Accredited Standards Committee X9
Title: X9-Financial Services
Accredited by the
American National Standards Institute

September 20, 1998

Working Draft
AMERICAN NATIONAL STANDARD
X9.62-1998
Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)©

Notice -- This document is a draft document. It has not yet been processed through the consensus procedures of X9 and ANSI.
Many changes which may greatly affect its contents can occur before this document is completed. The working group may not be held responsible for the contents of this document.
Implementation or design based on this draft is at the risk of the user. No advertisement or citation implying compliance with a "Standard" should appear as it is erroneous and misleading to so state.
Copies of the draft proposed American National Standard will be available from the X9 Secretariat when the document is finally announced for two months public comment. Notice of this announcement will be in the trade press.

Secretariat: American Bankers Association
Standards Department
1120 Connecticut Ave., N.W.
Washington, DC 20036

Foreword

(Informative)

Business practice has changed with the introduction of computer-based technologies. The substitution of electronic transactions for their paper-based predecessors has reduced costs and improved efficiency. Trillions of dollars in funds and securities are transferred daily by telephone, wire services, and other electronic communication mechanisms. The high value or sheer volume of such transactions within an open environment exposes the financial community and its customers to potentially severe risks from accidental or deliberate alteration, substitution or destruction of data. This risk is compounded by interconnected networks, and the increased number and sophistication of malicious adversaries.

Some of the conventional "due care" controls used with paper-based transactions are unavailable in electronic transactions. Examples of such controls are safety paper which protects integrity, and handwritten signatures or embossed seals which indicate the intent of the originator to be legally bound. In an electronic-based environment, controls must be in place that provide the same degree of assurance and certainty as in a paper environment. The financial community is responding to these needs.

This Standard, X9.62-1998, *Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm* (*ECDSA*), defines a technique for generating and validating digital signatures.

This Standard describes a method for digital signatures using the elliptic curve analog of the Digital Signature Algorithm (DSA) (ANSI X9.30 Part 1 [3]).

Elliptic curve systems are public-key (asymmetric) cryptographic algorithms that are typically used:

1. to create digital signatures (in conjunction with a hash algorithm), and
2. to establish secret keys securely for use in symmetric-key cryptosystems.

When implemented with proper controls, the techniques of this Standard provide:

1. data integrity,
2. data origin authentication, and
3. non-repudiation of the message origin and the message contents.

Additionally, when used in conjunction with a Message Identifier (ANSI X9.9 [2]), the techniques of this Standard provide the capability of detecting duplicate transactions. It is the Committee's belief that the proper implementation of this Standard should also contribute to the enforceability of some legal obligations.

The use of this Standard, together with appropriate controls, may have a legal effect, including the apportionment of liability for erroneous or fraudulent transactions and the satisfaction of statutory or contractual "due care" requirements. The legal implications associated with the use of this Standard may be affected by case law and legislation, including the Uniform Commercial Code Article 4A on Funds Transfers (Article 4A).

The details of Article 4A address, in part, the use of commercially reasonable security procedures and the effect of using such procedures on the apportionment of liability between a customer and a bank. A security procedure is provided by Article 4A-201 "for the purpose of (i) verifying that a payment order or communication amending or canceling a payment order originated is that of the customer, or (ii) detecting an error in the transmission or the content of the payment order or communication." The commercial reasonableness of a security procedure is determined by the criteria established in Article 4A-201.

While the techniques specified in this Standard are designed to maintain the integrity of financial messages and provide the service of non-repudiation, the Standard does not guarantee that a particular implementation is secure. It is the responsibility of the financial institution to put an overall process in place with the necessary controls to ensure that the process is securely implemented. Furthermore, the controls should include the application of appropriate audit tests in order to verify compliance with this Standard.

Suggestions for the improvement or revision of this Standard are welcome. They should be sent to the X9 Committee Secretariat, American Bankers Association, 1120 Connecticut Avenue, N.W., Washington D.C. 20036.

This Standard was processed and approved for submittal to ANSI by the Accredited Standards Committee on Financial Services, X9. Committee approval of the Standard does not necessarily imply that all the committee members voted for its approval. At the time that this Standard was approved, the X9 Committee had the following members:

Harold Deal, Chairman
Alice Droogan, Vice Chairman
Cynthia Fuller, Secretariat

Organization Represented,	**Representative**
[to be furnished]	

The X9F subcommittee on Data and Information Security had the following members:
Glenda Barnes, Chairman

Organization Represented,	**Representative**
[to be furnished]	

The X9F1 working group which developed this standard had the following members:
M. Blake Greenlee, Chairman

Organization Represented,	**Representative**
[to be furnished]	

Contents

Figures

Tables

X9.62-1998, Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)

1 Scope

This Standard defines methods for digital signature (signature) generation and verification for the protection of messages and data using the Elliptic Curve Digital Signature Algorithm (ECDSA). ECDSA is the elliptic curve analogue of the Digital Signature Algorithm (ANSI X9.30 Part 1 [3]); see Annex G.

The ECDSA shall be used in conjunction with the hash function SHA-1 defined in ANSI X9.30 Part 2 [4]. In addition, this ECDSA Standard provides the criteria for the generation of public and private keys that are required by the algorithm and the procedural controls required for the secure use of the algorithm.

2 Definitions, Abbreviations and References

2.1 Definitions and Abbreviations

addition rule

An *addition rule* describes the addition of two elliptic curve points P_1 and P_2 to produce a third elliptic curve point P_3. (See Annexes B.3 and B.4.)

asymmetric cryptographic algorithm

A cryptographic algorithm that uses two related keys, a public key and a private key; the two keys have the property that, given the public key, it is computationally infeasible to derive the private key.

base point (*G*)

A distinguished point on an elliptic curve of large prime order n.

basis

A representation of the elements of the finite field F_{2^m}. Two special kinds of basis are *polynomial basis* and *normal basis.* (See Annex B.2.)

binary polynomial

A polynomial whose coefficients are in the field F_2. When adding, multiplying, or dividing two binary polynomials, the coefficient arithmetic is performed modulo 2.

bit string

A bit string is an ordered sequence of 0's and 1's.

certificate

The public key and identity of an entity together with some other information, rendered unforgeable by signing the certificate with the private key of the Certification Authority which issued that certificate. In this Standard the term certificate shall mean a public-key certificate.

Certification Authority (CA)

A Center trusted by one or more entities to create and assign certificates.

characteristic 2 finite field

A finite field containing 2^m elements, where $m \geq 1$ is an integer.

compressed form

Octet string representation for a point using the point compression technique described in Section 4.2. (See also Section 4.3.6.)

cryptographic hash function

A (mathematical) function which maps values from a large (possibly very large) domain into a smaller range. The function satisfies the following properties:

1. it is computationally infeasible to find any input which maps to any pre-specified output;
2. it is computationally infeasible to find any two distinct inputs which map to the same output.

cryptographic key (key)

A parameter that determines the operation of a cryptographic function such as:

1. the transformation from plaintext to ciphertext and vice versa,
2. the synchronized generation of keying material,
3. a digital signature computation or verification.

cryptography

The discipline which embodies principles, means and methods for the transformation of data in order to hide its information content, prevent its undetected modification, prevent its unauthorized use, or a combination thereof.

cryptoperiod

The time span during which a specific key is authorized for use or in which the keys for a given system may remain in effect.

cyclic group

The group of points $E(F_q)$ is said to be *cyclic* if there exists a point $P \in E(F_q)$ of order n, where $n = \#E(F_q)$. In this case, $E(F_q) = \{kP: 0 \le k \le n-1\}$.

digital signature

The result of a cryptographic transformation of data which, when properly implemented, provides the services of:

1. origin authentication,
2. data integrity, and
3. signer non-repudiation.

ECDLP

Elliptic Curve Discrete Logarithm Problem. (See Annex H.)

ECDSA

Elliptic Curve Digital Signature Algorithm.

elliptic curve

An *elliptic curve* over F_q is a set of points which satisfy a certain equation specified by 2 parameters a and b, which are elements of a field F_q. (See Section 4.2.)

elliptic curve key pair (Q, d)

Given particular elliptic curve domain parameters, an *elliptic curve key pair* consists of an elliptic curve public key (Q) and the corresponding elliptic curve private key (d).

elliptic curve private key (d)

Given particular elliptic curve domain parameters, an *elliptic curve private key, d,* is a statistically unique and unpredictable integer in the interval [1, n-1], where n is the prime order of the base point G.

elliptic curve public key (Q)

Given particular elliptic curve domain parameters, and an elliptic curve private key d, the corresponding *elliptic curve* public *key, Q,* is the elliptic curve point $Q = dG$, where G is the base point. Note that Q will never equal $\mathcal{O}$, since $1 \le d \le n-1$.

elliptic curve domain parameters

Elliptic curve domain parameters are comprised of a field size q, indication of basis used (in the case $q = 2^m$), an optional SEED, two elements a, b in F_q which define an elliptic curve E over F_q, a point $G = (x_G, y_G)$ of prime order in $E(F_q)$, the order n of G, and the cofactor h.

See Sections 5.1.1.1 and 5.1.2.1 for a complete specification of elliptic curve domain parameters.

elliptic curve point

If E is an elliptic curve defined over a field F_q, then an *elliptic curve point P* is either: a pair of field elements (x_p, y_p) (where $x_p, y_p \in F_q$) such that the values $x = x_p$ and $y = y_p$ satisfy the equation defining E, or a special point $\mathcal{O}$ called the *point at infinity*.

Gaussian normal basis (GNB)

A type of normal basis that can be used to represent the elements of the finite field F_{2^m}. (See Section 4.1.2.2.)

hash function

See cryptographic hash function.

hash value

The result of applying a cryptographic hash function to a message.

hybrid form
Octet string representation for both the compressed and uncompressed forms of an elliptic curve point. (See Section 4.3.6.)
irreducible binary polynomial
A binary polynomial $f(x)$ is *irreducible* if it does not factor as a product of two or more binary polynomials, each of degree less than the degree of $f(x)$.
key
See cryptographic key.
keying material
The data (e.g., keys, certificates and initialization vectors) necessary to establish and maintain cryptographic keying relationships.
message
The data to be signed.
message identifier (MID)
A field which may be used to identify a message. Typically, this field is a sequence number.
non-repudiation
This service provides proof of the integrity and origin of data which can be verified by a third party.
normal basis (NB)
A type of basis that can be used to represent the elements of the finite field F_{2^m}. (See Annex B.2.3.)
octet
An *octet* is a bit string of length 8. An octet is represented by a hexadecimal string of length 2. The first hexadecimal digit represents the four leftmost bits of the octet, and the second hexadecimal digit represents the four rightmost bits of the octet. For example, 9*D* represents the bit string 10011101. An octet also represents an integer in the interval [0, 255]. For example, 9*D* represents the integer 157.
octet string
An octet string is an ordered sequence of octets.
optimal normal basis (ONB)
A type of Gaussian normal basis that can be used to represent the elements of the finite field F_{2^m}. (See Section 4.1.2.2.) There are two kinds of ONB, called Type I ONB and Type II ONB.
order of a curve
The *order of an elliptic curve E* defined over the field F_q is the number of points on E, including $\mathcal{O}$. This is denoted by $\#E(F_q)$.
order of a point
The *order of a point P* is the smallest positive integer n such that $nP = \mathcal{O}$ (the point at infinity).
owner
The entity whose identity is associated with a private/public key pair.
pentanomial
A polynomial of the form $x^m + x^{k3} + x^{k2} + x^{k1} + 1$, where $1 \leq k1 < k2 < k3 \leq m-1$.
pentanomial basis (PPB)
A type of polynomial basis that can be used to represent the elements of the finite field F_{2^m}. (See Annex B.2.2.)
point compression
Point compression allows a point $P = (x_p, y_p)$ to be represented compactly using x_p and a single additional bit $\tilde{y}_p$ derived from x_p and y_p. (See Section 4.2.)
polynomial basis (PB)
A type of basis that can be used to represent the elements of the finite field F_{2^m}. (See Annex B.2.1.)
prime finite field
A finite field containing p elements, where p is an odd prime number.
private key
In an asymmetric (public) key system, that key of an entity's key pair which is known only by that entity.
public key
In an asymmetric key system, that key of an entity's key pair which is publicly known.

reduction polynomial
The irreducible binary polynomial $f(x)$ of degree m that is used to determine a polynomial basis representation of F_{2^m}.
scalar multiplication
If k is a positive integer, then kP denotes the point obtained by adding together k copies of the point P. The process of computing kP from P and k is called *scalar multiplication*.
Secure Hash Algorithm, Revision 1 (SHA-1)
SHA-1 implements a hash function which maps messages of a length less than 2^{64} bits to hash values of a length which is exactly 160 bits.
SEED
Random value input into a pseudo-random bit generator (PRBG) algorithm.
signatory
The entity that generates a digital signature on data.
statistically unique
For the generation of n-bit quantities, the probability of two values repeating is less than or equal to the probability of two n-bit random quantities repeating.
trinomial
A polynomial of the form $x^m + x^k + 1$, where $1 \leq k \leq m\text{-}1$.
trinomial basis (TPB)
A type of polynomial basis that can be used to represent the elements of the finite field F_{2^m}. (See Annex B.2.2.)
type I ONB
A kind of optimal normal basis. (See Section 4.1.2.2.)
type II ONB
A kind of optimal normal basis. (See Section 4.1.2.2.)
uncompressed form
Octet string representation for an uncompressed elliptic curve point. (See Section 4.3.6.)
valid elliptic curve domain parameters
A set of elliptic curve domain parameters that have been validated using the method specified in Section 5.1.1.2 or Section 5.1.2.2.
verifier
The entity that verifies the authenticity of a digital signature.
XOR
Bitwise exclusive-or (also bitwise addition mod 2) of two bit strings of the same bit length.
***x*-coordinate**
The *x-coordinate* of an elliptic curve point, $P = (x_p, y_p)$, is x_p.
***y*-coordinate**
The *y-coordinate* of an elliptic curve point, $P = (x_p, y_p)$, is y_p.

2.2 Symbols and Notation

$[x, y]$	The interval of integers between and including x and y.
$\lceil x \rceil$	Ceiling: the smallest integer $\geq x$. For example, $\lceil 5 \rceil = 5$ and $\lceil 5.3 \rceil = 6$.
$\lfloor x \rfloor$	Floor: the largest integer $\leq x$. For example, $\lfloor 5 \rfloor = 5$ and $\lfloor 5.3 \rfloor = 5$.
$x \bmod n$	The unique remainder r, $0 \leq r \leq n - 1$, when integer x is divided by n. For example, 23 mod 7 = 2.
$x \equiv y \pmod{n}$	x is congruent to y modulo n. That is, $(x \bmod n) = (y \bmod n)$.
$x^{-1} \bmod n$	If $\gcd(x, n) = 1$, then $x^{-1} \bmod n$ is the unique integer y, $1 \leq y \leq n - 1$, such that $xy \equiv 1 \pmod{n}$.
a, b	Elements of F_q that define an elliptic curve E over F_q.
B	MOV threshold. A positive integer B such that taking discrete logarithms over F_{q^B} is at least as difficult as taking elliptic curve logarithms over F_q. For this Standard, B shall be ≥ 20.
d	Elliptic curve private key.
e	Result of applying hash function to message M.
e'	Result of applying hash function to message M'.

E	An elliptic curve over the field F_q defined by a and b.
$E(F_q)$	The set of all points on an elliptic curve E defined over F_q and including the point at infinity $\mathcal{O}$.
$\#E(F_q)$	If E is defined over F_q, then $\#E(F_q)$ denotes the number of points on the curve (including the point at infinity $\mathcal{O}$). $\#E(F_q)$ is called the order of the curve E.
F_{2^m}	The finite field containing $q = 2^m$ elements, where m is a positive integer.
F_p	The finite field containing $q = p$ elements, where p is a prime.
F_q	The finite field containing q elements. For this Standard, q shall either be an odd prime number ($q = p$, $p > 3$) or a power of 2 ($q = 2^m$).
G	A distinguished point on an elliptic curve called the *base point* or *generating point*.
gcd(x, y)	The greatest common divisor of integers x and y.
h	$h = \#E(F_q)/n$, where n is the order of the base point G. h is called the *cofactor*.
k	Per-message secret value. For this Standard, k shall be a statistically unique and unpredictable integer in the interval [1, n-1].
l	The length of a field element in octets; $l = \lceil t / 8 \rceil$.
l_{max}	Upper bound on the largest prime divisor of the cofactor h.
$\log_2 x$	The logarithm of x to the base 2.
m	The *degree* of the finite field F_{2^m}.
M	Message to be signed.
M'	Message as received.
MID	Message Identifier.
mod	Modulo.
mod $f(x)$	Arithmetic modulo the polynomial $f(x)$. If $f(x)$ is a binary polynomial, then all coefficient arithmetic is performed modulo 2.
mod n	Arithmetic modulo n.
n	The order of the base point G. For this Standard, n shall be greater than 2^{160} and $4\sqrt{q}$, and shall be a prime number. n is the primary security parameter. The strength of ECDSA rests on two fundamental assumptions, the difficulty of finding a collision using the one-way hash function and the difficulty of solving the ECDLP. The difficulty of finding a collision using SHA-1 is thought to take 2^{80} steps. The difficulty of solving the ECDLP is related to the size of n – as n increases, the difficulty of the ECDLP increases. See Annex H for more information.
$\mathcal{O}$	A special point on an elliptic curve, called the point at infinity. This is the additive identity of the elliptic curve group.
p	An odd prime number.
q	The number of elements in the field F_q.
Q	Elliptic Curve public key.
r_{min}	Lower bound on the desired (prime) order n of the base point G. For this Standard r_{min} shall be $>2^{160}$.
t	The length of a field element in bits; $t = \lceil \log_2 q \rceil$. In particular, if $q = 2^m$, then a field element in F_{2^m} can be represented as a bit string of bit length $t = m$.
T	In the probabilistic primality test (Annex A.2.1), the number of independent test rounds to execute. For this Standard T shall be ≥ 50.
Tr	Trace function. (See Annex D.1.5.)
x_p	The x-coordinate of a point P.
$\|\|X\|\|$	Length in octets of the octet string X.
$X\|\|Y$	Concatenation of two strings X and Y. X and Y are either both bit strings, or both octet strings.
$X \oplus Y$	Bitwise exclusive-or (also bitwise addition mod 2) of two bit strings X and Y of the same bit length.
y_p	The y-coordinate of a point P.
$\tilde{y}_p$	The representation of the y-coordinate of a point P when point compression is used.
Z_p	The set of integers modulo p, where p is an odd prime number.

2.3 References

The following standards contain provisions which, through reference in this text, constitute provisions of this American National Standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this American National Standard are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below. Accredited Standards Committee X9 (ASC X9) maintains a register of currently valid financial industry standards.

ANSI X3.92-1981, *Data Encryption Algorithm*.

ANSI X9.30-1993, Part 2: *Public key cryptography using irreversible algorithms for the financial services industry*: *The Secure Hash Algorithm* 1 (*SHA*-1) (*Revised*).

3 Application

3.1 General

When information is transmitted from one party to another, the recipient may desire to know that the information has not been altered in transit. Furthermore, the recipient may wish to be certain of the originator's identity. The use of public-key cryptography digital signatures can provide assurance (1) of the identity of the signer, and (2) that the received message has not been altered during transmission.

A digital signature is an electronic analog to a written signature. The digital signature may be used in proving to a third party that the information was, in fact, signed by the claimed originator. Unlike their written counterparts, digital signatures also verify the integrity of information. Digital signatures may also be generated for stored data and programs so that the integrity of the data and programs may be verified at any later time.

3.2 The Use of the ECDSA Algorithm

The ECDSA is used by a *signatory* to generate a digital signature on data and by a *verifier* to verify the authenticity of the signature. Each signatory has a public and private key. The private key is used in the signature generation process, and the public key is used in the signature verification process. For both signature generation and verification, the message, M, is compressed by means of the Secure Hash Algorithm (SHA-1) specified in ANSI X9.30 Part 2 [4], prior to the signature generation and verification process.

An adversary, who does not know the private key of the signatory, cannot feasibly generate the correct signature of the signatory. In other words, signatures cannot be forged. However, by using the signatory's public key, anyone can verify a validly signed message.

The user of the public key of a private/public key pair requires assurance that the public key represents the owner of that key pair. That is, there must be a binding of an owner's identity and the owner's public key. This binding may be certified by a mutually trusted party. This may be accomplished by using a Certification Authority which generates a certificate in accordance with ANSI X9.57 [5].

This Standard provides the capability of detecting duplicate messages and preventing the acceptance of replayed messages when the signed message includes:

1. the identity of the intended recipient, and
2. a message identifier (MID).

The MID shall not repeat during the cryptoperiod of the underlying private/public key pair. Annex F of ANSI X9.9 [2] provides information on the use of unique MIDs.

3.3 Control of Keying Material

The signatory shall provide and maintain the proper control of all keying material. In the ECDSA asymmetric cryptographic system, the integrity of signed data is dependent upon:

1. the prevention of unauthorized disclosure, use, modification, substitution, insertion, and deletion of the private key, *d*, the per-message value, *k*, and (optional) seeds input to their generation, and
2. the prevention of unauthorized modification, substitution, insertion, and deletion of elliptic curve domain parameters for the ECDSA (see Section 5.1) computation procedures.

Therefore, if d is disclosed, the integrity of any message signed using that d can no longer be assured. Similarly, the values for the elliptic curve domain parameters must be protected.

NOTE— Key generation should be performed on physically isolated equipment such that in the event of a hardware or software failure, no partial information is retained. For example, if a system crash causes a core dump, some of the keying material data may be captured.

3.4 Annexes

The Annexes to this Standard provide additional requirements and information on the ECDSA and its implementation.

The following Normative annex is an integral part of the standard which, for reasons of convenience, is placed after all other normative elements.

Annex	Contents
A	Normative Number-Theoretic Algorithms

The following Informative annexes give additional information which may be useful to implementors of this Standard.

Annex	Contents
B	Mathematical Background
C	Tables of Trinomials, Pentanomials and Gaussian Normal Bases
D	Informative Number-Theoretic Algorithms
E	Complex Multiplication (CM) Elliptic Curve Generation Method
F	An Overview of Elliptic Curve Systems
G	The Elliptic Curve Analog of the DSA (ECDSA)
H	Security Considerations
I	Small Examples of the ECDSA
J	Examples of ECDSA and Sample Curves
K	References

4 Mathematical Conventions

4.1 Finite Field Arithmetic

This section describes the representations that shall be used for the purposes of conversion for the elements of the underlying finite field F_q. For this Standard, q shall either be an odd prime ($q = p$, $p > 3$) or a power of 2 ($q = 2^m$). Implementations with different internal representations that produce equivalent results are allowed. Mathematics background and examples are provided in Annex B.

4.1.1 The Finite Field F_p

If $q = p$ is an odd prime, then the elements of the finite field F_p shall be represented by the integers 0, 1, 2, ..., $p-1$.

1. The multiplicative identity element is the integer 1.
2. The zero element is the integer 0.
3. Addition of field elements is integer addition modulo p: that is, if $a, b \in F_p$, then $a + b = (a + b) \bmod p$.

4. Multiplication of field elements is integer multiplication modulo p: that is, if $a, b \in F_p$, then $a \cdot b = (a \cdot b) \bmod p$.

4.1.2 The Finite Field F_{2^m}

If $q = 2^m$, then the elements of the finite field F_{2^m} shall be represented by the bit strings of bit length m.

There are numerous methods for interpreting the elements of the finite field F_{2^m}. Two such methods are a *polynomial basis* (PB) *representation* (see Annex B.2.1) and a *normal basis* (NB) *representation* (see Annex B.2.3). A *trinomial basis* (TPB) and a *pentanomial basis* (PPB) are special types of polynomial bases; these bases are described in Section 4.1.2.1. A *Gaussian normal basis* (GNB) is a special type of normal basis; these bases are described in Section 4.1.2.2.

One of TPB, PPB, or GNB shall be used as the basis for representing the elements of the finite field F_{2^m} in implementing this Standard, as described in Sections 4.1.2.1 and 4.1.2.2.

NOTES:

1. TPB, PPB, and GNB have been chosen because they are apparently the most common representations currently used for F_{2^m} over F_2, and because they lead to efficient arithmetic for F_{2^m} over F_2.
2. An *optimal normal basis* (ONB) is a special type of *Gaussian normal basis* that yields efficient field arithmetic. Table C-4 in Annex C lists the values of m, $160 \le m \le 2000$, for which the field F_{2^m} has both an ONB representation and a TPB representation.
3. Annex D.2.3 describes one method for converting the elements of F_{2^m} from one representation to another.
4. When doing computations in F_{2^m}, all integer arithmetic is performed modulo 2.

4.1.2.1 Trinomial and Pentanomial Basis Representation

A *polynomial basis representation* of F_{2^m} over F_2 is determined by an irreducible binary polynomial $f(x)$ of degree m; $f(x)$ is called the *reduction polynomial*. The set of polynomials $\{x^{m-1}, x^{m-2}, \ldots, x, 1\}$ forms a basis of F_{2^m} over F_2, called a *polynomial basis*. The elements of F_{2^m} are the bit strings of a bit length which is exactly m. A typical element $a \in F_{2^m}$ is represented by the bit string $a = (a_{m-1}a_{m-2} \ldots a_1a_0)$, which corresponds to the polynomial $a(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_1x + a_0$.

1. The multiplicative identity element (1) is represented by the bit string (00…001).
2. The zero element (0) is represented by the bit string of all 0's.
3. Addition of two field elements is accomplished by XORing the bit strings.
4. Multiplication of field elements a and b is defined as follows. Let $r(x)$ be the remainder polynomial obtained upon dividing the product of the polynomials $a(x)$ and $b(x)$ by $f(x)$ over F_2 (i.e. the coefficient arithmetic is performed modulo 2). Then $a \cdot b$ is defined to be the bit string corresponding to the polynomial $r(x)$.

See Annex B.2.1 for further details and an example of a polynomial basis representation.

A *trinomial* over F_2 is a polynomial of the form $x^m + x^k + 1$, where $1 \le k \le m-1$. A *pentanomial* over F_2 is a polynomial of the form $x^m + x^{k3} + x^{k2} + x^{k1} + 1$ where $1 \le k1 < k2 < k3 \le m-1$.

A *trinomial basis representation* of F_{2^m} is a polynomial basis representation determined by an irreducible trinomial $f(x) = x^m + x^k + 1$ of degree m over F_2. Such trinomials only exist for certain values of m. Table C-2 in Annex C lists an irreducible trinomial of degree m over F_2 for each m, $160 \le m \le 2000$, for which an irreducible trinomial of degree m exists. For each such m, the table lists the smallest k for which $x^m + x^k + 1$ is irreducible over F_2.

A *pentanomial basis representation* of F_{2^m} is a polynomial basis representation determined by an irreducible pentanomial $f(x) = x^m + x^{k3} + x^{k2} + x^{k1} + 1$ of degree m over F_2. Such pentanomials exist for all values of $m \ge 4$.

Table C-3 in Annex C lists an irreducible pentanomial of degree m over F_2 for each m, $160 \le m \le 2000$, for which an irreducible trinomial of degree m does not exist. For each such m, the table lists the triple ($k1$, $k2$, $k3$) for which (i) $x^m + x^{k3} + x^{k2} + x^{k1} + 1$ is irreducible over F_2; (ii) $k1$ is as small as possible; (iii) for this particular value of $k1$, $k2$ is as small as possible; and (iv) for these particular values of $k1$ and $k2$, $k3$ is as small as possible.

Rules for selecting the polynomial basis

1. If a polynomial basis representation is used for F_{2^m} where there exists an irreducible trinomial of degree m over F_2, then the reduction polynomial $f(x)$ shall be an irreducible trinomial of degree m over F_2. To maximize the chances for interoperability, the reduction polynomial used should be $x^m + x^k + 1$ for the smallest possible k. Examples of such polynomials are given in Table C-2 in Annex C.
2. If a polynomial basis representation is used for F_{2^m} where there does not exist an irreducible trinomial of degree m over F_2, then the reduction polynomial $f(x)$ shall be an irreducible pentanomial of degree m over F_2. To maximize the chances for interoperability, the reduction polynomial used should be $x^m + x^{k3} + x^{k2} + x^{k1} + 1$, where (i) $k1$ is as small as possible; (ii) for this particular value of $k1$, $k2$ is as small as possible; and (iii) for these particular values of $k1$ and $k2$, $k3$ is as small as possible. Examples of such polynomials are given in Table C-3 in Annex C.

4.1.2.2 Gaussian Normal Basis Representation

A *normal basis* for F_{2^m} over F_2 is a basis of the form $N = \{\alpha, \alpha^2, \alpha^{2^2}, \ldots, \alpha^{2^{m-1}}\}$, where $\alpha \in F_{2^m}$. Normal basis representations have the computational advantage that squaring an element can be done very efficiently (see Annex B.2.3). Multiplying distinct elements, on the other hand, can be cumbersome in general. For this reason, it is common to specialize to a class of normal bases, called *Gaussian normal bases*, for which multiplication is both simpler and more efficient.

Gaussian normal bases for F_{2^m} exist whenever m is not divisible by 8. The *type* of a Gaussian normal basis 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 F_{2^m} can have at most one Gaussian normal basis of type T. Thus it is proper to speak of *the* type T Gaussian normal basis over F_{2^m}. The Gaussian normal bases of types 1 and 2 have the most efficient multiplication rules of all normal bases. For this reason, they are called *optimal* normal bases. The type 1 Gaussian normal bases are called *Type I optimal normal bases*, and the type 2 Gaussian normal bases are called *Type II optimal normal bases*.

The elements of the finite field F_{2^m} are the bit strings of bit length which is exactly m. A typical element $a \in F_{2^m}$ is represented by the bit string $a = (a_0 a_1 \ldots a_{m-2} a_{m-1})$.

1. The multiplicative identity element (1) is represented by the bit string of all 1's.
2. The zero element (0) is represented by the bit string of all 0's.
3. Addition of two field elements is accomplished by XORing the bit strings.
4. Multiplication of field elements is described in Sections 4.1.2.2.2 and 4.1.2.2.3.

Rules for selecting the normal basis representation

1. If there exists a GNB of type 2 for F_{2^m}, then this basis shall be used.
2. If there does not exist a GNB of type 2 for F_{2^m}, but there does exist a GNB of type 1, then the type 1 GNB shall be used.
3. If neither a type 1 GNB nor a type 2 GNB exists for F_{2^m}, then the GNB of smallest type shall be used.

Table C-1 in Annex C lists the type of the GNB that shall be used for F_{2^m} for each m, $160 \le m \le 2000$, for which m is not divisible by 8.

4.1.2.2.1 Checking for a Gaussian Normal Basis

If $m > 1$ is not divisible by 8, the following algorithm tests for the existence of a Gaussian normal basis for F_{2^m} of a given type.

Input: An integer $m > 1$ not divisible by 8; a positive integer T.

Output: If a type T Gaussian normal basis for F_{2^m} exists, the message "true"; otherwise "false."

1. Set $p = Tm + 1$.
2. If p is not prime then output "false" and stop.
3. Compute via Annex D.1.8 the order k of 2 modulo p.
4. Set $h = Tm / k$.

5. Compute $d = \gcd(h, m)$.
6. If $d = 1$ then output “true”; else output “false”.

4.1.2.2.2 The Multiplication Rule for a Gaussian Normal Basis

The following procedure produces the rule for multiplication with respect to a given Gaussian normal basis.

Input: Integers $m > 1$ and T for which there exists a type T Gaussian normal basis B for F_{2^m}.

Output: An explicit formula for the first coordinate of the product of two elements with respect to B.

1. Set $p = Tm + 1$.
2. Generate via Annex D.1.9 an integer u having order T modulo p.
3. Compute the sequence $F(1), F(2), \ldots, F(p\text{–}1)$ as follows:
   - 3.1 Set $w = 1$.
   - 3.2 For j from 0 to T–1 do
     - Set $n = w$.
     - For i from 0 to m–1 do
       - Set $F(n) = i$.
       - Set $n = 2n \bmod p$.
     - Set $w = uw \bmod p$.
4. If T is even, then set $J = 0$, else set

$$J = \sum_{k=1}^{m/2} \left(a_{k-1} b_{m/2+k-1} + a_{m/2+k-1} b_{k-1} \right)$$

5. Output the formula

$$c_0 = J + \sum_{k=1}^{p-2} a_{F(k+1)} b_{F(p-k)}$$

4.1.2.2.3 A Multiplication Algorithm for a Gaussian Normal Basis

The formula given in Section 4.1.2.2.2 for c_0 can be used to multiply field elements as follows. For

$u = (u_0\ u_1 \ldots u_{m-1})$, $v = (v_0\ v_1 \ldots v_{m-1})$,

let $F(u, v)$ be the expression derived with $c_0 = F(a, b)$.

Then the product $(c_0\ c_1 \ldots c_{m-1}) = (a_0\ a_1 \ldots a_{m-1}) \times (b_0\ b_1 \ldots b_{m-1})$ can be computed as follows.

1. Set $(u_0\ u_1 \ldots u_{m-1}) = (a_0\ a_1 \ldots a_{m-1})$.
2. Set $(v_0\ v_1 \ldots v_{m-1}) = (b_0\ b_1 \ldots b_{m-1})$.
3. For k from 0 to $m - 1$ do
   - 3.1 Compute $c_k = F(u, v)$.
   - 3.2 Set u = `LeftShift`(u) and v = `LeftShift`(v), where `LeftShift` denotes the circular left shift operation.
4. Output $c = (c_0\ c_1 \ldots c_{m-1})$.

4.2 Elliptic Curves and Points

An *elliptic curve E defined over* F_q is a set of points $P = (x_p, y_p)$ where x_p and y_p are elements of F_q that satisfy a certain equation, together with the *point at infinity* denoted by $\mathcal{O}$. F_q is sometimes called the *underlying field*.

If $q = p$ is an odd prime (so the underlying field is F_p) and $p > 3$, then a and b shall satisfy $4a^3 + 27b^2 \not\equiv 0 \pmod p$, and every point $P = (x_p, y_p)$ on E (other than the point $\mathcal{O}$) shall satisfy the following equation in F_p:

$$y_p^2 = x_p^3 + ax_p + b.$$

If $q = 2^m$ is a power of 2 (so the underlying field is F_{2^m}), then b shall be non-zero in F_{2^m}, and every point $P = (x_p, y_p)$ on E (other than the point $\mathcal{O}$) shall satisfy the following equation in F_{2^m}:

$$y_p^2 + x_p y_p = x_p^3 + ax_p^2 + b.$$

For further background on elliptic curves, see Annex B.3 and B.4.

An elliptic curve point P (which is not the point at infinity $\mathcal{O}$) is represented by two field elements, the x-coordinate of P and the y-coordinate of P: $P = (x_p, y_p)$. The point can be represented compactly by storing only the x-coordinate x_p and a certain bit $\tilde{y}_p$ derived from the x-coordinate x_p and the y-coordinate y_p. The next subsections describe the technique that shall be used to recover the full y-coordinate y_p from x_p and $\tilde{y}_p$, if point compression is used.

4.2.1 Point Compression Technique for Elliptic Curves over F_p (Optional)

Let $P = (x_p, y_p)$ be a point on the elliptic curve $E : y^2 = x^3 + ax + b$ defined over a prime field F_p. Then $\tilde{y}_p$ is defined to be the rightmost bit of y_p.

When the x-coordinate x_p of P and the bit $\tilde{y}_p$ are provided, then y_p can be recovered as follows.

1. Compute the field element $\alpha = x_p^3 + ax_p + b \bmod p$.
2. Compute a square root β of $\alpha \bmod p$. (See Annex D.1.4.) It is an error if the output of Annex D.1.4 is "no square roots exist".
3. If the rightmost bit of β is equal to $\tilde{y}_p$, then set $y_p = \beta$. Otherwise, set $y_p = p - \beta$.

