks

Van Oorschot and Wiener [80] examined the feasibility of implementing parallelized Pollard's rho algorithm using special-purpose hardware. They estimated that if $n \approx 10^{36} \approx 2^{120}$, then a machine with $r = 330\,000$ processors could be built for about U.S. $10 million that could compute a single elliptic curve discrete logarithm in about 32 days. Since ANSI X9.62 mandates that the parameter n should satisfy $n > 2^{160}$, such hardware attacks appear to be infeasible with today's technology.

8.2 Attacks on the hash function

Definition. A *(cryptographic) hash function* H is a function that maps bit strings of arbitrary lengths to bit strings of a fixed length t such that:

1. H can be computed efficiently;
2. (*Preimage resistance*) For essentially all $y \in \{0,1\}^t$ it is computationally infeasible to find a bit string x such that $H(x) = y$; and
3. (*Collision resistance*) It is computationally infeasible to find distinct bit strings x_1 and x_2 such that $H(x_1) = H(x_2)$.

SHA-1 security requirements. The following explains how attacks on ECDSA can be successfully launched if SHA-1 is not preimage resistant or not collision resistant.

1. If SHA-1 is not preimage resistant, then an adversary E may be able to forge A's signatures as follows. E selects an arbitrary integer l and computes r as the x coordinate of $Q + lG$ reduced modulo n. E sets $s = r$ and computes $e = rl \bmod n$. If E can find a message m such that $e = \text{SHA-1}(m)$, then (r, s) is a valid signature for m.
2. If SHA-1 is not collision resistant, then an entity A may be able to repudiate signatures as follows. A first generates two messages m and m' such that $\text{SHA-1}(m) = \text{SHA-1}(m')$; such a pair of messages is called a *collision* for SHA-1. She then signs m and later claims to have signed m' (note that every signature for m is also a signature for m').

Ideal security. A t-bit hash function is said to be have *ideal security* [65] if both: (1) given a hash output, producing a preimage requires approximately 2^t operations; and (2) producing a collision requires approximately $2^{t/2}$ operations. SHA-1 is a 160-bit hash function and is believed to have ideal security. The fastest method known for attacking ECDSA by exploiting properties of SHA-1 is to find collisions for SHA-1. Since this is believed to take 2^{80} steps, attacking ECDSA in this way is computationally infeasible. Note, however, that this attack imposes an upper bound of 2^{80} on the security level of ECDSA, regardless of the size of the primary security parameter n. Of course, this is also the case with all present signature

schemes with appendix since the only hash functions that are widely accepted as being both secure and practical are SHA-1 and RIPEMD-160 (see Dobbertin et al. [22]), both of which are 160-bit hash functions.

Variable output length hash functions. It is expected that SHA-1 will soon be replaced by a family of hash functions H_l, where H_l is an l-bit hash function having ideal security. If one uses ECDSA with parameter n, then one would use H_l, where $l = \lfloor \log_2 n \rfloor$, as the hash function. In this case, attacking ECDSA by solving the ECDLP and attacking ECDSA by finding collisions for H_l both take approximately the same amount of time. The new family will have output lengths of 256, 384, and 512 bits [76].

8.3 Other attacks

Security requirements for per-message secrets. The per-message secrets k in ECDSA signature generation have the same security requirements as the private key d. This is because if an adversary E learns a single per-message secret k which was used by A to generate a signature (r, s) on some message m, then E can recover A's private key since $d = r^{-1}(ks - e) \bmod n$ where $e = \text{SHA-1}(m)$ (see step 6 of ECDSA signature generation). Hence per-message secrets must be securely generated, securely stored, and securely destroyed after they have been used.

Repeated use of per-message secrets. The per-message secrets k used to sign two or more messages should be generated independently of each other. In particular, a different per-message secret k should be generated for each different message signed; otherwise, the private key d can be recovered. Note that if a secure random or pseudorandom number generator is used, then the chance of generating a repeated k value is negligible. To see how private keys can be recovered if per-message secrets are repeated, suppose that the same per-message secret k was used to generate ECDSA signatures (r, s_1) and (r, s_2) on two different messages m_1 and m_2. Then $s_1 \equiv k^{-1}(e_1 + dr) \pmod n$ and $s_2 \equiv k^{-1}(e_2 + dr) \pmod n$, where $e_1 = \text{SHA-1}(m_1)$ and $e_2 = \text{SHA-1}(m_2)$. Then $ks_1 \equiv e_1 + dr \pmod n$ and $ks_2 \equiv e_2 + dr \pmod n$. Subtraction gives $k(s_1 - s_2) \equiv e_1 - e_2 \pmod n$. If $s_1 \not\equiv s_2 \pmod n$, which occurs with overwhelming probability, then $k \equiv (s_1 - s_2)^{-1}(e_1 - e_2) \pmod n$. Thus, an adversary can determine k and then use this to recover d.

Vaudenay's attacks. Vaudenay [109] demonstrated a theoretical weakness in DSA based on his insight that the actual hash function used in the DSA is SHA-1 modulo q, not just SHA-1, where q is a 160-bit prime. (Since SHA-1 is a 160-bit hash function, some of its outputs, when converted to integers, are larger than q. Hence, in general, $\text{SHA-1}(m) \neq (\text{SHA-1}(m) \bmod q)$.) This weakness allows the selective forgery of one message if the adversary can select the domain parameters. This weakness is not present in ECDSA because of the requirement that n (the analogous quantity to q in the DSA) be greater than 2^{160}.

Duplicate-signature key selection. A signature scheme S is said to have the duplicate-signature key selection (DSKS) property if given A's public key P_A and given A's signature s_A on a message M, an adversary E is able to select a valid key pair (P_E, S_E) for S such that s_A is also E's signature on M. Note that this definition requires that S_E is known to E. Blake-Wilson and Menezes [11] showed how this property can be exploited to attack a key agreement protocol which employs a signature scheme. They also demonstrated that if entities are permitted to select their own domain parameters, then ECDSA possesses the DSKS property. To see this, suppose that A's domain parameters are $D_A = (q, \text{FR}, a, b, G, n, h)$, A's key pair is (Q_A, d_A), and (r, s) is A's signature on M. The adversary E selects an arbitrary integer c, $1 \leq c \leq n - 1$, such that $t := ((s^{-1}e + s^{-1}rc) \bmod n) \neq 0$, computes $X = s^{-1}eG + s^{-1}rQ$ (where $e = \text{SHA-1}(M)$) and $\overline{G} = (t^{-1} \bmod n)X$. E then forms $D_E = (q, \text{FR}, a, b, \overline{G}, n, h)$ and $Q_E = c\overline{G}$. Then it is easily verified that D_E and Q_E are valid and that (r, s) is also E's signature on M.

If one mandates that the generating point G be selected verifiably at random during domain parameter generation (using a method akin to those in Sect. 5.2 for generating elliptic curves verifiably at random), then it appears that ECDSA no longer possesses the DSKS property. It must be emphasized that possession of the DSKS property does not constitute a weakness of the signature scheme – the goal of a signature scheme is to be existentially unforgeable against an adaptive chosen-message attack. Rather, it demonstrates the importance of auditing domain parameter and public key generation.

Implementation attacks. ANSI X9.62 does not address attacks that could be launched against implementations of ECDSA such as timing attacks (Kocher [53]), differential fault analysis (Boneh et al. [13]), differential power analysis (Kocher et al. [54]), and attacks which exploit weak random or pseudorandom number generators (Kelsey et al. [48]).

9 Implementation considerations

Before implementing ECDSA, several basic choices have to be made including:

1. Type of underlying finite field $\mathbb{F}_q$ ($\mathbb{F}_p$ or $\mathbb{F}_{2^m}$).
2. Field representation (e.g., polynomial or normal basis for $\mathbb{F}_{2^m}$).
3. Type of elliptic curve E over $\mathbb{F}_q$ (e.g., random curve or Koblitz curve).
4. Elliptic curve point representation (e.g., affine or projective coordinates [39]).

There are many factors that can influence the choices made. All of these must be considered simultaneously in

order to arrive at the best solution for a particular application. The factors include:

- Security considerations.
- Suitability of methods available for optimizing finite field arithmetic (addition, multiplication, squaring, and inversion).
- Suitability of methods available for optimizing elliptic curve arithmetic (point addition, point doubling, and scalar multiplication).
- Application platform (software, hardware, or firmware).
- Constraints of a particular computing environment (e.g., processor speed, storage, code size, gate count, power consumption).
- Constraints of a particular communications environment (e.g., bandwidth, response time).

Selected references to the literature. The most detailed and comprehensive reference available on techniques for efficient finite field and elliptic curve arithmetic is IEEE 1363-2000 [39]. See Gordon [36] for a detailed survey of various methods for scalar multiplication. For an implementation report of elliptic curve operations over $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$, see Schroeppel et al. [92], De Win et al. [112], Hasegawa et al. [38], Brown et al. [16, 17], and Hankerson et al. [37].

10 Interoperability considerations

The goals of cryptographic standards are twofold:

1. To facilitate the widespread use of cryptographically sound and well-specified techniques.
2. To promote interoperability between different implementations.

Factors affecting interoperability. Interoperability is encouraged by completely specifying the steps of the cryptographic schemes and the formats for shared data such as domain parameters, keys, and exchanged messages, and by limiting the number of options available to the implementor. For elliptic curve cryptography and, in particular, the ECDSA, the factors that can impact interoperability include:

1. The number, and types, of allowable finite fields.
2. The number of allowable representations for the elements of an allowable finite field.
3. The number of allowable elliptic curves over an allowable finite field.
4. The formats for specifying field elements, elliptic curve points, domain parameters, public keys, and signatures.

10.1 ECDSA standards

Among the standards and draft standards which specify ECDSA, the ones which have been officially approved by their respective accredited organizations are ANSI X9.62 [3], FIPS 186-2 [74], IEEE 1363-2000 [39], and ISO 14888-3 [42]. ECDSA has also been standardized by the Standards for Efficient Cryptography Group (SECG) [103], which is a consortium of companies formed to address potential interoperability problems with cryptographic standards.

The salient features of these standards are described first, and then the standards are compared with regards to their compatibility with each other. This is followed by a brief overview of some other standards that specify or use ECDSA.

Core ECDSA standards.

1. *ANSI X9.62*: This project began in 1995 and was adopted as an official ANSI standard in January 1999. The primary objectives of ANSI X9.62 were to achieve a high level of security and interoperability. The underlying field is restricted to being a prime finite field $\mathbb{F}_p$ or a binary finite field $\mathbb{F}_{2^m}$. The elements of $\mathbb{F}_{2^m}$ may be represented using a polynomial or a normal basis over $\mathbb{F}_2$. If a polynomial basis is desired, ANSI X9.62 mandates that the reduction polynomial be an irreducible trinomial, provided one exists, and an irreducible pentanomial otherwise. To facilitate interoperability, a specific reduction polynomial is recommended for each field $\mathbb{F}_{2^m}$. If a normal basis is desired, ANSI X9.62 mandates that a specific Gaussian normal basis be used. The primary security requirement imposed on elliptic curves in ANSI X9.62 is that n, the order of the base point G, be greater than 2^{160}. Elliptic curves may either be selected arbitrarily (subject to the security constraints mentioned in Sect. 5.1) or verifiably at random (using the procedure described in Sect. 5.3). ANSI X9.62 defines a mandatory octet string representation for elliptic points in either compressed, uncompressed, or hybrid form. Optional ASN.1 (abstract syntax notation one) syntax is provided for unambiguously describing domain parameters, public keys, and signatures.
2. *FIPS 186-2*: In May 1997, NIST announced plans to revise FIPS 186 by including RSA and elliptic curve signature algorithms. In December 1998, FIPS 186 was revised to include both the DSA and RSA signature schemes (as specified in ANSI X9.31 [2]); the revised standard was called FIPS 186-1 [73]. Shortly after that, in June 1999, NIST presented a list of 15 elliptic curves that were recommended for U.S. Federal Government use. These curves are compliant with the ANSI X9.62 formats (and therefore also with IEEE 1363-2000 formats) and are discussed further in Sect. 10.2. In February 2000, FIPS 186-1 was revised to include ECDSA as specified in ANSI X9.62 with the aforementioned recommended elliptic curves; the revised standard is called FIPS 186-2.
3. *IEEE 1363-2000*: This project was formally approved as an IEEE standard in August 2000. IEEE 1363's

scope is very broad and includes public-key cryptographic techniques for encryption, key agreement, and signatures based on the intractability of integer factorization, discrete logarithms in finite fields, and elliptic curve discrete logarithms. It differs fundamentally from ANSI X9.62 and FIPS 186-2 in that it does not mandate minimum security requirements (e.g., lower bounds on the order n of the base point G) and has an abundance of options. Consequently, 1363-2000 should neither be viewed as a security standard nor as an interoperability standard, but rather as a reference for specifications of a variety of techniques from which applications may select. With regards to the elliptic curve schemes and, in particular, ECDSA, the underlying field is restricted to being a prime finite field $\mathbb{F}_p$ or a binary finite field $\mathbb{F}_{2^m}$. The elements of $\mathbb{F}_{2^m}$ may be represented with respect to any polynomial or normal basis over $\mathbb{F}_2$. The representation of $\mathbb{F}_p$ elements as integers and $\mathbb{F}_{2^m}$ elements as bit strings are consistent with ANSI X9.62 and FIPS 186-2 conventions.

4. *ISO/IEC 14888-3* [42]: This standard contains high-level descriptions of some signature algorithms including ECDSA, whose description is consistent with that of ANSI X9.62.
5. *SEC 1* [103] *and SEC 2* [104]: SEC 1 describes the ECDSA, and also elliptic curve public-key encryption and key agreement protocols. A specific list of recommended elliptic curve domain parameters are provided in SEC 2. SEC 1 ECDSA is compliant with ANSI X9.62, except that the former permits some fields of bitlength less than 160.

Compatibility. Any ECDSA implementation that is conformant with FIPS 186-2 is also conformant with SEC 1; however, the converse is not necessarily true. Any ECDSA implementation that is conformant with SEC 1 (with $n > 2^{160}$) is conformant with ANSI X9.62; however, the converse is not necessarily true. Furthermore, any ECDSA implementation that is conformant with ANSI X9.62 is also conformant with IEEE 1363-2000; however, the converse is not necessarily true. Finally, any ECDSA implementation that is conformant with IEEE 1363-2000 is also conformant with ISO 14888-3, but the converese is not necessarily true. This conformance relationship between the five ECDSA standards is depicted in Fig. 3.

Other ECDSA standards. ECDSA is being considered for inclusion in numerous core cryptography and applications standards. These include:

1. *ISO/IEC 15946* [43]: This draft standard specifies various cryptographic techniques based on elliptic curves including signature schemes, public-key encryption schemes, and key establishment protocols. ISO/IEC 15946 allows any finite field, unlike ANSI X9.62, IEEE 1363-2000, and FIPS 186-2 where the underlying field is required to be either a prime field or a binary field. It is expected that the ECDSA description will be consistent with that of ANSI X9.62.
2. *IETF PKIX* (Internet Engineering Task Force Public Key Infrastructure X.509-Based): An internet draft [7] profiles the format of ECDSA domain parameters and public keys for use in X.509 certificates. The formats are consistent with those present in ANSI X9.62.
3. *IETF TLS* (Internet Engineering Task Force Transport Layer Security): This is the IETF's adoption of SSL (secure sockets layer) which provides confidentiality, integrity, and authentication for network connections. ANSI X9.62 ECDSA is currently being considered for inclusion as one of the signature algorithms [20].
4. *WAP WTLS* [110] (Wireless Application Protocol Wireless Transport Layer Security): Provides transport layer security for an architecture that enables secure web browsing for mobile devices such as cellular phones, personal device assistants, and pagers. ANSI X9.62 ECDSA is used for authentication.