4.2.2 Point Compression Technique for Elliptic Curves over F_{2^m} (Optional)

Let $P = (x_p, y_p)$ be a point on the elliptic curve $E : y^2 + xy = x^3 + ax^2 + b$ defined over a field F_{2^m}. Then $\tilde{y}_p$ is defined to be 0 if $x_p = 0$; if $x_p \neq 0$, then $\tilde{y}_p$ is defined to be the rightmost bit of the field element $y_p.x_p^{-1}$.

When the x-coordinate x_p of P and the bit $\tilde{y}_p$ are provided, then y_p can be recovered as follows.

1. If $x_p = 0$, then $y_p = b^{2^{m-1}}$. (y_p is the square root of b in F_{2^m}.)
2. If $x_p \neq 0$, then do the following:
   - 2.1. Compute the field element $\beta = x_p + a + bx_p^{-2}$ in F_{2^m}.
   - 2.2. Find a field element z such that $z^2 + z = \beta$ using the algorithm described in Annex D.1.6. It is an error if the output of Annex D.1.6 is "no solutions exist".
   - 2.3. Let $\tilde{z}$ be the rightmost bit of z.
   - 2.4. If $\tilde{y}_p \neq \tilde{z}$, then set $z = z + 1$, where 1 is the multiplicative identity.
   - 2.5. Compute $y_p = x_p.z$.

4.3 Data Conversions

The data types in this Standard are octet strings, integers, field elements and elliptic curve points. Figure 1 provides a cross-reference for the sections defining conversions between data types that shall be used in the algorithms specified in this Standard. The number on a line is the section number where the conversion technique is specified. Examples of conversions are provided in Annex J.1.

4.3.1 Integer-to-Octet-String Conversion

Input: A non-negative integer x, and the intended length k of the octet string satisfying:

$$2^{8k} > x.$$

Output: An octet string M of length k octets.

1. Let $M_1, M_2, ..., M_k$ be the octets of M from leftmost to rightmost.
2. The octets of M shall satisfy:

$$x = \sum_{i=1}^{k} 2^{8(k-i)} M_i .$$

4.3.2 Octet-String-to-Integer Conversion

Input: An octet string M of length k octets.

Output: An integer x.

1. Let $M_1, M_2, ..., M_k$ be the octets of M from leftmost to rightmost.
2. M shall be converted to an integer x satisfying:

$$x = \sum_{i=1}^{k} 2^{8(k-i)} M_i .$$

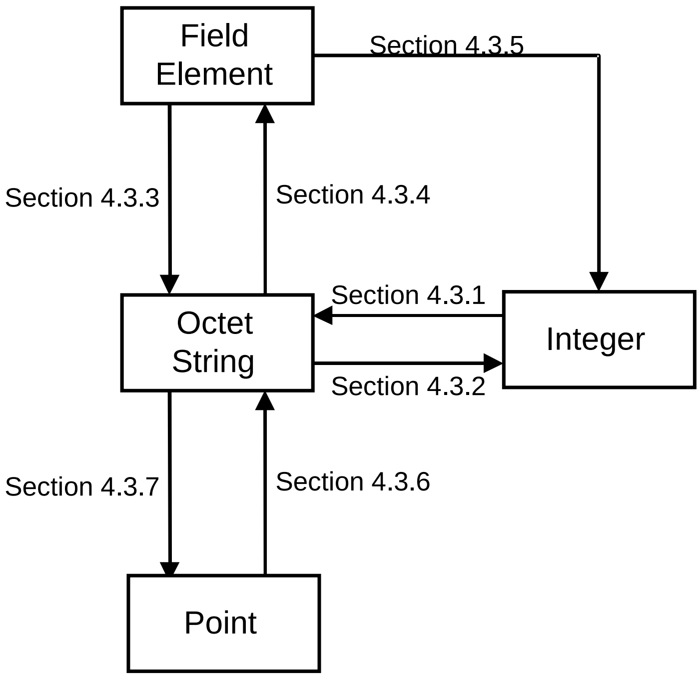

Figure 1 – Data Types and Conversion Conventions

4.3.3 Field-Element-to-Octet-String Conversion

Input: An element α in the field F_q.

Output: An octet string S of length $l = \lceil t / 8 \rceil$ octets, where $t = \lceil \log_2 q \rceil$.

1. If q is an odd prime, then α must be an integer in the interval $[0, q - 1]$; α shall be converted to an octet string of length l octets using the technique specified in Section 4.3.1.
2. If $q = 2^m$, then α must be a bit string of length m bits. Let $s_1, s_2, \ldots, s_m$ be the bits of α from leftmost to rightmost. Let $S_1, S_2, \ldots, S_l$ be the octets of S from leftmost to rightmost. The rightmost bit s_m shall become the rightmost bit of the last octet S_l, and so on through the leftmost bit s_1, which shall become the $(8l - m + 1)^{th}$ bit of the first octet S_1. The leftmost $(8l - m)$ bits of the first octet S_1 shall be zero.

4.3.4 Octet-String-to-Field-Element Conversion

Input: An indication of the field F_q used, and an octet string S of length $l = \lceil t / 8 \rceil$ octets, where $t = \lceil \log_2 q \rceil$.

Output: An element α in F_q.

1. If q is an odd prime, then convert S to an integer α using the technique specified in Section 4.2.2. It is an error if α does not lie in the interval $[0, q - 1]$.
2. If $q = 2^m$, then α shall be a bit string of length m bits. Let $s_1, s_2, \ldots, s_m$ be the bits of α from leftmost to rightmost. Let $S_1, S_2, \ldots, S_l$ be the octets of S from leftmost to rightmost. The rightmost bit of the last octet S_l shall become the rightmost bit s_m, and so on through the $(8l - m + 1)^{th}$ bit of the first octet S_1, which shall become the leftmost bit s_1. The leftmost $(8l - m)$ bits of the first octet S_1 are not used.

4.3.5 Field-Element-to-Integer Conversion

Input: An element α in the field F_q.

Output: An integer x.

1. If q is an odd prime then $x = \alpha$ (no conversion is required).
2. If $q = 2^m$, then α must be a bit string of length m bits. Let $s_1, s_2, \ldots, s_m$ be the bits of α from leftmost to rightmost. α shall be converted to an integer x satisfying:

$$x = \sum_{i=1}^{m} 2^{(m-i)} s_i \, .$$

4.3.6 Point-to-Octet-String Conversion

The octet string representation of the point at infinity $\mathcal{O}$ shall be a single zero octet PC = 00.

An elliptic curve point $P = (x_p , y_p)$ which is not the point at infinity shall be represented as an octet string in one of the following three forms:

1. compressed form.
2. uncompressed form.
3. hybrid form.

NOTE— The hybrid form contains information of both compressed and uncompressed forms. It allows an implementation to convert to either compressed form or to uncompressed form.

Input: An elliptic curve point $P = (x_p ,y_p)$, not the point at infinity.

Output: An octet string PO of length l + 1 octets if the compressed form is used, or of length 2l +1 octets if the uncompressed or hybrid form is used. ($l = \lceil (\log_2 q)/8 \rceil$.)

1. Convert the field element x_p to an octet string X_1. (See Section 4.3.3.)
2. If the compressed form is used, then do the following:
   - 2.1. Compute the bit $\tilde{y}_p$. (See Section 4.2.)
   - 2.2. Assign the value 02 to the single octet PC if $\tilde{y}_p$ is 0, or the value 03 if $\tilde{y}_p$ is 1.
   - 2.3. The result is the octet string $PO = PC \,||\, X_1$.
3. If the uncompressed form is used, then do the following:
   - 3.1. Convert the field element y_p to an octet string Y_1. (See Section 4.3.3.)
   - 3.2. Assign the value 04 to the single octet PC.
   - 3.3. The result is the octet string $PO = PC \,||\, X_1 \,||\, Y_1$.
4. If the hybrid form is used, then do the following:
   - 4.1. Convert the field element y_p to an octet string Y_1. (See Section 4.3.3.)
   - 4.2. Compute the bit $\tilde{y}_p$. (See Section 4.2.)
   - 4.3. Assign the value 06 to the single octet if $\tilde{y}_p$ is 0, or the value 07 if $\tilde{y}_p$ is 1.
   - 4.4. The result is the octet string $PO = PC \,||\, X_1 \,||\, Y_1$.

4.3.7 Octet-String-to-Point Conversion

Input: An octet string PO of length l + 1 octets if the compressed form is used, or of length 2l + 1 octets if the uncompressed or hybrid form is used ($l = \lceil (\log_2 q) / 8 \rceil$), and field elements a, b which define an elliptic curve over F_q.

Output: An elliptic curve point $P = (x_p , y_p)$, not the point at infinity.

1. If the compressed form is used, then parse PO as follows: $PO = PC \,||\, X_1$, where PC is a single octet, and X_1 is an octet string of length l octets. If uncompressed or hybrid form is used, then parse PO as follows: $PO = PC \,||\, X_1 \,||\, Y_1$, where PC is a single octet, and X_1 and Y_1 are octet strings each of length l octets.
2. Convert X_1 to a field element x_p. (See Section 4.3.4.)
3. If the compressed form is used, then do the following:
   - 3.1. Verify that PC is either 02 or 03. (It is an error if this is not the case.)
   - 3.2. Set the bit $\tilde{y}_p$ to be equal to 0 if PC = 02, or 1 if PC = 03.
   - 3.3. Convert $(x_p, \tilde{y}_p)$ to an elliptic curve point (x_p ,y_p). (See Section 4.2.)

4. If the uncompressed form is used, then do the following:
   4.1. Verify that PC is 04. (It is an error if this is not the case.)
   4.2. Convert Y_1 to a field element y_p. (See Section 4.3.4.)
5. If the hybrid form is used, then do the following:
   5.1. Verify that PC is either 06 or 07. (It is an error if this is not the case.)
   5.2. Perform either step 5.2.1 or step 5.2.2:
      5.2.1. Convert Y_1 to a field element y_p. (See Section 4.3.4.)
      5.2.2. Set the bit $\tilde{y}_p$ to be equal to 0 if PC = 06, or 1 if PC = 07. Convert (x_p, $\tilde{y}_p$) to an elliptic curve point (x_p, y_p). (See Section 4.2.)
6. If q is a prime, verify that $y_p^2 = x_p^3 + ax_p + b \pmod p$. (It is an error if this is not the case.)
   If $q = 2^m$, verify that $y_p^2 + x_p y_p = x_p^3 + ax_p^2 + b$ in F_{2^m}. (It is an error if this is not the case.)
7. The result is $P = (x_p, y_p)$.

NOTE— If hybrid form is used, an implementation may optionally check that y_p and $\tilde{y}_p$ are consistent (see steps 5.2.1 and 5.2.2). This may be particularly appropriate prior to elliptic curve domain parameter validation and public key validation.

5 The Elliptic Curve Digital Signature Algorithm (ECDSA)

This section specifies the following processes:

— Elliptic curve domain parameter generation and their validation.
— Key generation and validation.
— Signature generation.
— Signature verification.

NOTE— Equivalent computations that result in identical output are allowed.

5.1 Elliptic Curve Domain Parameter Generation and Validation

Elliptic curve domain parameters may be public; the security of the system does not rely on these parameters being secret. There is a security risk associated with multiple users sharing the same elliptic curve domain parameters; see Annex H.2 for more information. Two cases are distinguished:

1. Elliptic curve domain parameters over F_p: when the underlying field is F_p (p an odd prime); and
2. Elliptic curve domain parameters over F_{2^m}: when the underlying field is F_{2^m}.

Note that n is the primary security parameter. In general, as n increases, the security of ECDSA also increases. See Annex H for more information.

5.1.1 Elliptic Curve Domain Parameters and their Validation over F_p

5.1.1.1 Elliptic curve domain parameters over F_p

Elliptic curve domain parameters over F_p shall consist of the following parameters:

1. A field size $q = p$ which defines the underlying finite field F_q, where $p > 3$ shall be a prime number;
2. (Optional) A bit string SEED of length at least 160 bits, if the elliptic curve was randomly generated in accordance with Annex A.3.3;
3. Two field elements a and b in F_q which define the equation of the elliptic curve E: $y^2 = x^3 + ax + b$;
4. Two field elements x_G and y_G in F_q which define a point $G = (x_G, y_G)$ of prime order on E (note that $G \neq \mathcal{O}$);
5. The order n of the point G (it must be the case that $n > 2^{160}$ and $n > 4\sqrt{q}$); and
6. (Optional) The cofactor $h = \#E(F_q)/n$.

Annex A.3.2 specifies the method that shall be used for generating an elliptic curve E over F_p and the point G of order n.

5.1.1.2 Elliptic curve domain parameter validation over F_p

The following conditions shall be verified by the generator of the elliptic curve domain parameters. These conditions may alternately be verified by a user of the elliptic curve domain parameters.

Input: A set of elliptic curve domain parameters over F_p.

Output: The message "valid" if the elliptic curve domain parameters are valid; otherwise the message "invalid".

1. Verify that $q = p$ is an odd prime number. (See Annex A.2.1.)

2. Verify that a, b, x_G and y_G are integers in the interval $[0, p-1]$.
3. If the elliptic curve was randomly generated in accordance with Annex A.3.3, verify that SEED is a bit string of length at least 160 bits, and that a and b were suitably derived from SEED. (See Annex A.3.4.2.)
4. Verify that $(4a^3 + 27b^2) \not\equiv 0 \pmod{p}$.
5. Verify that $y_G^2 \equiv x_G^3 + ax_G + b \pmod{p}$.
6. Verify that n is prime, and that $n > 2^{160}$ and $n > 4\sqrt{p}$. (See Annex A.2.1.)
7. Verify that $nG = \mathcal{O}$. (See Annex D.3.2.)
8. (Optional) Compute $h' = \lfloor (\sqrt{p}+1)^2/n \rfloor$ and verify that $h = h'$.
9. Verify that the MOV and Anomalous conditions hold. (See Annex A.1.)
10. If any of the above verifications fail, then output "invalid". If all the verifications pass, then output "valid".

NOTES:

1. The cofactor h is not used in ECDSA, but is included here for compatibility with ANSI X9.63 [6] where it may be needed.
2. Step 8 of Section 5.1.1.2 (and also step 8 of Section 5.1.2.2) verifies that the value of the cofactor h is correct in the case that $n > 4\sqrt{q}$.

5.1.2 Elliptic Curve Domain Parameters and their Validation over F_{2^m}

5.1.2.1 Elliptic curve domain parameters over F_{2^m}

Elliptic curve domain parameters over F_{2^m} shall consist of the following parameters:

1. A field size $q = 2^m$ which defines the underlying finite field F_q, an indication of the basis used to represent the elements of the field (TPB, PPB or GNB), and a reduction polynomial of degree m over F_2 if the basis used is a TPB or PPB;
2. (Optional) A bit string SEED of length at least 160 bits, if the elliptic curve was randomly generated in accordance with Annex A.3.3;
3. Two field elements a and b in F_q which define the equation of the elliptic curve E: $y^2 + xy = x^3 + ax^2 + b$;
4. Two field elements x_G and y_G in F_q which define a point $G = (x_G, y_G)$ of prime order on E (note that $G \neq \mathcal{O}$);
5. The order n of the point G (it must be the case that $n>2^{160}$ and $n>4\sqrt{q}$); and
6. (Optional) The cofactor $h = \#E(F_q)/n$.

Annex A.3.2 specifies the method that shall be used for generating an elliptic curve E over F_{2^m} and the point G of order n.

5.1.2.2 Elliptic curve domain parameter validation over F_{2^m}

The following conditions shall be verified by the generator of the elliptic curve domain parameters. These conditions may alternately be verified by a user of the elliptic curve domain parameters.

Input: A set of elliptic curve domain parameters over F_{2^m}.

Output: The message "valid" if the elliptic curve domain parameters are valid; otherwise the message "invalid".

1. Verify that $q = 2^m$ for some m. If the basis used is a TPB, verify that the reduction polynomial is a trinomial and is irreducible over F_2 (see Table C-2 or Annex D.2.4). If the basis used is a PPB, verify that an irreducible trinomial of degree m does not exist, and that the reduction polynomial is a pentanomial and is irreducible over F_2 (see Table C-3 or Annex D.2.4). If the basis used is a GNB, verify that m is not divisible by 8.
2. Verify that a, b, x_G and y_G are bit strings of length m bits.
3. If the elliptic curve was randomly generated in accordance with A.3.3, verify that SEED is a bit string of length at least 160 bits, and that b was suitably derived from SEED. (See Annex A.3.4.1.)
4. Verify that $b \neq 0$.
5. Verify that $y_G^2 + x_G y_G = x_G^3 + ax_G^2 + b$ in F_{2^m}.
6. Verify that n is prime, and that $n > 2^{160}$ and $n > 4\sqrt{q}$. (See Annex A.2.1.)
7. Verify that $nG = \mathcal{O}$. (See Annex D.3.2.)
8. (Optional) Compute $h' = \lfloor (\sqrt{q}+1)^2/n \rfloor$ and verify that $h = h'$.
9. Verify that the MOV and Anomalous conditions hold. (See Annex A.1.)

10. If any of the above verifications fail, then output "invalid". If all the verifications pass, then output "valid".

5.2 Key Pair Generation and Public Key Validation

5.2.1 Key Pair Generation

Input: A valid set of elliptic curve domain parameters.

Output: A key pair (Q, d) associated with the elliptic curve domain parameters.

1. Select a statistically unique and unpredictable integer d in the interval [1, n-1]. It is acceptable to use a random or pseudorandom number. If a pseudorandom number is used, it shall be generated using one of the procedures of Annex A.4 or in an ANSI X9 approved standard. If a pseudorandom number is used, optional information to store with the private key are the seed values and the particular pseudorandom generation method used. Storing this optional information helps allow auditing of the key generation process.
   If a pseudorandom generation method is used, the seed values used in the generation of d may be determined by internal means, be supplied by the caller, or both—this is an implementation choice. In all cases, the seed values have the same security requirements as the private key value. That is, they must be protected from unauthorized disclosure and be unpredictable.
2. Compute the point $Q = (x_Q, y_Q) = dG$. (See Annex D.3.2.)
3. The key pair is (Q, d), where Q is the public key, and d is the private key.

5.2.2 Public Key Validation (Optional)

When an application is deemed to require the validation of the public key, for a given valid set of elliptic curve domain parameters and an associated public key Q, the public key shall be validated as follows.

Input: A valid set of elliptic curve domain parameters, and an associated public key Q.

Output: The message "valid" if Q is a valid public key for the given set of elliptic curve domain parameters; otherwise the message "invalid".

1. Verify that Q is not the point at infinity $\mathcal{O}$.
2. Verify that x_Q and y_Q are elements in the field F_q, where x_Q and y_Q are the x and y coordinates of Q, respectively. (That is, verify that x_Q and y_Q are integers in the interval [0, p-1] in the case that $q = p$ is an odd prime, or that x_Q and y_Q are bit strings of length m bits in the case that $q = 2^m$.)
3. If $q = p$ is an odd prime, verify that $y_Q^2 \equiv x_Q^3 + ax_Q + b \pmod{p}$. If $q = 2^m$, verify that $y_Q^2 + x_Q y_Q = x_Q^3 + ax_Q^2 + b$ in F_{2^m}.
4. Verify that $nQ = \mathcal{O}$. (See Annex D.3.2.)
5. If any one of the above verifications fail, then output "invalid". If all the verifications pass, then output "valid".

NOTE— If there is more than one public key available, it may also be checked that no two public keys are the same.

5.3 Signature Generation

This section describes the ECDSA signature generation process.

The signature generation process consists of:

1. Message digesting.
2. Elliptic curve computations.
3. Modular computations.

The inputs to the signature process are:

1. The message, M, of an arbitrary length, which is represented by a bit string.
2. A valid set of elliptic curve domain parameters.
3. An elliptic curve private key, d, associated with the elliptic curve domain parameters.

The output of the signature process are two integers r and s (the digital signature), where $1 \le r \le n–1$, $1 \le s \le n–1$.

5.3.1 Message Digesting

Compute the hash value $e = H(\mathrm{M})$ using the hash function SHA-1 as specified in ANSI X9.30 Part 2 [4]. e is represented as an integer with a length of 160 bits.

5.3.2 Elliptic Curve Computations

1. Select a statistically unique and unpredictable integer k in the interval $[1,n-1]$. It is acceptable to use a random or pseudorandom number. If a pseudorandom number is used, it shall be generated using one of the procedures of Annex A.4 or in an ANSI X9 approved standard.
   If a pseudorandom generation method is used, the seed values used in the generation of k may either be determined by internal means, be supplied by the caller, or both—this is an implementation choice. In all cases, the seed values have the same security requirements as the private key value. That is, they must be protected from unauthorized disclosure and be unpredictable.
   If the implementation allows a seed supplied by the caller, then the physical security of the device is of utmost importance. This is because if an adversary gained access to the signature generation device and were able to generate a signature with a seed of its choice for the per-message secret k, then the adversary could easily recover the private key.
2. Compute the elliptic curve point $(x_1, y_1) = kG$. (See Annex D.3.2.)

5.3.3 Modular Computations

1. Convert the field element x_1 to an integer $\overline{x}_1$, as described in Section 4.3.5.
2. Set $r = \overline{x}_1 \bmod n$.
3. If $r = 0$, then go to step 1 of Section 5.3.2.
4. Compute $s = k^{-1}(e + dr) \bmod n$. (See Annex D.1.2. for one method to compute $k^{-1} \bmod n$.)
5. If $s = 0$, then go to step 1 of Section 5.3.2.

5.3.4 The Signature

The signature for M shall be the two integers, r and s, as computed in Section 5.3.3.

NOTES:

1. In step 3 of Section 5.3.3, the probability that $r = 0$ is approximately $1/n$.
2. In step 5 of Section 5.3.3, the probability that $s = 0$ is approximately $1/n$.
3. As an optional security check (to guard against malicious or non-malicious errors in the signature generation process), the signer may verify that (r, s) is indeed a valid signature for message M using the signature verification process described in Section 5.4.

5.4 Signature Verification

This section describes the ECDSA signature verification process.

The signature verification process consists of:

1. Message digesting.
2. Modular computations.
3. Elliptic curve computations.
4. Signature checking.

The input to the signature verification process is:

1. The received message, M', represented as a bit string.
2. The received signature for M', represented as the two integers, r' and s'.
3. A valid set of elliptic curve domain parameters.
4. A valid public key, Q, associated with the elliptic curve domain parameters.

The output of the signature verification process is an indication of signature verification success or failure.

5.4.1 Message Digesting

Compute the hash value $e' = H(M')$ using the hash function SHA-1 as specified in ANSI X9.30 Part 2 [4]. e' is represented as an integer with a length of 160 bits.

5.4.2 Modular Computations

1. If r' is not an integer in the interval $[1, n-1]$, then reject the signature.
2. If s' is not an integer in the interval $[1, n-1]$, then reject the signature.
3. Compute $c = (s')^{-1} \bmod n$. (See Annex D.1.2.)
4. Compute $u_1 = e'c \bmod n$ and $u_2 = r'c \bmod n$.

5.4.3 Elliptic Curve Computations

1. Compute the elliptic curve point $(x_1, y_1) = u_1G + u_2Q$ (see Annex D.3.2). (If $u_1G + u_2Q$ is the point at infinity, then reject the signature.)

5.4.4 Signature Checking

1. Convert the field element x_1 to an integer $\overline{x}_1$, as described in Section 4.3.5.
2. Compute $v = \overline{x}_1 \bmod n$.
3. If $r' = v$, then the signature is verified, and the verifier has a high level of confidence that the received message was sent by the party holding the secret key d corresponding to Q**.**
   If r' does not equal v, then the message may have been modified, the message may have been incorrectly signed by the signatory, or the message may have been signed by an impostor. The message shall be considered invalid.

6 ASN.1 Syntax

This section provides the syntax for elliptic curve domain parameters and keys according to Abstract Syntax Notation One (**ASN.1**). While it is not required that elliptic curve domain parameters and keys be represented with **ASN.1** syntax, if they are so represented, then their syntax shall be as defined here. While it is likely that these **ASN.1** definitions will be encoded using the Distinguished Encoding Rules (**DER**), other encoding rules may also be used.

The object identifier **ansi-X9-62** represents the root of the tree containing all object identifiers defined in this Standard, and has the following value:

```
ansi-X9-62 OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) 10045 }
```

6.1 Syntax for Finite Field Identification

This section provides the abstract syntax definitions for the finite fields defined in this Standard.

A finite field shall be identified by a value of type **FieldID**:

```
FieldID { FIELD-ID:IOSet } ::= SEQUENCE {                    -- Finite field
    fieldType           FIELD-ID.&id({IOSet}),
    parameters          FIELD-ID.&Type({IOSet}{@fieldType})
}

FieldTypes FIELD-ID ::= {
    { Prime-p               IDENTIFIED BY  prime-field } |
    { Characteristic-two    IDENTIFIED BY  characteristic-two-field },
    ...
}

FIELD-ID ::= TYPE-IDENTIFIER
```

Note: **FieldID** is a parameterized type composed of two components, **fieldType** and **parameters**. These components are specified by the fields **&id** and **&Type**, which form a template for defining sets of information objects, instances of the class **FIELD-ID**. This class is based on the useful information object class **TYPE-IDENTIFIER**, described in X.681, Annex A. In an instance of **FieldID**, "**fieldType**" will contain an object identifier value that uniquely identifies the type contained in "**parameters**". The effect of referencing "**fieldType**" in both components of the **FieldID** sequence is to tightly bind the object identifier and its type.

The information object set **FieldTypes** is used as the single parameter in a reference to type **FieldID** and contains two objects followed by the extension marker ("**...**"). Each object, which represents a finite field, contains a unique object identifier and its associated type. The values of these objects define all of the valid values that may appear in an instance of **FieldID**. The extension marker allows backward compatibility with future versions of this Standard which may define objects to represent additional kinds of finite fields.

The object identifier **id-fieldType** represents the root of a tree containing the object identifiers of each field type. It has the following value:

```
id-fieldType OBJECT IDENTIFIER ::= { ansi-X9-62 fieldType(1) }
```

The object identifiers **prime-field** and **characteristic-two-field** name the two kinds of fields defined in this Standard. They have the following values:

```
prime-field OBJECT IDENTIFIER ::= { id-fieldType 1 }

characteristic-two-field OBJECT IDENTIFIER ::= { id-fieldType 2 }
```

Each of these unique object identifiers is associated with one **ASN.1** type, **Prime-p** and **Characteristic-two** which together specify the values of the finite fields defined in this Standard. These types have the following definitions:

```
Prime-p ::= INTEGER                                  -- Field size p

Characteristic-two ::= SEQUENCE {
    m               INTEGER,                         -- Field size 2^m
    basis           CHARACTERISTIC-TWO.&id({BasisTypes}),
    parameters      CHARACTERISTIC-TWO.&Type({BasisTypes}{@basis})
}

BasisTypes CHARACTERISTIC-TWO::= {
    { NULL          IDENTIFIED BY   gnBasis } |
    { Trinomial     IDENTIFIED BY   tpBasis } |
    { Pentanomial   IDENTIFIED BY   ppBasis },
    ...
}

Trinomial ::= INTEGER

Pentanomial ::= SEQUENCE {
    k1              INTEGER,
    k2              INTEGER,
    k3              INTEGER
}

CHARACTERISTIC-TWO ::= TYPE-IDENTIFIER
```

The object identifier **id-characteristic-two-basis** represents the root of a tree containing the object identifiers for each type of basis for the characteristic-two finite fields. It has the following value:

```
id-characteristic-two-basis OBJECT IDENTIFIER ::= {
    characteristic-two-field basisType(3) }
```

The object identifiers **gnBasis**, **tpBasis** and **ppBasis** name the three kinds of basis for characteristic-two finite fields defined in this Standard. They have the following values:

```
gnBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 1 }

tpBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 2 }

ppBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 3 }
```

Notes:

1. For the finite field F_p, where p is an odd prime, the parameter p is specified by a value of type **Prime-p**.
2. For the finite field F_{2^m}, the components of **Characteristic-two** are:

— **m**: degree of the field.
— **basis**: the type of representation used (GNB, TPB, or PPB).
— **parameters**: the values associated with each characteristic two basis type.

The information object set **BasisTypes** constrains the values of **Characteristic-two** components **basis** and **parameters** to only the valid values defined by this Standard. This set contains three objects followed by the extension marker ("**...**"). Each object, which represents a basis type, contains a unique object identifier and its associated type. An extension marker allows backward compatibility with future versions of this standard which may define objects to represent additional types of basis for characteristic-two finite fields.

3. For a Gaussian basis representation of F_{2^m}, **NULL** indicates that no specific values are required.
4. For a trinomial basis representation of F_{2^m}, **Trinomial** specifies the integer k where $x^m + x^k + 1$ is the reduction polynomial.
5. For a pentanomial basis representation of F_{2^m}, the components **k1**, **k2**, and **k3** of **Pentanomial** specify the integers $k1$, $k2$, and $k3$, respectively, where $x^m + x^{k3} + x^{k2} + x^{k1} + 1$ is the reduction polynomial.

6.2 Syntax for Finite Field Elements and Elliptic Curve Points

A finite field element shall be represented by a value of type **FieldElement**:

```
FieldElement ::= OCTET STRING                          -- Finite field element
```

The value of **FieldElement** shall be the octet string representation of a field element following the conversion routine in Section 4.3.3.

An elliptic curve point shall be represented by a value of type **ECPoint**:

```
ECPoint ::= OCTET STRING                               -- Elliptic curve point
```

The value of **ECPoint** shall be the octet string representation of an elliptic curve point following the conversion routine in Section 4.3.6.

6.3 Syntax for Elliptic Curve Domain Parameters

This section provides syntax for representing elliptic curve domain parameters.

Elliptic curve domain parameters shall be represented by a value of type **ECParameters**.

```
ECParameters ::= SEQUENCE {
    version         INTEGER { ecpVer1(1) } (ecpVer1),
    fieldID         FieldID {{FieldTypes}},
    curve           Curve,
    base            ECPoint,
    order           INTEGER,
    cofactor        INTEGER OPTIONAL,
    ...
}

Curve ::= SEQUENCE {
    a               FieldElement,
    b               FieldElement,
    seed            BIT STRING      OPTIONAL
}
```

The components of type **ECParameters** have the following meanings:

— **version** specifies the version number of the elliptic curve domain parameters. It shall have the value 1 for this version of the Standard. The notation above creates an **INTEGER** named **ecpVer1** and gives it a value of one. It is used to constrain **version** to a single value.
— **fieldID** identifies the finite field over which the elliptic curve is defined. Finite fields are represented by values of the parameterized type **FieldID**, constrained to the values of the objects defined in the information object set **FieldTypes**.

— **curve** specifies the coefficients a and b of the elliptic curve E. Each coefficient shall be represented as a value of type **FieldElement.** The value **seed** is an optional parameter used to derive the coefficients of a randomly generated elliptic curve.
— **base** specifies the base point G on the elliptic curve. The base point shall be represented as a value of type **ECPoint.**
— **order** specifies the order n of the base point.
— **cofactor** is the integer $h = \#E(F_q)/n$.

6.4 Syntax for Public Keys

This section provides the syntax for the public keys defined in this Standard.

A public key may be represented in a variety of ways using **ASN.1** syntax. When a public key is represented as the X.509 type **SubjectPublicKeyInfo**, then the public key shall have the following syntax:

```
SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm           AlgorithmIdentifier {{ECPKAlgorithms}},
    subjectPublicKey    BIT STRING
}
```

The elliptic curve public key (a value of type **ECPoint** which is an **OCTET STRING**) is mapped to a **subjectPublicKey** (a value of type **BIT STRING**) as follows: the most significant bit of the **OCTET STRING** value becomes the most significant bit of the **BIT STRING** value, etc.; the least significant bit of the **OCTET STRING** becomes the least significant bit of the **BIT STRING**.

A reference to parameterized type **AlgorithmIdentifier {}** tightly binds a set of **algorithm** object identifiers to their associated **parameters** types. Type **AlgorithmIdentifier {}** is defined as:

```
AlgorithmIdentifier { ALGORITHM:IOSet } ::= SEQUENCE {
    algorithm           ALGORITHM.&id({IOSet}),
    parameters          ALGORITHM.&Type({IOSet}{@algorithm})
}
```

A single parameter in the reference of type **AlgorithmIdentifier {}**, the information object set of class **ALGORITHM**, **ECPKAlgorithms**, specifies all of the pairs of valid values of this type. This set contains only one object **ecPublicKeyType**, as defined in this Standard.

```
ECPKAlgorithms ALGORITHM ::= {
    ecPublicKeyType,
    ...
}

ecPublicKeyType ALGORITHM ::= {
    Parameters IDENTIFIED BY id-ecPublicKey
}

ALGORITHM ::= TYPE-IDENTIFIER
```

The object identifier **id-publicKeyType** represents the tree containing the object identifiers for each public key. It has the following value:

```
id-publicKeyType OBJECT IDENTIFIER ::= { ansi-X9-62 keyType(2) }
```

The object identifier **id-ecPublicKey** names the public key type defined in this Standard. It has the following value:

```
id-ecPublicKey OBJECT IDENTIFIER ::= { id-publicKeyType 1 }
```

The public key **Parameters** are defined in this Standard as a choice of three alternatives. This allows detailed specification of all required values using choice **ecParameters**, the use of a **namedCurve** as an object identifier substitute for a particular set of elliptic curve domain parameters, or **implicitlyCA** to indicate that the parameters are explicitly defined elsewhere.

```
Parameters ::= CHOICE {
    ecParameters          ECParameters,
    namedCurve            CURVES.&id({CurveNames}),
    implicitlyCA          NULL
}
```

The valid values for the **namedCurve** choice alternative are specified by the information object set of class **CURVES**, **CurveNames**, and represent all of the example elliptic curves defined in this Standard. Each curve object is repesented as a unique object identifier value.

```
CurveNames CURVES ::= {
    { ID c2pnb163v1 } |       -- J.4.1,  example 1 --
    { ID c2pnb163v2 } |       -- J.4.1,  example 2 --
    { ID c2pnb163v3 } |       -- J.4.1,  example 3 --
    { ID c2pnb176w1 } |       -- J.4.2,  example 1 --
    { ID c2tnb191v1 } |       -- J.4.3,  example 1 --
    { ID c2tnb191v2 } |       -- J.4.3,  example 2 --
    { ID c2tnb191v3 } |       -- J.4.3,  example 3 --
    { ID c2onb191v4 } |       -- J.4.3,  example 4 --
    { ID c2onb191v5 } |       -- J.4.3,  example 5 --
    { ID c2pnb208w1 } |       -- J.4.4,  example 1 --
    { ID c2tnb239v1 } |       -- J.4.5,  example 1 --
    { ID c2tnb239v2 } |       -- J.4.5,  example 2 --
    { ID c2tnb239v3 } |       -- J.4.5,  example 3 --
    { ID c2onb239v4 } |       -- J.4.5,  example 4 --
    { ID c2onb239v5 } |       -- J.4.5,  example 5 --
    { ID c2pnb272w1 } |       -- J.4.6,  example 1 --
    { ID c2pnb304w1 } |       -- J.4.7,  example 1 --
    { ID c2tnb359v1 } |       -- J.4.8,  example 1 --
    { ID c2pnb368w1 } |       -- J.4.9,  example 1 --
    { ID c2tnb431r1 } |       -- J.4.10, example 1 --
    { ID prime192v1 } |       -- J.5.1,  example 1 --
    { ID prime192v2 } |       -- J.5.1,  example 2 --
    { ID prime192v3 } |       -- J.5.1,  example 3 --
    { ID prime239v1 } |       -- J.5.2,  example 1 --
    { ID prime239v2 } |       -- J.5.2,  example 2 --
    { ID prime239v3 } |       -- J.5.2,  example 3 --
    { ID prime256v1 },        -- J.5.3,  example 1 --
    ...                       -- others --
}
```

Curve identifier names are prefixed to indicate a type of finite field, '**c2**' for characteristic two followed by a three character basis type, or '**prime**'. The next three characters are numeric digits that indicate the field size in bits. The final two characters indicate how the curve was selected and the example number in Appendix J which fully defines the elliptic curve domain parameters for each named curve.