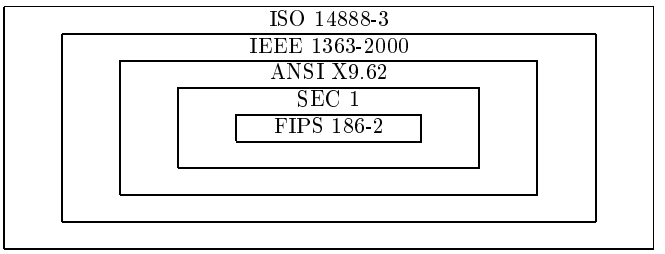

Fig. 3. Compatibility of FIPS 186-2, SEC 1, ANSI X9.62, IEEE 1363-2000, and ISO 14888-3 specifications of ECDSA

10.2 NIST recommended curves

This section presents the 15 elliptic curves that were recommended (but not mandated) by NIST for U.S. Federal Government use [74].

Recommended finite fields. There are 10 recommended finite fields:

1. The prime fields $\mathbb{F}_p$ for $p = 2^{192} - 2^{64} - 1$, $p = 2^{224} - 2^{96} + 1$, $p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$, $p = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$, and $p = 2^{521} - 1$.
2. The binary fields $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$, $\mathbb{F}_{2^{283}}$, $\mathbb{F}_{2^{409}}$, and $\mathbb{F}_{2^{571}}$.

The factors which influenced the choices of fields were:

(i) The fields were selected so that the bitlengths of their orders are twice the key lengths of common symmetric-key block ciphers – this is because exhaustive key search of a k-bit block cipher is expected to take roughly the same time as the solution of an instance of the elliptic curve discrete logarithm problem using Pollard's rho algorithm for an appropriately selected elliptic curve over a finite field

whose order has bitlength $2k$. The correspondence between symmetric cipher key lengths and field sizes is given in Table 1.

(ii) For prime fields $\mathbb{F}_p$, the prime moduli p are of a special type (called *generalized Mersenne numbers*) for which modular multiplication can be carried out more efficiently than in general; see [74] and [101].

(iii) For binary fields $\mathbb{F}_{2^m}$, m was chosen so that there exists a Koblitz curve of almost prime order over $\mathbb{F}_{2^m}$. Since $\#E(\mathbb{F}_{2^l})$ divides $\#E(\mathbb{F}_{2^m})$ whenever l divides m, this requirement imposes the condition that m be prime.

Recommended elliptic curves. There are three types of elliptic curves:

1. Random elliptic curves over $\mathbb{F}_p$.
2. Koblitz elliptic curves over $\mathbb{F}_{2^m}$.
3. Random elliptic curves over $\mathbb{F}_{2^m}$.

The parameters of these curves are presented below. In these subsections, parameters are either given in decimal form or in hexadecimal form preceded by '0x'. For the binary fields, the additive and multiplicative identities are simply denoted by 0 and 1. A method for converting between polynomial and normal basis representations for $\mathbb{F}_{2^m}$ is given at the end of this section.

10.2.1 Random elliptic curves over $\mathbb{F}_p$

The following parameters are given for each elliptic curve:

p	The order of the prime field $\mathbb{F}_p$.
`seedE`	The seed used to randomly generate the coefficients of the elliptic curve using Algorithm 1.
r	The output of SHA-1 in Algorithm 1.
a, b	The coefficients of the elliptic curve $y^2 = x^3 + ax + b$ satisfying $rb^2 \equiv a^3 \bmod p$. The selection $a = -3$ was made for reasons of efficiency; see IEEE 1363-2000 [39].
x_G, y_G	The x and y coordinates of the base point G.
n	The (prime) order of G.
h	The co-factor.

Table 1. Recommended field sizes for U.S. Federal Government use.

Symmetric cipher key length	Example algorithm	Bitlength of p in prime field $\mathbb{F}_p$	Dimension m of binary field $\mathbb{F}_{2^m}$
80	SKIPJACK [77]	192	163
112	Triple-DES	224	233
128	AES small [75]	256	283
192	AES medium [75]	384	409
256	AES large [75]	521	571

Curve P-192 ($p = 2^{192} - 2^{64} - 1$)

```
p      6277101735386680763835789423207666416083908700390324961279
seedE  0x 3045ae6f c8422f64 ed579528 d38120ea e12196d5
r      0x 3099d2bb bfcb2538 542dcd5f b078b6ef 5f3d6fe2 c745de65
a      -3
b      0x 64210519 e59c80e7 0fa7e9ab 72243049 feb8deec c146b9b1
xG     0x 188da80e b03090f6 7cbf20eb 43a18800 f4ff0afd 82ff1012
yG     0x 07192b95 ffc8da78 631011ed 6b24cdd5 73f977a1 1e794811
n      6277101735386680763835789423176059013767194773182842284081
h      1
```

Curve P-224 ($p = 2^{224} - 2^{96} + 1$)

```
p      26959946667150639794667015087019630673557916260026308143510066298881
seedE  0x bd713447 99d5c7fc dc45b59f a3b9ab8f 6a948bc5
r      0x 5b056c7e 11dd68f4 0469ee7f 3c7a7d74 f7d12111 6506d031 218291fb
a      -3
b      0x b4050a85 0c04b3ab f5413256 5044b0b7 d7bfd8ba 270b3943 2355ffb4
xG     0x b70e0cbd 6bb4bf7f 321390b9 4a03c1d3 56c21122 343280d6 115c1d21
yG     0x bd376388 b5f723fb 4c22dfe6 cd4375a0 5a074764 44d58199 85007e34
n      26959946667150639794667015087019625940457807714424391721682722368061
h      1
```

Curve P-256 ($p = 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$)

```
p     1157920892103562487626974469494075735300861434152903141955336313088670978539 51
seedE 0x c49d3608 86e70493 6a6678e1 139d26b7 819f7e90
r     0x 7efba166 2985be94 03cb055c 75d4f7e0 ce8d84a9 c5114abc af317768 0104fa0d
a     -3
b     0x 5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b
xG    0x 6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0 f4a13945 d898c296
yG    0x 4fe342e2 fe1a7f9b 8ee7eb4a 7c0f9e16 2bce3357 6b315ece cbb64068 37bf51f5
n     1157920892103562487626974469494075735299969552241357603424222590610685120443 69
h     1
```

Curve P-384 ($p = 2^{384} - 2^{128} - 2^{96} + 2^{32} - 1$)

```
p     3940200619639447921227904010014361380507973927046544666794829340424572177149687032 9
      0472660882589380018616069731123 19
seedE 0x a335926a a319a27a 1d00896a 6773a482 7acdac73
r     0x 79d1e655 f868f02f ff48dcde e14151dd b80643c1 406d0ca1 0dfe6fc5 2009540a 495e8042
         ea5f744f 6e184667 cc722483
a     -3
b     0x b3312fa7 e23ee7e4 988e056b e3f82d19 181d9c6e fe814112 0314088f 5013875a c656398d
         8a2ed19d 2a85c8ed d3ec2aef
xG    0x aa87ca22 be8b0537 8eb1c71e f320ad74 6e1d3b62 8ba79b98 59f741e0 82542a38 5502f25d
         bf55296c 3a545e38 72760ab7
yG    0x 3617de4a 96262c6f 5d9e98bf 9292dc29 f8f41dbd 289a147c e9da3113 b5f0b8c0 0a60b1ce
         1d7e819d 7a431d7c 90ea0e5f
n     3940200619639447921227904010014361380507973927046544666794690527962765939911326356 9
      3989563081522949135544336539426 43
h     1
```

Curve P-521 ($p = 2^{521} - 1$)

```
p     6864797660130609714981900799081393217269435300143305409394463459185543183397656052 1
      2255964066145455497729631139148085803712198799971664381257402829111505715 1
seedE 0x d09e8800 291cb853 96cc6717 393284aa a0da64ba
r     0x      0b4 8bfa5f42 0a349495 39d2bdfc 264eeeeb 077688e4 4fbf0ad8 f6d0edb3 7bd6b533
         28100051 8e19f1b9 ffbe0fe9 ed8a3c22 00b8f875 e523868c 70c1e5bf 55bad637
a     -3
b     0x      051 953eb961 8e1c9a1f 929a21a0 b68540ee a2da725b 99b315f3 b8b48991 8ef109e1
         56193951 ec7e937b 1652c0bd 3bb1bf07 3573df88 3d2c34f1 ef451fd4 6b503f00
xG    0x      0c6 858e06b7 0404e9cd 9e3ecb66 2395b442 9c648139 053fb521 f828af60 6b4d3dba
         a14b5e77 efe75928 fe1dc127 a2ffa8de 3348b3c1 856a429b f97e7e31 c2e5bd66
yG    0x      118 39296a78 9a3bc004 5c8a5fb4 2c7d1bd9 98f54449 579b4468 17afbd17 273e662c
         97ee7299 5ef42640 c550b901 3fad0761 353c7086 a272c240 88be9476 9fd16650
n     6864797660130609714981900799081393217269435300143305409394463459185543183397655394 2
      4505774633321719753296399637136332111386476861244038034037280889270700544 9
h     1
```

10.2.2 Koblitz elliptic curves over $\mathbb{F}_{2^m}$

The parameters of the (same) Koblitz curve and base point are given in both normal basis representation (indicated by FR) and in polynomial basis representation (indicated by FR2). A method for converting between the two representations is given at the end of this section. The following parameters are given for each Koblitz curve:

m	The extension degree of the binary field $\mathbb{F}_{2^m}$.
FR	An indication of the representation used for the elements of $\mathbb{F}_{2^m}$ in accordance with ANSI X9.62.
a, b	The coefficients of the elliptic curve $y^2 + xy = x^3 + ax^2 + b$.
x_G, y_G	The x and y coordinates of the base point G.
n	The (prime) order of G.
h	The co-factor.

FR2 An indication of the second representation used for the elements of $\mathbb{F}_{2^m}$ in accordance with ANSI X9.62.

$a2, b2$ The coefficients of the (same) elliptic curve using representation FR2.

x_G2, y_G2 The x and y coordinates of the (same) base point G using representation FR2.

10.2.3 Random elliptic curves over $\mathbb{F}_{2^m}$

Each random elliptic curve over $\mathbb{F}_{2^m}$ was generated using Algorithm 3. The output of SHA-1 was interpreted as an element of a binary field represented with a Gaussian normal basis. The parameters of the (same) elliptic curve and base point are given in both normal basis representation

Curve K-163

```
m   163
FR  Gaussian Normal Basis, T=4
a   1
b   1
xG  0x 0 5679b353 caa46825 fea2d371 3ba450da 0c2a4541
yG  0x 2 35b7c671 00506899 06bac3d9 dec76a83 5591edb2
n   5846006549323611672814741753598448348329118574063
h   2
FR2 Polynomial basis with reduction polynomial f(x) = x^163 + x^7 + x^6 + x^3 + 1
a2  1
b2  1
xG2 0x 2 fe13c053 7bbc11ac aa07d793 de4e6d5e 5c94eee8
yG2 0x 2 89070fb0 5d38ff58 321f2e80 0536d538 ccdaa3d9
```

Curve K-233

```
m   233
FR  Gaussian Normal Basis, T=2
a   0
b   1
xG  0x 0fd e76d9dcd 26e643ac 26f1aa90 1aa12978 4b71fc07 22b2d056 14d650b3
yG  0x 064 3e317633 155c9e04 47ba8020 a3c43177 450ee036 d6335014 34cac978
n   3450873173395281893717377931138512760570940988862252126328087024741343
h   4
FR2 Polynomial basis with reduction polynomial f(x) = x^233 + x^74 + 1
a2  0
b2  1
xG2 0x 172 32ba853a 7e731af1 29f22ff4 149563a4 19c26bf5 0a4c9d6e efad6126
yG2 0x 1db 537dece8 19b7f70f 555a67c4 27a8cd9b f18aeb9b 56e0c110 56fae6a3
```

Curve K-283

```
m   283
FR  Gaussian Normal Basis, T=6
a   0
b   1
xG  0x  3ab9593 f8db09fc 188f1d7c 4ac9fcc3 e57fcd3b db15024b 212c7022 9de5fcd9 2eb0ea60
yG  0x  2118c47 55e7345c d8f603ef 93b98b10 6fe8854f feb9a3b3 04634cc8 3a0e759f 0c2686b1
n   38853377844514581418389238136470378132848117337930613242958749975298158297044226038 73
h   4
FR2 Polynomial basis with reduction polynomial f(x) = x^283 + x^12 + x^7 + x^5 + 1
a2  0
b2  1
xG2 0x  503213f 78ca4488 3f1a3b81 62f188e5 53cd265f 23c1567a 16876913 b0c2ac24 58492836
yG2 0x  1ccda38 0f1c9e31 8d90f95d 07e5426f e87e45c0 e8184698 e4596236 4e341161 77dd2259
```

Curve K-409

```
m   409
FR  Gaussian Normal Basis, T=4
a   0
b   1
xG  0x  1b559c7 cba2422e 3affe133 43e808b5 5e012d72 6ca0b7e6 a63aeafb c1e3a98e 10ca0fcf
        98350c3b 7f89a975 4a8e1dc0 713cec4a
yG  0x  16d8c42 052f07e7 713e7490 eff318ba 1abd6fef 8a5433c8 94b24f5c 817aeb79 852496fb
        ee803a47 bc8a2038 78ebf1c4 99afd7d6
n   33052798439512429947595765401638551991420234148214060964232439502288071128924919105
    06732584577774580140963665906177313586 71
h   4
FR2 Polynomial basis with reduction polynomial f(x) = x^409 + x^87 + 1
a2  0
b2  1
xG2 0x  060f05f 658f49c1 ad3ab189 0f718421 0efd0987 e307c84c 27accfb8 f9f67cc2 c460189e
        b5aaaa62 ee222eb1 b35540cf e9023746
yG2 0x  1e36905 0b7c4e42 acba1dac bf04299c 3460782f 918ea427 e6325165 e9ea10e3 da5f6c42
        e9c55215 aa9ca27a 5863ec48 d8e0286b
```

Curve K-571

```
m   571
FR  Gaussian Normal Basis, T=10
a   0
b   1
xG  0x  04bb2db a418d0db 107adae0 03427e5d 7cc139ac b465e593 4f0bea2a b2f3622b c29b3d5b
        9aa7a1fd fd5d8be6 6057c100 8e71e484 bcd98f22 bf847642 37673674 29ef2ec5 bc3ebcf7
yG  0x  44cbb57 de20788d 2c952d7b 56cf39bd 3e89b189 84bd124e 751ceff4 369dd8da c6a59e6e
        745df44d 8220ce22 aa2c852c fcbbef49 ebaa98bd 2483e331 80e04286 feaa2530 50caff60
n   19322687615086291723476759454659936721494636648532174993286176257257595711447802122
    68133978522706711834706712800825351461273674974066617311929682421617092503555733685
    276673
h   4
FR2 Polynomial basis with reduction polynomial f(x) = x^571 + x^10 + x^5 + x^2 + 1
a2  0
b2  1
xG2 0x  26eb7a8 59923fbc 82189631 f8103fe4 ac9ca297 0012d5d4 60248048 01841ca4 43709584
        93b205e6 47da304d b4ceb08c bbd1ba39 494776fb 988b4717 4dca88c7 e2945283 a01c8972
yG2 0x  349dc80 7f4fbf37 4f4aeade 3bca9531 4dd58cec 9f307a54 ffc61efc 006d8a2c 9d4979c0
        ac44aea7 4fbebbb9 f772aedc b620b01a 7ba7af1b 320430c8 591984f6 01cd4c14 3ef1c7a3
```

(indicated by FR) and in polynomial basis representation (indicated by FR2). A method for converting between the two representations is given at the end of this section. The following parameters are given for each elliptic curve:

m	The extension degree of the binary field $\mathbb{F}_{2^m}$.
FR	An indication of the representation used for the elements of $\mathbb{F}_{2^m}$ in accordance with ANSI X9.62.
`seedE`	The seed used to randomly generate the coefficients of the elliptic curve using Algorithm 3.
a, b	The coefficients of the elliptic curve $y^2 + xy = x^3 + ax^2 + b$.
x_G, y_G	The x and y coordinates of the base point G.
n	The (prime) order of G.
h	The co-factor.
FR2	An indication of the second representation used for the elements of $\mathbb{F}_{2^m}$ in accordance with ANSI X9.62.
$a2, b2$	The coefficients of the (same) elliptic curve using representation FR2.
x_G2, y_G2	The x and y coordinates of the (same) base point G using representation FR2.