For the curve selection character, a '**v**' indicates a curve whose coefficients were selected verifiably at random using a seeded hash. An '**r**' indicates a curve whose coefficients were selected at random but were not selected verifiably at random using a seeded hash. The letter '**w**' indicates a curve whose coefficients were selected using the Weil method. (See Annex A.3.2.)

```
CURVES ::= CLASS {
    &id                       OBJECT IDENTIFIER UNIQUE
}

WITH SYNTAX { ID &id }
```

The object identifier **ellipticCurve** represents the tree containing the object identifiers for each example elliptic curve specified in this Standard. It has the following value:

```
ellipticCurve OBJECT IDENTIFIER ::= { ansi-X9-62 curves(3) }
```

The object identifier **c-TwoCurve** represents the tree containing the object identifiers for each example elliptic curves over the field F_{2^m} specified in this Standard. It has the following value:

```
c-TwoCurve OBJECT IDENTIFIER ::= {
    ellipticCurve characteristicTwo(0) }
```

The object identifier **primeCurve** represents the tree containing the object identifiers for each example elliptic curve over field F_p specified in this Standard. It has the following value:

```
primeCurve OBJECT IDENTIFIER ::= { ellipticCurve prime(1) }
```

```
c2pnb163v1   OBJECT IDENTIFIER  ::=  { c-TwoCurve  1 }

c2pnb163v2   OBJECT IDENTIFIER  ::=  { c-TwoCurve  2 }

c2pnb163v3   OBJECT IDENTIFIER  ::=  { c-TwoCurve  3 }

c2pnb176w1   OBJECT IDENTIFIER  ::=  { c-TwoCurve  4 }

c2tnb191v1   OBJECT IDENTIFIER  ::=  { c-TwoCurve  5 }

c2tnb191v2   OBJECT IDENTIFIER  ::=  { c-TwoCurve  6 }

c2tnb191v3   OBJECT IDENTIFIER  ::=  { c-TwoCurve  7 }

c2onb191v4   OBJECT IDENTIFIER  ::=  { c-TwoCurve  8 }

c2onb191v5   OBJECT IDENTIFIER  ::=  { c-TwoCurve  9 }

c2pnb208w1   OBJECT IDENTIFIER  ::=  { c-TwoCurve 10 }

c2tnb239v1   OBJECT IDENTIFIER  ::=  { c-TwoCurve 11 }

c2tnb239v2   OBJECT IDENTIFIER  ::=  { c-TwoCurve 12 }

c2tnb239v3   OBJECT IDENTIFIER  ::=  { c-TwoCurve 13 }

c2onb239v4   OBJECT IDENTIFIER  ::=  { c-TwoCurve 14 }

c2onb239v5   OBJECT IDENTIFIER  ::=  { c-TwoCurve 15 }

c2pnb272w1   OBJECT IDENTIFIER  ::=  { c-TwoCurve 16 }

c2pnb304w1   OBJECT IDENTIFIER  ::=  { c-TwoCurve 17 }

c2tnb359v1   OBJECT IDENTIFIER  ::=  { c-TwoCurve 18 }

c2pnb368w1   OBJECT IDENTIFIER  ::=  { c-TwoCurve 19 }

c2tnb431r1   OBJECT IDENTIFIER  ::=  { c-TwoCurve 20 }

prime192v1   OBJECT IDENTIFIER  ::=  { primeCurve  1 }

prime192v2   OBJECT IDENTIFIER  ::=  { primeCurve  2 }

prime192v3   OBJECT IDENTIFIER  ::=  { primeCurve  3 }
```

```
prime239v1    OBJECT IDENTIFIER  ::=  { primeCurve  4 }

prime239v2    OBJECT IDENTIFIER  ::=  { primeCurve  5 }

prime239v3    OBJECT IDENTIFIER  ::=  { primeCurve  6 }

prime256v1    OBJECT IDENTIFIER  ::=  { primeCurve  7 }
```

6.5 Syntax for Digital Signatures

This section provides the syntax for the digital signatures defined in this Standard.

A signature may be represented in a variety of ways using the **ASN.1** notation. The X.509 certificate and CRL types include an **ASN.1** algorithm object identifier to identify the signature type and format. When ECDSA and SHA-1 are used to sign an X.509 certificate or CRL, the signature shall be identified by the value **ecdsa-with-SHA1**, as defined below:

```
id-ecSigType        OBJECT IDENTIFIER ::= { ansi-X9-62 signatures(4) }

ecdsa-with-SHA1   OBJECT IDENTIFIER ::= { id-ecSigType 1 }
```

When the **ecdsa-with-SHA1** OID appears in the **algorithm** field of the **ASN.1** type **AlgorithmIdentifier**, and the **parameters** field is a value of type **NULL**, the ECDSA parameters for signature verification must be obtained from other sources, such as the **subjectPublicKeyInfo** field of the certificate of the issuer.

When a digital signature is identified by the OID **ecdsa-with-SHA1**, the digital signature shall be **ASN.1** encoded using the following syntax:

```
ECDSA-Sig-Value ::= SEQUENCE {
    r                   INTEGER,
    s                   INTEGER
}
```

X.509 certificates and CRLs represent signatures as a bit string. Where a certificate or CRL is signed with ECDSA and SHA-1, the entire encoding of a value of **ASN.1** type **ECDSA-Sig-Value** shall be the value of the bit string.

6.6 ASN.1 Module

The following **ASN.1** module contains all of the syntax defined in this Standard:

```
ANSI-X9-62 { iso(1) member-body(2) us(840) 10045 module(4) 1 }
    DEFINITIONS EXPLICIT TAGS ::= BEGIN

-- EXPORTS All;

-- IMPORTS None;


ansi-X9-62 OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) us(840) 10045 }

FieldID { FIELD-ID:IOSet } ::= SEQUENCE {                 -- Finite field
    fieldType           FIELD-ID.&id({IOSet}),
    parameters          FIELD-ID.&Type({IOSet}{@fieldType})
}
```

```
FieldTypes FIELD-ID ::= {
    { Prime-p                IDENTIFIED BY  prime-field                 } |
    { Characteristic-two     IDENTIFIED BY  characteristic-two-field    },
    ...
}

FIELD-ID ::= TYPE-IDENTIFIER             -- ISO/IEC 8824-2:1995(E), Annex A

id-fieldType OBJECT IDENTIFIER ::= { ansi-X9-62 fieldType(1) }

prime-field OBJECT IDENTIFIER ::= { id-fieldType 1 }

characteristic-two-field OBJECT IDENTIFIER ::= { id-fieldType 2 }

Prime-p ::= INTEGER    -- Finite field F(p), where p is an odd prime

Characteristic-two ::= SEQUENCE {
    m                        INTEGER,                    -- Field size 2^m
    basis                    CHARACTERISTIC-TWO.&id({BasisTypes}),
    parameters               CHARACTERISTIC-TWO.&Type({BasisTypes}{@basis})
}

BasisTypes CHARACTERISTIC-TWO::= {
    { NULL                   IDENTIFIED BY  gnBasis    } |
    { Trinomial              IDENTIFIED BY  tpBasis    } |
    { Pentanomial            IDENTIFIED BY  ppBasis    },
    ...
}

-- Trinomial basis representation of F2^m

-- Integer k for reduction polynomial xm + xk + 1

--

Trinomial ::= INTEGER

Pentanomial ::= SEQUENCE {
    --
    -- Pentanomial basis representation of F2^m
    -- reduction polynomial integers k1, k2, k3
    -- f(x) = x**m + x**k3 + x**k2 + x**k1 + 1
    --
    k1                       INTEGER,
    k2                       INTEGER,
    k3                       INTEGER
}

CHARACTERISTIC-TWO ::= TYPE-IDENTIFIER

id-characteristic-two-basis OBJECT IDENTIFIER ::= {
    characteristic-two-field basisType(3) }

-- The object identifiers gnBasis, tpBasis and ppBasis name

-- three kinds of basis for characteristic-two finite fields

gnBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 1 }
```

```
tpBasis  OBJECT IDENTIFIER ::= { id-characteristic-two-basis 2 }

ppBasis OBJECT IDENTIFIER ::= { id-characteristic-two-basis 3 }

FieldElement ::= OCTET STRING                              -- Finite field element

ECPoint  ::= OCTET STRING                                  -- Elliptic curve point

ECParameters  ::= SEQUENCE {                    -- Elliptic curve parameters
    version                 INTEGER { ecpVer1(1) } (ecpVer1),
    fieldID                 FieldID {{FieldTypes}},
    curve                   Curve,
    base                    ECPoint,            -- Base point G
    order                   INTEGER,            -- Order n of the base point
    cofactor                INTEGER OPTIONAL, -- The integer h = #E(Fq)/n
    ...
}

Curve  ::= SEQUENCE {
    a                       FieldElement,       -- Elliptic curve coefficient a
    b                       FieldElement,       -- Elliptic curve coefficient b
    seed                    BIT STRING  OPTIONAL
}

ECDSA-Sig-Value ::= SEQUENCE {
    r                       INTEGER,
    s                       INTEGER
}

id-ecSigType OBJECT IDENTIFIER ::= { ansi-X9-62 signatures(4) }

ecdsa-with-SHA1 OBJECT IDENTIFIER ::= { id-ecSigType 1 }

SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm               AlgorithmIdentifier {{ECPKAlgorithms}},
    subjectPublicKey        BIT STRING
}

AlgorithmIdentifier { ALGORITHM:IOSet } ::= SEQUENCE {
    algorithm               ALGORITHM.&id({IOSet}),
    parameters              ALGORITHM.&Type({IOSet}{@algorithm})
}

ECPKAlgorithms ALGORITHM ::= {
    ecPublicKeyType,
    ...
}

ecPublicKeyType ALGORITHM ::= {
    Parameters IDENTIFIED BY id-ecPublicKey
}

ALGORITHM ::= TYPE-IDENTIFIER

id-publicKeyType OBJECT IDENTIFIER  ::= { ansi-X9-62 keyType(2) }

id-ecPublicKey OBJECT IDENTIFIER ::= { id-publicKeyType 1 }
```

```
Parameters ::= CHOICE {
    ecParameters          ECParameters,
    namedCurve            CURVES.&id({CurveNames}),
    implicitlyCA          NULL
}

CurveNames CURVES ::= {
    { ID c2pnb163v1 }  |    -- J.4.1,  example 1 --
    { ID c2pnb163v2 }  |    -- J.4.1,  example 2 --
    { ID c2pnb163v3 }  |    -- J.4.1,  example 3 --
    { ID c2pnb176w1 }  |    -- J.4.2,  example 1 --
    { ID c2tnb191v1 }  |    -- J.4.3,  example 1 --
    { ID c2tnb191v2 }  |    -- J.4.3,  example 2 --
    { ID c2tnb191v3 }  |    -- J.4.3,  example 3 --
    { ID c2onb191v4 }  |    -- J.4.3,  example 4 --
    { ID c2onb191v5 }  |    -- J.4.3,  example 5 --
    { ID c2pnb208w1 }  |    -- J.4.4,  example 1 --
    { ID c2tnb239v1 }  |    -- J.4.5,  example 1 --
    { ID c2tnb239v2 }  |    -- J.4.5,  example 2 --
    { ID c2tnb239v3 }  |    -- J.4.5,  example 3 --
    { ID c2onb239v4 }  |    -- J.4.5,  example 4 --
    { ID c2onb239v5 }  |    -- J.4.5,  example 5 --
    { ID c2pnb272w1 }  |    -- J.4.6,  example 1 --
    { ID c2pnb304w1 }  |    -- J.4.7,  example 1 --
    { ID c2tnb359v1 }  |    -- J.4.8,  example 1 --
    { ID c2pnb368w1 }  |    -- J.4.9,  example 1 --
    { ID c2tnb431r1 }  |    -- J.4.10, example 1 --
    { ID prime192v1 }  |    -- J.5.1,  example 1 --
    { ID prime192v2 }  |    -- J.5.1,  example 2 --
    { ID prime192v3 }  |    -- J.5.1,  example 3 --
    { ID prime239v1 }  |    -- J.5.2,  example 1 --
    { ID prime239v2 }  |    -- J.5.2,  example 2 --
    { ID prime239v3 }  |    -- J.5.2,  example 3 --
    { ID prime256v1 },      -- J.5.3,  example 1 --
    ... -- others --
}

CURVES ::= CLASS {
      &id  OBJECT IDENTIFIER UNIQUE
}

WITH SYNTAX { ID &id }

ellipticCurve OBJECT IDENTIFIER ::= { ansi-X9-62 curves(3) }

c-TwoCurve OBJECT IDENTIFIER ::= {
    ellipticCurve characteristicTwo(0) }

primeCurve OBJECT IDENTIFIER ::= { ellipticCurve prime(1) }

c2pnb163v1    OBJECT IDENTIFIER   ::=    { c-TwoCurve  1 }

c2pnb163v2    OBJECT IDENTIFIER   ::=    { c-TwoCurve  2 }

c2pnb163v3    OBJECT IDENTIFIER   ::=    { c-TwoCurve  3 }
```

```
c2pnb176w1   OBJECT IDENTIFIER   ::=   { c-TwoCurve  4 }
c2tnb191v1   OBJECT IDENTIFIER   ::=   { c-TwoCurve  5 }
c2tnb191v2   OBJECT IDENTIFIER   ::=   { c-TwoCurve  6 }
c2tnb191v3   OBJECT IDENTIFIER   ::=   { c-TwoCurve  7 }
c2onb191v4   OBJECT IDENTIFIER   ::=   { c-TwoCurve  8 }
c2onb191v5   OBJECT IDENTIFIER   ::=   { c-TwoCurve  9 }
c2pnb208w1   OBJECT IDENTIFIER   ::=   { c-TwoCurve 10 }
c2tnb239v1   OBJECT IDENTIFIER   ::=   { c-TwoCurve 11 }
c2tnb239v2   OBJECT IDENTIFIER   ::=   { c-TwoCurve 12 }
c2tnb239v3   OBJECT IDENTIFIER   ::=   { c-TwoCurve 13 }
c2onb239v4   OBJECT IDENTIFIER   ::=   { c-TwoCurve 14 }
c2onb239v5   OBJECT IDENTIFIER   ::=   { c-TwoCurve 15 }
c2pnb272w1   OBJECT IDENTIFIER   ::=   { c-TwoCurve 16 }
c2pnb304w1   OBJECT IDENTIFIER   ::=   { c-TwoCurve 17 }
c2tnb359v1   OBJECT IDENTIFIER   ::=   { c-TwoCurve 18 }
c2pnb368w1   OBJECT IDENTIFIER   ::=   { c-TwoCurve 19 }
c2tnb431r1   OBJECT IDENTIFIER   ::=   { c-TwoCurve 20 }
prime192v1   OBJECT IDENTIFIER   ::=   { primeCurve  1 }
prime192v2   OBJECT IDENTIFIER   ::=   { primeCurve  2 }
prime192v3   OBJECT IDENTIFIER   ::=   { primeCurve  3 }
prime239v1   OBJECT IDENTIFIER   ::=   { primeCurve  4 }
prime239v2   OBJECT IDENTIFIER   ::=   { primeCurve  5 }
prime239v3   OBJECT IDENTIFIER   ::=   { primeCurve  6 }
prime256v1   OBJECT IDENTIFIER   ::=   { primeCurve  7 }
END
```

Annex A
(normative)
Normative Number-Theoretic Algorithms

A.1 Avoiding Cryptographically Weak Curves

Two conditions, the *MOV condition* and the *Anomalous condition*, are described to ensure that a particular elliptic curve is not vulnerable to two known attacks on special instances of the elliptic curve discrete logarithm problem.

A.1.1 The MOV Condition

The reduction attacks of Menezes, Okamoto and Vanstone [29] and Frey and Ruck reduce the discrete logarithm problem in an elliptic curve over F_q to the discrete logarithm in the finite field F_{q^B} for some $B \geq 1$. The attack is only practical if B is small; this is not the case for most elliptic curves. The *MOV condition* ensures that an elliptic curve is not vulnerable to these reduction attacks. Most elliptic curves over a field F_q will indeed satisfy the MOV condition.

Before performing the algorithm, it is necessary to select an MOV threshold. This is a positive integer B such that taking discrete logarithms over F_{q^B} is at least as difficult as taking elliptic discrete logarithms over F_q. For this Standard, a value $B \geq 20$ is required. Selecting $B \geq 20$ also limits the selection of curves to non-supersingular curves (see Annex H.1). This algorithm is used in elliptic curve domain parameter validation (see Section 5.1) and elliptic curve domain parameter generation (see Annex A.3.2).

Input: An MOV threshold B, a prime-power q, and a prime n. (n is a prime divisor of $\#E(F_q)$, where E is an elliptic curve defined over F_q.)

Output: The message “true” if the MOV condition is satisfied for an elliptic curve over F_q with a base point of order n; the message “false” otherwise.

1. Set $t = 1$.
2. For i from 1 to B do
    2.1. Set $t = t.q \bmod n$.
    2.2. If $t = 1$, then output “false” and stop.
3. Output “true”.

A.1.2 The Anomalous Condition

Smart [38] and Satoh and Araki [37] showed that the elliptic curve discrete logarithm problem in anomalous curves can be efficiently solved. An elliptic curve E defined over F_q is said to be F_q-*anomalous* if $\#E(F_q) = q$. The *Anomalous condition* checks that $\#E(F_q) \neq q$; this ensures that an elliptic curve is not vulnerable to the Anomalous attack. Most elliptic curves over a field F_q will indeed satisfy the Anomalous condition.

Input: An elliptic curve E defined over F_q, and the order $u = \#E(F_q)$.

Output: The message “true” if the Anomalous condition is satisfied for E over F_q; the message “false” otherwise.

1. If $u = q$ then output “false”; otherwise output “true”.

A.2 Primality

A.2.1 A Probabilistic Primality Test

If n is a large positive integer, the following probabilistic algorithm (the *Miller-Rabin test*) [22, p.379] will determine whether n is prime or composite. This algorithm is used in elliptic curve domain parameter validation (see Section 5.1), and in checking for near primality (see Annex A.2.2).

Input: A large odd integer n, and a positive integer T.

Output: The message “probable prime” or “composite”.

1. Compute v and an odd value for w such that $n-1 = 2^v w$.
2. For j from 1 to T do
    2.1. Choose random a in the interval $[2, n-1]$.
    2.2. Set $b = a^w \bmod n$.

2.3. If $b = 1$ or n-1, go to Step 2.6.
2.4. For i from 1 to v-1 do
2.4.1 Set $b = b^2 \bmod n$.
2.4.2 If $b = n$-1, go to Step 2.6.
2.4.3 If $b = 1$, output "composite" and stop.
2.4.4 Next i.
2.5. Output "composite" and stop.
2.6. Next j.
3. Output "probable prime".

If the algorithm outputs "composite", then n is a composite integer. The probability that the algorithm outputs "probable prime" when n is a composite integer is less than 2^{-2T}. Thus, the probability of an error can be made negligible by taking a large enough value for T. For this Standard, a value of $T \geq 50$ shall be used.

The probabilistic and deterministic primality tests to appear in a forthcoming ANSI X9 Standard on prime generation [7] may be used instead of the test described in this section.

A.2.2 Checking for Near Primality

Given a trial division bound l_{max}, a positive integer h is said to be l_{max}*-smooth* if every prime divisor of h is at most l_{max}. Given a positive integer r_{min}, the positive integer u is said to be *nearly prime* if $u = hn$ for some probable prime value of n such that $n \geq r_{min}$ and some l_{max}-smooth integer h. The following algorithm checks for near primality. The algorithm is used in elliptic curve domain parameter generation (see Annex A.3.2).

Input: Positive integers u, l_{max}, and r_{min}.

Output: If u is nearly prime, a probable prime $n \geq r_{min}$ and a l_{max}-smooth integer h such that $u = hn$. If u is not nearly prime, the message "not nearly prime".

1. Set $n = u$, $h = 1$.
2. For l from 2 to l_{max} do
2.1. If l is composite, then go to Step 2.3.
2.2. While (l divides n)
2.2.1 Set $n = n / l$ and $h = h.l$.
2.2.2 If $n < r_{min}$, then output "not nearly prime" and stop.
2.3. Next l.
3. If n is a probable prime (see Annex A.2.1), then output h and n and stop.
4. Output "not nearly prime".

A.3 Elliptic Curve Algorithms

A.3.1 Finding a Point of Large Prime Order

If the order $\#E(F_q) = u$ of an elliptic curve E is nearly prime, the following algorithm efficiently produces a random point on E whose order is the large prime factor n of $u = hn$. The algorithm is used in elliptic curve domain parameter generation (see Annex A.3.2).

Input: A prime n, a positive integer h not divisible by n, and an elliptic curve E over the field F_q with $\#E(F_q) = u$.

Output: If $u = hn$, a point G on E of order n. If not, the message "wrong order".

1. Generate a random point R (not $\mathcal{O}$) on E. (See Annex D.3.1.)
2. Set $G = hR$.
3. If $G = \mathcal{O}$, then go to Step 1.
4. Set $Q = nG$.
5. If $Q \neq \mathcal{O}$, then output "wrong order" and stop.
6. Output G.

A.3.2 Selecting an Appropriate Curve and Point

Given a field size q, a lower bound r_{min} for the point order, and a trial division bound l_{max}, the following procedure shall be used for choosing a curve and arbitrary point. The algorithm is used to generate elliptic curve domain parameters (see Sections 5.1.1.1 and 5.1.2.1).

Input: A field size q, lower bound r_{min}, and trial division bound l_{max}. (See the notes below for guidance on selecting r_{min} and l_{max}.)

Output: Field elements $a, b \in F_q$ which define an elliptic curve over F_q, a point G of prime order $n \geq r_{min}$ on the curve, and the cofactor $h = \#E(F_q)/n$.

1. If it is desired that an elliptic curve be generated verifiably at random, then select parameters (SEED, a, b) using the technique specified in Annex A.3.3.1 in the case that $q = 2^m$, or the technique specified in Annex A.3.3.2 in the case that $q = p$ is an odd prime. Compute the order u of the curve defined by a and b (see Note 5 below).

   Otherwise, use any alternative technique to select $a, b \in F_q$ which define an elliptic curve of known order u. (See Note 7 and Note 8 for two such techniques.)
2. In the case that q is a prime, verify that $(4a^3+27b^2) \not\equiv 0 \pmod p$. The curve equation for E is:
   $$y^2 = x^3 + ax + b.$$
   In the case that $q = 2^m$, verify that $b \neq 0$. The curve equation for E is:
   $$y^2 + xy = x^3 + ax^2 + b.$$
3. Test u for near primality using the technique defined in Annex A.2.2. If the result is "not nearly prime", then go to Step 1. Otherwise, $u = hn$ where h is l_{max}-smooth, and $n \geq r_{min}$ is probably prime.
4. Check the MOV condition (see Annex A.1.1) with inputs $B \geq 20$, q, and n. If the result is "false", then go to Step 1.

   Check the Anomalous condition (see Annex A.1.2). If the result is "false", then go to Step 1.
5. Find a point G on E of order n. (See Annex A.3.1.)
6. Output the curve E, the point G, the order n, and the cofactor h.

NOTES:

1. r_{min} shall be selected so that $r_{min} > 2^{160}$ and $r_{min} > 4\sqrt{q}$. The security level of the resulting elliptic curve discrete logarithm problem can be increased by selecting a larger r_{min} (e.g. $r_{min} > 2^{200}$).
2. If q is prime, then the order u of an elliptic curve E over F_q satisfies $q+1-2\sqrt{q} \leq u \leq q+1+2\sqrt{q}$. Hence for a given q, r_{min} should be $\leq q+1-2\sqrt{q}$.
3. If $q = 2^m$, then the order u of an elliptic curve E over F_q satisfies $q+1-2\sqrt{q} \leq u \leq q+1+2\sqrt{q}$, and u is even. Hence for a given q, r_{min} should be $\leq (q+1-2\sqrt{q})/2$.
4. l_{max} is typically a small integer (e.g. $l_{max} = 255$).
5. The order $\#E(F_q)$ can be computed by using Schoof's algorithm [36]. Although the basic algorithm is quite inefficient, several dramatic improvements and extensions of this method have been discovered in recent years. Currently, it is feasible to compute orders of elliptic curves over F_p where p is as large as 10^{499}, and orders of elliptic curves over F_{2^m} where m is as large as 1300. Cryptographically suitable elliptic curves over fields as large as $F_{2^{196}}$ can be randomly generated in about 5 hours on a workstation (see [24] and [25]).
6. One technique for selecting an elliptic curve of known order is to use the Weil Theorem which states the following. Let E be an elliptic curve defined over F_q, and let $t = q + 1 - \#E(F_q)$. Let α and β be the complex numbers $\alpha = (t+\sqrt{(t^2-4q)})/2$ and $\beta = (t - \sqrt{(t^2-4q)})/2$. Then $\#E(F_{q^k}) = q^k+1-\alpha^k-\beta^k$ for all $k \geq 1$.
7. The Weil Theorem can be used to select a curve over F_{2^m} when m is divisible by a small number l as follows. First select a random elliptic curve E: $y^2 + xy = x^3 + ax^2 + b$, $b \neq 0$, where $a, b \in F_{2^l}$. Note that since l divides m, F_{2^l} is contained in F_{2^m}. Compute $\#E(F_{2^l})$; this can easily be done exhaustively since l is small. Then compute $\#E(F_{2^m})$ using the Weil Theorem with $q = 2^l$ and $k = m/l$. This method of selecting curves is called the Weil method.
8. Another technique for selecting an elliptic curve of known order is to use the Complex Multiplication (CM) method. This method is described in detail in Annex E.

Annex J.4 and Annex J.5 present sample elliptic curves over a 192-bit prime field, a 239-bit prime field, a 256-bit prime field, and the fields $F_{2^{163}}$, $F_{2^{176}}$, $F_{2^{191}}$, $F_{2^{208}}$, $F_{2^{239}}$, $F_{2^{272}}$, $F_{2^{304}}$, $F_{2^{359}}$, $F_{2^{368}}$ and $F_{2^{431}}$ which may be used to ensure the correct implementation of this Standard.

A.3.3 Selecting an Elliptic Curve Verifiably at Random

In order to verify that a given elliptic curve was indeed generated at random, the defining parameters of the elliptic curve are defined to be outputs of the hash function SHA-1 (as specified in ANSI X9.30 Part 2 [4]). The input (SEED) to SHA-1 then serves as proof (under the assumption that SHA-1 cannot be inverted) that the parameters were indeed generated at random. (See Annex A.3.4.) The algorithms in this section are used in Annex A.3.2.

A.3.3.1 Elliptic curves over F_{2^m}

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

Output: A bit string SEED and field elements $a, b \in F_{2^m}$ which define an elliptic curve over F_{2^m}.

Let $t = m$, $s = \lfloor (t-1)/160 \rfloor$, and $h = t - 160.s$.

1. Choose an arbitrary bit string SEED of bit length at least 160 bits. Let g be the length of SEED in bits.
2. Compute H = SHA-1(SEED), and let b_0 denote the bit string of length h bits obtained by taking the h rightmost bits of H.
3. For i from 1 to s do:
   Compute b_i = SHA-1((SEED + i) mod 2^g).
4. Let b be the field element obtained by the concatenation of $b_0,b_1,\ldots,b_s$ as follows:
   $b = b_0 \| b_1 \| \ldots \| b_s$.
5. If $b = 0$, then go to step 1.
6. Let a be an arbitrary element in F_{2^m}.
7. The elliptic curve chosen over F_{2^m} is:
   E: $y^2+xy = x^3+ax^2+b$.
8. Output (SEED, a, b).

A.3.3.2 Elliptic curves over F_p

Input: A prime field size p.

Output: A bit string SEED and field elements $a, b \in F_p$ which define an elliptic curve over F_p.

Let $t = \lfloor \log_2 p \rfloor$, $s = \lfloor (t-1)/160 \rfloor$, and $h = t - 160.s$.

1. Choose an arbitrary bit string SEED of bit length at least 160 bits. Let g be the length of SEED in bits.
2. Compute H = SHA-1(SEED), and let c_0 denote the bit string of length h bits obtained by taking the h rightmost bits of H.
3. Let W_0 denote the bit string of length h bits obtained by setting the leftmost bit of c_0 to 0. (This ensures that $r < p$.)
4. For i from 1 to s do:
   Compute W_i = SHA-1((SEED + i) mod 2^g).
5. Let W be the bit string obtained by the concatenation of $W_0, W_1,\ldots, W_s$ as follows:
   $W = W_0 \| W_1 \| \ldots \| W_s$.
6. Let $w_1, w_2, \ldots, w_t$ be the bits of W from leftmost to rightmost. Let r be the integer $r = \sum_{i=1}^{t} w_i 2^{t-i}$.
7. Choose integers $a, b \in F_p$ such that $r.b^2 \equiv a^3 \pmod p$. (It is not necessary that a and b be chosen at random.)
8. If $4a^3+27b^2 \equiv 0 \pmod p$, then go to step 1.
9. The elliptic curve chosen over F_p is:
   $E : y^2 = x^3+ax+b$.
10. Output (SEED, a, b).

A.3.4 Verifying that an Elliptic Curve was Generated at Random

The technique specified in this section verifies that the defining parameters of an elliptic curve were indeed selected using the method specified in Annex A.3.3.

A.3.4.1 Elliptic curves over F_{2^m}

Input: A bit string SEED and a field element $b \in F_{2^m}$.

Output: Acceptance or rejection of the input parameters.

Let $t = m$, $s = \lfloor (t-1)/160 \rfloor$, and $h = t - 160.s$.

1. Compute H = SHA-1(SEED), and let b_0 denote the bit string of length h bits obtained by taking the h rightmost bits of H.
2. For i from 1 to s do:
   Compute b_i = SHA-1((SEED + i) mod 2^g).
3. Let b' be the field element obtained by the concatenation of $b_0,b_1,\ldots,b_s$ as follows:

$b' = b_0 \| b_1 \| \ldots \| b_s$.

4. If $b = b'$, then accept; otherwise reject.

A.3.4.2 Elliptic curves over F_p

Input: A bit string SEED and field elements $a, b \in F_p$.

Output: Acceptance or rejection of the input parameters.

Let $t = \lfloor \log_2 p \rfloor$, $s = \lfloor (t-1)/160 \rfloor$, and $h = t - 160 \cdot s$.

1. Compute H = SHA-1(SEED) and let c_0 denote the bit string of length h bits obtained by taking the h rightmost bits of H.
2. Let W_0 denote the bit string of length h bits obtained by setting the leftmost bit of c_0 to 0.
3. For i from 1 to s do:
   Compute W_i = SHA-1((SEED + i) mod 2^g).
4. Let W' be the bit string obtained by the concatenation of $W_0, W_1, \ldots, W_s$ as follows:
   $W' = W_0 \| W_1 \| \ldots \| W_s$.
5. Let $w_1, w_2, \ldots, w_t$ be the bits of W from leftmost to rightmost. Let r' be the integer $r' = \sum_{i=1}^{t} w_i 2^{t-i}$.
6. If $r'.b^2 \equiv a^3 \pmod{p}$, then accept; otherwise reject.

A.4 Pseudorandom Number Generation

Any implementation of the ECDSA requires the ability to generate random or pseudorandom integers. Such numbers are used to derive a user's private key, d, and a user's per-message secret number k. These randomly or pseudorandomly generated integers are selected to be between 1 and n-1 inclusive, where n is a prime number. If pseudorandom numbers are desired, they shall be generated by the techniques given in this section or in an ANSI X9 approved standard.

A.4.1 Algorithm Derived from FIPS 186

The algorithm described in this section employs a one-way function $G(t, c)$, where t is 160 bits, c is b bits ($160 \leq b \leq 512$), and $G(t, c)$ is 160 bits. One way to construct G is via the Secure Hash Algorithm (SHA-1), as defined in ANSI X9.30 Part 2 [4]. A second method for constructing G is to use the Data Encryption Algorithm (DEA) as specified in ANSI X3.92 [1]. The construction of G by these techniques is described in Annexes A.4.1.1 and A.4.1.2, respectively.

In the algorithm specified below, a secret b-bit seed-key XKEY is used. If G is constructed via SHA-1 as defined in Annex A.4.1.1, then b shall be between 160 and 512. If DEA is used to construct G as defined in Annex A.4.1.2, then b shall be equal to 160. The algorithm optionally allows the use of a user provided input.

Input: A prime number n, positive integer l, and integer b ($160 \leq b \leq 512$).

Output: l pseudorandom integers $k_1, k_2, \ldots, k_l$ in the interval [1, n-1].

1. Let $s = \lfloor \log_2 n \rfloor + 1$ and $f = \lceil s/160 \rceil$.
2. Choose a new, secret value for the seed-key, XKEY. (XKEY is of length b bits.)
3. In hexadecimal notation, let:
   t = 67452301 EFCDAB89 98BADCFE 10325476 C3D2E1F0.
   This is the initial value for $H_0 \| H_1 \| H_2 \| H_3 \| H_4$ in SHA-1.
4. For i from 1 to l do the following:
   4.1. For j from 1 to f do the following:
      4.1.1. $\text{XSEED}_{i,j}$ = optional user input.
      4.1.2. XVAL = (XKEY + $\text{XSEED}_{i,j}$) mod 2^b.
      4.1.3. $x_j = G(t, \text{XVAL})$.
      4.1.4. XKEY = (1 + XKEY + x_j) mod 2^b.
   4.2. Set $k_i = ((x_1 \| x_2 \| \ldots \| x_f) \bmod (n-1)) + 1$.
5. Output $(k_1, k_2, \ldots, k_l)$.

NOTE— The optional user input $XSEED_{i,j}$ in step 4.1.1 permits a user to augment the seed-key XKEY with random or pseudorandom numbers derived from alternate sources. The values of $XSEED_{i,j}$ must have the same security requirements as the seed-key XKEY. That is, they must be protected from unauthorized disclosure and be unpredictable.

A.4.1.1 Constructing the Function G from the SHA-1

$G(t,c)$ may be constructed using steps (a)-(e) in Annex 3.3 of ANSI X9.30 Part 2 [4]. Before executing these steps, $\{H_j\}$ and M_1 must be initialized as follows:

1. Initialize the $\{H_j\}$ by dividing the 160-bit value t into five 32-bit segments as follows:

   $t = t_0 \| t_1 \| t_2 \| t_3 \| t_4$.

   Then $H_j = t_j$ for $j = 0$ through 4.
2. There will be only one message block, M_1, which is initialized as follows:

   $M_1 = c \| 0^{512-b}$.

   (The first b bits of M_1 contain c, and the remaining (512-b) bits are set to zero.)

Then steps (a) through (e) of Section 3.3 of ANSI X9.30 Part 2 [4] are executed, and $G(t,c)$ is the 160-bit string represented by the five words:

$H_0 \| H_1 \| H_2 \| H_3 \| H_4$

at the end of step (e).

A.4.1.2 Constructing the Function G from the DEA

$G(t, c)$ may be constructed using the DEA (Data Encryption Algorithm) as specified in ANSI X3.92 [1].

Let $a \oplus b$ denote the bitwise exclusive-or of bit strings a and b, and let $a \| b$ denote the concatenation of bit strings.

If b_1 is a 32-bit string, then b_1' denotes the 24 least significant bits of b_1.

In the following, $DEA_K(A)$ represents ordinary DEA encryption of the 64-bit block A using the 56-bit key K. Now suppose t and c are each 160 bits. To compute $G(t,c)$:

1. Write:

   $t = t_1 \| t_2 \| t_3 \| t_4 \| t_5$.

   $c = c_1 \| c_2 \| c_3 \| c_4 \| c_5$.

   In the above, t_i and c_i are each 32 bits in length.
2. For i from 1 to 5 do:

   $x_i = t_i \oplus c_i$.
3. For i from 1 to 5 do:

   $b_1 = c_{((i+3) \bmod 5)+1}$

   $b_2 = c_{((i+2) \bmod 5)+1}$

   $a_1 = x_i$

   $a_2 = x_{(i \bmod 5)+1} \oplus x_{((i+3) \bmod 5)+1}$

   $y_{i,1} \| y_{i,2} = DEA_{b_1' \| b_2}(a_1 \| a_2)$,

   where $y_{i,1}$ and $y_{i,2}$ are each 32 bits in length.
4. For i from 1 to 5 do:

   $z_i = y_{i,1} \oplus y_{((i+1) \bmod 5)+1,2} \oplus y_{((i+2) \bmod 5)+1,1}$.
5. Let $G(t,c) = z_1 \| z_2 \| z_3 \| z_4 \| z_5$.