10.2.4 Converting between polynomial and normal basis representations

This section describes one method, utilizing multiplication by a change-of-basis matrix, for converting the elements of $\mathbb{F}_{2^m}$ represented with respect to a particular polynomial basis to the elements of $\mathbb{F}_{2^m}$ represented with respect to a particular normal basis, and vice versa.

Curve B-163

```
m     163
FR    Gaussian Normal Basis, T=4
seedE 0x 85e25bfe 5c86226c db12016f 7553f9d0 e693a268
a     1
b     0x        6 645f3cac f1638e13 9c6cd13e f61734fb c9e3d9fb
xG    0x        0 311103c1 7167564a ce77ccb0 9c681f88 6ba54ee8
yG    0x        3 33ac13c6 447f2e67 613bf700 9daf98c8 7bb50c7f
n     5846006549323611672814742442876390689256843201587
h     2
FR2   Polynomial basis with reduction polynomial f(x) = x^163 + x^7 + x^6 + x^3 + 1
a2    1
b2    0x        2 0a601907 b8c953ca 1481eb10 512f7874 4a3205fd
xG2   0x        3 f0eba162 86a2d57e a0991168 d4994637 e8343e36
yG2   0x        0 d51fbc6c 71a0094f a2cdd545 b11c5c0c 797324f1
```

Curve B-233

```
m     233
FR    Gaussian Normal Basis, T=2
seedE 0x 74d59ff0 7f6b413d 0ea14b34 4b20a2db 049b50c3
a     1
b     0x      1a0 03e0962d 4f9a8e40 7c904a95 38163adb 82521260 0c7752ad 52233279
xG    0x      18b 863524b3 cdfefb94 f2784e0b 116faac5 4404bc91 62a363ba b84a14c5
yG    0x      049 25df77bd 8b8ff1a5 ff519417 822bfedf 2bbd7526 44292c98 c7af6e02
n     6901746346790563787434755862277025555839812737345013555379383634485463
h     2
FR2   Polynomial basis with reduction polynomial f(x) = x^233 + x^74 + 1
a2    1
b2    0x      066 647ede6c 332c7f8c 0923bb58 213b333b 20e9ce42 81fe115f 7d8f90ad
xG2   0x      0fa c9dfcbac 8313bb21 39f1bb75 5fef65bc 391f8b36 f8f8eb73 71fd558b
yG2   0x      100 6a08a419 03350678 e58528be bf8a0bef f867a7ca 36716f7e 01f81052
```

Curve B-283

```
m     283
FR    Gaussian Normal Basis, T=6
seedE 0x 77e2b073 70eb0f83 2a6dd5b6 2dfc88cd 06bb84b
a     1
b     0x  157261b 894739fb 5a13503f 55f0b3f1 0c560116 66331022 01138cc1 80c0206b dafbc951
xG    0x  749468e 464ee468 634b21f7 f61cb700 701817e6 bc36a236 4cb8906e 940948ea a463c35d
yG    0x  62968bd 3b489ac5 c9b859da 68475c31 5bafcdc4 ccd0dc90 5b70f624 46f49c05 2f49c08c
n     77706755689029162836778476272940756265696259243769048891091965267700442777873786928 71
h     2
FR2   Polynomial basis with reduction polynomial f(x) = x^283 + x^12 + x^7 + x^5 + 1
a2    1
b2    0x  27b680a c8b8596d a5a4af8a 19a0303f ca97fd76 45309fa2 a581485a f6263e31 3b79a2f5
xG2   0x  5f93925 8db7dd90 e1934f8c 70b0dfec 2eed25b8 557eac9c 80e2e198 f8cdbecd 86b12053
yG2   0x  3676854 fe24141c b98fe6d4 b20d02b4 516ff702 350eddb0 826779c8 13f0df45 be8112f4
```

The change-of-basis matrices for converting between the polynomial basis and normal basis representations of the fields $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$, $\mathbb{F}_{2^{283}}$, $\mathbb{F}_{2^{409}}$, and $\mathbb{F}_{2^{571}}$ are presented. There are other methods available for performing the conversions; e.g., see Kaliski and Yin [47].

Normal basis to polynomial basis conversion. Suppose that α is an element of the field $\mathbb{F}_{2^m}$. Let a be its bit string representation with respect to a given normal basis, and let $\overline{a}$ be its bit string representation with respect to a given polynomial basis. Then $\overline{a}$ can be derived from a via the matrix computation $\overline{a} = aA$, where A is an $m \times m$ binary matrix. The matrix A, which depends only on the bases, can be computed easily given its top row R as follows. Let β be the element of $\mathbb{F}_{2^m}$ whose representation with respect to the polynomial basis is R. Then the rows of A, from top to bottom, are the bit strings representing the elements $\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}$ with respect to the polynomial basis.

Curve B-409

```
m     409
FR    Gaussian Normal Basis, T=4
seedE 0x 4099b5a4 57f9d69f 79213d09 4c4bcd4d 4262210b
a     1
b     0x  124d065 1c3d3772 f7f5a1fe 6e715559 e2129bdf a04d52f7 b6ac7c53 2cf0ed06 f610072d
         88ad2fdc c50c6fde 72843670 f8b3742a
xG    0x  0ceacbc 9f475767 d8e69f3b 5dfab398 13685262 bcacf22b 84c7b6dd 981899e7 318c96f0
         761f77c6 02c016ce d7c548de 830d708f
yG    0x  199d64b a8f089c6 db0e0b61 e80bb959 34afd0ca f2e8be76 d1c5e9af fc7476df 49142691
         ad303902 88aa09bc c59c1573 aa3c009a
n     661055968790248598951915308032771039828404682964281219284648798304157774827374805208143723762179110965979867288366567526771
h     2
FR2   Polynomial basis with reduction polynomial f(x) = x^409 + x^87 + 1
a2    1
b2    0x  021a5c2 c8ee9feb 5c4b9a75 3b7b476b 7fd6422e f1f3dd67 4761fa99 d6ac27c8 a9a197b2
         72822f6c d57a55aa 4f50ae31 7b13545f
xG2   0x  15d4860 d088ddb3 496b0c60 64756260 441cde4a f1771d4d b01ffe5b 34e59703 dc255a86
         8a118051 5603aeab 60794e54 bb7996a7
yG2   0x  061b1cf ab6be5f3 2bbfa783 24ed106a 7636b9c5 a7bd198d 0158aa4f 5488d08f 38514f1f
         df4b4f40 d2181b36 81c364ba 0273c706
```

Curve B-571

```
m     571
FR    Gaussian Normal Basis, T=10
seedE 0x 2aa058f7 3a0e33ab 486b0f61 0410c53a 7f132310
a     1
b     0x  3762d0d 47116006 179da356 88eeaccf 591a5cde a7500011 8d9608c5 9132d434 26101a1d
         fb377411 5f586623 f75f0000 1ce61198 3c1275fa 31f5bc9f 4be1a0f4 67f01ca8 85c74777
xG    0x  0735e03 5def5925 cc33173e b2a8ce77 67522b46 6d278b65 0a291612 7dfea9d2 d361089f
         0a7a0247 a184e1c7 0d417866 e0fe0feb 0ff8f2f3 f9176418 f97d117e 624e2015 df1662a8
yG    0x  04a3642 0572616c df7e606f ccadaecf c3b76dab 0eb1248d d03fbdfc 9cd3242c 4726be57
         9855e812 de7ec5c5 00b4576a 24628048 b6a72d88 0062eed0 dd34b109 6d3acbb6 b01a4a97
n     3864537523017258344695351890931987344298927329706434998657235251451519142289560424536143999389415773083133881121926944486246872462816813070234528288303332411393191105285703
h     2
FR2   Polynomial basis with reduction polynomial f(x) = x^571 + x^10 + x^5 + x^2 + 1
a2    1
b2    0x  2f40e7e 2221f295 de297117 b7f3d62f 5c6a97ff cb8ceff1 cd6ba8ce 4a9a18ad 84ffabbd
         8efa5933 2be7ad67 56a66e29 4afd185a 78ff12aa 520e4de7 39baca0c 7ffeff7f 2955727a
xG2   0x  303001d 34b85629 6c16c0d4 0d3cd775 0a93d1d2 955fa80a a5f40fc8 db7b2abd bde53950
         f4c0d293 cdd711a3 5b67fb14 99ae6003 8614f139 4abfa3b4 c850d927 e1e7769c 8eec2d19
yG2   0x  37bf273 42da639b 6dccfffe b73d69d7 8c6c27a6 009cbbca 1980f853 3921e8a6 84423e43
         bab08a57 6291af8f 461bb2a8 b3531d2f 0485c19b 16e2f151 6e23dd3c 1a4827af 1b8ac15b
```

The following gives the top row R for each conversion from the normal bases indicated by FR to the polynomial bases indicated by FR2.

Polynomial basis to normal basis conversion. Suppose that α is an element of the field $\mathbb{F}_{2^m}$. Let a be its bit string representation with respect to a given normal basis, and let $\overline{a}$ be its bit string representation with respect to a given polynomial basis. Then a can be derived from $\overline{a}$ via the matrix computation $a = \overline{a}B$, where B is an $m \times m$ binary matrix. The matrix B, which depends only on the bases, can be computed easily given its second-to-last row S as follows. Let β be the element of $\mathbb{F}_{2^m}$ whose representation with respect to the normal basis is S. Then the rows of B, from top to bottom, are the bit strings representing the elements $\beta^{m-1}, \beta^{m-2}, \ldots, \beta^2, \beta, 1$ with respect to the normal basis.

The following gives the second-to-last row S for each conversion from the polynomial bases indicated by FR2 to the normal bases indicated by FR.

11 Conclusions

ECDSA is now an ANSI, IEEE, NIST, and ISO standard and is being standardized by several other standards organizations. This paper described the ANSI X9.62 ECDSA, presented rationale for some design decisions, and discussed related security, implementation, and in-

```
m=163 0x        7 15169c10 9c612e39 0d347c74 8342bcd3 b02a0bef

m=233 0x      149 9e398ac5 d79e3685 59b35ca4 9bb7305d a6c0390b cf9e2300 253203c9

m=283 0x  31e0ed7 91c3282d c5624a72 0818049d 053e8c7a b8663792 bc1d792e ba9867fc 7b317a99

m=409 0x  0dfa06b e206aa97 b7a41fff b9b0c55f 8f048062 fbe8381b 4248adf9 2912ccc8 e3f91a24
          e1cfb395 0532b988 971c2304 2e85708d

m=571 0x  452186b bf5840a0 bcf8c9f0 2a54efa0 4e813b43 c3d41496 06c4d27b 487bf107 393c8907
          f79d9778 beb35ee8 7467d328 8274caeb da6ce05a eb4ca5cf 3c3044bd 4372232f 2c1a27c4


m=163 0x        3 e173bfaf 3a86434d 883a2918 a489ddbd 69fe84e1

m=233 0x      0be 19b89595 28bbc490 038f4bc4 da8bdfc1 ca36bb05 853fd0ed 0ae200ce

m=283 0x  3347f17 521fdabc 62ec1551 acf156fb 0bceb855 f174d4c1 7807511c 9f745382 add53bc3

m=409 0x  0eb00f2 ea95fd6c 64024e7f 0b68b81f 5ff8a467 acc2b4c3 b9372843 6265c7ff a06d896c
          ae3a7e31 e295ec30 3eb9f769 de78bef5

m=571 0x  7940ffa ef996513 4d59dcbf e5bf239b e4fe4b41 05959c5d 4d942ffd 46ea35f3 e3cdb0e1
          04a2aa01 cef30a3a 49478011 196bfb43 c55091b6 1174d7c0 8d0cdd61 3bf6748a bad972a4
```

teroperability issues. We hope that this paper contributes to an increased understanding of the properties of ECDSA, and facilitates its use in practice.

Acknowledgements. The authors would like to thank the members of the ANSI X9F1 and IEEE P1363 working groups, and, in particular, Jerry Solinas, for their many comments and contributions during the development of the ECDSA standards.

References

1. Adleman L, DeMarrais J, Huang M (1994) A subexponential algorithm for discrete logarithms over the rational subgroup of the jacobians of large genus hyperelliptic curves over finite fields. In: Algorithmic Number Theory, Lecture Notes in Computer Science, vol 877. Springer, Berlin Heidelberg New York, pp 28–40
2. ANSI X9.31 (1998) Digital signatures using reversible public key cryptography for the financial services industry (rDSA)
3. ANSI X9.62 (1999) Public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA)
4. ANSI X9.63 (2000) Public key cryptography for the financial services industry: elliptic curve key agreement and key transport protocols. Working draft
5. Ash D, Blake I, Vanstone S (1989) Low complexity normal bases. Discrete Appl Math25:191–210
6. Balasubramanian R, Koblitz N (1998) The improbability that an elliptic curve has subexponential discrete log problem under the Menezes–Okamoto–Vanstone algorithm. J Cryptology 11:141–145
7. Bassham L, Johnson D, Polk T (1999) Representation of Elliptic Curve Digital Signature Algorithm (ECDSA) Keys and Signatures in Internet X.509 Public Key Infrastructure Certificates. Internet Draft, Available at `http://www.ietf.org`
8. Bellare M, Canetti R, Krawczyk H (1998) A modular approach to the design and analysis of authentication and key exchange protocols. In: Proceedings of the 30th Annual ACM Symposium on the Theory of Computing, Dallas. ACM Press, pp 419–428
9. Blake I, Seroussi G, Smart N (1999) Elliptic curves in cryptography. Cambridge University Press, Cambridge
10. Blake-Wilson S, Menezes A (1997) Entity authentication and authenticated key transport protocols employing asymmetric techniques. In: Proceedings of the 5th International Workshop on Security Protocols, Lecture Notes in Computer Science, vol 1361. Springer, Berlin Heidelberg New York, pp 137–158
11. Blake-Wilson S, Menezes A (1999) Unknown key-share attacks on the station-to-station (STS) protocol. In: Public Key Cryptography – Proceedings of PKC '99, Lecture Notes in Computer Science, vol 1560. Springer, Berlin Heidelberg New York, pp 154–170
12. Bleichenbacher D (1996) Generating ElGamal signatures without knowing the secret key. In: Advances in Cryptology – Eurocrypt '96, Lecture Notes in Computer Science, vol 1070. Springer, Berlin Heidelberg New York, pp 10–18
13. Boneh D, DeMillo R, Lipton R (1997) On the importance of checking cryptographic protocols for faults. In: Advances in Cryptology – Eurocrypt '97, Lecture Notes in Computer Science, vol 1233. Springer, Berlin Heidelberg New York, pp 37–51
14. Brickell E, Pointcheval D, Vaudenay S, Yung M (2000) Design validations for discrete logarithm based signature schemes. In: Public Key Cryptography – Proceedings of PKC 2000, Lecture Notes in Computer Science, vol 1751. Springer, Berlin Heidelberg New York, pp 276–292
15. Brown D (2000) The exact security of ECDSA. Technical report CORR 2000-54, Department of C&O, University of Waterloo. Available from `http://www.cacr.math.uwaterloo.ca`
16. Brown M, Cheung D, Hankerson D, Hernandez J, Kirkup M, Menezes A (2000) PGP in constrained wireless devices. In: Proceedings of the Ninth USENIX Security Symposium, Denver. USENIX Association, pp 247-261
17. Brown M, Hankerson D, Hernandez J, Menezes A (2001) Software implementation of the NIST elliptic curves over prime fields. In: Topics in Cryptology – CT-RSA 2001, Lecture Notes in Computer Science, vol 2020, Springer, Berlin Heidelberg New York, pp 250–265
18. Certicom ECC Challenge (1997) `http://www.certicom.com`
19. Chaum D, Evertse J-H, van de Graaf J (1988) An improved protocol for demonstrating possession of discrete logarithms and some generalizations. In: Advances in Cryptology – Eurocrypt '87, Lecture Notes in Computer Science, vol 304. Springer, Berlin Heidelberg New York, pp 127–141
20. Dierks T, Anderson B (1998) ECC cipher suites for TLS. Internet Draft, Available at `http://www.ietf.org`
21. Diffie W, van Oorschot P, Wiener M (1992) Authentication and authenticated key exchanges. Des Codes Cryptography 2:107–125
22. Dobbertin H, Bosselaers A, Preneel B (1996) RIPEMD-160: a strengthened version of RIPEMD. In: Fast Software Encryp-