Annex B
(informative)
Mathematical Background

B.1 The Finite Field F_p

Let p be a prime number. There are many ways to represent the elements of the finite field with p elements. The most commonly used representation is the one defined in this section.

The finite field F_p is comprised of the set of integers:

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

with the following arithmetic operations:

— *Addition*: If $a, b \in F_p$, then $a + b = r$, where r is the remainder when the integer $a + b$ is divided by p, $r \in [0, p-1]$. This is known as addition modulo p (mod p).

— *Multiplication*: If $a, b \in F_p$, then $ab = s$, where s is the remainder when the integer ab is divided by p, $s \in [0, p-1]$. This is known as multiplication modulo p (mod p).

Let F_p^* denote all the non-zero elements in F_p. In F_p, there exists at least one element g such that any non-zero element of F_p can be expressed as a power of g. Such an element g is called a *generator* (or *primitive element*) of F_p^*. That is:

$$F_p^* = \{g^i : 0 \le i \le p-2\}.$$

The *multiplicative inverse* of $a = g^i \in F_p^*$, where $0 \le i \le p - 2$, is:

$$a^{-1} = g^{p-1-i}.$$

Example 1: The finite field F_2.

$F_2 = \{0, 1\}$. The addition and multiplication tables for F_2 are:

+	0	1
0	0	1
1	1	0

•	0	1
0	0	0
1	0	1

Example 2: The finite field F_{23}.

$F_{23} = \{0,1,2,...,22\}$. Examples of the arithmetic operations in F_{23} are:

1. $12 + 20 = 32 \bmod 23 = 9$, since the remainder is 9 when 32 is divided by 23.
2. $8 \cdot 9 = 72 \bmod 23 = 3$, since the remainder is 3 when 72 is divided by 23.

The element 5 is a generator of F_{23}^*. The powers of 5 modulo 23 are:

$5^0 = 1$	$5^1 = 5$	$5^2 = 2$	$5^3 = 10$	$5^4 = 4$	$5^5 = 20$
$5^6 = 8$	$5^7 = 17$	$5^8 = 16$	$5^9 = 11$	$5^{10} = 9$	$5^{11} = 22$
$5^{12} = 18$	$5^{13} =21$	$5^{14} = 13$	$5^{15} = 19$	$5^{16} = 3$	$5^{17} = 15$
$5^{18} = 6$	$5^{19} =7$	$5^{20} = 12$	$5^{21} = 14$	$5^{22} =1$.	

B.2 The Finite Field F_{2^m}

There are many ways to construct a finite field with 2^m elements. The field F_{2^m} can be viewed as a vector space of dimension m over F_2. That is, there exist m elements $\alpha_0, \alpha_1, \ldots, \alpha_{m-1}$ in F_{2^m} such that each element $\alpha \in F_{2^m}$ can be uniquely written in the form:

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

Such a set $\{\alpha_0, \alpha_1, \ldots, \alpha_{m-1}\}$ of elements is called a *basis* of F_{2^m} over F_2. Given such a basis, we can represent a field element α as the binary vector $(a_0, a_1, \ldots, a_{m-1})$. Addition of field elements is performed by bitwise XOR-ing the vector representations.

There are many different bases of F_{2^m} over F_2. Some bases lead to more efficient software and/or hardware implementations of the arithmetic in F_{2^m} than other bases. In this section, three kinds of bases are discussed. Annex B.2.1 introduces *polynomial bases* which use polynomial addition, multiplication, division and remainder. Annex B.2.2 introduces special kinds of polynomial bases called *trinomial* and *pentanomial bases*. Annex B.2.3 introduces *normal bases*. Annex B.2.4 introduces special kinds of normal bases called *Gaussian normal bases* (GNB).

B.2.1 Polynomial Bases

Let $f(x)= x^m+ f_{m-1}x^{m-1}+ \ldots+ f_2x^2+ f_1x+ f_0$ (where $f_i \in F_2$ for $i = 0, ..., m-1$) be an irreducible polynomial of degree m over F_2, i.e., $f(x)$ cannot be factored as a product of two or more polynomials over F_2, each of degree less than m. $f(x)$ is called the *reduction polynomial*. The *finite field* F_{2^m} is comprised of all polynomials over F_2 of degree less than m:

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

The field element $(a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + ... + a_1x + a_0)$ is usually denoted by the bit string $(a_{m-1}...a_1a_0)$ of length m, so that:

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

Thus the elements of F_{2^m} can be represented by the set of all bit strings of length m. The multiplicative identity element (1) is represented by the bit string (00…01), while the zero element is represented by the bit string of all 0's.

Field elements are added and multiplied as follows:

B.2.1.1 Field addition

Field elements are added as follows:

$$(a_{m-1}...a_1a_0) + (b_{m-1}\ldots b_1b_0) = (c_{m-1}\ldots c_1c_0)$$

where $c_i = a_i \oplus b_i$. That is, field addition is performed componentwise.

B.2.1.2 Field multiplication

Field elements are multiplied as follows:

$$(a_{m-1}...a_1a_0) \,.\, (b_{m-1}\ldots b_1b_0) = (r_{m-1}\ldots r_1r_0),$$

where the polynomial $(r_{m-1}x^{m-1}+\ldots+ r_1x+ r_0)$ is the remainder when the polynomial:

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

is divided by $f(x)$ over F_2.

This method of representing F_{2^m} is called a *polynomial basis representation*, and $\{x^{m-1}, \ldots, x^2, x, 1\}$ is called a *polynomial basis* of F_{2^m} over F_2.

Note that F_{2^m} contains exactly 2^m elements. Let $F_{2^m}^*$ denote the set of all non-zero elements in F_{2^m}. There exists at least one element g in F_{2^m} such that any non-zero element of F_{2^m} can be expressed as a power of g. Such an element g is called a *generator* (or *primitive element*) of F_{2^m}. That is:

$$F_{2^m}^* = \{g^i : 0 \leq i \leq 2^m - 2\}.$$

The *multiplicative inverse* of $a = g^i \in F_{2^m}^*$, where $0 \leq i \leq 2^m - 2$, is:

$$a^{-1} = g^{2^m-1-i}.$$

Example 3: The finite field F_{2^4} using a polynomial basis representation.

Take $f(x) = x^4 + x +1$ over F_2; it can be verified that $f(x)$ is irreducible over F_2. Then the elements of F_{2^4} are:

(0000)	(1000)	(0100)	(1100)	(0010)	(1010)	(0110)	(1110)
(0001)	(1001)	(0101)	(1101)	(0011)	(1011)	(0111)	(1111).

As examples of field arithmetic, we have:

$$(1101) + (1001) = (0100), \text{ and}$$

$$(1101) \times (1001) = (1111)$$

since:

$$\begin{aligned}(x^3+x^2+1)(x^3+1) &= x^6+x^5+x^2+1\\ &= (x^4+x+1)(x^2+x)+(x^3+x^2+x+1)\\ &= x^3+x^2+x+1 \bmod f(x)\end{aligned}$$

i.e., $x^3 + x^2 + x + 1$ is the remainder when $(x^3 + x^2 + 1) \times (x^3 + 1)$ is divided by $f(x)$.
The multiplicative identity is (0001).

$F^*_{2^4}$ can be generated by the element $\alpha = x$. The powers of α are:

α^0 = (0001)	α^1 = (0010)	α^2 = (0100)	α^3 = (1000)
α^4 = (0011)	α^5 = (0110)	α^6 = (1100)	α^7 = (1011)
α^8 = (0101)	α^9 = (1010)	α^{10} = (0111)	α^{11} = (1110)
α^{12} = (1111)	α^{13} = (1101)	α^{14} = (1001).	

B.2.2 Trinomial and Pentanomial Bases

A *trinomial basis* (TPB) and a *pentanomial basis* (PPB) are special types of polynomial bases. A *trinomial* over F_2 is a polynomial of the form $x^m + x^k + 1$, where $1 \le k \le m-1$. A *pentanomial* over F_2 is a polynomial of the form $x^m + x^{k3} + x^{k2} + x^{k1} + 1$, where $1 \le k1 < k2 < k3 \le m-1$.

A *trinomial basis representation* of F_{2^m} is a polynomial basis representation determined by an irreducible trinomial $f(x) = x^m + x^k + 1$ of degree m over F_2. Such trinomials only exist for certain values of m. Example 3 above is an example of a trinomial basis representation of the finite field F_{2^4}.

A *pentanomial basis representation* of F_{2^m} is a polynomial basis representation determined by an irreducible pentanomial $f(x) = x^m + x^{k3} + x^{k2} + x^{k1} + 1$ of degree m over F_2. Such pentanomials exist for all values of $m \ge 4$.

B.2.3 Normal Bases

A *normal basis* of F_{2^m} over F_2 is a basis of the form:

$$\{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}\},$$

where $\beta \in F_{2^m}$. Such a basis always exists. Given any element $\alpha \in F_{2^m}$, we can write $\alpha = \sum_{i=0}^{m-1} a_i \beta^{2^i}$, where $a_i \in \{0,1\}$. This field element α is denoted by the binary string $(a_0 a_1 a_2 \ldots a_{m-1})$ of length m, so that:

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

Note that, by convention, the ordering of bits is different from that of a polynomial basis representation (Annex B.2.1).

The multiplicative identity element (1) is represented by the bit string of all 1's (11. . .11), while the zero element is represented by the bit string of all 0's.

Since squaring is a linear operator in F_{2^m}, we have:

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

with indices reduced modulo m. Hence a normal basis representation of F_{2^m} is advantageous because squaring a field element can then be accomplished by a simple rotation of the vector representation, an operation that is easily implemented in hardware.

B.2.4 Gaussian Normal Bases

In Example 3, the field F_{2^4} was described using polynomial multiplication, division and remainders. A Gaussian normal basis representation, as defined in Section 4.1.2.2, may also be used to construct the field F_{2^4}.

Example 4: The finite field F_{2^4} using a Gaussian normal basis representation.

As in Example 3, the elements of F_{2^4} are the binary 4-tuples:

(0000)	(0001)	(0010)	(0011)	(0100)	(0101)	(0110)	(0111)
(1000)	(1001)	(1010)	(1011)	(1100)	(1101)	(1110)	(1111).

Field elements are added and multiplied as follows:

Field addition:

$$(a_0a_1a_2a_3) + (b_0b_1b_2b_3) = (c_0c_1c_2c_3)$$

where $c_i = a_i \oplus b_i$. In other words, field addition is performed by simply XORing the vector representation.

Field multiplication: The setup for multiplication is done as follows. See Section 4.1.2.2 for a description of the steps that are performed.

(See Section 4.1.2.2.2 for a description of the setup steps performed below.)

For the type 3 normal basis for F_{2^4}, the values of F are given by:

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

Therefore, after simplifying one obtains:

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

Here c_0 is the first coordinate of the product:

$$(c_0\ c_1 \ldots c_{m-1}) = (a_0\ a_1 \ldots a_{m-1}) \times (b_0\ b_1 \ldots b_{m-1}).$$

The other coordinates of the product are obtained from the formula for c_0 by cycling the subscripts modulo m. Thus:

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

(See Section 4.1.2.2.3 for a description of the setup steps performed below.)

We have $F(u, v) = u_0 (v_1 + v_2 + v_3) + u_1 (v_0 + v_2) + u_2 (v_0 + v_1) + u_3 (v_0 + v_3)$.

If:

$$a = (1000) \text{ and } b = (1101),$$

then:

$$c_0 = F((1000), (1101)) = 0,$$
$$c_1 = F((0001), (1011)) = 0,$$
$$c_2 = F((0010), (0111)) = 1,$$
$$c_3 = F((0100), (1110)) = 0,$$

so that $c = ab = (0010)$.

B.3 Elliptic Curves over F_p

Let $p > 3$ be a prime number. Let $a, b \in F_p$ be such that $4a^3 + 27b^2 \neq 0$ in F_p. An *elliptic curve* $E(F_p)$ over F_p defined by the parameters a and b is the set of solutions (x, y), for $x,y \in F_p$, to the equation: $y^2 = x^3 + ax + b$, together with an extra point $\mathcal{O}$, the *point at infinity*. The number of points in $E(F_p)$ is denoted by $\#E(F_p)$. The Hasse Theorem tells us that:

$$p+1-2\sqrt{p} \leq \#E(F_p) \leq p+1+2\sqrt{p}.$$

The set of points $E(F_p)$ forms a group with the following addition rules:

1. $\mathcal{O} + \mathcal{O} = \mathcal{O}$.
2. $(x, y) + \mathcal{O} = \mathcal{O} + (x, y) = (x, y)$ for all $(x, y) \in E(F_p)$.
3. $(x, y) + (x,-y) = \mathcal{O}$ for all $(x, y) \in E(F_p)$ (i.e., the negative of the point (x, y) is $-(x, y) = (x,-y)$).
4. (Rule for adding two distinct points that are not inverses of each other)

   Let $(x_1,y_1) \in E(F_p)$ and $(x_2,y_2) \in E(F_p)$ be two points such that $x_1 \neq x_2$.
   Then $(x_1,y_1) + (x_2,y_2) = (x_3,y_3)$, where:

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

5. (Rule for doubling a point)

   Let $(x_1, y_1) \in E(F_p)$ be a point with $y_1 \neq 0$.
   Then $2(x_1, y_1) = (x_3, y_3)$, where:

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

The group $E(F_p)$ is *abelian*, which means that $P_1+P_2 = P_2+P_1$ for all points P_1 and P_2 in $E(F_p)$. The curve is said to be *supersingular* if $\#E(F_p) = p+1$; otherwise it is *non-supersingular*. Only non-supersingular curves shall be in compliance with this standard (see Annex H).

Example 5: An elliptic curve over F_{23}.
Let $y^2 = x^3 + x + 1$ be an equation over F_{23}. Here $a = 1$ and $b = 1$. Then the solutions over F_{23} to the equation of the elliptic curve are:

(0,1)	(0,22)	(1,7)	(1,16)	(3,10)	(3,13)	(4,0)	(5,4)	(5,19)
(6,4)	(6,19)	(7,11)	(7,12)	(9,7)	(9,16)	(11,3)	(11,20)	(12,4)
(12,19)	(13,7)	(13,16)	(17,3)	(17,20)	(18,3)	(18,20)	(19,5)	(19,18).

The solutions were obtained by trial and error. The group $E(F_{23})$ has 28 points (including the point at infinity $\mathcal{O}$). The following are examples of the group operation.

1. Let $P_1 = (3,10)$, $P_2 = (9,7)$, $P_1 + P_2 = (x_3, y_3)$. Compute:

   $$\lambda = \frac{y_2 - y_1}{x_2 - x_1} = \frac{7-10}{9-3} = \frac{-3}{6} = \frac{-1}{2} = 11 \in F_{23},$$

   $$x_3 = \lambda^2 - x_1 - x_2 = 11^2 - 3 - 9 = 6 - 3 - 9 = -6 = 17,$$

   $$y_3 = \lambda(x_1 - x_3) - y_1 = 11(3 - 17) - 10 = 11(9) - 10 = 89 = 20.$$

   Therefore $P_1 + P_2 = (17, 20)$.

2. Let $P_1 = (3, 10)$, $2P_1 = (x_3, y_3)$. Compute:

   $$\lambda = \frac{3x_1^2 + a}{2y_1} = \frac{3(3^2)+1}{20} = \frac{5}{20} = \frac{1}{4} = 6,$$

   $$x_3 = \lambda^2 - 2x_1 = 6^2 - 6 = 30 = 7,$$

   $$y_3 = \lambda(x_1 - x_3) - y_1 = 6(3 - 7) - 10 = -24 - 10 = -11 = 12.$$

   Therefore $2P_1 = (7, 12)$.

B.4 Elliptic Curves over F_{2^m}

A non-supersingular *elliptic curve* $E(F_{2^m})$ over F_{2^m} defined by the parameters $a, b \in F_{2^m}$, $b \neq 0$, is the set of solutions (x, y), $x \in F_{2^m}$, $y \in F_{2^m}$, to the equation $y^2 + xy = x^3 + ax^2 + b$ together with an extra point $\mathcal{O}$, the *point at infinity*. The number of points in $E(F_{2^m})$ is denoted by $\#E(F_{2^m})$. The Hasse Theorem tells us that:

$$q + 1 - 2\sqrt{q} \leq \#E(F_{2^m}) \leq q + 1 + 2\sqrt{q},$$

where $q = 2^m$. Furthermore, $\#E(F_{2^m})$ is even.

The set of points $E(F_{2^m})$ forms a group with the following addition rules:

1. $\mathcal{O} + \mathcal{O} = \mathcal{O}$.
2. $(x, y) + \mathcal{O} = \mathcal{O} + (x, y) = (x, y)$ for all $(x, y) \in E(F_{2^m})$.
3. $(x, y) + (x, x + y) = \mathcal{O}$ for all $(x, y) \in E(F_{2^m})$ (i.e., the negative of the point (x, y) is $-(x, y) = (x, x + y)$).
4. (Rule for adding two distinct points that are not inverses of each other)
   Let $(x_1, y_1) \in E(F_{2^m})$ and $(x_2, y_2) \in E(F_{2^m})$ be two points such that $x_1 \neq x_2$. Then $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$, where:
   $$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a,\ \ y_3 = \lambda(x_1 + x_3) + x_3 + y_1, \text{ and } \lambda = \frac{y_1 + y_2}{x_1 + x_2}.$$
5. (Rule for doubling a point)
   Let $(x_1, y_1) \in E(F_{2^m})$ be a point with $x_1 \neq 0$. Then $2(x_1, y_1) = (x_3, y_3)$, where:
   $$x_3 = \lambda^2 + \lambda + a,\ \ y_3 = x_1^2 + (\lambda + 1)\, x_3, \text{ and } \lambda = x_1 + \frac{y_1}{x_1}.$$

The group $E(F_{2^m})$ is *abelian*, which means that $P_1 + P_2 = P_2 + P_1$ for all points P_1 and P_2 in $E(F_{2^m})$.

We now give two examples of elliptic curves over F_{2^4}. Example 6 uses a trinomial basis representation for the field, and Example 7 uses an optimal normal basis representation.

Example 6: An elliptic curve over F_{2^4}.

A trinomial basis representation is used for the elements of F_{2^4}. Consider the field F_{2^4} generated by the root $\alpha = x$ of the irreducible polynomial:

$$f(x) = x^4 + x + 1.$$

(See Example 3.) The powers of α are:

$\alpha^0 = (0001)$	$\alpha^1 = (0010)$	$\alpha^2 = (0100)$	$\alpha^3 = (1000)$
$\alpha^4 = (0011)$	$\alpha^5 = (0110)$	$\alpha^6 = (1100)$	$\alpha^7 = (1011)$
$\alpha^8 = (0101)$	$\alpha^9 = (1010)$	$\alpha^{10} = (0111)$	$\alpha^{11} = (1110)$
$\alpha^{12} = (1111)$	$\alpha^{13} = (1101)$	$\alpha^{14} = (1001)$	$\alpha^{15} = \alpha^0 = (0001)$.

Consider the non-supersingular elliptic curve over F_{2^4} with defining equation:

$$y^2 + xy = x^3 + \alpha^4 x^2 + 1.$$

Here, $a = \alpha^4$ and $b = 1$. The notation for this equation can be expressed as follows, since the multiplicative identity is (0001):

$$(0001)\, y^2 + (0001)\, xy = (0001)\, x^3 + (0011)\, x^2 + (0001).$$

Then the solutions over F_{2^4} to the equation of the elliptic curve are:

$(0, 1)$ $(1, \alpha^6)$ $(1, \alpha^{13})$ (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^1) (α^{10}, α^8) $(\alpha^{12}, 0)$ $(\alpha^{12}, \alpha^{12})$.

The group $E(F_{2^4})$ has 16 points (including the point at infinity $\mathcal{O}$). The following are examples of the group operation.

1. Let $P_1 = (x_1, y_1) = (\alpha^6, \alpha^8)$, $P_2 = (x_2, y_2) = (\alpha^3, \alpha^{13})$, and $P_1 + P_2 = (x_3, y_3)$. Then:
   $$\lambda = \frac{y_1 + y_2}{x_1 + x_2} = \frac{\alpha^8 + \alpha^{13}}{\alpha^6 + \alpha^3} = \alpha,$$
   $$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a = \alpha^2 + \alpha + \alpha^6 + \alpha^3 + \alpha^4 = 1,$$

$$y_3 = \lambda(x_1 + x_3) + x_3 + y_1 = \alpha(\alpha^6 + 1) + 1 + \alpha^8 = \alpha^{13}.$$

2. If $2P_1 = (x_3, y_3)$, then:

$$\lambda = x_1 + \frac{y_1}{x_1} = \alpha^6 + \frac{\alpha^8}{\alpha^6} = \alpha^3,$$

$$x_3 = \lambda^2 + \lambda + a = \alpha^6 + \alpha^3 + \alpha^4 = \alpha^{10},$$

$$y_3 = x_1^2 + (\lambda+1)x_3 = \alpha^{12} + (\alpha^3+1)\alpha^{10} = \alpha^8.$$

Example 7: An elliptic curve over F_{2^4}.

An optimal normal basis representation is used for the elements of F_{2^4}. Consider the field F_{2^4} given by the Type I optimal normal basis representation. α = (1100) is a generator for the non-zero elements, and (1111) is the multiplicative identity. The powers of α are:

α^0 = (1111)	α^1 = (1100)	α^2 = (0110)	α^3 = (0100)
α^4 = (0011)	α^5 = (1010)	α^6 = (0010)	α^7 = (0111)
α^8 = (1001)	α^9 = (1000)	α^{10} = (0101)	α^{11} = (1110)
α^{12} = (0001)	α^{13} = (1101)	α^{14} = (1011)	$\alpha^{15} = \alpha^0$ = (1111).

Consider the non-supersingular curve over F_{2^4} defined by the equation:

$$E : y^2 + xy = x^3 + \alpha^3.$$

Here, $a = 0$ and $b = \alpha^3$. The notation for this equation can be expressed as follows since the multiplicative identity is (1111):

$$(1111)\ y^2 + (1111)\ xy = (1111)\ x^3 + (0100).$$

The solutions over F_{2^4} to the elliptic curve equation are:

$(0, \alpha^9)$	$(\alpha, 0)$	(α,α)	(α^3, α^5)	(α^3, α^{11})	(α^4, α^3)	(α^4, α^7)
(α^5, α^3)	(α^5, α^{11})	$(\alpha^6, 0)$	(α^6, α^6)	(α^8, α^3)	(α^8, α^{13})	
$(\alpha^{11}, 0)$	$(\alpha^{11}, \alpha^{11})$	(α^{12}, α^8)	(α^{12}, α^9)	(α^{13}, α^2)	$(\alpha^{13}, \alpha^{14})$.	

Since there are 19 solutions to the equation in F_{2^4}, the group $E(F_{2^4})$ has 19 + 1 = 20 elements (including the point at infinity). This group turns out to be a cyclic group. If we take $G = (\alpha^3, \alpha^5)$ and use the addition formulae, we find that:

$1G = (\alpha^3, \alpha^5)$	$2G = (\alpha^4, \alpha^3)$	$3G = (\alpha^{13}, \alpha^2)$	$4G = (\alpha, 0)$	$5G = (\alpha^{12}, \alpha^8)$
$6G = (\alpha^8, \alpha^3)$	$7G = (\alpha^{11}, 0)$	$8G = (\alpha^5, \alpha^{11})$	$9G = (\alpha^6, 0)$	$10G = (0, \alpha^9)$
$11G = (\alpha^6, \alpha^6)$	$12G = (\alpha^5, \alpha^3)$	$13G = (\alpha^{11}, \alpha^{11})$	$14G = (\alpha^8, \alpha^{13})$	$15G = (\alpha^{12}, \alpha^9)$
$16G = (\alpha,\alpha)$	$17G = (\alpha^{13}, \alpha^{14})$	$18G = (\alpha^4, \alpha^7)$	$19G = (\alpha^3, \alpha^{11})$	$20G = \mathcal{O}$.

Annex C
(informative)
Tables of Trinomials, Pentanomials, and Gaussian Normal Bases

C.1 Table of GNB for F_{2^m}

Table C-1 – The type of GNB that shall be used for F_{2^m}.

Table C-1.a: This table lists each m, $160 \le m \le 300$, for which m is not divisible by 8.

m	*type*	*m*	*type*	*m*	*type*	*m*	*type*
161	6	196	1	230	2	266	6
162	1	197	18	231	2	267	8
163	4	198	22	233	2	268	1
164	5	199	4	234	5	269	8
165	4	201	8	235	4	270	2
166	3	202	6	236	3	271	6
167	14	203	12	237	10	273	2
169	4	203	12	238	7	274	9
170	6	204	3	239	2	275	14
171	12	205	4	241	6	276	3
172	1	206	3	242	6	277	4
173	2	207	4	243	2	278	2
174	2	209	2	244	3	279	4
175	4	210	2	245	2	281	2
177	4	211	10	246	11	282	6
178	1	212	5	247	6	283	6
180	1	214	3	250	9	285	10
181	6	215	6	251	2	286	3
182	3	217	6	252	3	287	6
183	2	218	5	253	10	289	12
185	8	219	4	254	2	290	5
186	2	220	3	255	6	291	6
187	6	221	2	257	6	292	1
188	5	222	10	258	5	293	2
189	2	223	12	259	10	294	3
190	10	225	22	260	5	295	16
191	2	226	1	261	2	297	6
193	4	227	24	262	3	298	6
194	2	228	9	263	6	299	2
195	6	229	12	265	4	300	19

Table C-1.b: The type of GNB that shall be used for F_{2^m}.

This table lists each m, $301 \leq m \leq 474$, for which m is not divisible by 8.

m	*type*	*m*	*type*	*m*	*type*	*m*	*type*
301	10	345	4	388	1	431	2
302	3	346	1	389	24	433	4
303	2	347	6	390	3	434	9
305	6	348	1	391	6	435	4
306	2	349	10	393	2	436	13
307	4	350	2	394	9	437	18
308	15	351	10	395	6	438	2
309	2	353	14	396	11	439	10
310	6	354	2	397	6	441	2
311	6	355	6	398	2	442	1
313	6	356	3	399	12	443	2
314	5	357	10	401	8	444	5
315	8	358	10	402	5	445	6
316	1	359	2	403	16	446	6
317	26	361	30	404	3	447	6
318	11	362	5	405	4	449	8
319	4	363	4	406	6	450	13
321	12	364	3	407	8	451	6
322	6	365	24	409	4	452	11
323	2	366	22	410	2	453	2
324	5	367	6	411	2	454	19
325	4	369	10	412	3	455	26
326	2	370	6	413	2	457	30
327	8	371	2	414	2	458	6
329	2	372	1	415	28	459	8
330	2	373	4	417	4	460	1
331	6	374	3	418	1	461	6
332	3	375	2	419	2	462	10
333	24	377	14	420	1	463	12
334	7	378	2	421	10	465	4
335	12	379	12	422	11	466	1
337	10	380	5	423	4	467	6
338	2	381	8	425	6	468	21
339	8	382	6	426	2	469	4
340	3	383	12	427	16	470	2
341	8	385	6	428	5	471	8
342	6	386	2	429	2	473	2
343	4	387	4	430	3	474	5

Table C-1.c: The type of GNB that shall be used for F_{2^m}.

This table lists each m, $475 \leq m \leq 647$, for which m is not divisible by 8.

m	*type*	*m*	*type*	*m*	*type*	*m*	*type*
475	4	518	14	562	1	605	6
476	5	519	2	563	14	606	2
477	46	521	32	564	3	607	6
478	7	522	1	565	10	609	4
479	8	523	10	566	3	610	10
481	6	524	5	567	4	611	2
482	5	525	8	569	12	612	1
483	2	526	3	570	5	613	10
484	3	527	6	571	10	614	2
485	18	529	24	572	5	615	2
486	10	530	2	573	4	617	8
487	4	531	2	574	3	618	2
489	12	532	3	575	2	619	4
490	1	533	12	577	4	620	3
491	2	534	7	578	6	621	6
492	13	535	4	579	10	622	3
493	4	537	8	580	3	623	12
494	3	538	6	581	8	625	36
495	2	539	12	582	3	626	21
497	20	540	1	583	4	627	20
498	9	541	18	585	2	628	7
499	4	542	3	586	1	629	2
500	11	543	2	587	14	630	14
501	10	545	2	588	11	631	10
502	10	546	1	589	4	633	34
503	6	547	10	590	11	634	13
505	10	548	5	591	6	635	8
506	5	549	14	593	2	636	13
507	4	550	7	594	17	637	4
508	1	551	6	595	6	638	2
509	2	553	4	596	3	639	2
510	3	554	2	597	4	641	2
511	6	555	4	598	15	642	6
513	4	556	1	599	8	643	12
514	33	557	6	601	6	644	3
515	2	558	2	602	5	645	2
516	3	559	4	603	12	646	6
517	4	561	2	604	7	647	14

Table C-1.d: The type of GNB that shall be used for F_{2^m}.

This table lists each m, $648 \leq m \leq 821$, for which m is not divisible by 8.

m	*type*	*m*	*type*	*m*	*type*	*m*	*type*
649	10	692	5	735	8	779	2
650	2	693	6	737	6	780	13
651	2	694	3	738	5	781	16
652	1	695	18	739	4	782	3
653	2	697	4	740	3	783	2
654	14	698	5	741	2	785	2
655	4	699	4	742	15	786	1
657	10	700	1	743	2	787	6
658	1	701	18	745	10	788	11
659	2	702	14	746	2	789	14
660	1	703	6	747	6	790	3
661	6	705	6	748	7	791	2
662	3	706	21	749	2	793	6
663	14	707	6	750	14	794	14
665	14	708	1	751	6	795	10
666	22	709	4	753	16	796	1
667	6	710	3	754	10	797	6
668	11	711	8	755	2	798	6
669	4	713	2	756	1	799	22
670	6	714	5	757	16	801	12
671	6	715	4	758	6	802	6
673	4	716	5	759	4	803	2
674	5	717	18	761	2	804	5
675	22	718	15	762	10	805	6
676	1	719	2	763	22	806	11
677	8	721	6	764	3	807	14
678	10	722	26	765	2	809	2
679	10	723	2	766	6	810	2
681	22	724	13	767	6	811	10
682	6	725	2	769	10	812	3
683	2	726	2	770	5	813	4
684	3	727	4	771	2	814	15
685	4	729	24	772	1	815	8
686	2	730	13	773	6	817	6
687	10	731	8	774	2	818	2
689	12	732	11	775	6	819	20
690	2	733	10	777	16	820	1
691	10	734	3	778	21	821	8

Table C-1.e: The type of GNB that shall be used for F_{2^m}.

This table lists each m, $822 \le m \le 995$, for which m is not divisible by 8.

m	*type*	*m*	*type*	*m*	*type*	*m*	*type*
822	3	866	2	909	4	953	2
823	10	867	4	910	18	954	49
825	6	868	19	911	2	955	10
826	1	869	12	913	6	956	15
827	14	870	2	914	18	957	6
828	1	871	6	915	10	958	6
829	10	873	2	916	3	959	8
830	14	874	9	917	6	961	16
831	2	875	12	918	10	962	14
833	2	876	1	919	4	963	4
834	2	877	16	921	6	964	9
835	6	878	15	922	10	965	2
836	15	879	2	923	2	966	7
837	6	881	18	924	5	967	16
838	7	882	1	925	4	969	4
839	12	883	4	926	6	970	9
841	12	884	27	927	4	971	6
842	5	885	28	929	8	972	5
843	6	886	3	930	2	973	6
844	13	887	6	931	10	974	2
845	8	889	4	932	3	975	2
846	2	890	5	933	2	977	8
847	30	891	2	934	3	978	6
849	8	892	3	935	2	979	4
850	6	893	2	937	6	980	9
851	6	894	3	938	2	981	32
852	1	895	4	939	2	982	15
853	4	897	8	940	1	983	14
854	18	898	21	941	6	985	10
855	8	899	8	942	10	986	2
857	8	900	11	943	6	987	6
858	1	901	6	945	8	988	7
859	22	902	3	946	1	989	2
860	9	903	4	947	6	990	10
861	28	905	6	948	7	991	18
862	31	906	1	949	4	993	2
863	6	907	6	950	2	994	10
865	4	908	21	951	16	995	14

Table C-1.f: The type of GNB that shall be used for F_{2^m}.

This table lists each m, 996 ≤ m ≤ 1169, for which m is not divisible by 8.

m	*type*	*m*	*type*	*m*	*type*	*m*	*type*
996	43	1039	4	1083	10	1126	7
997	4	1041	2	1084	3	1127	6
998	2	1042	18	1085	18	1129	4
999	8	1043	2	1086	7	1130	5
1001	6	1044	7	1087	4	1131	8
1002	5	1045	6	1089	4	1132	13
1003	4	1046	6	1090	1	1133	2
1004	5	1047	36	1091	6	1134	2
1005	4	1049	2	1092	15	1135	10
1006	3	1050	10	1093	4	1137	6
1007	18	1051	12	1094	15	1138	6
1009	10	1052	5	1095	14	1139	24
1010	5	1053	12	1097	14	1140	5
1011	6	1054	3	1098	9	1141	12
1012	3	1055	2	1099	4	1142	23
1013	2	1057	4	1100	5	1143	16
1014	2	1058	14	1101	6	1145	8
1015	6	1059	14	1102	3	1146	2
1017	16	1060	1	1103	2	1147	6
1018	1	1061	6	1105	18	1148	5
1019	2	1062	3	1106	2	1149	14
1020	9	1063	4	1107	10	1150	19
1021	10	1065	2	1108	1	1151	6
1022	3	1066	6	1109	12	1153	22
1023	4	1067	8	1110	2	1154	2
1025	6	1068	7	1111	22	1155	2
1026	2	1069	10	1113	10	1156	3
1027	6	1070	2	1114	22	1157	8
1028	17	1071	10	1115	6	1158	6
1029	8	1073	30	1116	1	1159	4
1030	7	1074	13	1117	6	1161	12
1031	2	1075	6	1118	2	1162	9
1033	4	1076	3	1119	2	1163	32
1034	2	1077	18	1121	2	1164	9
1035	6	1078	6	1122	1	1165	6
1036	7	1079	14	1123	4	1166	2
1037	8	1081	12	1124	3	1167	8
1038	6	1082	9	1125	8	1169	2

Table C-1.g: The type of GNB that shall be used for F_{2^m}.

This table lists each m, 1170≤ m ≤ 1342, for which m is not divisible by 8.

m	*type*	*m*	*type*	*m*	*type*	*m*	*type*
1170	1	1213	12	1257	14	1300	1
1171	6	1214	3	1258	1	1301	20
1172	3	1215	14	1259	14	1302	3
1173	6	1217	24	1260	7	1303	16
1174	7	1218	2	1261	10	1305	12
1175	24	1219	4	1262	6	1306	1
1177	18	1220	5	1263	24	1307	8
1178	2	1221	8	1265	2	1308	7
1179	8	1222	6	1266	17	1309	18
1180	21	1223	2	1267	6	1310	2
1181	12	1225	10	1268	17	1311	22
1182	3	1226	5	1269	2,4	1313	6
1183	10	1227	34	1270	6	1314	5
1185	2	1228	1	1271	2	1315	4
1186	1	1229	2	1273	6	1316	5
1187	8	1230	3	1274	2	1317	10
1188	19	1231	16	1275	2	1318	7
1189	24	1233	2	1276	1	1319	18
1190	3	1234	25	1277	20	1321	6
1191	28	1235	6	1278	2	1322	6
1193	6	1236	1	1279	10	1323	2
1194	2	1237	16	1281	6	1324	15
1195	12	1238	2	1282	1	1325	6
1196	17	1239	4	1283	6	1326	7
1197	4	1241	20	1284	3	1327	4
1198	7	1242	5	1285	18	1329	2
1199	2	1243	4	1286	6	1330	9
1201	6	1244	3	1287	18	1331	2
1202	5	1245	14	1289	2	1332	11
1203	4	1246	6	1290	1	1333	4
1204	3	1247	18	1291	10	1334	3
1205	12	1249	10	1292	3	1335	44
1206	6	1250	18	1293	6	1337	14
1207	6	1251	2	1294	7	1338	2
1209	38	1252	19	1295	2	1339	12
1210	9	1253	26	1297	4	1340	3
1211	2	1254	10	1298	5	1341	2
1212	1	1255	12	1299	22	1342	3

Table C-1.h: The type of GNB that shall be used for F_{2^m}.