tion – FSE '96, Lecture Notes in Computer Science, vol 1039. Springer, Berlin Heidelberg New York, pp 71–82
23. ElGamal T (1985) A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans Inf Theory 31:469–472
24. Enge A (1999) Elliptic curves and their applications to cryptography – an introduction. Kluwer, Boston
25. Escott A, Sager J, Selkirk A, Tsapakidis D (1999) Attacking elliptic curve cryptosystems using the parallel Pollard rho method. CryptoBytes – The Technical Newsletter of RSA Laboratories 4(2):15–19; Also available at `http://www.rsasecurity.com`
26. Fouquet M, Gaudry P, Harley R (2000) On Satoh's algorithm and its implementation, J Ramanujan Math Soc 15:281–318
27. Frey G (1998) How to disguise an elliptic curve (Weil descent). Talk at ECC '98. Workshop on Elliptic Curve Cryptography. Slides available at `http://www.cacr.math.uwaterloo.ca`
28. Frey G (2001) Applications of arithmetical geometry to cryptographic constructions. Proceedings of the Fifth International Conference on Finite Fields and Applications. Springer, Berlin Heidelberg New York, pp 128–161
29. Frey G, Rück H (1994) A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves. Mathematics Comput 62:865–874
30. Galbraith S, Smart N (1999) A cryptographic application of Weil descent. In: Codes and Cryptography, Lecture Notes in Computer Science, vol 1746. Springer, Berlin Heidelberg New York, pp 191-200
31. Gallant R, Lambert R, Vanstone S (2000) Improving the parallelized Pollard lambda search on anomalous binary curves. Mathematics Computation 69:1699–1705
32. Gaudry P, Hess F, Smart N (2000) Constructive and destructive facets of Weil descent on elliptic curves, preprint. Available from `http://www.hpl.hp.com/techreports/2000/HPL-2000-10.html`
33. Goldwasser S, Micali S, Rivest R (1988) A digital signature scheme secure against adaptive chosen message attacks, SIAM J Comput 17:281–308
34. Gordon D (1993) Designing and detecting trapdoors for discrete log cryptosystems. In: Advances in Cryptology – Crypto '92, Lecture Notes in Computer Science, vol 740. Springer, Berlin Heidelberg New York, pp 66-75
35. Gordon D (1993) Discrete logarithms in $GF(p)$ using the number field sieve. SIAM J Discrete Math 6:124–138
36. Gordon D (1998) A survey of fast exponentiation methods. J Algorithms 27:129–146
37. Hankerson D, Hernandez J, Menezes A (2001) Software implementation of elliptic curve cryptography over binary fields. In: Proceedings of CHES 2000. Lecture Notes in Computer Science, vol 1965. Springer, Berlin Heidelberg New York, pp 1–24
38. Hasegawa T, Nakajima J, Matsui M (1998) A practical implementation of elliptic curve cryptosystems over $GF(p)$ on a 16-bit microcomputer. In: Public Key Cryptography – Proceedings of PKC '98, Lecture Notes in Computer Science, vol 1431. Springer, Berlin Heidelberg New York, pp 182–194
39. IEEE 1363 (2000) Standard Specifications for Public-Key Cryptography. `http://grouper.ieee.org/groups/1363/index.html`
40. ISO/IEC 9798-3 (1993) Information technology – security techniques – entity authentication mechanisms. Part 3: Entity authentication using a public-key algorithm, 1st edn. ISO, International Organization for Standardization, Geneva
41. ISO/IEC 11770-3 (1999) Information technology – security techniques – key management. Part 3: Mechanisms using asymmetric techniques. ISO, International Organization for Standardization, Geneva
42. ISO/IEC 14888-3 (1998) Information technology – security techniques – digital signatures with appendix. Part 3: Certificate based-mechanisms. ISO, International Organization for Standardization, Geneva
43. ISO/IEC 15946 (1999) Information technology – security techniques – cryptographic techniques based on elliptic curves, committee draft. ISO, International Organization for Standardization, Geneva
44. Izu T, Kogure J, Noro M, Yokoyama K (1999) Efficient implementation of Schoof's algorithm. In: Advances in Cryptology – Asiacrypt '98, Lecture Notes in Computer Science, vol 1514. Springer, Berlin Heidelberg New York, pp 66–79
45. Jacobson M, Koblitz N, Silverman J, Stein A, Teske E (2000) Analysis of the xedni calculus attack. Des Codes Cryptography 20:41–64.
46. Johnson D (1997) Key validation. Contribution to ANSI X9F1 working group
47. Kaliski B, Yin Y (1999) Storage-efficient finite field basis conversion. In: Selected Areas in Cryptography, Lecture Notes in Computer Science, vol 1556. Springer, Berlin Heidelberg New York, pp 81–93
48. Kelsey J, Schneier B, Wagner D, Hall C (1998) Cryptanalytic attacks on pseudorandom number generators. In: Fast Software Encryption – FSE '98, Lecture Notes in Computer Science, vol 1372. Springer, Berlin Heidelberg New York, pp 168–188
49. Koblitz N (1987) Elliptic curve cryptosystems. Math Comput 48:203–209
50. Koblitz N (1991) Constructing elliptic curve cryptosystems in characteristic 2. In: Advances in Cryptology – Crypto '90, Lecture Notes in Computer Science, vol 537. Springer, Berlin Heidelberg New York, pp 156–167
51. Koblitz N (1992) CM-curves with good cryptographic properties. In: Advances in Cryptology – Crypto '91, Lecture Notes in Computer Science, vol 576. Springer, Berlin Heidelberg New York, pp 279–287
52. Koblitz N (1994) A course in number theory and cryptography, 2nd edn. Springer, Berlin Heidelberg New York
53. Kocher P (1996) Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Advances in Cryptology – Crypto '96, Lecture Notes in Computer Science, vol 1109. Springer, Berlin Heidelberg New York, pp 104–113
54. Kocher P, Jaffe J, Jun B (1999) Differential power analysis. In: Advances in Cryptology – Crypto '99, Lecture Notes in Computer Science, vol 1666. Springer, Berlin Heidelberg New York, pp 388–397
55. Lay G, Zimmer H (1994) Constructing elliptic curves with given group order over large finite fields. In: Algorithmic Number Theory, Lecture Notes in Computer Science, vol 877. Springer, Berlin Heidelberg New York, pp 250–263
56. Lercier R (1996) Computing isogenies in $\mathbb{F}_{2^n}$. In: Algorithmic Number Theory, Lecture Notes in Computer Science, vol 1122. Springer, Berlin Heidelberg New York, pp 197–212
57. Lercier R (1997) Finding good random elliptic curves for cryptosystems defined $\mathbb{F}_{2^n}$. In: Advances in Cryptology – Eurocrypt '97, Lecture Notes in Computer Science, vol 1233. Springer, Berlin Heidelberg New York, pp 379–392
58. Lercier R, Morain F (1995) Counting the number of points on elliptic curves over finite fields: strategies and performances. In: Advances in Cryptology – Eurocrypt '95, Lecture Notes in Computer Science, vol 921. Springer, Berlin Heidelberg New York, pp 79–94
59. Lidl R, Niederreitter H (1984) Introduction to finite fields and their applications, Cambridge University Press, Cambridge
60. Lim C, Lee P (1997) A key recovery attack on discrete log-based schemes using a prime order subgroup. In: Advances in Cryptology – Crypto '97, Lecture Notes in Computer Science, vol 1294. Springer, Berlin Heidelberg New York, pp 249–263
61. McEliece R (1987) Finite fields for computer scientists and engineers. Kluwer, Boston
62. Meier W, Staffelbach O (1993) Efficient multiplication on certain nonsupersingular elliptic curves. In: Advances in Cryptology – Crypto '92, Lecture Notes in Computer Science, vol 740. Springer, Berlin Heidelberg New York, pp 333–344
63. Menezes A (1993) Elliptic curve public key cryptosystems. Kluwer, Boston
64. Menezes A, Okamoto T, Vanstone S Reducing elliptic curve logarithms to logarithms in a finite field. IEEE Trans Inf Theory 39:1639–1646
65. Menezes A, van Oorschot P, Vanstone S Handbook of applied cryptography. CRC Press, Boca Raton, FL
66. Menezes A, Qu M (2001) Analysis of the Weil descent attack of Gaudry, Hess and Smart. In: Topics in Cryptology –

CT-RSA 2001, Lecture Notes in Computer Science, vol 2020, Springer, Berlin Heidelberg New York, pp 308–318
67. Miller V (1986) Uses of elliptic curves in cryptography. In: Advances in Cryptology – Crypto '85, Lecture Notes in Computer Science, vol 218. Springer, Berlin Heidelberg New York, pp 417–426
68. Morain F (1991) Building cyclic elliptic curves modulo large primes. In: Advances in Cryptology – Eurocrypt '91, Lecture Notes in Computer Science, vol 547. Springer, Berlin Heidelberg New York, pp 328–336
69. Mullin R, Onyszchuk I, Vanstone S, Wilson R (1988/89) Optimal normal bases in $GF(p^n)$. Discrete Appl Math 22:149–161
70. National Institute of Standards and Technology (1994) Digital signature standard. FIPS Publication 186, available from `http://csrc.nist.gov/encryption/`
71. National Institute of Standards and Technology (1995) Secure hash standard (SHS). FIPS Publication 180-1, available from `http://csrc.nist.gov/encryption/`
72. National Institute of Standards and Technology (1997) Entity authentication using public key cryptography. FIPS Publication 196, available from `http://csrc.nist.gov/encryption/`
73. National Institute of Standards and Technology (1998) Digital signature standard. FIPS Publication 186-1, available from `http://csrc.nist.gov/encryption/`
74. National Institute of Standards and Technology (2000) Digital signature standard. FIPS Publication 186-2, available from `http://csrc.nist.gov/encryption/`
75. National Institute of Standards and Technology, Advanced Encryption Standard, work in progress, available from `http://csrc.nist.gov/encryption/`
76. National Institute of Standards and Technology (2000) Descriptions of SHA-256, SHA-384, and SHA-512, preprint
77. National Security Agency (1998) SKIPJACK and KEA algorithm specification, Version 2.0, 29 May 1998
78. Nyberg K, Rueppel R (1993) A new signature scheme based on the DSA giving message recovery. In: 1st ACM Conference on Computer and Communications Security, Fairfax, VA. ACM Press, pp 58–61
79. Nyberg K, Rueppel R (1996) Message recovery for signature schemes based on the discrete logarithm problem. Des Codes Cryptography 7:61–81
80. van Oorschot P, Wiener M (1999) Parallel collision search with cryptanalytic applications. J Cryptology 12:1–28
81. Pohlig S, Hellman M (1978) An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. IEEE Trans Inf Theory 24:106–110
82. Pointcheval D, Stern J (1993) Security proofs for signature schemes. In: Advances in Cryptology – Eurocrypt '96, Lecture Notes in Computer Science, vol 1070. Springer, Berlin Heidelberg New York, pp 387–398
83. Pollard J (1978) Monte Carlo methods for index computation mod p. Math Comput 32:918–924
84. Rabin M (1979) Digitalized signatures and public-key functions as intractable as factorization, MIT/LCS/TR-212. MIT Laboratory for Computer Science, Cambridge, MA
85. Rivest R, Shamir A, Adleman L (1978) A method for obtaining digital signatures and public-key cryptosystems. Commun ACM 21:120–126
86. Rueppel R, Lenstra A, Smid M, McCurley K, Desmedt Y, Odlyzko A, Landrock P (1993) The Eurocrypt '92 controversial issue – trapdoor primes and moduli. In: Advances in Cryptology – Eurocrypt '92, Lecture Notes in Computer Science, vol 658. Springer, Berlin Heidelberg New York, pp 194–199
87. Satoh T (2000) The canonical lift of an ordinary elliptic curve over a prime field and its point counting. J Ramanujan Math Soc 15:247–270
88. Satoh T, Araki K (1998) Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves. Comment Math Univ Sancti Pauli 47:81–92
89. Schirokauer O (1993) Discrete logarithms and local units. Philos Trans R Soc London A 345:409–423
90. Schnorr C (1991) Efficient signature generation by smart cards. J Cryptology 4:161–174
91. Schoof R (1985) Elliptic curves over finite fields and the computation of square roots mod p. Math Comput 44:483–494
92. Schroeppel R, Orman H, O'Malley S, Spatscheck O (1995) Fast key exchange with elliptic curve systems. In: Advances in Cryptology – Crypto '95, Lecture Notes in Computer Science, vol 963. Springer, Berlin Heidelberg New York, pp 43–56
93. Semaev I (1998) Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p. Math Comput 67:353–356
94. Silverman J (1986) The arithmetic of elliptic curves. Springer, New York
95. Silverman J (2000) The xedni calculus and the elliptic curve discrete logarithm problem. Des Codes Cryptography20:5–40
96. Silverman J, Suzuki J (1999) Elliptic curve discrete logarithms and the index calculus. Advances in Cryptology – Asiacrypt '98, Lecture Notes in Computer Science, vol 1514. Springer, Berlin Heidelberg New York, pp 110–125
97. Silverman R, Stapleton J (1997) Contribution to ANSI X9F1 working group (unpublished)
98. Smart N (1999) The discrete logarithm problem on elliptic curves of trace one. J Cryptology 12:193–196
99. Smid M, Branstad D (1993) Response to comments on the NIST proposed digital signature standard. Advances in Cryptology – Crypto '92, Lecture Notes in Computer Science, vol 740. Springer, Berlin Heidelberg New York, pp 76–88
100. Solinas J (1997) An improved algorithm for arithmetic on a family of elliptic curves. In: Advances in Cryptology – Crypto '97, Lecture Notes in Computer Science, vol 1294. Springer, Berlin Heidelberg New York, pp 357–371
101. Solinas J (1999) Generalized Mersenne numbers. Technical report CORR 99-39, Department of C&O, University of Waterloo. Available from `http://www.cacr.math.uwaterloo.ca`
102. Solinas J (2000) Efficient arithmetic on Koblitz curves. Des Codes Cryptography 19:195–249
103. Standards for Efficient Cryptography Group (2000) SEC 1: elliptic curve cryptography, version 1.0. Available at `http://www.secg.org`
104. Standards for Efficient Cryptography Group (2000) SEC 2: recommended elliptic curve domain parameters, version 1.0. Available at `http://www.secg.org`
105. Stein A (1997) Equivalences between elliptic curves and real quadratic congruence function fields. J Théor Nombres Bordeaux 9:75–95
106. Stein A, Müller V, Thiel C (1999) Computing discrete logarithms in real quadratic congruence function fields of large genus. Math Comput 68:807–822
107. Teske E (1998) Speeding up Pollard's rho method for computing discrete logarithms. In: Algorithmic Number Theory, Lecture Notes in Computer Science, vol 1423. Springer, Berlin Heidelberg New York, pp 541–554
108. Vanstone S (1992) Responses to NIST's proposal. Commun ACM 35:50–52
109. Vaudenay S (1996) Hidden collisions on DSS. In: Advances in Cryptology – Crypto '96, Lecture Notes in Computer Science, vol 1109. Springer, Berlin Heidelberg New York, pp 83–88
110. WAP WTLS (1999) Wireless application protocol wireless transport layer security specification. Wireless Application Protocol Forum. Drafts available at `http://www.wapforum.org`
111. Wiener M, Zuccherato R (1999) Faster attacks on elliptic curve cryptosystems. In: Selected Areas in Cryptography, Lecture Notes in Computer Science, vol 1556. Springer, Berlin Heidelberg New York, pp 190–200
112. De Win E, Mister S, Preneel B, Wiener M (1998) On the performance of signature schemes based on elliptic curves. In: Algorithmic Number Theory, Lecture Notes in Computer Science, vol 1423. Springer, Berlin Heidelberg New York, pp 252–266
113. Zuccherato R (1998) The equivalence between elliptic curve and quadratic function field discrete logarithms in characteristic 2. In: Algorithmic Number Theory, Lecture Notes in Computer Science, vol 1423. Springer, Berlin Heidelberg New York, pp 621–638