This table lists each m, 1343 ≤ m ≤ 1516, for which m is not divisible by 8.

m	*type*	*m*	*type*	*m*	*type*	*m*	*type*
1343	6	1387	16	1430	2	1474	9
1345	10	1388	11	1431	40	1475	8
1346	2	1389	4	1433	6	1476	25
1347	14	1390	10	1434	9	1477	6
1348	7	1391	12	1435	4	1478	2
1349	2	1393	4	1436	11	1479	8
1350	11	1394	2	1437	4	1481	2
1351	16	1395	20	1438	6	1482	1
1353	2	1396	13	1439	2	1483	10
1354	18	1397	8	1441	6	1484	17
1355	2	1398	2	1442	5	1485	10
1356	5	1399	18	1443	2	1486	15
1357	16	1401	2	1444	13	1487	6
1358	11	1402	9	1445	12	1489	10
1359	2	1403	6	1446	6	1490	5
1361	6	1404	7	1447	24	1491	16
1362	14	1405	6	1449	8	1492	1
1363	6	1406	3	1450	1	1493	14
1364	3	1407	6	1451	2	1494	3
1365	12	1409	2	1452	1	1495	6
1366	3	1410	42	1453	4	1497	18
1367	8	1411	6	1454	2	1498	1
1369	4	1412	29	1455	6	1499	2
1370	2	1413	26	1457	8	1500	7
1371	10	1414	3	1458	22	1501	6
1372	1	1415	8	1459	10	1502	3
1373	12	1417	40	1460	11	1503	10
1374	7	1418	2	1461	8	1505	2
1375	4	1419	8	1462	10	1506	10
1377	6	1420	3	1463	2	1507	4
1378	6	1421	2	1465	30	1508	5
1379	20	1422	10	1466	5	1509	2
1380	1	1423	4	1467	4	1510	10
1381	6	1425	2	1468	19	1511	2
1382	6	1426	1	1469	2	1513	4
1383	10	1427	6	1470	6	1514	9
1385	6	1428	21	1471	16	1515	12
1386	17	1429	4	1473	6	1516	3

Table C-1.i: The type of GNB that shall be used for F_{2^m}.

This table lists each m, 1517 ≤ m ≤ 1690, for which m is not divisible by 8.

m	*type*	*m*	*type*	*m*	*type*	*m*	*type*
1517	6	1561	16	1604	3	1647	6
1518	2	1562	21	1605	32	1649	2
1519	12	1563	12	1606	7	1650	6
1521	6	1564	7	1607	6	1651	6
1522	1	1565	6	1609	10	1652	3
1523	14	1566	6	1610	6	1653	2
1524	5	1567	4	1611	8	1654	7
1525	4	1569	4	1612	15	1655	6
1526	11	1570	1	1613	6	1657	16
1527	14	1571	8	1614	7	1658	5
1529	14	1572	25	1615	16	1659	2
1530	1	1573	6	1617	4	1660	7
1531	6	1574	3	1618	1	1661	2
1532	3	1575	8	1619	8	1662	3
1533	2	1577	6	1620	1	1663	4
1534	3	1578	25	1621	18	1665	10
1535	8	1579	4	1622	6	1666	1
1537	16	1580	5	1623	10	1667	8
1538	5	1581	12	1625	8	1668	1
1539	2	1582	18	1626	2	1669	10
1540	3	1583	2	1627	18	1670	3
1541	2	1585	22	1628	9	1671	16
1542	11	1586	18	1629	8	1673	2
1543	4	1587	8	1630	7	1674	33
1545	28	1588	7	1631	6	1675	4
1546	6	1589	8	1633	12	1676	17
1547	6	1590	7	1634	5	1677	4
1548	1	1591	6	1635	38	1678	6
1549	4	1593	2	1636	1	1679	2
1550	3	1594	9	1637	38	1681	10
1551	8	1595	12	1638	10	1682	6
1553	6	1596	3	1639	28	1683	4
1554	10	1597	4	1641	28	1684	7
1555	12	1598	11	1642	9	1685	2
1556	11	1599	4	1643	6	1686	3
1557	4	1601	2	1644	3	1687	10
1558	6	1602	6	1645	6	1689	8
1559	2	1603	6	1646	15	1690	6

Table C-1.j: The type of GNB that shall be used for F_{2^m}.

This table lists each m, 1691 ≤ m ≤ 1863, for which m is not divisible by 8.

m	*type*	*m*	*type*	*m*	*type*	*m*	*type*
1691	42	1734	2	1778	2	1821	2
1692	1	1735	10	1779	2	1822	18
1693	6	1737	4	1780	15	1823	6
1694	15	1738	6	1781	6	1825	10
1695	4	1739	8	1782	11	1826	6
1697	8	1740	1	1783	12	1827	14
1698	10	1741	22	1785	2	1828	9
1699	12	1742	3	1786	1	1829	2
1700	3	1743	6	1787	6	1830	18
1701	10	1745	2	1788	15	1831	6
1702	3	1746	1	1789	10	1833	26
1703	2	1747	10	1790	2	1834	10
1705	16	1748	5	1791	2	1835	2
1706	2	1749	2	1793	12	1836	5
1707	4	1750	6	1794	5	1837	4
1708	9	1751	8	1795	6	1838	2
1709	12	1753	4	1796	21	1839	8
1710	18	1754	9	1797	10	1841	6
1711	6	1755	2	1798	6	1842	25
1713	20	1756	21	1799	12	1843	6
1714	9	1757	8	1801	12	1844	5
1715	8	1758	2	1802	5	1845	2
1716	17	1759	18	1803	14	1846	7
1717	4	1761	6	1804	3	1847	6
1718	11	1762	9	1805	6	1849	12
1719	24	1763	2	1806	2	1850	2
1721	20	1764	5	1807	4	1851	28
1722	14	1765	18	1809	4	1852	3
1723	10	1766	2	1810	6	1853	14
1724	27	1767	4	1811	2	1854	2
1725	22	1769	2	1812	13	1855	6
1726	3	1770	14	1813	4	1857	14
1727	14	1771	40	1814	3	1858	6
1729	4	1772	5	1815	6	1859	2
1730	2	1773	2	1817	6	1860	1
1731	12	1774	3	1818	2	1861	40
1732	1	1775	6	1819	10	1862	6
1733	2	1777	4	1820	9	1863	2

Table C-1.k: The type of GNB that shall be used for F_{2^m}.

This table lists each m, 1864 ≤ m ≤ 2000, for which m is not divisible by 8.

m	*type*	*m*	*type*	*m*	*type*	*m*	*type*
1865	14	1901	2	1937	8	1972	1
1866	2	1902	35	1938	2	1973	2
1867	10	1903	10	1939	4	1974	3
1868	5	1905	4	1940	11	1975	4
1869	4	1906	1	1941	18	1977	8
1870	10	1907	6	1942	3	1978	1
1871	8	1908	25	1943	20	1978	1
1873	6	1909	22	1945	16	1979	20
1874	5	1910	11	1946	6	1980	5
1875	12	1911	22	1947	4	1981	6
1876	1	1913	14	1948	1	1982	11
1877	8	1914	10	1949	18	1983	2
1878	7	1915	6	1950	3	1985	8
1879	4	1916	3	1950	3	1986	1
1881	16	1917	4	1951	22	1987	4
1882	25	1918	10	1953	2	1988	5
1883	2	1919	12	1954	10	1989	10
1884	5	1921	6	1955	2	1990	7
1885	4	1922	9	1956	3	1991	18
1886	3	1923	2	1957	4	1993	6
1887	4	1923	2	1958	2	1994	2
1889	2	1924	7	1959	2	1995	18
1890	9	1925	2	1961	2	1996	1
1891	10	1926	2	1962	50	1997	44
1892	5	1927	18	1963	4	1998	19
1893	4	1929	4	1964	29	1999	10
1894	3	1930	1	1965	2		
1895	8	1931	2	1966	7		
1897	4	1932	5	1967	8		
1898	2	1933	12	1969	4		
1899	18	1934	14	1970	5		
1900	1	1935	14	1971	6		

C.2 Irreducible Trinomials over F_2

Table C-2 – Irreducible trinomials $x^m + x^k + 1$ over F_2.

Table C-2.a: For each m, $160 \le m \le 609$, for which an irreducible trinomial of degree m exists, the table lists the smallest k for which $x^m + x^k + 1$ is irreducible over F_2.

m	*k*	*m*	*k*	*m*	*k*	*m*	*k*	*m*	*k*	*m*	*k*
161	18	236	5	308	15	383	90	458	203	527	47
162	27	238	73	310	93	385	6	460	19	529	42
166	37	239	36	313	79	386	83	462	73	532	1
167	6	241	70	314	15	388	159	463	93	534	161
169	34	242	95	316	63	390	9	465	31	537	94
170	11	244	111	318	45	391	28	468	27	538	195
172	1	247	82	319	36	393	7	470	9	540	9
174	13	249	35	321	31	394	135	471	1	543	16
175	6	250	103	322	67	396	25	473	200	545	122
177	8	252	15	324	51	399	26	474	191	550	193
178	31	253	46	327	34	401	152	476	9	551	135
180	3	255	52	329	50	402	171	478	121	553	39
182	81	257	12	330	99	404	65	479	104	556	153
183	56	258	71	332	89	406	141	481	138	558	73
185	24	260	15	333	2	407	71	484	105	559	34
186	11	263	93	337	55	409	87	486	81	561	71
191	9	265	42	340	45	412	147	487	94	564	163
193	15	266	47	342	125	414	13	489	83	566	153
194	87	268	25	343	75	415	102	490	219	567	28
196	3	270	53	345	22	417	107	492	7	569	77
198	9	271	58	346	63	418	199	494	17	570	67
199	34	273	23	348	103	420	7	495	76	574	13
201	14	274	67	350	53	422	149	497	78	575	146
202	55	276	63	351	34	423	25	498	155	577	25
204	27	278	5	353	69	425	12	500	27	580	237
207	43	279	5	354	99	426	63	503	3	582	85
209	6	281	93	358	57	428	105	505	156	583	130
210	7	282	35	359	68	431	120	506	23	585	88
212	105	284	53	362	63	433	33	508	9	588	35
214	73	286	69	364	9	436	165	510	69	590	93
215	23	287	71	366	29	438	65	511	10	593	86
217	45	289	21	367	21	439	49	513	26	594	19
218	11	292	37	369	91	441	7	514	67	596	273
220	7	294	33	370	139	444	81	516	21	599	30
223	33	295	48	372	111	446	105	518	33	601	201
225	32	297	5	375	16	447	73	519	79	602	215
228	113	300	5	377	41	449	134	521	32	604	105
231	26	302	41	378	43	450	47	522	39	606	165
233	74	303	1	380	47	455	38	524	167	607	105
234	31	305	102	382	81	457	16	526	97	609	31

Table C-2.b: Irreducible trinomials $x^m + x^k + 1$ over F_2.

For each m, $610 \le m \le 1060$, for which an irreducible trinomial of degree m exists, the table lists the smallest k for which $x^m + x^k + 1$ is irreducible over F_2.

m	*k*	*m*	*k*	*m*	*k*	*m*	*k*	*m*	*k*	*m*	*k*
610	127	684	209	754	19	833	149	903	35	988	121
612	81	686	197	756	45	834	15	905	117	990	161
614	45	687	13	758	233	838	61	906	123	991	39
615	211	689	14	759	98	839	54	908	143	993	62
617	200	690	79	761	3	841	144	911	204	994	223
618	295	692	299	762	83	842	47	913	91	996	65
620	9	694	169	767	168	844	105	916	183	998	101
622	297	695	177	769	120	845	2	918	77	999	59
623	68	697	267	772	7	846	105	919	36	1001	17
625	133	698	215	774	185	847	136	921	221	1007	75
626	251	700	75	775	93	849	253	924	31	1009	55
628	223	702	37	777	29	850	111	926	365	1010	99
631	307	705	17	778	375	852	159	927	403	1012	115
633	101	708	15	780	13	855	29	930	31	1014	385
634	39	711	92	782	329	857	119	932	177	1015	186
636	217	713	41	783	68	858	207	935	417	1020	135
639	16	714	23	785	92	860	35	937	217	1022	317
641	11	716	183	791	30	861	14	938	207	1023	7
642	119	718	165	793	253	862	349	942	45	1025	294
646	249	719	150	794	143	865	1	943	24	1026	35
647	5	721	9	798	53	866	75	945	77	1028	119
649	37	722	231	799	25	868	145	948	189	1029	98
650	3	724	207	801	217	870	301	951	260	1030	93
651	14	726	5	804	75	871	378	953	168	1031	68
652	93	727	180	806	21	873	352	954	131	1033	108
654	33	729	58	807	7	876	149	956	305	1034	75
655	88	730	147	809	15	879	11	959	143	1036	411
657	38	732	343	810	159	881	78	961	18	1039	21
658	55	735	44	812	29	882	99	964	103	1041	412
660	11	737	5	814	21	884	173	966	201	1042	439
662	21	738	347	815	333	887	147	967	36	1044	41
663	107	740	135	817	52	889	127	969	31	1047	10
665	33	742	85	818	119	890	183	972	7	1049	141
668	147	743	90	820	123	892	31	975	19	1050	159
670	153	745	258	822	17	894	173	977	15	1052	291
671	15	746	351	823	9	895	12	979	178	1054	105
673	28	748	19	825	38	897	113	982	177	1055	24
676	31	750	309	826	255	898	207	983	230	1057	198
679	66	751	18	828	189	900	1	985	222	1058	27
682	171	753	158	831	49	902	21	986	3	1060	439

Table C-2.c: Irreducible trinomials $x^m + x^k + 1$ over F_2.

For each m, $1061 \le m \le 1516$, for which an irreducible trinomial of degree m exists, the table lists the smallest k for which $x^m + x^k + 1$ is irreducible over F_2.

m	*k*	*m*	*k*	*m*	*k*	*m*	*k*	*m*	*k*	*m*	*k*
1062	49	1140	141	1212	203	1287	470	1366	1	1441	322
1063	168	1142	357	1214	257	1289	99	1367	134	1442	395
1065	463	1145	227	1215	302	1294	201	1369	88	1444	595
1071	7	1146	131	1217	393	1295	38	1372	181	1446	421
1078	361	1148	23	1218	91	1297	198	1374	609	1447	195
1079	230	1151	90	1220	413	1298	399	1375	52	1449	13
1081	24	1153	241	1223	255	1300	75	1377	100	1452	315
1082	407	1154	75	1225	234	1302	77	1380	183	1454	297
1084	189	1156	307	1226	167	1305	326	1383	130	1455	52
1085	62	1158	245	1228	27	1306	39	1385	12	1457	314
1086	189	1159	66	1230	433	1308	495	1386	219	1458	243
1087	112	1161	365	1231	105	1310	333	1388	11	1460	185
1089	91	1164	19	1233	151	1311	476	1390	129	1463	575
1090	79	1166	189	1234	427	1313	164	1391	3	1465	39
1092	23	1167	133	1236	49	1314	19	1393	300	1466	311
1094	57	1169	114	1238	153	1319	129	1396	97	1468	181
1095	139	1170	27	1239	4	1321	52	1398	601	1470	49
1097	14	1174	133	1241	54	1324	337	1399	55	1471	25
1098	83	1175	476	1242	203	1326	397	1401	92	1473	77
1100	35	1177	16	1246	25	1327	277	1402	127	1476	21
1102	117	1178	375	1247	14	1329	73	1404	81	1478	69
1103	65	1180	25	1249	187	1332	95	1407	47	1479	49
1105	21	1182	77	1252	97	1334	617	1409	194	1481	32
1106	195	1183	87	1255	589	1335	392	1410	383	1482	411
1108	327	1185	134	1257	289	1337	75	1412	125	1486	85
1110	417	1186	171	1260	21	1338	315	1414	429	1487	140
1111	13	1188	75	1263	77	1340	125	1415	282	1489	252
1113	107	1190	233	1265	119	1343	348	1417	342	1490	279
1116	59	1191	196	1266	7	1345	553	1420	33	1492	307
1119	283	1193	173	1268	345	1348	553	1422	49	1495	94
1121	62	1196	281	1270	333	1350	237	1423	15	1497	49
1122	427	1198	405	1271	17	1351	39	1425	28	1500	25
1126	105	1199	114	1273	168	1353	371	1426	103	1503	80
1127	27	1201	171	1276	217	1354	255	1428	27	1505	246
1129	103	1202	287	1278	189	1356	131	1430	33	1508	599
1130	551	1204	43	1279	216	1358	117	1431	17	1510	189
1134	129	1206	513	1281	229	1359	98	1433	387	1511	278
1135	9	1207	273	1282	231	1361	56	1434	363	1513	399
1137	277	1209	118	1284	223	1362	655	1436	83	1514	299
1138	31	1210	243	1286	153	1364	239	1438	357	1516	277

Table C-2.d: Irreducible trinomials $x^m + x^k + 1$ over F_2.

For each m, $1517 \le m \le 2000$, for which an irreducible trinomial of degree m exists, the table lists the smallest k for which $x^m + x^k + 1$ is irreducible over F_2.

m	k	m	k	m	k	m	k	m	k	m	k
1518	69	1590	169	1673	90	1756	99	1838	53	1927	25
1519	220	1591	15	1674	755	1759	165	1839	836	1929	31
1521	229	1593	568	1676	363	1764	105	1841	66	1932	277
1524	27	1596	3	1678	129	1767	250	1844	339	1934	413
1526	473	1599	643	1679	20	1769	327	1846	901	1935	103
1527	373	1601	548	1681	135	1770	279	1847	180	1937	231
1529	60	1602	783	1687	31	1772	371	1849	49	1938	747
1530	207	1604	317	1689	758	1774	117	1854	885	1940	113
1534	225	1606	153	1692	359	1775	486	1855	39	1943	11
1535	404	1607	87	1694	501	1777	217	1857	688	1945	91
1537	46	1609	231	1695	29	1778	635	1860	13	1946	51
1540	75	1612	771	1697	201	1780	457	1862	149	1948	603
1542	365	1615	103	1698	459	1782	57	1863	260	1950	9
1543	445	1617	182	1700	225	1783	439	1865	53	1951	121
1545	44	1618	211	1703	161	1785	214	1866	11	1953	17
1548	63	1620	27	1705	52	1788	819	1870	121	1956	279
1550	189	1623	17	1708	93	1790	593	1871	261	1958	89
1551	557	1625	69	1710	201	1791	190	1873	199	1959	371
1553	252	1628	603	1711	178	1793	114	1878	253	1961	771
1554	99	1630	741	1713	250	1798	69	1879	174	1962	99
1556	65	1631	668	1716	221	1799	312	1881	370	1964	21
1558	9	1633	147	1719	113	1801	502	1884	669	1966	801
1559	119	1634	227	1721	300	1802	843	1886	833	1967	26
1561	339	1636	37	1722	39	1804	747	1887	353	1969	175
1562	95	1638	173	1724	261	1806	101	1889	29	1974	165
1564	7	1639	427	1726	753	1807	123	1890	371	1975	841
1566	77	1641	287	1729	94	1809	521	1895	873	1977	238
1567	127	1642	231	1734	461	1810	171	1900	235	1980	33
1569	319	1647	310	1735	418	1814	545	1902	733	1983	113
1570	667	1649	434	1737	403	1815	163	1903	778	1985	311
1572	501	1650	579	1738	267	1817	479	1905	344	1986	891
1575	17	1652	45	1740	259	1818	495	1906	931	1988	555
1577	341	1655	53	1742	869	1820	11	1908	945	1990	133
1578	731	1657	16	1743	173	1823	684	1911	67	1991	546
1580	647	1660	37	1745	369	1825	9	1913	462	1993	103
1582	121	1663	99	1746	255	1828	273	1918	477	1994	15
1583	20	1665	176	1748	567	1830	381	1919	105	1996	307
1585	574	1666	271	1750	457	1831	51	1921	468	1999	367
1586	399	1668	459	1751	482	1833	518	1924	327		
1588	85	1671	202	1753	775	1836	243	1926	357		

C.3 Irreducible Pentanomials over F_2

Table C-3 – Irreducible pentanomials $x^m + x^{k1} + x^{k2} + x^{k3} + 1$ over F_2.

Table C-3.a: For each m, $160 \le m \le 488$, for which an irreducible trinomial of degree m does not exist, a triple of exponents $k1$, $k2$, $k3$ is given for which the pentanomial $x^m + x^{k1} + x^{k2} + x^{k3} + 1$ is irreducible over F_2.

m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)
160	1, 2, 117	243	1, 2, 17	326	1, 2, 67	410	1, 2, 16
163	1, 2, 8	245	1, 2, 37	328	1, 2, 51	411	1, 2, 50
164	1, 2, 49	246	1, 2, 11	331	1, 2, 134	413	1, 2, 33
165	1, 2, 25	248	1, 2, 243	334	1, 2, 5	416	1, 3, 76
168	1, 2, 65	251	1, 2, 45	335	1, 2, 250	419	1, 2, 129
171	1, 3, 42	254	1, 2, 7	336	1, 2, 77	421	1, 2, 81
173	1, 2, 10	256	1, 2, 155	338	1, 2, 112	424	1, 2, 177
176	1, 2, 43	259	1, 2, 254	339	1, 2, 26	427	1, 2, 245
179	1, 2, 4	261	1, 2, 74	341	1, 2, 57	429	1, 2, 14
181	1, 2, 89	262	1, 2, 207	344	1, 2, 7	430	1, 2, 263
184	1, 2, 81	264	1, 2, 169	347	1, 2, 96	432	1, 2, 103
187	1, 2, 20	267	1, 2, 29	349	1, 2, 186	434	1, 2, 64
188	1, 2, 60	269	1, 2, 117	352	1, 2, 263	435	1, 2, 166
189	1, 2, 49	272	1, 3, 56	355	1, 2, 138	437	1, 2, 6
190	1, 2, 47	275	1, 2, 28	356	1, 2, 69	440	1, 2, 37
192	1, 2, 7	277	1, 2, 33	357	1, 2, 28	442	1, 2, 32
195	1, 2, 37	280	1, 2, 113	360	1, 2, 49	443	1, 2, 57
197	1, 2, 21	283	1, 2, 200	361	1, 2, 44	445	1, 2, 225
200	1, 2, 81	285	1, 2, 77	363	1, 2, 38	448	1, 3, 83
203	1, 2, 45	288	1, 2, 191	365	1, 2, 109	451	1, 2, 33
205	1, 2, 21	290	1, 2, 70	368	1, 2, 85	452	1, 2, 10
206	1, 2, 63	291	1, 2, 76	371	1, 2, 156	453	1, 2, 88
208	1, 2, 83	293	1, 3, 154	373	1, 3, 172	454	1, 2, 195
211	1, 2, 165	296	1, 2, 123	374	1, 2, 109	456	1, 2, 275
213	1, 2, 62	298	1, 2, 78	376	1, 2, 77	459	1, 2, 332
216	1, 2, 107	299	1, 2, 21	379	1, 2, 222	461	1, 2, 247
219	1, 2, 65	301	1, 2, 26	381	1, 2, 5	464	1, 2, 310
221	1, 2, 18	304	1, 2, 11	384	1, 2, 299	466	1, 2, 78
222	1, 2, 73	306	1, 2, 106	387	1, 2, 146	467	1, 2, 210
224	1, 2, 159	307	1, 2, 93	389	1, 2, 159	469	1, 2, 149
226	1, 2, 30	309	1, 2, 26	392	1, 2, 145	472	1, 2, 33
227	1, 2, 21	311	1, 3, 155	395	1, 2, 333	475	1, 2, 68
229	1, 2, 21	312	1, 2, 83	397	1, 2, 125	477	1, 2, 121
230	1, 2, 13	315	1, 2, 142	398	1, 3, 23	480	1, 2, 149
232	1, 2, 23	317	1, 3, 68	400	1, 2, 245	482	1, 2, 13
235	1, 2, 45	320	1, 2, 7	403	1, 2, 80	483	1, 2, 352
237	1, 2, 104	323	1, 2, 21	405	1, 2, 38	485	1, 2, 70
240	1, 3, 49	325	1, 2, 53	408	1, 2, 323	488	1, 2, 123

Table C-3.b: Irreducible pentanomials $x^m + x^{k1} + x^{k2} + x^{k3} + 1$ over F_2.

For each m, $490 \le m \le 811$, for which an irreducible trinomial of degree m does not exist, a triple of exponents $k1$, $k2$, $k3$ is given for which the pentanomial $x^m + x^{k1} + x^{k2} + x^{k3} + 1$ is irreducible over F_2.

m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)
491	1, 2, 270	571	1, 2, 408	653	1, 2, 37	734	1, 2, 67
493	1, 2, 171	572	1, 2, 238	656	1, 2, 39	736	1, 2, 359
496	1, 3, 52	573	1, 2, 220	659	1, 2, 25	739	1, 2, 60
499	1, 2, 174	576	1, 3, 52	661	1, 2, 80	741	1, 2, 34
501	1, 2, 332	578	1, 2, 138	664	1, 2, 177	744	1, 2, 347
502	1, 2, 99	579	1, 3, 526	666	1, 2, 100	747	1, 2, 158
504	1, 3, 148	581	1, 2, 138	667	1, 2, 161	749	1, 2, 357
507	1, 2, 26	584	1, 2, 361	669	1, 2, 314	752	1, 2, 129
509	1, 2, 94	586	1, 2, 14	672	1, 2, 91	755	1, 4, 159
512	1, 2, 51	587	1, 2, 130	674	1, 2, 22	757	1, 2, 359
515	1, 2, 73	589	1, 2, 365	675	1, 2, 214	760	1, 2, 17
517	1, 2, 333	591	1, 2, 38	677	1, 2, 325	763	1, 2, 17
520	1, 2, 291	592	1, 2, 143	678	1, 2, 95	764	1, 2, 12
523	1, 2, 66	595	1, 2, 9	680	1, 2, 91	765	1, 2, 137
525	1, 2, 92	597	1, 2, 64	681	1, 2, 83	766	1, 3, 280
528	1, 2, 35	598	1, 2, 131	683	1, 2, 153	768	1, 2, 115
530	1, 2, 25	600	1, 2, 239	685	1, 3, 4	770	1, 2, 453
531	1, 2, 53	603	1, 2, 446	688	1, 2, 71	771	1, 2, 86
533	1, 2, 37	605	1, 2, 312	691	1, 2, 242	773	1, 2, 73
535	1, 2, 143	608	1, 2, 213	693	1, 2, 250	776	1, 2, 51
536	1, 2, 165	611	1, 2, 13	696	1, 2, 241	779	1, 2, 456
539	1, 2, 37	613	1, 2, 377	699	1, 2, 40	781	1, 2, 209
541	1, 2, 36	616	1, 2, 465	701	1, 2, 466	784	1, 2, 59
542	1, 3, 212	619	1, 2, 494	703	1, 2, 123	786	1, 2, 118
544	1, 2, 87	621	1, 2, 17	704	1, 2, 277	787	1, 2, 189
546	1, 2, 8	624	1, 2, 71	706	1, 2, 27	788	1, 2, 375
547	1, 2, 165	627	1, 2, 37	707	1, 2, 141	789	1, 2, 5
548	1, 2, 385	629	1, 2, 121	709	1, 2, 9	790	1, 2, 111
549	1, 3, 274	630	1, 2, 49	710	1, 3, 29	792	1, 2, 403
552	1, 2, 41	632	1, 2, 9	712	1, 2, 623	795	1, 2, 137
554	1, 2, 162	635	1, 2, 64	715	1, 3, 458	796	1, 2, 36
555	1, 2, 326	637	1, 2, 84	717	1, 2, 320	797	1, 2, 193
557	1, 2, 288	638	1, 2, 127	720	1, 2, 625	800	1, 2, 463
560	1, 2, 157	640	1, 3, 253	723	1, 2, 268	802	1, 2, 102
562	1, 2, 56	643	1, 2, 153	725	1, 2, 331	803	1, 2, 208
563	1, 4, 159	644	1, 2, 24	728	1, 2, 51	805	1, 2, 453
565	1, 2, 66	645	1, 2, 473	731	1, 2, 69	808	1, 3, 175
568	1, 2, 291	648	1, 2, 235	733	1, 2, 92	811	1, 2, 18

Table C-3.c: Irreducible pentanomials $x^m + x^{k1} + x^{k2} + x^{k3} + 1$ over F_2.

For each m, $812 \le m \le 1131$, for which an irreducible trinomial of degree m does not exist, a triple of exponents $k1$, $k2$, $k3$ is given for which the pentanomial $x^m + x^{k1} + x^{k2} + x^{k3} + 1$ is irreducible over F_2.

m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)
813	1, 2, 802	901	1, 2, 581	973	1, 2, 113	1053	1, 2, 290
816	1, 3, 51	904	1, 3, 60	974	1, 2, 211	1056	1, 2, 11
819	1, 2, 149	907	1, 3, 26	976	1, 2, 285	1059	1, 3, 6
821	1, 2, 177	909	1, 3, 168	978	1, 2, 376	1061	1, 2, 166
824	1, 2, 495	910	1, 2, 357	980	1, 2, 316	1064	1, 2, 946
827	1, 2, 189	912	1, 2, 569	981	1, 2, 383	1066	1, 2, 258
829	1, 2, 560	914	1, 2, 4	984	1, 2, 349	1067	1, 2, 69
830	1, 2, 241	915	1, 2, 89	987	1, 3, 142	1068	1, 2, 223
832	1, 2, 39	917	1, 2, 22	989	1, 2, 105	1069	1, 2, 146
835	1, 2, 350	920	1, 3, 517	992	1, 2, 585	1070	1, 3, 94
836	1, 2, 606	922	1, 2, 24	995	1, 3, 242	1072	1, 2, 443
837	1, 2, 365	923	1, 2, 142	997	1, 2, 453	1073	1, 3, 235
840	1, 2, 341	925	1, 2, 308	1000	1, 3, 68	1074	1, 2, 395
843	1, 2, 322	928	1, 2, 33	1002	1, 2, 266	1075	1, 2, 92
848	1, 2, 225	929	1, 2, 36	1003	1, 2, 410	1076	1, 2, 22
851	1, 2, 442	931	1, 2, 72	1004	1, 2, 96	1077	1, 2, 521
853	1, 2, 461	933	1, 2, 527	1005	1, 2, 41	1080	1, 2, 151
854	1, 2, 79	934	1, 3, 800	1006	1, 2, 63	1083	1, 2, 538
856	1, 2, 842	936	1, 3, 27	1008	1, 2, 703	1088	1, 2, 531
859	1, 2, 594	939	1, 2, 142	1011	1, 2, 17	1091	1, 2, 82
863	1, 2, 90	940	1, 2, 204	1013	1, 2, 180	1093	1, 2, 173
864	1, 2, 607	941	1, 2, 573	1016	1, 2, 49	1096	1, 2, 351
867	1, 2, 380	944	1, 2, 487	1017	1, 2, 746	1099	1, 2, 464
869	1, 2, 82	946	1, 3, 83	1018	1, 2, 27	1101	1, 2, 14
872	1, 2, 691	947	1, 2, 400	1019	1, 2, 96	1104	1, 2, 259
874	1, 2, 110	949	1, 2, 417	1021	1, 2, 5	1107	1, 2, 176
875	1, 2, 66	950	1, 2, 859	1024	1, 2, 515	1109	1, 2, 501
877	1, 2, 140	952	1, 3, 311	1027	1, 2, 378	1112	1, 2, 1045
878	1, 2, 343	955	1, 2, 606	1032	1, 2, 901	1114	1, 2, 345
880	1, 3, 221	957	1, 2, 158	1035	1, 2, 76	1115	1, 2, 268
883	1, 2, 488	958	1, 2, 191	1037	1, 2, 981	1117	1, 2, 149
885	1, 2, 707	960	1, 2, 491	1038	1, 2, 41	1118	1, 2, 475
886	1, 2, 227	962	1, 2, 18	1040	1, 2, 429	1120	1, 3, 386
888	1, 2, 97	963	1, 2, 145	1043	1, 3, 869	1123	1, 2, 641
891	1, 2, 364	965	1, 2, 213	1045	1, 2, 378	1124	1, 2, 156
893	1, 2, 13	968	1, 2, 21	1046	1, 2, 39	1125	1, 2, 206
896	1, 2, 19	970	1, 2, 260	1048	1, 3, 172	1128	1, 3, 7
899	1, 3, 898	971	1, 2, 6	1051	1, 3, 354	1131	1, 2, 188

Table C-3.d: Irreducible pentanomials $x^m + x^{k1} + x^{k2} + x^{k3} + 1$ over F_2.

For each m, $1132 \le m \le 1456$, for which an irreducible trinomial of degree m does not exist, a triple of exponents $k1$, $k2$, $k3$ is given for which the pentanomial $x^m + x^{k1} + x^{k2} + x^{k3} + 1$ is irreducible over F_2.

m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)
1132	1, 2, 20	1219	1, 2, 225	1296	1, 2, 379	1376	1, 2, 1201
1133	1, 2, 667	1221	1, 2, 101	1299	1, 2, 172	1378	1, 2, 362
1136	1, 2, 177	1222	1, 2, 215	1301	1, 2, 297	1379	1, 2, 400
1139	1, 2, 45	1224	1, 2, 157	1303	1, 2, 306	1381	1, 2, 56
1141	1 2 134	1227	1, 2, 361	1304	1, 3, 574	1382	1, 3, 58
1143	1, 2, 7	1229	1, 2, 627	1307	1, 2, 157	1384	1, 2, 1131
1144	1, 2, 431	1232	1, 2, 225	1309	1, 2, 789	1387	1, 2, 33
1147	1, 2, 390	1235	1, 2, 642	1312	1, 2, 1265	1389	1, 2, 41
1149	1, 2, 221	1237	1, 2, 150	1315	1, 2, 270	1392	1, 2, 485
1150	1, 2, 63	1240	1, 2, 567	1316	1, 2, 12	1394	1, 2, 30
1152	1, 2, 971	1243	1, 2, 758	1317	1, 2, 254	1395	1, 2, 233
1155	1, 2, 94	1244	1, 2, 126	1318	1, 3, 94	1397	1, 2, 397
1157	1, 2, 105	1245	1, 2, 212	1320	1, 2, 835	1400	1, 2, 493
1160	1, 2, 889	1248	1, 2, 1201	1322	1, 2, 538	1403	1, 2, 717
1162	1, 2, 288	1250	1, 2, 37	1323	1, 2, 1198	1405	1, 2, 558
1163	1, 2, 33	1251	1, 2, 1004	1325	1, 2, 526	1406	1, 2, 13
1165	1, 2, 494	1253	1, 2, 141	1328	1, 2, 507	1408	1, 3, 45
1168	1, 2, 473	1254	1, 2, 697	1330	1, 2, 609	1411	1, 2, 200
1171	1, 2, 396	1256	1, 2, 171	1331	1, 2, 289	1413	1, 2, 101
1172	1, 2, 426	1258	1, 2, 503	1333	1, 2, 276	1416	1, 3, 231
1173	1, 2, 673	1259	1, 2, 192	1336	1, 2, 815	1418	1, 2, 283
1176	1, 2, 19	1261	1, 2, 14	1339	1, 2, 284	1419	1, 2, 592
1179	1, 2, 640	1262	1, 2, 793	1341	1, 2, 53	1421	1, 2, 30
1181	1, 2, 82	1264	1, 2, 285	1342	1, 2, 477	1424	1, 2, 507
1184	1, 2, 1177	1267	1, 2, 197	1344	1, 2, 469	1427	1, 2, 900
1187	1, 2, 438	1269	1, 2, 484	1346	1, 2, 57	1429	1, 2, 149
1189	1, 2, 102	1272	1, 2, 223	1347	1, 2, 61	1432	1, 2, 251
1192	1, 3, 831	1274	1, 2, 486	1349	1, 2, 40	1435	1, 2, 126
1194	1, 2, 317	1275	1, 2, 25	1352	1, 2, 583	1437	1, 2, 545
1195	1, 2, 293	1277	1, 2, 451	1355	1, 2, 117	1439	1, 2, 535
1197	1, 2, 269	1280	1, 2, 843	1357	1, 2, 495	1440	1, 3, 1023
1200	1, 3, 739	1283	1, 2, 70	1360	1, 2, 393	1443	1, 2, 413
1203	1, 2, 226	1285	1, 2, 564	1363	1, 2, 852	1445	1, 2, 214
1205	1, 2, 4	1288	1, 2, 215	1365	1, 2, 329	1448	1, 3, 212
1208	1, 2, 915	1290	1, 2, 422	1368	1, 2, 41	1450	1, 2, 155
1211	1, 2, 373	1291	1, 2, 245	1370	1, 2, 108	1451	1, 2, 193
1213	1, 2, 245	1292	1, 2, 78	1371	1, 2, 145	1453	1, 2, 348
1216	1, 2, 155	1293	1, 2, 26	1373	1, 2, 613	1456	1, 2, 1011

Table C-3.e: Irreducible pentanomials $x^m + x^{k1} + x^{k2} + x^{k3} + 1$ over F_2.

For each m, $1458 \le m \le 1761$, for which an irreducible trinomial of degree m does not exist, a triple of exponents $k1$, $k2$, $k3$ is given for which the pentanomial $x^m + x^{k1} + x^{k2} + x^{k3} + 1$ is irreducible over F_2.

m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)
1459	1, 2, 1032	1536	1, 2, 881	1619	1, 2, 289	1690	1, 2, 200
1461	1, 2, 446	1538	1, 2, 6	1621	1, 2, 1577	1691	1, 2, 556
1462	1, 2, 165	1539	1, 2, 80	1622	1, 2, 1341	1693	1, 2, 137
1464	1, 2, 275	1541	1, 2, 4	1624	1, 2, 1095	1696	1, 2, 737
1467	1, 2, 113	1544	1, 2, 99	1626	1, 2, 191	1699	1, 2, 405
1469	1, 2, 775	1546	1, 2, 810	1627	1, 2, 189	1701	1, 2, 568
1472	1, 2, 613	1547	1, 2, 493	1629	1, 2, 397	1702	1, 2, 245
1474	1, 2, 59	1549	1, 2, 426	1632	1, 2, 211	1704	1, 3, 55
1475	1, 2, 208	1552	1, 2, 83	1635	1, 2, 113	1706	1, 2, 574
1477	1, 2, 1325	1555	1, 2, 254	1637	1, 2, 234	1707	1, 2, 221
1480	1, 2, 285	1557	1, 2, 20	1640	1, 2, 715	1709	1, 2, 201
1483	1, 2, 1077	1560	1, 2, 11	1643	1, 2, 760	1712	1, 2, 445
1484	1, 2, 61	1563	1, 2, 41	1644	1, 2, 236	1714	1, 2, 191
1485	1, 2, 655	1565	1, 2, 18	1645	1, 2, 938	1715	1, 2, 612
1488	1, 2, 463	1568	1, 2, 133	1646	1, 2, 435	1717	1, 2, 881
1491	1, 2, 544	1571	1, 2, 21	1648	1, 2, 77	1718	1, 2, 535
1493	1, 2, 378	1573	1, 2, 461	1651	1, 2, 873	1720	1, 2, 525
1494	1, 2, 731	1574	1, 2, 331	1653	1, 2, 82	1723	1, 2, 137
1496	1, 2, 181	1576	1, 2, 147	1654	1, 3, 201	1725	1, 2, 623
1498	1, 2, 416	1579	1, 2, 374	1656	1, 2, 361	1727	1, 2, 22
1499	1, 2, 477	1581	1, 2, 160	1658	1, 2, 552	1728	1, 2, 545
1501	1, 2, 60	1584	1, 2, 895	1659	1, 2, 374	1730	1, 2, 316
1502	1, 2, 111	1587	1, 2, 433	1661	1, 2, 84	1731	1, 2, 925
1504	1, 2, 207	1589	1, 2, 882	1662	1, 3, 958	1732	1, 2, 75
1506	1, 2, 533	1592	1, 2, 223	1664	1, 2, 399	1733	1, 2, 285
1507	1, 2, 900	1594	1, 2, 971	1667	1, 2, 1020	1736	1, 2, 435
1509	1, 2, 209	1595	1, 2, 18	1669	1, 2, 425	1739	1, 2, 409
1512	1, 2, 1121	1597	1, 2, 42	1670	1, 2, 19	1741	1, 3, 226
1515	1, 2, 712	1598	1, 2, 385	1672	1, 2, 405	1744	1, 2, 35
1517	1, 2, 568	1600	1, 2, 57	1675	1, 2, 77	1747	1, 2, 93
1520	1, 2, 81	1603	1, 2, 917	1677	1, 2, 844	1749	1, 2, 236
1522	1, 2, 47	1605	1, 2, 46	1680	1, 2, 1549	1752	1, 2, 559
1523	1, 2, 240	1608	1, 2, 271	1682	1, 2, 354	1754	1, 2, 75
1525	1, 2, 102	1610	1, 2, 250	1683	1, 2, 1348	1755	1, 2, 316
1528	1, 2, 923	1611	1, 2, 58	1684	1, 2, 474	1757	1, 2, 21
1531	1, 2, 1125	1613	1, 2, 48	1685	1, 2, 493	1758	1, 2, 221
1532	1, 2, 466	1614	1, 2, 1489	1686	1, 2, 887	1760	1, 3, 1612
1533	1, 2, 763	1616	1, 2, 139	1688	1, 2, 921	1761	1, 2, 131

Table C-3.f: Irreducible pentanomials $x^m + x^{k1} + x^{k2} + x^{k3} + 1$ over F_2.

For each m, $1762 \le m \le 2000$, for which an irreducible trinomial of degree m does not exist, a triple of exponents $k1$, $k2$, $k3$ is given for which the pentanomial $x^m + x^{k1} + x^{k2} + x^{k3} + 1$ is irreducible over F_2.

m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)	m	(k_1, k_2, k_3)
1762	1, 2, 318	1826	1, 2, 298	1883	1, 2, 1062	1941	1, 2, 1133
1763	1, 2, 345	1827	1, 2, 154	1885	1, 2, 813	1942	1, 2, 147
1765	1, 2, 165	1829	1, 2, 162	1888	1, 2, 923	1944	1, 2, 617
1766	1, 2, 1029	1832	1, 3, 1078	1891	1, 2, 1766	1947	1, 2, 1162
1768	1, 2, 1403	1834	1, 2, 210	1892	1, 3, 497	1949	1, 2, 621
1771	1, 2, 297	1835	1, 2, 288	1893	1, 2, 461	1952	1, 3, 65
1773	1, 2, 50	1837	1, 2, 200	1894	1, 3, 215	1954	1, 2, 1226
1776	1, 2, 17	1840	1, 2 195	1896	1, 2, 451	1955	1, 2, 109
1779	1, 3, 1068	1842	1, 2, 799	1897	1, 2, 324	1957	1, 2, 17
1781	1, 2, 18	1843	1, 2, 872	1898	1, 2, 613	1960	1, 2, 939
1784	1, 2, 1489	1845	1, 2, 526	1899	1, 2, 485	1963	1, 2, 1137
1786	1, 2, 614	1848	1, 2, 871	1901	1, 2, 330	1965	1, 2, 364
1787	1, 2, 457	1850	1, 2, 79	1904	1, 2, 337	1968	1, 3, 922
1789	1, 2, 80	1851	1, 2, 250	1907	1, 2, 45	1970	1, 2, 388
1792	1, 2, 341	1852	1, 2, 339	1909	1, 2, 225	1971	1, 2, 100
1794	1, 2, 95	1853	1, 2, 705	1910	1, 3, 365	1972	1, 2, 474
1795	1, 2, 89	1856	1, 2, 585	1912	1, 2, 599	1973	1, 2, 438
1796	1, 2, 829	1858	1, 2, 1368	1914	1, 2, 544	1976	1, 3, 1160
1797	1, 2, 80	1859	1, 2, 120	1915	1, 2, 473	1978	1, 2, 158
1800	1, 2, 1013	1861	1, 2, 509	1916	1, 2, 502	1979	1, 2, 369
1803	1, 2, 248	1864	1, 2, 1379	1917	1, 2, 485	1981	1, 2, 96
1805	1, 2, 82	1867	1, 2, 117	1920	1, 2, 67	1982	1, 2, 1027
1808	1, 2, 25	1868	1, 2, 250	1922	1, 2, 36	1984	1, 2, 129
1811	1, 2, 117	1869	1, 2, 617	1923	1, 4, 40	1987	1, 2, 80
1812	1, 2, 758	1872	1, 3, 60	1925	1, 2, 576	1989	1, 2, 719
1813	1, 3, 884	1874	1, 2, 70	1928	1, 2, 763	1992	1, 2, 1241
1816	1, 2, 887	1875	1, 2, 412	1930	1, 2, 155	1995	1, 2, 37
1819	1, 2, 116	1876	1, 2, 122	1931	1, 2, 648	1997	1, 2, 835
1821	1, 2, 326	1877	1, 2, 796	1933	1, 2, 971	1998	1, 3, 1290
1822	1, 3, 31	1880	1, 2, 1647	1936	1, 2, 117	2000	1, 2, 981
1824	1, 2, 821	1882	1, 2, 128	1939	1, 2, 5		

C.4 Table of Fields F_{2^m} which have both an ONB and a TPB over F_2

Table C-4 – Values of m, $160 \leq m \leq 2000$, for which the field F_{2^m} has both an ONB and a TPB over F_2.

162	292	431	606	743	858	1034	1170	1306	1492	1703	1926
172	303	438	612	746	866	1041	1178	1310	1505	1734	1938
174	316	441	614	756	870	1049	1185	1329	1511	1740	1948
178	329	460	615	761	873	1055	1186	1338	1518	1745	1953
180	330	470	618	772	876	1060	1199	1353	1530	1746	1958
183	346	473	639	774	879	1065	1212	1359	1548	1769	1959
186	348	490	641	783	882	1090	1218	1372	1559	1778	1961
191	350	495	650	785	906	1103	1223	1380	1570	1785	1983
194	354	508	651	791	911	1106	1228	1398	1583	1790	1986
196	359	519	652	809	930	1108	1233	1401	1593	1791	1994
209	372	522	658	810	935	1110	1236	1409	1601	1806	1996
210	375	540	660	818	938	1116	1238	1425	1618	1818	
231	378	543	676	820	953	1119	1265	1426	1620	1838	
233	386	545	686	826	975	1121	1271	1430	1636	1854	
239	388	556	690	828	986	1122	1276	1452	1649	1860	
268	393	558	700	831	993	1134	1278	1454	1666	1863	
270	414	561	708	833	998	1146	1282	1463	1668	1866	
273	418	575	713	834	1014	1154	1289	1478	1673	1889	
278	420	585	719	846	1026	1166	1295	1481	1679	1900	
281	426	593	726	852	1031	1169	1300	1482	1692	1906	

Annex D
(informative)
Informative Number-Theoretic Algorithms

D.1 Finite Fields and Modular Arithmetic

D.1.1 Exponentiation in a Finite Field

If a is a positive integer and g is an element of the field F_q, then *exponentiation* is the process of computing g^a. Exponentiation can be performed efficiently by the *binary method* outlined below. The algorithm is used in Annexes D.1.2 and D.1.4.

Input: A positive integer a, a field F_q, and a field element g.

Output: g^a.

1. Set $e = a \bmod (q-1)$. If $e = 0$, then output 1.
2. Let $e = e_r e_{r-1} \ldots e_1 e_0$ be the binary representation of e, where the most significant bit e_r of e is 1.
3. Set $x = g$.
4. For i from r-1 down to 0 do
   - 4.1. Set $x = x^2$.
   - 4.2. If $e_i = 1$, then set $x = gx$.
5. Output x.

There are several variations of this method which can be used to speed up the computations. One such method which requires some precomputations is described in [12]. See also Knuth [22, pp. 441-466].

D.1.2 Inversion in a Finite Field

If $g \neq 0$ is an element of the field F_q, then the *inverse* g^{-1} is the field element c such that $gc = 1$. The inverse can be found efficiently by exponentiation since $c = g^{q-2}$. Note that if q is prime and g is an integer satisfying $1 \leq g \leq q - 1$, then g^{-1} is the integer c, $1 \leq c \leq q - 1$, such that $gc \equiv 1 \pmod q$. The algorithm is used in Sections 5.3.3 and 5.4.2.

Input: A field F_q, and a non-zero element $g \in F_q$.

Output: The inverse g^{-1}.

1. Compute $c = g^{q-2}$ (see Annex D.1.1).
2. Output c.

An even more efficient method is the extended Euclidean Algorithm [22, p. 325].

D.1.3 Generating Lucas Sequences

Let P and Q be nonzero integers. The *Lucas sequences* U_k and V_k for P, Q are defined by:

$U_0 = 0$, $U_1 = 1$, and $U_k = PU_{k-1} - QU_{k-2}$ for $k \geq 2$.

$V_0 = 2$, $V_1 = P$, and $V_k = PV_{k-1} - QV_{k-2}$ for $k \geq 2$.

This recursion is adequate for computing U_k and V_k for small values of k. The following algorithm can be used to efficiently compute U_k and V_k modulo an odd prime p for large values of k. The algorithm is used in Annex D.1.4.

Input: An odd prime p, integers P and Q, and a positive integer k.

Output: $U_k \bmod p$ and $V_k \bmod p$.

1. Set $\Delta = P^2 - 4Q$.
2. Let $k = k_r k_{r-1} \ldots k_1 k_0$ be the binary representation of k, where the leftmost bit k_r of k is 1.
3. Set $U = 1$, $V = P$.
4. For i from r - 1 down to 0 do
   - 4.1. Set $(U,V) = \left(UV \bmod p, \left(\frac{V^2 + \Delta U^2}{2}\right) \bmod p\right)$.
   - 4.2. If $k_i = 1$ then set $(U,V) = \left(\left(\frac{PU + V}{2}\right) \bmod p, \left(\frac{PV + \Delta U}{2}\right) \bmod p\right)$.
5. Output U and V.

D.1.4 Finding Square Roots Modulo a Prime

Let p be an odd prime, and let g be an integer with $0 \le g < p$. A square root (mod p) of g is an integer y with $0 \le y < p$ and:

$$y^2 \equiv g \pmod{p}.$$

If $g = 0$, then there is one square root (mod p), namely $y = 0$. If $g \neq 0$, then g has either 0 or 2 square roots (mod p). If y is one square root, then the other is p-y.

The following algorithm determines whether g has square roots (mod p) and, if so, computes one. The algorithm is used in Section 4.2.1 and Annex D.3.1.

Input: An odd prime p, and an integer g with $0 < g < p$.

Output: A square root (mod p) of g if one exists; otherwise, the message "no square roots exist."

Algorithm 1: *for* $p \equiv 3 \pmod 4$, *that is* $p = 4u + 3$ *for some positive integer* u.

1. Compute $y = g^{u+1} \bmod p$ via Annex D.1.1.
2. Compute $z = y^2 \bmod p$.
3. If $z = g$, then output y. Otherwise output the message "no square roots exist."

Algorithm 2: *for* $p \equiv 5 \pmod 8$, *that is* $p = 8u + 5$ *for some positive integer* u.

1. Compute $\gamma = (2g)^u \bmod p$ via Annex D.1.1.
2. Compute $i = 2g\gamma^2 \bmod p$.
3. Compute $y = g\gamma(i - 1) \bmod p$.
4. Compute $z = y^2 \bmod p$.
5. If $z = g$, then output y. Otherwise output the message "no square roots exist."

Algorithm 3: *for* $p \equiv 1 \pmod 4$, *that is* $p = 4u + 1$ *for some positive integer* u.

1. Set $Q = g$.
2. Generate random P with $0 \le P < p$.
3. Using Annex D.1.3, compute the Lucas sequence elements:
   $U = U_{2u+1} \bmod p$, $V = V_{2u+1} \bmod p$.
4. If $V^2 \equiv 4Q \pmod p$ then output $y = V/2 \bmod p$ and stop.
5. If $U \not\equiv \pm 1 \pmod p$ then output the message "no square roots exist" and stop.
6. Go to Step 2.

D.1.5 Trace and Half-Trace Functions

If α is an element of F_{2^m}, the *trace* of α is:

$$Tr(\alpha) = \alpha + \alpha^2 + \alpha^{2^2} + ... + \alpha^{2^{m-1}}.$$

The value of $Tr(\alpha)$ is 0 for half the elements of F_{2^m}, and 1 for the other half. The trace can be computed as follows. The methods are used in Annex D.1.6.

Normal basis representation used for elements of F_{2^m}:

If α has representation $(\alpha_0 \alpha_1 ... \alpha_{m-1})$, then:

$$Tr(\alpha) = \alpha_0 \oplus \alpha_1 \oplus ... \oplus \alpha_{m-1}.$$

Polynomial basis representation used for elements of F_{2^m}:

1. Set $T = \alpha$.
2. For i from 1 to $m - 1$ do
   2.1. $T = T^2 + \alpha$.
3. Output T.

If m is odd, the *half-trace* of α is:

$$\alpha + \alpha^{2^2} + \alpha^{2^4} + ... + \alpha^{2^{m-1}}.$$

If F_{2^m} is represented by a polynomial basis, the half-trace can be computed efficiently as follows. The method is used in Annex D.1.6.

1. Set $T = \alpha$.
2. For i from 1 to $(m - 1)/2$ do
   2.1. $T = T^2$.
   2.2. $T = T^2 + \alpha$.
3. Output T.

D.1.6 Solving Quadratic Equations over F_{2^m}

If β is an element of F_{2^m}, then the equation:

$$z^2 + z = \beta$$

has $2\text{-}2T$ solutions over F_{2^m}, where $T = Tr(\beta)$. Thus, there are either 0 or 2 solutions. If $\beta = 0$, then the solutions are 0 and 1. If $\beta \neq 0$ and z is a solution, then the other solution is $z+1$.

The following algorithms determine whether a solution z exists for a given β, and if so, computes one. The algorithms are used in point compression (see Section 4.2.2) and in Annex D.3.1.

Input: A field F_{2^m} along with a basis for representing its elements; and an element $\beta \neq 0$.

Output: An element z for which $z^2 + z = \beta$ if any exist; otherwise the message "no solutions exist".

Algorithm 1: *for normal basis representation*.

1. Let $(\beta_0\, \beta_1 \ldots \beta_{m-1})$ be the representation of β.
2. Set $z_0 = 0$.
3. For i from 1 to m-1 do
   3.1. Set $z_i = z_{i-1} \oplus \beta_i$.
4. Set $z = (z_0\, z_1 \ldots z_{m-1})$.
5. Compute $\gamma = z^2 + z$.
6. If $\gamma = \beta$, then output z. Otherwise, output the message "no solutions exist".

Algorithm 2: *for polynomial basis representation, with m odd*.

1. Compute z = half-trace of β via Annex D.1.5.
2. Compute $\gamma = z^2 + z$.
3. If $\gamma = \beta$, then output z. Otherwise, output the message "no solutions exist".

Algorithm 3: *works in any polynomial basis*.

1. Choose a random $\tau \in F_{2^m}$.
2. Set $z = 0$ and $w = \beta$.
3. For i from 1 to $m - 1$ do
   3.1. Set $z = z^2 + w^2\tau$.
   3.2. Set $w = w^2 + \beta$.
4. If $w \neq 0$, then output the message "no solutions exist" and stop.
5. Compute $\gamma = z^2 + z$.
6. If $\gamma = 0$, then go to Step 1.
7. Output z.

D.1.7 Checking the Order of an Integer Modulo a Prime

Let p be a prime and let g satisfy $1 < g < p$. The *order* of g modulo p is the smallest positive integer k such that $g^k \equiv 1 \pmod p$. The following algorithm tests whether or not g has order k modulo p. The algorithm is used in Annex D.1.8.

Input: A prime p, a positive integer k, and an integer g with $1 < g < p$.

Output: "true" if g has order k modulo p, and "false" otherwise.

1. Determine the prime divisors of k.
2. If $g^k \not\equiv 1 \pmod p$, then output "false" and stop.
3. For each prime l dividing k do

3.1. If $g^{k/l} \equiv 1 \pmod p$, then output "false" and stop.
4. Output "true".

D.1.8 Computing the Order of a Given Integer Modulo a Prime

Let p be a prime and let g satisfy $1 < g < p$. The following algorithm determines the order of g modulo p. The algorithm is efficient only for small p. It is used in Annex D.1.9.

Input: A prime p and an integer g with $1 < g < p$.
Output: The order k of g modulo p.
1. Set $b = g$ and $j = 0$.
2. Set $b = gb \bmod p$ and $j = j + 1$.
3. If $b > 1$ then go to Step 2.
4. Output j.

D.1.9 Constructing an Integer of a Given Order Modulo a Prime

Let p be a prime and let T divide p–1. The following algorithm generates an element of F_p of order T. The algorithm is efficient only for small p.

Input: A prime p and an integer T dividing p–1.
Output: An integer u having order T modulo p.
1. Generate a random integer g between 1 and p.
2. Compute via Annex D.1.8 the order k of g modulo p.
3. If T does not divide k then go to Step 1.
4. Output $u = g^{k/T} \bmod p$.

D.2 Polynomials over a Finite Field

D.2.1 GCD's over a Finite Field

If $f(t)$ and $g(t) \neq 0$ are two polynomials with coefficients in the field F_q, then there is a unique monic polynomial $d(t)$ with coefficient also in F_q of largest degree which divides both $f(t)$ and $g(t)$. The polynomial $d(t)$ is called the *greatest common divisor* or *gcd* of $f(t)$ and $g(t)$. The following algorithm (the Euclidean algorithm) computes the gcd of two polynomials. The algorithm is used in Annex D.2.2.

Input: A finite field F_q and two polynomials $f(t)$, $g(t) \neq 0$ over F_q.
Output: $d(t) = \gcd(f(t), g(t))$.
1. Set $a(t) = f(t)$, $b(t) = g(t)$.
2. While $b(t) \neq 0$
   2.1. Set $c(t)$ = the remainder when $a(t)$ is divided by $b(t)$.
   2.2. Set $a(t) = b(t)$.
   2.3. Set $b(t) = c(t)$.
3. Let α be the leading coefficient of $a(t)$ and output $\alpha^{-1}a(t)$.

D.2.2 Finding a Root in F_{2^m} of an Irreducible Binary Polynomial

If $f(t)$ is an irreducible binary polynomial of degree m, then $f(t)$ has m distinct roots in the field F_{2^m}. A random root can be found efficiently using the following algorithm. The algorithm is used in Annex D.2.3.

Input: An irreducible binary polynomial $f(t)$ of degree m, and a field F_{2^m}.
Output: A random root of $f(t)$ in F_{2^m}.
1. Set $g(t) = f(t)$.
2. While $\deg(g) > 1$
   2.1. Choose random $u \in F_{2^m}$.
   2.2. Set $c(t) = ut$.
   2.3. For i from 1 to m-1 do
      2.3.1. $c(t) = (c(t)^2 + ut) \bmod g(t)$.
   2.4. Set $h(t) = \gcd(c(t), g(t))$.

2.5. If $h(t)$ is constant or $\deg(g) = \deg(h)$, then go to step 2.1.
2.6. If $2\deg(h) > \deg(g)$, then set $g(t) = g(t)/h(t)$; else $g(t) = h(t)$.

3. Output $g(0)$.

D.2.3 Change of Basis

Given a field F_{2^m} and two (polynomial or normal) bases B_1 and B_2 for the field over F_2, the following algorithm allows conversion between bases B_1 and B_2.

1. Let $f(t)$ be the field polynomial of B_2. That is,
   1.1. If B_2 is a *polynomial basis*, let $f(t)$ be the (irreducible) reduction polynomial of degree m over F_2.
   1.2. If B_2 is a *Type I optimal normal basis*, let:
   $$f(t) = t^m + t^{m-1} + t^{m-2} + \ldots + t + 1.$$
   1.3. If B_2 is a *Type II optimal normal basis*, let:
   $$f(t) = \sum_{\substack{0 \le j \le m \\ m-j \prec m+j}} t^j$$
   where the notation $a \prec b$ means that in the binary representations
   $$a = \sum u_i 2^i, \; b = \sum w_i 2^i,$$
   we have $u_i \le w_i$ for all i.
   1.4. If B_2 is a Gaussian normal basis of Type $T \ge 3$, then:
      1.4.1. Set $p = T m + 1$.
      1.4.2. Generate via Annex D.1.9 an integer u having order T modulo p.
      1.4.3. For k from 1 to m do
      $$e_k = \sum_{j=0}^{T-1} \exp\left(\frac{2^k u^j \pi i}{p}\right)$$
      1.4.4. Compute the polynomial
      $$g(t) = \prod_{k=1}^{m} (t - e_k)$$
      (The polynomial $g(t)$ has integer coefficients.)
      1.4.5. Output $f(t) = g(t) \bmod 2$.
      Note: The complex numbers e_k must be computed with sufficient accuracy to identify each coefficient of the polynomial $g(t)$. Since each such coefficient is an integer, this means that the error incurred in calculating each coefficient should be less than 1/2.

2. Let γ be a root of $f(t)$ computed with respect to B_1. (γ can be computed using the technique defined in Annex D.2.2.)

3. Let Γ be the matrix:
$$\Gamma = \begin{pmatrix} \gamma_{0,0} & \gamma_{0,1} & \cdots & \gamma_{0,m-1} \\ \gamma_{1,0} & \gamma_{1,1} & \cdots & \gamma_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{m-1,0} & \gamma_{m-1,1} & \cdots & \gamma_{m-1,m-1} \end{pmatrix}$$
where the entries $\gamma_{i,j}$ are defined as follows:
   3.1. If B_2 is a polynomial basis, then:

$$1 = (\gamma_{0,0}\ \gamma_{0,1} \ldots \gamma_{0,m-1})$$
$$\gamma = (\gamma_{1,0}\ \gamma_{1,1} \ldots \gamma_{1,m-1})$$
$$\gamma^2 = (\gamma_{2,0}\ \gamma_{2,1} \ldots \gamma_{2,m-1})$$
$$\vdots$$
$$\gamma^{m-1} = (\gamma_{m-1,0}\ \gamma_{m-1,1} \ldots \gamma_{m-1,m-1})$$

with respect to B_1. (The entries $\gamma_{i,j}$ are computed by repeated multiplication by γ.)

3.2. If B_2 is a Gaussian normal basis (of any type $T \geq 1$), then:

$$\gamma = (\gamma_{0,0}\ \gamma_{0,1} \ldots \gamma_{0,m-1})$$
$$\gamma^2 = (\gamma_{1,0}\ \gamma_{1,1} \ldots \gamma_{1,m-1})$$
$$\gamma^4 = (\gamma_{2,0}\ \gamma_{2,1} \ldots \gamma_{2,m-1})$$
$$\vdots$$
$$\gamma^{2^{m-1}} = (\gamma_{m-1,0}\ \gamma_{m-1,1} \ldots \gamma_{m-1,m-1})$$

with respect to B_1. (The entries $\gamma_{i,j}$ are computed by repeated squaring of γ.)

4. If an element has representation $(\beta_0\beta_1 \ldots \beta_{m-1})$ with respect to B_2, then its representation with respect to B_1 is

$$(\alpha_0\alpha_1 \ldots \alpha_{m-1}) = (\beta_0\beta_1 \ldots \beta_{m-1})\ \Gamma.$$

If an element has representation $(\alpha_0\alpha_1 \ldots \alpha_{m-1})$ with respect to B_1, then its representation with respect to B_2 is

$$(\beta_0\beta_1 \ldots \beta_{m-1}) = (\alpha_0\alpha_1 \ldots \alpha_{m-1})\ \Gamma^{-1},$$

where Γ^{-1} denotes the mod 2 inverse of Γ.

Example:
Suppose that B_1 is the polynomial basis (mod $t^4 + t + 1$), and B_2 is the Type I optimal normal basis for F_{2^4}. Then $f(t) = t^4 + t^3 + t^2 + t + 1$, and a root is given by $\gamma = (1100)$ with respect to B_1. Then:

$$\gamma = (1100)$$
$$\gamma^2 = (1111)$$
$$\gamma^4 = (1010)$$
$$\gamma^8 = (1000)$$

so that:

$$\Gamma = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

and:

$$\Gamma^{-1} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

If λ = (1001) with respect to B_2, then its representation with respect to B_1 is:

(0100) = (1001) Γ.

If λ = (1011) with respect to B_1, then its representation with respect to B_2 is:

(1101) = (1011) Γ^{-1}.

D.2.4 Checking Binary Polynomials for Irreducibility

If $f(x)$ is a binary polynomial, then $f(x)$ can be tested efficiently for irreducibility using the following algorithm. The algorithm is used in Section 5.1.2.2.

Input: A binary polynomial $f(x)$.

Output: The message "true" if $f(x)$ is irreducible over F_2; the message "false" otherwise.

1. Set d = degree of $f(x)$.
2. Set $u(x) = x$.
3. For i from 1 to $\lfloor d/2 \rfloor$ do
    3.1. Set $u(x) = u(x)^2 \bmod f(x)$.
    3.2. Set $g(x) = \gcd(u(x) + x, f(x))$.
    3.3. If $g(x) \neq 1$, then output "false" and stop.
4. Output "true".

D.3 Elliptic Curve Algorithms

D.3.1 Finding a Point on an Elliptic Curve

The following algorithms provide an efficient method for finding an arbitrary point (other than $\mathcal{O}$) on a given elliptic curve over a finite field. These algorithms are used in Annexes A.3.1 and A.3.2.

Case I: Curves over F_p

Input: A prime p and the parameters a and b of an elliptic curve E over F_p.

Output: An arbitrary point (other than $\mathcal{O}$) on E.

1. Choose a random integer x with $0 \leq x < p$.
2. Set $\alpha = x^3 + ax + b \bmod p$.
3. If $\alpha = 0$ then output $(x, 0)$ and stop.
4. Apply the appropriate algorithm from Annex D.1.4 to look for a square root (mod p) of α.
5. If the output of Step 4 is "no square roots exist," then go to Step 1. Otherwise the output of Step 4 is an integer y with $0 < y < p$ such that $y^2 \equiv \alpha \pmod p$.
6. Output (x, y).

Case II: Curves over F_{2^m}.

Input: A field F_{2^m} and the parameters a and b of an elliptic curve E over F_{2^m}.

Output: A randomly generated point (other than $\mathcal{O}$) on E.

1. Choose a random element x in F_{2^m}.
2. If $x = 0$, then output $(0, b^{2^{m-1}})$ and stop.
3. Set $\alpha = x^3 + ax^2 + b$.
4. If $\alpha = 0$, then output $(x, 0)$ and stop.
5. Set $\beta = x^{-2}\alpha$.
6. Apply the appropriate algorithm from Annex D.1.6 to look for an element z for which $z^2 + z = \beta$.
7. If the output of Step 6 is "no solutions exist," then go to Step 1. Otherwise the output of Step 6 is a solution z.
8. Set $y = xz$.
9. Output (x, y).

D.3.2 Scalar Multiplication (Computing a Multiple of an Elliptic Curve Point)

If k is a positive integer and P is an elliptic curve point, then kP is the point obtained by adding together k copies of P. This computation can be performed efficiently by the *addition-subtraction method* outlined below. These algorithms are used, for example, in Sections 5.1.1, 5.1.2, 5.3, and 5.4.

Input: A positive integer k and an elliptic curve point P.

Output: The elliptic curve point kP.

1. Set $e = k \bmod n$, where n is the order of P. (If n is unknown, then set $e = k$ instead.)
2. Let $h_r\, h_{r-1} \ldots h_1\, h_0$ be the binary representation of $3e$, where the leftmost bit h_r is 1.
3. Let $e_r\, e_{r-1} \ldots e_1\, e_0$ be the binary representation of e.
4. Set $R = P$.
5. For i from r-1 down to 1 do
    - 5.1. Set $R = 2R$.
    - 5.2. If $h_i = 1$ and $e_i = 0$, then set $R = R + P$.
    - 5.3. If $h_i = 0$ and $e_i = 1$, then set $R = R - P$.
6. Output R.

Note: To subtract the point (x, y), just add the point $(x, -y)$ (for the field F_p) or $(x, x + y)$ (for the field F_{2^m}).

There are several variations of this method which can be used to speed up the computations. One such method which requires some precomputations is described in [12]. See also Knuth [22, pages 441-466].

Annex E
(informative)
Complex Multiplication (CM) Elliptic Curve Generation Method

This Annex describes a method for generating an elliptic curve with known order. The method may be used for selecting an appropriate elliptic curve and point (see Annex A.3.2).

E.1 Miscellaneous Number-Theoretic Algorithms

This section collects together some number-theoretic algorithms that are used in Annexes E.2 and E.3. These algorithms are not used in any other sections of this Standard.

E.1.1 Evaluating Jacobi Symbols

The Legendre symbol:

If $p > 2$ is prime, and a is any integer, then the *Legendre symbol* $\left(\frac{a}{p}\right)$ is defined as follows. If p divides a, then $\left(\frac{a}{p}\right) = 0$. If p does not divide a, then $\left(\frac{a}{p}\right)$ equals 1 if a is a square modulo p and –1 otherwise. (Despite the similarity in notation, a Legendre symbol should not be confused with a rational fraction; the distinction must be made from the context.)

The Jacobi symbol:

The *Jacobi symbol* $\left(\frac{a}{n}\right)$ is a generalization of the Legendre symbol. If $n > 1$ is odd with prime factorization:

$$n = \prod_{i=1}^{t} p_i^{e_i},$$

and a is any integer, then the Jacobi symbol is defined to be

$$\left(\frac{a}{n}\right) = \prod_{i=1}^{t} \left(\frac{a}{p_i}\right)^{e_i},$$

where the symbols $\left(\frac{a}{p_i}\right)$ are Legendre symbols. (Despite the similarity in notation, a Jacobi symbol should not be confused with a rational fraction; the distinction must be made from the context.)

The values of the Jacobi symbol are ± 1 if a and n are relatively prime and 0 otherwise. The values 1 and –1 are achieved equally often (unless n is a square, in which case the value –1 does not occur at all).

The following algorithm efficiently computes the Jacobi symbol.

Input: An integer a and an odd integer $n > 1$.

Output: The Jacobi symbol $\left(\frac{a}{n}\right)$

1. If $\gcd(a, n) > 1$ then output 0 and stop.
2. Set $x = a$, $y = n$, $J = 1$.
3. Set $x = (x \bmod y)$.
4. If $x > y/2$ then
   - 4.1 Set $x = y - x$.
   - 4.2. If $y \equiv 3 \pmod 4$ then set $J = -J$.
5. While 4 divides x
   - 5.1 Set $x = x/4$.

6. If 2 divides x then
   6.1 Set $x = x/2$.
   6.2 If $y \equiv \pm 3 \pmod 8$ then set $J = -J$.
7. If $x = 1$ then output J and stop.
8. If $x \equiv 3 \pmod 4$ and $y \equiv 3 \pmod 4$ then set $J = -J$.
9. Switch x and y.
10. Go to Step 3.

If n is equal to a prime p, the Jacobi symbol can also be found efficiently using exponentiation via:

$$\left(\frac{a}{p}\right) = a^{(p-1)/2} \bmod p.$$

E.1.2 Finding Square Roots Modulo a Power of 2

If $r > 2$ and $a < 2^r$ is a positive integer congruent to 1 modulo 8, then there is a unique positive integer b less than 2^{r-2} such that $b^2 \equiv a \pmod{2^r}$. The number b can be computed efficiently using the following algorithm. The binary representations of the integers a, b, h are denoted as

$a = a_{r-1}...a_1a_0$,
$b = b_{r-1}...b_1b_0$,
$h = h_{r-1}...h_1h_0$.

Input: An integer $r > 2$, and a positive integer $a \equiv 1 \pmod 8$ less than 2^r.

Output: The positive integer b less than 2^{r-2} such that $b^2 \equiv a \pmod{2^r}$.

1. Set $h = 1$.
2. Set $b = 1$.
3. For j from 2 to $r - 2$ do
   If $h_{j+1} \neq a_{j+1}$ then
      Set $b_j = 1$.
      If $j < r/2$
         then $h = (h + 2^{j+1}b - 2^{2j}) \bmod 2^r$.
         else $h = (h + 2^{j+1}b) \bmod 2^r$.
4. If $b_{r-2} = 1$ then set $b = 2^{r-1} - b$.
5. Output b.

E.1.3 Exponentiation Modulo a Polynomial

If k is a positive integer and $f(t)$ and $m(t)$ are polynomials with coefficients in the field F_q, then $f(t)^k \bmod m(t)$ can be computed efficiently by the *binary method* outlined below.

Input: A positive integer k, a field F_q, and polynomials $f(t)$ and $m(t)$ with coefficients in F_q.

Output: The polynomial $f(t)^k \bmod m(t)$.

1. Let $k = k_r\, k_{r-1} \dots k_1\, k_0$ be the binary representation of k, where the most significant bit k_r of k is 1.
2. Set $u(t) = f(t) \bmod m(t)$.
3. For i from r–1 downto 0 do
   3.1 Set $u(t) = u(t)^2 \bmod m(t)$.
   3.2 If $k_i = 1$ then set $u(t) = u(t)\, f(t) \bmod m(t)$.
4. Output $u(t)$.

E.1.4 Factoring Polynomials over F_p (Special Case)

Let $f(t)$ be a polynomial with coefficients in the field F_p, and suppose that $f(t)$ factors into distinct irreducible polynomials of degree d. (This is the special case needed in Annex E.3.) The following algorithm finds a random degree-d factor of $f(t)$ efficiently.

Input: A prime $p > 2$, a positive integer d, and a polynomial $f(t)$ which factors modulo p into distinct irreducible polynomials of degree d.

Output: A random degree-d factor of $f(t)$.

1. Set $g(t) = f(t)$.
2. While $\deg(g) > d$

2.1 Choose $u(t)$ = a random monic polynomial of degree $2d - 1$.
2.2 Compute (via Annex E.1.3.)

$$c(t) = u(t)^{(p^d - 1)/2} \bmod g(t).$$

2.3 Compute $h(t) = \gcd(c(t) - 1, g(t))$ via Annex D.2.1.
2.4 If $h(t)$ is constant or $\deg(g) = \deg(h)$ then go to Step 2.1.
2.5 If $2 \deg(h) > \deg(g)$ then set $g(t) = g(t) / h(t)$; else $g(t) = h(t)$.

3. Output $g(t)$.

E.1.5 Factoring Polynomials over F_2 (Special Case)

Let $f(t)$ be a polynomial with coefficients in the field F_2, and suppose that $f(t)$ factors into distinct irreducible polynomials of degree d. (This is the special case needed in Annex E.3.) The following algorithm finds a random degree-d factor of $f(t)$ efficiently.

Input: A positive integer d, and a polynomial $f(t)$ which factors modulo 2 into distinct irreducible polynomials of degree d.

Output: A random degree-d factor of $f(t)$.

1. Set $g(t) = f(t)$.
2. While $\deg(g) > d$

2.1 Choose $u(t)$ = a random monic polynomial of degree $2d - 1$.
2.2 Set $c(t) = u(t)$.
2.3 For i from 1 to $d - 1$ do
2.3.1 $c(t) = c(t)^2 + u(t) \bmod g(t)$.
2.4 Compute $h(t) = \gcd(c(t), g(t))$ via Annex D.2.1.
2.5 If $h(t)$ is constant or $\deg(g) = \deg(h)$ then go to Step 2.1.
2.6 If $2 \deg(h) > \deg(g)$ then set $g(t) = g(t) / h(t)$; else $g(t) = h(t)$.

3. Output $g(t)$.

E.2 Class Group Calculations

The following computations are necessary for the complex multiplication technique described in Annex E.3.

E.2.1 Overview

A *reduced symmetric matrix* is one of the form

$$S = \begin{pmatrix} A & B \\ B & C \end{pmatrix}$$

where the integers A, B, C satisfy the following conditions:

1. $\gcd(A, 2B, C) = 1$,
2. $|2B| \le A \le C$,
3. If either $A = |2B|$ or $A = C$, then $B \ge 0$.

We will abbreviate S as $[A, B, C]$ when typographically convenient.

The determinant $D = AC - B^2$ of S will be assumed throughout this section to be positive and *squarefree* (i.e., containing no square factors).

Given D, the *class group* $H(D)$ is the set of all reduced symmetric matrices of determinant D. The *class number* $h(D)$ is the number of matrices in $H(D)$.

The class group is used to construct the *reduced class polynomial*. This is a polynomial $w_D(t)$ with integer coefficients of degree $h(D)$. The reduced class polynomial is used in Annex E.3 to construct elliptic curves with known orders.

E.2.2 Class Group and Class Number

The following algorithm produces a list of the reduced symmetric matrices of a given determinant D.

Input: A squarefree determinant $D > 0$.

Output: The class group $H(D)$.

1. Let s be the largest integer less than $\sqrt{D/3}$.

2. For B from 0 to s do
   - 2.1. List the positive divisors $A_1, \ldots, A_r$ of $D + B^2$ that satisfy $2B \le A \le \sqrt{D + B^2}$.
   - 2.2. For i from 1 to r do
     - 2.2.1. Set $C = (D + B^2) / A_i$.
     - 2.2.2. If $\gcd(A_i, 2B, C) = 1$ then
       list $[A_i, B, C]$.
       if $0 < 2B < A_i < C$ then list $[A_i, -B, C]$.
3. Output list.

Example:

$D = 71$. We need to check $0 \le B < 5$.

— For $B = 0$, we have $A = 1$, leading to [1,0,71].

— For $B = 1$, we have $A = 2,3,4,6,8$, leading to $[3, \pm 1, 24]$ and $[8, \pm 1, 9]$.

— For $B = 2$, we have $A = 5$, leading to $[5, \pm 2, 15]$.

— For $B = 3$, we have $A = 8$, but no reduced matrices.

— For $B = 4$, we have no divisors A in the right range.

Thus the class group is:

$H(71) = \{[1,0,71], [3, \pm 1,24], [8, \pm 1,9], [5, \pm 2, 15]\}$

and the class number is:

$h(71) = 7$.

E.2.3 Reduced Class Polynomials

Let:

$$F(z) = 1 + \sum_{j=1}^{\infty} (-1)^j \left(z^{(3j^2 - j)/2} + z^{(3j^2 + j)/2} \right)$$
$$= 1 - z - z^2 + z^5 + z^7 - z^{12} - z^{15} + \ldots$$

and:

$$\theta = \exp\left(\frac{-\sqrt{D} + Bi}{A} \pi \right)$$

Let:

$\mathfrak{f}_0(A, B, C) = \theta^{-1/24} F(-\theta) / F(\theta^2)$,

$\mathfrak{f}_1(A, B, C) = \theta^{-1/24} F(\theta) / F(\theta^2)$,

$\mathfrak{f}_2(A, B, C) = \sqrt{2}\, \theta^{1/12} F(\theta^4) / F(\theta^2)$.

Note: Since $|\theta| < e^{-\pi\sqrt{3}/2} \approx 0.0658287$, the series $F(z)$ used in computing the numbers $\mathfrak{f}_J(A, B, C)$ converges as quickly as a power series in $e^{-\pi\sqrt{3}/2}$.

If $[A, B, C]$ is a matrix of determinant D, then its *class invariant* is

$\mathfrak{C}(A, B, C) = (N \lambda^{BL} 2^{-I/6} (\mathfrak{f}_J (A, B, C))^K)^G$,

where:

$G = \gcd(D,3)$,

$$I = \begin{cases} 3 & \text{if } D \equiv 1,2,6,7 \bmod 8, \\ 0 & \text{if } D \equiv 3 \bmod 8 \text{ and } D \not\equiv 0 \bmod 3, \\ 2 & \text{if } D \equiv 3 \bmod 8 \text{ and } D \equiv 0 \bmod 3, \\ 6 & \text{if } D \equiv 5 \bmod 8, \end{cases}$$

$$J = \begin{cases} 0 & \text{for } AC \text{ odd}, \\ 1 & \text{for } C \text{ even}, \\ 2 & \text{for } A \text{ even}, \end{cases}$$

$$K = \begin{cases} 2 & \text{if } D \equiv 1,2,6 \bmod 8, \\ 1 & \text{if } D \equiv 3,7 \bmod 8, \\ 4 & \text{if } D \equiv 5 \bmod 8, \end{cases}$$

$$L = \begin{cases} A - C + A^2C & \text{if } AC \text{ odd or } D \equiv 5 \bmod 8 \text{ and } C \text{ even}, \\ A + 2C - AC^2 & \text{if } D \equiv 1,2,3,6,7 \bmod 8 \text{ and } C \text{ even}, \\ A - C + 5A^2C & \text{if } D \equiv 3 \bmod 8 \text{ and } A \text{ even}, \\ A - C - AC^2 & \text{if } D \equiv 1,2,5,6,7 \bmod 8 \text{ and } A \text{ even}, \end{cases}$$

$$M = \begin{cases} (-1)^{(A^2-1)/8} & \text{if } A \text{ odd}, \\ (-1)^{(C^2-1)/8} & \text{if } A \text{ even}, \end{cases}$$

$$N = \begin{cases} 1 & \text{if } D \equiv 5 \bmod 8 \\ & \text{or } D \equiv 3 \bmod 8 \text{ and } AC \text{ odd} \\ & \text{or } D \equiv 7 \bmod 8 \text{ and } AC \text{ even}, \\ M & \text{if } D \equiv 1,2,6 \bmod 8 \\ & \text{or } D \equiv 7 \bmod 8 \text{ and } AC \text{ odd} \\ -M & \text{if } D \equiv 3 \bmod 8 \text{ and } AC \text{ even}, \end{cases}$$

$$\lambda = e^{-\pi i K/24}.$$

If $[A_1, B_1, C_1]$, ..., $[A_h, B_h, C_h]$ are the reduced symmetric matrices of (positive squarefree) determinant D, then the *reduced class polynomial* for D is:

$$w_D(t) = \prod_{j=1}^{h} \left(t - \mathfrak{C}(A_j, B_j, C_j)\right).$$

The reduced class polynomial has integer coefficients.

Note: The above computations must be performed with sufficient accuracy to identify each coefficient of the polynomial $w_D(t)$. Since each such coefficient is an integer, this means that the error incurred in calculating each coefficient should be less than 1/2.

Example:

$$w_{71}(t) \quad = \quad \left(t - \frac{1}{\sqrt{2}} f_0(1,0,71)\right)$$

$$\left(t - \frac{e^{-i\pi/8}}{\sqrt{2}}\,\mathrm{f}_1(3,1,24)\right)\left(t - \frac{e^{i\pi/8}}{\sqrt{2}}\,\mathrm{f}_1(3,-1,24)\right)$$
$$\left(t - \frac{e^{-23i\pi/24}}{\sqrt{2}}\,\mathrm{f}_2(8,1,9)\right)\left(t - \frac{e^{23i\pi/24}}{\sqrt{2}}\,\mathrm{f}_2(8,-1,9)\right)$$
$$\left(t + \frac{e^{-5i\pi/12}}{\sqrt{2}}\,\mathrm{f}_0(5,2,15)\right)\left(t + \frac{e^{5i\pi/12}}{\sqrt{2}}\,\mathrm{f}_0(5,-2,15)\right)$$

= $(t - 2.13060682983889533005591468688942503...)$
$(t - (0.95969178530567025250797047645507504...) +$
$(0.34916071001269654799855316293926907...)\ i)$
$(t - (0.95969178530567025250797047645507504...) -$
$(0.34916071001269654799855316293926907...)\ i)$
$(t + (0.75613568804001789053564010985317772...) +$
$(0.07375086316308890052407649445676 75...)\ i)$
$(t + (0.75613568804001789053564010985317772...) -$
$(0.07375086316308890052407649445676 75...)\ i)$
$(t + (0.26885951218510002700028771004661 02...) -$
$(0.84108577401329800103648634224905292...)\ i)$
$(t + (0.26885951218510002700028771004661 02...) +$
$(0.\ 84108577401329800103648634224905292...)\ i)$

= $t^7 - 2t^6 - t^5 + t^4 + t^3 + t^2 - t - 1.$

E.3 Complex Multiplication

E.3.1 Overview

If E is a non-supersingular elliptic curve over F_q of order u, then:

$$Z = 4q - (q+1-u)^2$$

is positive by the Hasse Theorem (see Annex C.3 and Annex C.4). Thus there is a unique factorization:

$$Z = DV^2$$

where D is squarefree (i.e. contains no square factors). Thus, for each non-supersingular elliptic curve over F_q of order u, there exists a unique squarefree positive integer D such that:

(*) $4q = W^2 + DV^2$,

(**) $u = q + 1 \pm W$

for some W and V.

We say that E has *complex multiplication* by D (or, more properly, by $\sqrt{-D}$). We call D a *CM discriminant* for q. If one knows D for a given curve E, one can compute its order via (*) and (**). As we shall see, one can construct the curves with CM by small D. Therefore one can obtain curves whose orders u satisfy (*) and (**) for small D. The near-primes are plentiful enough that one can find curves of nearly prime order with small enough D to construct.

Over F_q, the CM technique is also called the *Atkin-Morain method*. Over F_{2^m}, it is also called the *Lay-Zimmer method*. Although it is possible (over F_p) to choose the order first and then the field, it is preferable to choose the field first since there are fields in which the arithmetic is especially efficient.

There are two basic steps involved: finding an appropriate order, and constructing a curve having that order. More precisely, one begins by choosing the field size q, the minimum point order r_{min}, and trial division bound l_{max}. Given those quantities, we say that D is *appropriate* if there exists an elliptic curve over F_q with CM by D and having nearly prime order.

Step 1:

(Annex E.3.2 and Annex E.3.3, Finding a Nearly Prime Order):

Find an appropriate D. When one is found, record D, the large prime r, and the positive integer k such that $u = kr$ is the nearly prime curve order.

Step 2:

(Annex E.3.4 and Annex E.3.5, Constructing a Curve and Point):
Given D, k and r, construct an elliptic curve over F_q and a point of order r.

E.3.2 Finding a Nearly Prime Order over F_p

E.3.2.1 Congruence Conditions

A squarefree positive integer D can be a CM discriminant for p only if it satisfies the following congruence conditions. Let:

$$K = \left\lfloor \frac{(\sqrt{p}+1)^2}{r_{\min}} \right\rfloor.$$

— If $p \equiv 3 \pmod 8$, then $D \equiv 2, 3,$ or $7 \pmod 8$.

— If $p \equiv 5 \pmod 8$, then D is odd.

— If $p \equiv 7 \pmod 8$, then $D \equiv 3, 6,$ or $7 \pmod 8$.

— If $K = 1$, then $D \equiv 3 \pmod 8$.

— If $K = 2$ or 3, then $D \not\equiv 7 \pmod 8$.

Thus the possible squarefree D's are as follows:

If $K = 1$, then
D = 3, 11, 19, 35, 43, 51, 59, 67, 83, 91, 107, 115, ….

If $p \equiv 1 \pmod 8$ and $K = 2$ or 3, then
D = 1, 2, 3, 5, 6, 10, 11, 13, 14, 17, 19, 21, ….

If $p \equiv 1 \pmod 8$ and $K \geq 4$, then
D = 1, 2, 3, 5, 6, 7, 10, 11, 13, 14, 15, 17, ….

If $p \equiv 3 \pmod 8$ and $K = 2$ or 3, then
D = 2, 3, 10, 11, 19, 26, 34, 35, 42, 43, 51, 58, ….

If $p \equiv 3 \pmod 8$ and $K \geq 4$, then
D = 2, 3, 7, 10, 11, 15, 19, 23, 26, 31, 34, 35, ….

If $p \equiv 5 \pmod 8$ and $K = 2$ or 3, then
D = 1, 3, 5, 11, 13, 17, 19, 21, 29, 33, 35, 37, ….

If $p \equiv 5 \pmod 8$ and $K \geq 4$, then
D = 1, 3, 5, 7, 11, 13, 15, 17, 19, 21, 23, 29, ….

If $p \equiv 7 \pmod 8$ and $K = 2$ or 3, then
D = 3, 6, 11, 14, 19, 22, 30, 35, 38, 43, 46, 51, ….

If $p \equiv 7 \pmod 8$ and $K \geq 4$, then
D = 3, 6, 7, 11, 14, 15, 19, 22, 23, 30, 31, 35, ….

E.3.2.2 Testing for CM Discriminants (Prime Case)

Input: A prime p and a squarefree positive integer D satisfying the congruence conditions from Annex E.3.2.1.
Output: If D is a CM discriminant for p, an integer W such that:

$$4p = W^2 + DV^2$$

for some V. (In the cases $D = 1$ or 3, the output also includes V.) If not, the message "not a CM discriminant."

1. Apply the appropriate technique from Annex D.1.4 to find a square root modulo p of $-D$ or determine that none exist.
2. If the result of Step 1 indicates that no square roots exist, then output "not a CM discriminant" and stop. Otherwise, the output of Step 1 is an integer B modulo p.
3. Let $A = p$ and $C = (B^2 + D) / p$.
4. Let $S = \begin{bmatrix} A & B \\ B & C \end{bmatrix}$ and $U = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

5. Until $|2B| \le A \le C$, repeat the following steps.

   5.1. Let $\delta = \left\lfloor \frac{B}{A} + \frac{1}{2} \right\rfloor$

   5.2. Let $T = \begin{pmatrix} 0 & -1 \\ 1 & \delta \end{pmatrix}$

   5.3. Replace U by $T^{-1}U$.

   5.4. Replace S by $T^{t} S\, T$, where T^{t} denotes the transpose of T.

6. If $D = 11$ and $A = 3$, let $\delta = 0$ and repeat steps 5.2, 5.3 and 5.4.
7. Let X and Y be the entries of U. That is,

$$U = \begin{pmatrix} X \\ Y \end{pmatrix}$$

8. If $D = 1$ or 3 then output $W = 2X$ and $V = 2Y$ and stop.
9. If $A = 1$ then output $W = 2X$ and stop.
10. If $A = 4$ then output $W = 4X + BY$ and stop.
11. Output "not a CM discriminant."

E.3.2.c Finding a Nearly Prime Order (Prime Case)

Input: A prime p, a trial division bound l_{max}, and lower bound r_{min} for base point order.

Output: A squarefree positive integer D, a prime r with $r_{\min} \le r$, and a smooth integer k such that $u = kr$ is the order of an elliptic curve modulo p with complex multiplication by D.

1. Choose a squarefree positive integer D, not already chosen, satisfying the congruence conditions of Annex E.3.2.1.
2. Compute Annex E.1.1 the Jacobi symbol $J = \left(\frac{-D}{p}\right)$. If $J = -1$ then go to Step 1.
3. List the odd primes l dividing D.
4. For each l, compute Annex E.1.1 the Jacobi symbol $J = \left(\frac{p}{l}\right)$. If $J = -1$ for some l, then go to Step 1.
5. Test Annex E.3.2.2, whether D is a CM discriminant for p. If the result is "not a CM discriminant," go to Step 1. (Otherwise, the result is the integer W, along with V if $D = 1$ or 3.)
6. Compile a list of the possible orders, as follows.
   — If $D = 1$, the orders are:

   $$p + 1 \pm W,\ p + 1 \pm V.$$

   — If $D = 3$, the orders are:

   $$p + 1 \pm W,\ p + 1 \pm (W + 3V)/2,\ p + 1 \pm (W - 3V)/2.$$

   — Otherwise, the orders are $p + 1 \pm W$.
7. Test each order for near-primality (Annex A.2.2.) If any order is nearly prime, output (D, k, r) and stop.
8. Go to Step 1.

Example:

Let $p = 2^{192} - 2^{64} - 1$. Then:

$$p = 4X^2 - 2XY + \frac{1+D}{4}\, Y^2 \text{ and } p + 1 - (4X - Y) = r$$

where $D = 235$,

$X = -31037252937617930835957687234$,

$Y = 5905046152393184521033305113$,

and r is the prime:

$r = 6277101735386680763835789423337720473986773608255189015329$.

Thus there is a curve modulo p of order r having complex multiplication by D.

E.3.3 Finding a Nearly Prime Order over F_{2^m}

E.3.3.1 Testing for CM Discriminants (Binary Case)

Input: A field degree d and a squarefree positive integer $D \equiv 7 \pmod 8$.

Output: If D is a CM discriminant for 2^d, an odd integer W such that:

$$2^{d+2} = W^2 + DV^2,$$

for some odd V. If not, the message "not a CM discriminant."

1. Compute via Annex E.1.2 an integer B such that $B^2 \equiv -D \pmod{2^{d+2}}$.
2. Let $A = 2^{d+2}$ and $C = (B^2 + D) / 2^{d+2}$.
3. Let $S = \begin{bmatrix} A & B \\ B & C \end{bmatrix}$ and $U = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$.
4. Until $|2B| \le A \le C$, repeat the following steps.
   - 4.1 Let $\delta = \left\lfloor \frac{B}{C} + \frac{1}{2} \right\rfloor$.
   - 4.2 Let $T = \begin{bmatrix} 0 & -1 \\ 1 & \delta \end{bmatrix}$.
   - 4.3 Replace U by $T^{-1}U$.
   - 4.4 Replace S by $T^{t} S\, T$, where T^{t} denotes the transpose of T.
5. Let X and Y be the entries of U. That is,

$$U = \begin{bmatrix} X \\ Y \end{bmatrix}$$

6. If $A = 1$, then output $W = X$ and stop.
7. If $A = 4$ and Y is even, then output $W = (4X + BY) / 2$ and stop.
8. Output "not a CM discriminant."

E.3.3.2 Finding a Nearly Prime Order (Binary Case)

Input: A field degree d, a trial division bound l_{max}, and lower bound r_{min} for base point order.

Output: A squarefree positive integer D, a prime r with $r_{min} \le r$, and a smooth integer k such that $u = kr$ is the order of an elliptic curve over F_{2^d} with complex multiplication by D.

1. Choose a squarefree positive integer $D \equiv 7 \pmod 8$, not already chosen.
2. Compute H = the class group for D via Annex E.2.2.
3. Set h = the number of elements in H.
4. If d does not divide h, then go to Step 1.
5. Test via Annex E.3.3.1 whether D is a CM discriminant for 2^d. If the result is "not a CM discriminant," go to Step 1. (Otherwise, the result is the integer W.)
6. The possible orders are $2^d + 1 \pm W$.
7. Test each order for near-primality via Annex A.2.2. If any order is nearly prime, output (D, k, r) and stop.
8. Go to Step 1.

Example:

Let $q = 2^{155}$. Then:

$$4q = X^2 + DY^2 \text{ and } q + 1 - X = 4r$$

where:

$D = 942679$,
$X = 229529878683046820398181$,
$Y = -371360755031779037497$,

and r is the prime:

r = 114179815416476790484662303731262903293568734470.

Thus there is a curve over F_q of order $4r$ having complex multiplication by D.

E.3.4 Constructing a Curve and Point (Prime Case)

E.3.4.1 Constructing a Curve with Prescribed CM (Prime Case)

Given a prime p and a CM discriminant D, the following technique produces an elliptic curve $y^2 \equiv x^3 + a_0x + b_0 \pmod p$ modulo p with CM by D. (Note that there are at least two possible orders among curves with CM by D. The curve constructed here will have the proper CM, but not necessarily the desired order. This curve will be replaced in Annex E.3.4.2 by one of the desired order.)

For nine values of D, the coefficients of E can be written down at once:

D	a_0	b_0
1	1	0
2	–30	56
3	0	1
7	–35	98
11	–264	1694
19	–152	722
43	–3440	77658
67	–29480	1948226
163	–8697680	9873093538

For other values of D, the following algorithm may be used.
Input: A prime modulus p and a CM discriminant $D > 3$ for p.
Output: a_0 and b_0 such that the elliptic curve:

$$y^2 \equiv x^3 + a_0 x + b_0 \pmod{p}$$

has CM by D.

1. Compute $w(t) = w_D(t) \bmod p$ via Annex E.2.3.
2. Let W be the output from Annex E.3.2.2.
3. If W is even, then use Annex E.1.4 with $d = 1$ to compute a root s of $w_D(t)$ modulo p. Let:

   $$V = (-1)^D\, 2^{4I/K}\, s^{24/(GK)} \bmod p,$$

   where G, I and K are as in Annex E.2.3. Finally, let:

   $$a_0 = -3(V + 64)(V + 16) \bmod p,$$
   $$b_0 = 2(V + 64)^2\,(V - 8) \bmod p.$$

4. If W is odd, then use Annex E.1.4 with $d = 3$ to find a cubic factor $g(t)$ of $w_D(t)$ modulo p. Perform the following computations, in which the coefficients of the polynomials are integers modulo p.

   $$V(t) = \begin{cases} -t^{24} \bmod g(t) & \text{if } 3 \nmid D, \\ -256t^{8} \bmod g(t) & \text{if } 3 \mid D, \end{cases}$$

   $$a_1(t) = -3(V(t) + 64)\,(V(t) + 256) \bmod g(t),$$
   $$b_1(t) = 2(V(t) + 64)^2\,(V(t) - 512) \bmod g(t),$$
   $$a_3(t) = a_1(t)^3 \bmod g(t),$$
   $$b_2(t) = b_1(t)^2 \bmod g(t).$$

   Now let σ be a nonzero coefficient from $a_3(t)$, and let τ be the corresponding coefficient from $b_2(t)$. Finally, let:

   $$a_0 = \sigma\tau \bmod p,$$
   $$b_0 = \sigma\tau^2 \bmod p.$$

5. Output (a_0, b_0).

Example:

If $D = 235$, then:

$$w_D(t) = t^6 - 10t^5 + 22t^4 - 24t^3 + 16t^2 - 4t + 4.$$

If $p = 2^{192} - 2^{64} - 1$, then:

$$w_D(t) \equiv (t^3 - (5 + \varphi)t^2 + (1 - \varphi)t - 2)\,(t^3 - (5 - \varphi)t^2 + (1 + \varphi)t - 2) \pmod{p},$$

where $\varphi = 1254098248316315745658220082226751383299177953632927607231$. The resulting coefficients are:

$$a_0 = -2089023816294079213892272128,$$
$$b_0 = -36750495627461354054044457602630966837248.$$

Thus the curve $y^2 \equiv x^3 + a_0x^2 + b_0$ modulo p has CM by $D = 235$.

E.3.4.2 Choosing the Curve and Point (Prime Case)

Input: EC parameters p, k, and r, and coefficients a_0, b_0 produced by Annex E.3.4.1.

Output: A curve E modulo p and a point G on E of order r, or a message “wrong order.”

1. Select an integer ξ with $0 < \xi < p$.
2. If $D = 1$ then set $a = a_0\xi \bmod p$ and $b = 0$.
   If $D = 3$ then set $a = 0$ and $b = b_0\xi \bmod p$.
   Otherwise, set $a = a_0\xi^2 \bmod p$ and $b = b_0\xi^3 \bmod p$.
3. Look for a point G of order r on the curve:
   $$y^2 \equiv x^3 + ax + b \pmod p$$
   via Annex A.3.1. (In the notation of Annex A.3.1, $h = k$ and $n = r$.)
4. If the output of Annex A.3.1 is “wrong order” then output the message “wrong order” and stop.
5. Output the coefficients a, b and the point G.

The method of selecting ξ in the first step of this algorithm depends on the kind of coefficients desired. Two examples follow.

— If $D \neq 1$ or 3, and it is desired that $a = -3$, then take ξ to be a solution of the congruence $a_0\xi^2 \equiv -3 \pmod p$, provided one exists. If one does not exist, or if this choice of ξ leads to the message “wrong order,” then select another curve as follows. If $p \equiv 3 \pmod 4$ and the result was “wrong order,” then choose $p - \xi$ in place of ξ; the result leads to a curve with $a = -3$ and the right order. If no solution ξ exists, or if $p \equiv 1 \pmod 4$, then repeat Annex E.3.4.1 with another root of the reduced class polynomial. The proportion of roots leading to a curve with $a = -3$ and the right order is roughly one-half if $p \equiv 3 \pmod 4$, and one-quarter if $p \equiv 1 \pmod 4$.

— If there is no restriction on the coefficients, then choose ξ at random. If the output is the message “wrong order,” then repeat the algorithm until a set of parameters a, b, G is obtained. This will happen for half the values of ξ, unless $D = 1$ (one-quarter of the values) or $D = 3$ (one-sixth of the values).

E.3.5 Constructing a Curve and Point (Binary Case)

E.3.5.1 Constructing a Curve with Prescribed CM (Binary Case)

Input: A field F_{2^m}, a CM discriminant D for 2^m, and the desired curve order u.

Output: a and b such that the elliptic curve:
$$y^2 + xy = x^3 + ax^2 + b$$
over F_{2^m} has order u.

1. Compute $w(t) = w_D(t) \bmod 2$ via Annex E.2.3.
2. Use Annex E.3.3.1 to find the smallest divisor d of m greater than $(\log_2 D) - 2$ such that D is a CM discriminant for 2^d.
3. Compute $p(t)$ = a degree d factor modulo 2 of $w(t)$. (If $d = h$, then $p(t)$ is just $w(t)$ itself. If $d < h$, $p(t)$ is found via Annex E.1.5.)
4. Compute α := a root in F_{2^m} of $p(t) = 0$ via Annex D.2.2.
5. If 3 divides D
   then set $b = \alpha$
   else set $b = \alpha^3$
6. If u is divisible by 4, then set $a = 0$
   else if m is odd, then set $a = 1$
   else generate via Annex D.1.5 a random element $a \in F_{2^m}$ of trace 1.
7. Output (a, b).

Example:

If D = 942679, then:

$$
\begin{aligned}
\mathrm{w}_D(t) \equiv 1 &+ t^{2} + t^{6} + t^{10} + t^{12} + t^{13} + t^{16} + t^{17} + t^{20} + t^{22} + t^{24} + t^{27} + t^{30} + t^{33} + t^{35} + t^{36} + t^{37} + \\
& t^{41} + t^{42} + t^{43} + t^{45} + t^{49} + t^{51} + t^{54} + t^{56} + t^{57} + t^{59} + t^{61} + t^{65} + t^{67} + t^{68} + t^{69} + t^{70} + t^{71} + t^{72} + t^{74} + t^{75} + \\
& t^{76} + t^{82} + t^{83} + t^{87} + t^{91} + t^{93} + t^{96} + t^{99} + t^{100} + t^{101} + t^{102} + t^{103} + t^{106} + t^{108} + t^{109} + t^{110} + t^{114} + t^{117} + \\
& t^{119} + t^{121} + t^{123} + t^{125} + t^{126} + t^{128} + t^{129} + t^{130} + t^{133} + t^{134} + t^{140} + t^{141} + t^{145} + t^{146} + t^{147} + t^{148} + t^{150} + \\
& t^{152} + t^{154} + t^{155} + t^{157} + t^{158} + t^{160} + t^{161} + t^{166} + t^{167} + t^{171} + t^{172} + t^{175} + t^{176} + t^{179} + t^{180} + t^{185} + t^{186} + \\
& t^{189} + t^{190} + t^{191} + t^{192} + t^{195} + t^{200} + t^{201} + t^{207} + t^{208} + t^{209} + t^{210} + t^{211} + t^{219} + t^{221} + t^{223} + t^{225} + t^{228} + \\
& t^{233} + t^{234} + t^{235} + t^{237} + t^{238} + t^{239} + t^{241} + t^{242} + t^{244} + t^{245} + t^{248} + t^{249} + t^{250} + t^{252} + t^{253} + t^{255} + t^{257} + \\
& t^{260} + t^{262} + t^{263} + t^{264} + t^{272} + t^{273} + t^{274} + t^{276} + t^{281} + t^{284} + t^{287} + t^{288} + t^{289} + t^{290} + t^{292} + t^{297} + t^{299} + \\
& t^{300} + t^{301} + t^{302} + t^{304} + t^{305} + t^{306} + t^{309} + t^{311} + t^{312} + t^{313} + t^{314} + t^{317} + t^{318} + t^{320} + t^{322} + t^{323} + t^{325} + \\
& t^{327} + t^{328} + t^{329} + t^{333} + t^{335} + t^{340} + t^{341} + t^{344} + t^{345} + t^{346} + t^{351} + t^{353} + t^{354} + t^{355} + t^{357} + t^{358} + t^{359} + \\
& t^{360} + t^{365} + t^{366} + t^{368} + t^{371} + t^{372} + t^{373} + t^{376} + t^{377} + t^{379} + t^{382} + t^{383} + t^{387} + t^{388} + t^{389} + t^{392} + t^{395} + \\
& t^{398} + t^{401} + t^{403} + t^{406} + t^{407} + t^{408} + t^{409} + t^{410} + t^{411} + t^{416} + t^{417} + t^{421} + t^{422} + t^{423} + t^{424} + t^{425} + t^{426} + \\
& t^{429} + t^{430} + t^{438} + t^{439} + t^{440} + t^{441} + t^{442} + t^{443} + t^{447} + t^{448} + t^{450} + t^{451} + t^{452} + t^{453} + t^{454} + t^{456} + t^{458} + \\
& t^{459} + t^{460} + t^{462} + t^{464} + t^{465} + t^{466} + t^{467} + t^{471} + t^{473} + t^{475} + t^{476} + t^{481} + t^{482} + t^{483} + t^{484} + t^{486} + t^{487} + \\
& t^{488} + t^{491} + t^{492} + t^{495} + t^{496} + t^{498} + t^{501} + t^{503} + t^{505} + t^{507} + t^{510} + t^{512} + t^{518} + t^{519} + t^{529} + t^{531} + t^{533} + \\
& t^{536} + t^{539} + t^{540} + t^{541} + t^{543} + t^{545} + t^{546} + t^{547} + t^{548} + t^{550} + t^{552} + t^{555} + t^{556} + t^{557} + t^{558} + t^{559} + t^{560} + \\
& t^{563} + t^{565} + t^{566} + t^{568} + t^{580} + t^{585} + t^{588} + t^{589} + t^{591} + t^{592} + t^{593} + t^{596} + t^{597} + t^{602} + t^{604} + t^{606} + t^{610} + \\
& t^{616} + t^{620} \pmod 2.
\end{aligned}
$$

This polynomial factors into 4 irreducibles over F_2, each of degree 155. One of these is:

$$
\begin{aligned}
\mathrm{p}(t) = 1 &+ t + t^{2} + t^{6} + t^{9} + t^{10} + t^{11} + t^{13} + t^{14} + t^{15} + t^{16} + t^{18} + t^{19} + t^{22} + t^{23} + t^{26} + t^{27} + \\
& t^{29} + t^{31} + t^{49} + t^{50} + t^{51} + t^{54} + t^{55} + t^{60} + t^{61} + t^{62} + t^{64} + t^{66} + t^{70} + t^{72} + t^{74} + t^{75} + t^{80} + t^{82} + t^{85} + t^{86} + \\
& t^{88} + t^{89} + t^{91} + t^{93} + t^{97} + t^{101} + t^{103} + t^{104} + t^{111} + t^{115} + t^{116} + t^{117} + t^{118} + t^{120} + t^{121} + t^{123} + t^{124} + t^{126} \\
& + t^{127} + t^{128} + t^{129} + t^{130} + t^{131} + t^{132} + t^{134} + t^{136} + t^{137} + t^{138} + t^{139} + t^{140} + t^{143} + t^{145} + t^{154} + t^{155}.
\end{aligned}
$$

If t is a root of $p(t)$, then the curve:

$$y^2+xy = x^3 + t^3$$

over $F_{2^{155}}$ has order $4r$, where r is the prime:

$$r = 11417981541647679048466230373126290329356873447.$$

E.3.5.2 Choosing the Curve and Point (Binary Case)

Input: A field size F_{2^m}, an appropriate D, the corresponding k and r from Annex E.3.3.2.

Output: A curve E over F_{2^m} and a point G on E of order r.

1. Compute a and b via Annex E.3.5.1 with $u = kr$.
2. Find a point G of order r via Annex A.3.1. (In the notation of Annex A.3.1, $h = k$ and $n = r$.)
3. Output the coefficients a, b and the point G.

Annex F
(informative)
An Overview of Elliptic Curve Systems

Many public-key cryptographic systems are based on exponentiation operations in large finite mathematical groups. The cryptographic strength of these systems is derived from the believed computational intractability of computing logarithms in these groups. The most common groups are the multiplicative groups of Z_p (the integers modulo a prime p) and F_{2^m} (characteristic 2 finite fields). The primary advantages of these groups are their rich theory, easily understood structure, and straightforward implementation. However, they are not the only groups that have the requisite properties. In particular, the mathematical structures known as elliptic curves have the requisite mathematical properties, a rich theory, and are especially amenable to efficient implementation in hardware or software.

The algebraic system defined on the points of an elliptic curve provides an alternate means to implement the ElGamal [13] and ElGamal-like public-key encryption and signature protocols. These protocols are described in the literature in the algebraic system Z_p, the integers modulo p, where p is a prime. For example, the Digital Signature Algorithm (DSA) defined in ANSI X9.30 Part 1 [3] is an ElGamal-like signature scheme defined over Z_p. The same protocol for signing can be defined over the points on an elliptic curve.

Elliptic curve systems as applied to ElGamal protocols were first proposed in 1985 independently by Neil Koblitz from the University of Washington, and Victor Miller, who was then at IBM, Yorktown Heights. The security of the cryptosystems using elliptic curves hinges on the intractability of the discrete logarithm problem in the algebraic system. Unlike the case of the discrete logarithm problem in finite fields, or the problem of factoring integers, there is no subexponential-time algorithm known for the elliptic curve discrete logarithm problem. The best algorithm known to date takes fully exponential time.

Associated with any finite field F_q there are on the order of q different (up to isomorphism) elliptic curves that can be formed and used for the cryptosystems. Thus, for a fixed finite field with q elements and with a large value of q, there are many choices for the elliptic curve group. Since each elliptic curve operation requires a number of more basic operations in the underlying finite field F_q, a finite field may be selected with a very efficient software or hardware implementation, and there remain an enormous number of choices for the elliptic curve.

This Standard describes the implementation of a signature algorithm which uses elliptic curves over a finite field F_q, where q is either a prime number or equal to 2^m for some positive integer m.

Annex G
(informative)
The Elliptic Curve Analog of the DSA (ECDSA)

The elliptic curve algorithm (ECDSA) described in this Standard is the elliptic curve analog of a discrete logarithm algorithm that is usually described in the setting of F_p^* (also denoted Z_p^*), the multiplicative group of the integers modulo a prime. The following tables show the correspondence between the elements and operations of the group F_p^* and the elliptic curve group $E(F_q)$.

Table G-1 – DSA and ECDSA Group Information

Group	F_p*	$E(F_q)$
Group elements	The set of integers {1,2,...,p-1}	Points (x,y) which satisfy the defining equation of the elliptic curve, plus the point at infinity $\mathcal{O}$.
Group operation	Multiplication modulo p	Addition of points
Notation	Elements: g, h Multiplication: $g \times h$ Exponentiation: g^a	Elements: P, Q Addition: $P + Q$ Multiple of a point (also called scalar multiplication): aP
Discrete logarithm problem	Given $g \in F_p^*$ and $h = g^a$ mod p, find the integer a.	Given $P \in E(F_q)$ and $Q = aP$, find the integer a.

Table G-2 – DSA and ECDSA Notation

DSA Notation	ECDSA Notation
q	n
g	G
x	d
y	Q

Table G-3 – DSA and ECDSA Setup

DSA Setup	ECDSA Setup
1. p and q are primes, q divides p-1. 2. g is an element of order q in F_p*. 3. The group used is: $\{g^0, g^1, g^2, ..., g^{q-1}\}$.	1. E is an elliptic curve defined over the field F_q. 2. G is a point of prime order n in $E(F_q)$. 3. The group used is: $\{ \mathcal{O}, G, 2G, ..., (n-1)G\}$.

Table G-4 – DSA and ECDSA Key Generation

DSA Key Generation	ECDSA Key Generation
1. Select a random integer x in the interval [1, q-1]. 2. Compute $y = g^x$ mod p. 3. The private key is x. 4. The public key is y.	1. Select a statistically unique and unpredictable integer d in the interval [1, n-1]. 2. Compute $Q = dG$. 3. The private key is d. 4. The public key is Q.

Table G-5 – DSA and ECDSA Signature Generation

DSA Signature Generation	ECDSA Signature Generation
1. Select a random integer k in the interval [1, q-1]. 2. Compute g^k mod p. 3. Compute $r = (g^k \bmod p) \bmod q$. 4. Compute $e = H(M)$. 5. Compute $s = k^{-1}(e + xr) \bmod q$. 6. The signature for M is (r, s).	1. Select a statistically unique and unpredictable integer k in the interval [1, n-1]. 2. Compute $kG = (x_1, y_1)$. 3. Compute $r = x_1 \bmod n$. 4. Compute $e = H(M)$. 5. Compute $s = k^{-1}(\mathrm{e}+dr) \bmod n$. 6. The signature for M is (r, s).

Table G-6 – DSA and ECDSA Signature Verification

DSA Signature Verification	ECDSA Signature Verification
1. Compute $e = H(M)$.	1. Compute $e = H(M)$.
2. Compute $s^{-1} \bmod q$.	2. Compute $s^{-1} \bmod n$.
3. Compute $u_1 = es^{-1} \bmod q$.	3. Compute $u_1 = es^{-1} \bmod n$.
4. Compute $u_2 = rs^{-1} \bmod q$.	4. Compute $u_2 = rs^{-1} \bmod n$.
5. Compute $v' = g^{u_1} y^{u_2} \bmod p$.	5. Compute $u_1G + u_2Q = (x_1, y_1)$.
6. Compute $v = v' \bmod q$.	6. Compute $v = x_1 \bmod n$.
7. Accept the signature if $v = r$.	7. Accept the signature if $v = r$.

Annex H
(informative)
Security Considerations

This annex is provided as initial guidance for implementers of this Standard. This information should be expected to change over time. Implementers should review the current state-of-the-art in attacks on elliptic curve systems at the time of implementation.
Annex H.1 summarizes the best attacks known on the elliptic curve discrete logarithm problem, which is the basis for the security of all elliptic curve systems. Annexes H.2 and H.3 discuss security issues for elliptic curve domain parameters and elliptic curve key pairs, respectively. The security considerations discussed in Annexes H.1, H.2 and H.3 affect all elliptic curve systems. Annex H.4 discusses security issues specific to the ECDSA.

H.1 The Elliptic Curve Discrete Logarithm Problem

Let E be an elliptic curve defined over a finite field F_q. Let $G \in E(F_q)$ be a point of order n, where n is a prime number and $n > 2^{160}$.

The elliptic curve discrete logarithm problem (ECDLP) is the following: given E, G and $Q \in E(F_q)$, determine the integer l, $0 \leq l \leq n-1$, such that $Q = lG$, provided that such an integer exists.

The best general algorithms known to date for ECDLP are the Pollard-ρ method [35] and the Pollard-λ method [35]. The Pollard-ρ method takes about $\sqrt{\pi n / 2}$ steps, where each step is an elliptic curve addition. The Pollard-ρ method can be parallelized (see [34]) so that if m processors are used, then the expected number of steps by each processor before a single discrete logarithm is obtained is $(\sqrt{\pi n / 2}) / m$. The Pollard λ method takes about 3.28 $\sqrt{n}$ steps. It can also be parallelized (see [34]) so that if m processors are used, then the expected number of steps by each processor before a single discrete logarithm is obtained is about $(2\sqrt{n}) / m$.

Some special classes of elliptic curves, including *supersingular curves*, have been prohibited in this Standard by the requirement of the MOV condition (see Annex A.1.1). These curves have been prohibited because there is a method for efficiently reducing the discrete logarithm problem in these curves to the discrete logarithm problem in a finite field.

Also, the special class of elliptic curves called F_q*-anomalous curves* have been prohibited by the requirement of the Anomalous condition (see Annex A.1.2) because there is an efficient algorithm for computing discrete logarithms in $E(F_q)$ where E is an anomalous curve over F_q (i.e. $\#E(F_q) = q$).

In April 1998, Gallant, Lambert, and Vanstone [14], and Wiener and Zuccherato [40] showed that the best algorithms known for the ECDLP (including Pollard-ρ) can be sped up by a factor of $\sqrt{2}$. Thus the expected running time of the Pollard-ρ method with this speedup is $\sqrt{\pi n / 4}$ steps. They also showed that if E is an elliptic curve defined over F_{2^e}, then the best algorithm known for the ECDLP in $E(F_{2^{ed}})$ can be sped up by a factor of $\sqrt{(2d)}$. This should be considered when doing a security analysis of curves generated using the Weil Theorem (see Note 6 in Annex A.3.2).

For example, the *binary anomalous curve* E: $y^2+xy = x^3+x^2+1$ has the property that $\#E(F_{2^{163}}) = 2n$, where n is a 162-bit prime. The ECDLP in $E(F_{2^{163}})$ can be solved in about 2^{77} elliptic curve operations, which is 16 times less work than the 2^{81} elliptic curve operations required to solve the ECDLP for a random curve of similar order. Now, a field operation in $F_{2^{163}}$ takes about the same time as a SHA-1 operation, and it takes about 6 field operations to do an elliptic curve operation and about 2 more field operations to operate in the equivalence relation posited by the above improved algorithm. Hence, it turns out that the improved algorithm takes roughly the same amount of work as it does to find a collision in SHA-1.

To guard against existing attacks on ECDLP, one should select an elliptic curve E over F_q such that:

1. The order $\#E(F_q)$ is divisible by a large prime $n > 2^{160}$;
2. The MOV condition (Annex A.1.1) holds; and
3 The Anomalous condition (Annex A.1.2) holds.

Furthermore, to guard against possible future attacks against special classes of non-supersingular curves, it is prudent to select an elliptic curve at random. Annex A.3.3 describes a method for selecting an elliptic curve *verifiably* at random.

H.1.1 Software Attacks

Assume that a 1 MIPS (Million Instructions Per Second) machine can perform $4\text{x}10^4$ elliptic curve additions per second. (This estimate is indeed high — an ASIC (Application Specific Integrated Circuit) built for performing elliptic curve operations over the field $F_{2^{155}}$ has a 40 MHz clock-rate and can perform roughly 40,000 elliptic additions per second.) Then, the number of elliptic curve additions that can be performed by a 1 MIPS machine in one year is

$$(4\times 10^4)\cdot(60\times 60\times 24\times 365)\approx 2^{40}.$$

Table H-1 shows the computing power required to compute a single discrete logarithm for various values of n. As an example, if 10,000 computers each rated at 1,000 MIPS are available, and $n \approx 2^{160}$, then an elliptic curve discrete logarithm can be computed in 85,000 years.

Odlyzko [33] has estimated that if 0.1% of the world's computing power were available for one year to work on a collaborative effort to break some challenge cipher, then the computing power available would be 10^8 MIPS years in 2004 and 10^{10} to 10^{11} MIPS years in 2014.

Field size (in bits)	Size of n (in bits)	$\sqrt{\pi n / 4}$	MIPS years
163	160	2^{80}	$8.5 \text{ x } 10^{11}$
191	186	2^{93}	$7.0 \text{ x } 10^{15}$
239	234	2^{117}	$1.2 \text{ x } 10^{23}$
359	354	2^{177}	$1.3 \text{ x } 10^{41}$
431	426	2^{213}	$9.2 \text{ x } 10^{51}$

Computing power required to compute elliptic curve logarithms with the Pollard-ρ method.

Note: The strength of any cryptographic algorithm relies on the best methods that are known to solve the hard mathematical problem that the cryptographic algorithm is based upon, The discovery and analysis of the best methods for any hard mathematical problem is a continuing research topic. Users of ECDSA should monitor the state of the art in solving the ECDLP, as it is subject to change. The purpose of the above discussion is to describe the current state of knowledge regarding attacks on the ECDLP as of June 1998.

H.1.2 Hardware Attacks

A more promising attack (for well-funded attackers) on elliptic curve systems would be to build special-purpose hardware for a parallel search. Van Oorschot and Wiener [34] provide a detailed study of such a possibility. In their 1994 study, they estimated that if $n \approx 10^{36} \approx 2^{120}$, then a machine with m = 325,000 processors that could be built for about $10 million would compute a single discrete logarithm in about 35 days.

It must be emphasized that these estimates were made for specific elliptic curve domain parameters having $n \approx 10^{36} \approx 2^{120}$. This Standard mandates that the parameter n should satisfy

$$n > 2^{160} \approx 10^{48},$$

and hence the hardware attacks are infeasible.

H.1.3 Key Length Considerations

It should be noted that for the software and hardware attacks described above, the computation of a single elliptic curve discrete logarithm has the effect of revealing a *single* user's private key. Roughly the same effort must be repeated in order to determine another user's private key.

If a single instance of the ECDLP (for a given elliptic curve E and base point G) is solved using the Pollard-λ 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 G). More precisely, if the first instance takes expected time t, then the second instance takes expected time $(\sqrt{2}-1)t \approx 0.41t$. Having solved these two instances, the third instance takes expected time $(\sqrt{3}-\sqrt{2})t \approx 0.32t$. Having solved these three instances, the fourth instance takes expected time $(\sqrt{4}-\sqrt{3})t \approx 0.27t$. And so on. Thus, subsequent instances of the ECDLP (for a given elliptic curve and base point G) 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 G) 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. Note also that the concern that successive logarithms become easier is addressed in this Standard by ensuring that the first instance is infeasible to solve (via the requirement that $n > 2^{160}$).

In [11], Blaze et al. report on the minimum key lengths required for secure symmetric-key encryption schemes (such as DES and IDEA). Their report provides the following conclusion:

> *To provide adequate protection against the most serious threats — well-funded commercial enterprises or government intelligence agencies — keys used to protect data today should be at least 75 bits long. To protect information adequately for the next 20 years in the face of expected advances in computing power, keys in newly-deployed systems should be at least 90 bits long.*

Extrapolating these conclusions to the case of elliptic curves, we see that n should be at least 150 bits for short-term security, and at least 180 bits for medium-term security. This extrapolation is justified by the following considerations:

1. Exhaustive search through a k-bit symmetric-key cipher takes about the same time as the Pollard-ρ or Pollard-λ algorithms applied to an elliptic curve having a $2k$-bit parameter n.
2. Both exhaustive search with a symmetric-key cipher and the Pollard-ρ and Pollard-λ algorithms can be parallelized with a linear speedup.
3. A basic operation with elliptic curves (addition of two points) is computationally more expensive than a basic operation in a symmetric-key cipher (encryption of one block).
4. In both symmetric-key ciphers and elliptic curve systems, a "break" has the same effect: it recovers a single private key.

H.2 Elliptic Curve Domain Parameters

Elliptic curve domain parameters are comprised of a field size q, an indication of basis used (in the case $q=2^m$), an optional SEED if the elliptic curve was generated verifiably at random, two elements a, b in F_q which define an elliptic curve E over F_q, a point $G=(x_G, y_G)$ of prime order in $E(F_q)$, the order n of G, and the cofactor h. See Sections 5.1.1.1 and 5.1.2.1 for a more detailed description of elliptic curve domain parameters.

1. Choice of basis. The basis of F_{2^m} specifies the way of interpreting the bit strings that make up the elements of F_{2^m}. There are two choices for the basis allowed in this Standard: a polynomial basis and a normal basis. It is not a security consideration which basis to use, but all users of a set of elliptic curve domain parameters must use the same basis externally. (Implementations with different internal representations that produce equivalent results are allowed.).
2. Use of the canonical seeded hash (Annex A.3.3) to determine the elliptic curve equation (described) by a and b). For the DSA, there is the possibility that a particularly poor choice of domain parameters could lead to an attack. To address this, the DSA requires the use of a canonical seeded hash to generate the domain parameters p and q, as this provides an assurance that p and q were generated arbitrarily. The analogous attack on the ECDSA does not apply as there are no known poor choices for the elliptic curve domain parameters that are not already excluded by this Standard. However, use of the canonical seeded hash can help mitigate fears about the possibility of new special-purpose attacks which might be discovered in the future.

   The use of a specific elliptic curve may allow performance improvements over the use of an arbitrary elliptic curve. For these reasons, this Standard allows both the choice of a particular elliptic curve or the generation of an arbitrary curve through the use of a canonical seeded hash function. An arbitrary curve may be used when security considerations are so preeminent that the possible performance impact is not a factor in the decision.
3. Choice of base point G. The choice of the base point G is not a security consideration as long as it has a large prime order as required by this Standard. However, all users of a set of elliptic curve domain parameters must use the same base point.
4. Elliptic curve domain parameter validation. The generator of a set of elliptic curve domain parameters should ensure that they meet the elliptic curve domain parameter validation criteria listed in Section 5.1. Whether anyone else needs to validate the elliptic curve domain parameters is a matter of the trust relationship between the generator and the user. For example, an untrusted party may generate a proposed set of elliptic curve domain parameters and a CA may subsequently validate the parameters for its potential users. Whether or not it validates elliptic curve domain parameters should be part of a CA's policy. If a set of elliptic curve domain parameters is supplied directly to a user in a situation where the user does not know that they are valid, then the user should validate the parameters before use; not doing so could leave the user open to the potential of an attack. As a minimum, a user should knowingly accept this risk if the elliptic curve domain parameters are not validated.
5. Elliptic curve domain parameter cryptoperiod considerations. A set of elliptic curve domain parameters may be used by one party to generate a single key pair or by that party to generate multiple key pairs. Alternatively, a group of parties could use the same set of parameters to generate multiple key pairs. How many users and how many key pairs should be allowed for a specific set of elliptic curve domain parameters is a policy decision.

   Just as a single elliptic curve key pair has a cryptoperiod which is deemed appropriate for its individual strength, so a set of elliptic curve domain parameters has a cryptoperiod which is deemed appropriate for its collective strength; that is, for all the key pairs expected to be generated using it. As noted in Annex H.1.3, for a given set of elliptic curve domain parameters, the cost to break k keys is only $\sqrt{k}$ times the cost to break one key. As more and more monetary value becomes protected by a specific set of elliptic curve domain parameters by allowing multiple users and multiple key pairs, there comes a point where it is appropriate for a user to use a different set of elliptic curve domain parameters (i.e. a different elliptic curve). This follows the general security principle of compartmentalization.

   Potential concerns about breaking a second key (or subsequent keys) given that a first key (which used the same elliptic curve domain parameters) has been broken are addressed in this Standard by the inability of an adversary to break the first key. As this Standard mandates that the order n of the base point G be greater than 2^{160}, breaking the first key is thought to be infeasible.
6. How large the MOV threshold B (see Annex A.1) should be. The MOV threshold B is a positive integer B such that taking discrete logarithms over F_{q^B} is at least as difficult as taking elliptic curve discrete logarithms over F_q. For this Standard, $B \geq 20$. For example, all elliptic curves over $F_{2^{191}}$, that are able to be mapped into finite fields with an order up to around 2^{3800} are eliminated from consideration. The value B = 20 is a conservative choice, and is sufficient to ensure resistance against the reduction attack.

7. What values to use for l_{max} and r_{min} when determining n, the order of the base point G (see Annex A.3.2). The value r_{min} is the minimum value that is appropriate for n, the order of the base point G in the elliptic curve domain parameters. For this Standard, $r_{min} > 2^{160}$. For example, if the order of the underlying field is 2^{191}, an appropriate value for r_{min} is $\approx 2^{185}$. When the order of the underlying field is larger, a larger r_{min} and therefore a larger n is appropriate. Mitigating the choice is the fact that finding a curve satisfying stricter requirements will take longer. The trial division bound l_{max} is the maximum size of all prime factors of the cofactor h. In this Standard, the order of an elliptic curve will be a number u such that $u = hn$, where n is a large prime factor (and the order of the base point G) and, h is a number whose prime factors are all less than l_{max}. For example, if the order of the underlying field is 2^{191} and r_{min} is 2^{185}, then an appropriate value for l_{max} is 255.
8. Point compression. The representation of a point in compressed, uncompressed, or hybrid form is not a security consideration.

H.3 Key Pairs

1. Associating public keys with elliptic curve domain parameters. It is very important that a public key and a private key be cryptographically bound to their associated elliptic curve domain parameters. The cryptographic binding of a public key with its associated elliptic curve domain parameters can be done by a CA, who includes the elliptic curve domain parameters in the data portion of the public-key certificate.
2. Public Key validation. There are potential attacks if a purported public key Q does not actually conform to the requirements of a public key. That is, Q should be an elliptic curve point of order n. For this reason, an optional public key validation routine has been specified in this Standard (Section 5.2.2). This routine assumes that the associated elliptic curve domain parameters have previously been validated. It checks the range and order of a purported public key to ensure that it is plausible that a private key could logically exists for this purported public key. Whether or not it validates public keys should be part of a CA's policy. It is recommended that a user validate all public keys that it does not know otherwise to be valid, as not doing so could leave the user open to the potential of an attack. As a minimum, a user should knowingly accept this risk if the public key is not validated.
3. Private key cryptoperiod considerations. It is appropriate to assign a cryptoperiod to a private key. That is, explicitly state an amount of time for which the private key can be used to generate digital signatures. The cryptoperiod defined for a particular private key is a policy decision. The strength of the key and the amount and value of information that will be protected by it are considerations to take into account when determining an appropriate cryptoperiod. Following the general security principle of compartmentalization, limiting the amount of information protected by a particular key limits the amount of damage that might occur if the private key is compromised. As the Standard mandates that the primary security parameter n be greater than 2^{160}, as of 1998, it is considered infeasible for the best methods known for solving the ECDLP to discover the private key. Users should monitor the state-of-the-art in solving the ECDLP to help determine an appropriate value of n.
4. Public key cryptoperiod considerations. A public key can be considered valid to verify digital signatures for any period of time after the associated private key was used to generate digital signatures. The appropriate cryptoperiod for a public key is a policy decision.
5. Repeated private keys. If two users are using the same elliptic curve domain parameters and somehow generate identical private key d values, then either could impersonate the other. As the private key d is a value between 1 and n-1 (inclusive), and n is required to be greater than 2^{160}, a duplicate private key is only expected to happen by chance (due to the birthday phenomenon) after about 2^{80} key pairs have been generated. As 2^{80} is over 1 million million million million, this is not expected to happen. However, it is possible that a private key might repeat due to a hardware or software error or a poorly-seeded pseudorandom number generator. If this occurred, the public key Q for the two users would also repeat. One way to address this concern is to use an ANSI X9 approved random or pseudorandom generation method. For an example of an ANSI X9 approved pseudorandom number generation method, see Annex A.4. Otherwise, a service that a Certificate Authority may choose to provide for users with high security requirements is to monitor public keys to ensure that there are no duplicates. If a duplicate public key is detected, then both parties should separately be told to revoke their current public key, determine if there has been an error, try to determine the cause of the error, decide what corrective action to take (if any), and regenerate new key pairs.

6. Non-repudiation issues. A particular value of a private key is required by this Standard to be a statistically unique and unpredictable value between 1 and n-1, where n is the prime order of the base generating point G. Any value that is the output of an ANSI X9 approved random or pseudorandom generator that is in the correct range is a valid value for a private key. Any given private key cannot be repudiated solely because of the particular integer value it might possess; that is, all potential private key values are valid key values if they should happen to be generated in conformance with this Standard.

H.4 ECDSA

1. Attacks on the hash function. This standard specifies the use of the Secure Hash Algorithm Revision 1 (SHA-1). If SHA-1 is broken, this Standard should not be used as is currently written.
2. Vaudenay's attack. Vaudenay [39] presented some attacks on the DSA where an adversary can forge one signature if she can select the elliptic curve domain parameters. One attack relies on the fact that the DSA signature hash function is actually SHA-1 mod q, not merely SHA-1. In ECDSA, Vaudenay's attack is thwarted because $n > 2^{160}$. Another attack of Vaudenay's is thwarted by cryptographic binding of public keys with the elliptic curve domain parameters with which it is associated.
3. Repeated per-message secrets. As with the possibility of repeated private keys (see Annex H.3), the possibility of a per-message secret k value repeating during signature generation may also be a concern. A k value has the same numeric and security constraints as a private key. If a k value repeats for two different messages, then the r value in the signature will also repeat and it is then possible for an adversary with access to both signatures to recover the associated private key. As with the private key, this event should never occur except by chance. As above, one way to address this concern is to use an ANSI X9 approved random or pseudorandom number generation method. Another way to address the possibility of an otherwise undetected hardware or software error or a poorly-seeded pseudorandom number generator is for a system intended for users with high security requirements to maintain a list of r values previously output by signature generation so that it can detect if an r value ever repeats. If a repeated r value is detected, the associated signature should not be output and a possible error indicated. The owner of the system should try to determine what happened and what corrective action to take, including whether to continue to operate the system.

Annex I
(informative)
Small Example of the ECDSA

I.1 System Setup

The underlying finite field is F_{23}, and the elliptic curve is $y^2 = x^3 + x + 1$, as described in Example 5 in Annex B.3. The point $G = (x_G, y_G) = (13, 7)$ is selected. Since $7G = \mathcal{O}$, the point G has order $n = 7$.

The domain parameters (the public information) are:

— the field F_{23},
— the curve E,
— the point G,
— the order $n = 7$, and
— the cofactor $h = 4$.

I.2 Key Generation

Entity A performs the following operations.

1. A selects a random integer $d = 3$ in the interval $[1, n\text{-}1] = [1, 6]$.
2. A computes the point $Q = dG = 3(13, 7) = (17, 3)$.
3. A makes public the point Q.
4. A′s private key is the integer $d = 3$.

I.3 Signature Generation for ECDSA

Entity A signs message M = 11100011010111100. Suppose that the decimal representation of the hash value $H(M)$ is $e = 6$.

Entity A:

1. Selects a random integer $k = 4$ in the interval $[1, n\text{-}1] = [1, 6]$.
2. Computes:

$$\begin{aligned}(x_1, y_1) &= kG \\ &= 4(13, 7) \\ &= (17, 20).\end{aligned}$$

3. Represents x_1 as the integer $\overline{x}_1 = 17$.
4. Sets $r = \overline{x}_1 \bmod n = 17 \bmod 7 = 3$.
5. Computes:

$$\begin{aligned}s &= k^{-1}(e + dr) \bmod n \\ &= 4^{-1}(6 + 3 \times 3) \bmod 7 \\ &= 2(15) \bmod 7 \\ &= 2.\end{aligned}$$

The signature on message M is $(r, s) = (3, 2)$.

I.4 Signature Verification for ECDSA

Entity B verifies signature $(r', s') = (3, 2)$ on M as follows.

Entity B:

1. Looks up A’s public key $Q = (17, 3)$.
2. Computes $e' = 7$, the decimal representation of $H(M)$.
3. Computes:

$$\begin{aligned}c &= (s')^{-1} \bmod n \\ &= 2^{-1} \bmod 7 \\ &= 4.\end{aligned}$$

4. Computes

$$u_1 = e' c \bmod n = 6 \times 4 \bmod 7 = 3$$

and

$$u_2 = r' c \bmod n = 3 \times 4 \bmod 7 = 5.$$

5. Computes the point:

$$(x_1, y_1) = u_1 G + u_2 Q = 3G + 5Q = 3(13, 7) + 5(17, 3) = (17, 20).$$

6. Represents x_1 as the integer $\overline{x}_1 = 17$.
7. Computes $v = \overline{x}_1 \bmod n = 17 \bmod 7 = 3$.
8. Accepts the signature since $v = r' = 3$.

Annex J
(informative)
Examples of ECDSA and Sample Curves

This annex contains 5 parts.

— Annex J.1 presents examples of data conversion methods.
— Annex J.2 presents 2 examples of ECDSA over the field F_{2^m}.
— Annex J.3 presents 2 examples of ECDSA over the field F_p , where p is odd prime.
— Annex J.4 presents sample elliptic curves over the field F_{2^m} with domain parameters for m = 163, 176, 191, 208, 239, 272, 304, 359, 368 and 431.
— Annex J.5 presents sample elliptic curves over field F_p with domain parameters for 192-bit, 239-bit, and 256-bit primes.

The sample curves in Annexes J.4 and J.5 may be used in an implementation of this Standard.

J.1 Examples of Data Conversion Methods

The following are examples of the data conversion techniques that shall be used in this Standard (See Figure 1).

Example of Integer-to-Octet-String Conversion. (See Section 4.3.1.)
Input: x = 123456789, k=4
Output: M= 075BCD15

Example of Octet-String-to-Integer Conversion. (See Section 4.3.2.)
Input: M= 0003ABF1CD
Output: x = 61600205

2 Examples of Field-Element-to-Octet-String Conversion. (See Section 4.3.3.)

1. **Input:** α = 94311, q =104729 (an odd prime).
   Output: S = 017067 (l=3).
2. **Input:** α = 11011011011101111001101111110110111110001 , q=2^{41}.
   Output: S = 01B6EF37EDF1 (l=6).

2 Examples of Octet-String-to-Field-Element Conversion. (See Section 4.3.4.)

1. **Input:** S = 01E74E (l=3), q = 224737 (an odd prime).
   Output: α = 124750.
2. **Input:** S = 0117B2939ACC (l=6), q=2^{41}.
   Output: α= 10001011110110010100100111001101011001100.

2 Examples of Field-Element-to-Integer Conversion. (See Section 4.3.5.)

1. **Input:** α = 136567, q = 287117, (an odd prime).
   Output: x = 136567.
2. **Input:** α = 11111111001000010011110000110011110101110 , q=2^{41}.
   Output: x = 2191548508078.

2 Examples of Point-to-Octet-String Conversion. (See Section 4.3.6.)

1. **Input:** p = 6277101735386680763835789423207666416083908700390324961279,
   and the curve E: $y^2=x^3+ax+b$ where:
   a = 6277101735386680763835789423207666416083908700390324961276
   b = 2455155546008943817740293915197451784769108058161191238065,
   and the point P=(x_p, y_p), where:

x_p = 6020462823756886567582134805875261119166989766368846848 18
y_p = 1740503322936220314048575522802194103640234889273866506 41.

Output: (compressed form)

PO = 03 188DA80E B03090F6 7CBF20EB 43A18800
F4FF0AFD 82FF1012.

Output: (uncompressed form)

PO = 04 188DA80E B03090F6 7CBF20EB 43A18800
F4FF0AFD 82FF1012 07192B95 FFC8DA78 631011ED
6B24CDD5 73F977A1 1E794811.

Output: (hybrid form)

PO = 07 188DA80E B03090F6 7CBF20EB 43A18800
F4FF0AFD 82FF1012 07192B95 FFC8DA78 631011ED
6B24CDD5 73F977A1 1E794811.

2. **Input:** $q = 2^{191}$,
and the irreducible polynomial which generates $F_{2^{191}}$:

f = 80000000 00000000 00000000 00000000 00000000
00000201,

and the curve E: $y^2+xy=x^3+ax^2+b$ over $F_{2^{191}}$, where:

a = 2866537B 67675263 6A68F565 54E12640 276B649E
F7526267,

b = 2E45EF57 1F00786F 67B0081B 9495A3D9 5462F5DE
0AA185EC,

and the point is $P=(x_p , y_p)$, where:

x_p = 0110110101100111101101011111000101000100011001 0000
0011011111001110001001111001010011001110101111 0110
0100001101010011100001101101001000100110111111 1001
01100100001001010111000011010101000001101,

y_p = 1110110010110111111001110011010000110011101100 11111
1100101011110001100110010100100110010111001110 0001
1101010001001000101110010100010010000011000111 0101
000001110111111001100000000000110001111 1011.

Output: (compressed form)

PO = 02 36B3DAF8 A23206F9 C4F299D7 B21A9C36
9137F2C8 4AE1AA0D.

Output: (uncompressed form)

PO = 04 36B3DAF8 A23206F9 C4F299D7 B21A9C36
9137F2C8 4AE1AA0D 765BE734 33B3F95E 332932E7
0EA245CA 2418EA0E F98018FB.

Output: (hybrid)

PO = 06 36B3DAF8 A23206F9 C4F299D7 B21A9C36
9137F2C8 4AE1AA0D 765BE734 33B3F95E 332932E7
0EA245CA 2418EA0E F98018FB.

2 Examples of Octet-String-to-Point Conversion. (See Section 4.3.7.)

1. **Input:** p = 6277101735386680763835789423207666416083908700390324961279,
and the curve E: $y^2=x^3+ax+b$ where:

a = 6277101735386680763835789423207666416083908700390324961276,
b = 2455155546008943817740293915197451784769108058161191238065,

and the octet string:

PO = 03 EEA2BAE7 E1497842 F2DE7769 CFE9C989
C072AD69 6F48034A.

Output: The point is $P=(x_p , y_p)$, where:

x_p = 5851329466723574623122023978072381191095567081251774399306,
y_p = 2487701625881228691269808880535093938601070911264778280469.

2. **Input:** $q=2^{191}$,

and the irreducible polynomial which generates $F_{2^{191}}$:

$f =$ 80000000 00000000 00000000 00000000 00000000 00000201,

and the curve E: $y^2+xy=x^3+ax^2+b$ over $F_{2^{191}}$, where:

$a =$ 40102877 4D7777C7 B7666D13 66EA4320 71274F89 FF01E718,

$b =$ 0620048D 28BCBD03 B6249C99 182B7C8C D19700C3 62C46A01,

and the octet string:

$PO =$ 02 3809B2B7 CC1B28CC 5A87926A AD83FD28 789E81E2 C9E3BF10.

Output: The point is $P=(x_p, y_p)$, where:

$x_p =$ 011100000000100110110010101101111100110000001101100101
000110011000101101010000111100100100110101010101011011
000001111111101001010000111100010011110100000001111100
0101100100111100011101111111000010000,

$y_p =$ 001011101000011010000111000011001100010011011010000010
100111100111101101111111000000001011101100000110110010
010000100111010001111100001110011110011011111010111101
10001000001101111101011001101000101 0.

J.2 Examples of ECDSA over the Field F_{2^m}

J.2.1 An Example with m = 191 (Trinomial Basis)

Elliptic Curve Domain Parameter Setup:

1. The field $F_{2^{191}}$ is generated by the irreducible polynomial:

$f =$ 80000000 00000000 00000000 00000000 00000000 00000201.

2. The curve is E : $y^2+xy=x^3+ax^2+b$ over $F_{2^{191}}$, where:

SEED = 4E13CA54 2744D696 E6768756 1517552F 279A8C84,

$a =$ 2866537B 67675263 6A68F565 54E12640 276B649E F7526267,

$b =$ 2E45EF57 1F00786F 67B0081B 9495A3D9 5462F5DE 0AA185EC.

3. Base point G (without point compression):

04 36B3DAF8 A23206F9 C4F299D7 B21A9C36 9137F2C8 4AE1AA0D 765BE734 33B3F95E 332932E7 0EA245CA 2418EA0E F98018FB.

G has prime order.

n = 1569275433846670190958947355803350458831205595451630533029.

h = 2.

Key Generation:

d = 1275552191113212300012030439187146164646146646466749494799.

$Q = dG = (x_Q, y_Q)$ (without point compression):

04 5DE37E75 6BD55D72 E3768CB3 96FFEB96 2614DEA4 CE28A2E7 55C0E0E0 2F5FB132 CAF416EF 85B229BB B8E13520 03125BA1.

Signature Generation:

M = "abc"

1. Message digesting.
   SHA-1 is applied to M to get:
   e = SHA-1(M) = 968236873715988614170569073515315707566766479517.
2. Elliptic curve computation.
   2.1. Select a k in the interval [1,n-1].
   k = 154272556521652398578923695626526526523567581194940404 0041.
   2.2. Compute $R = kG = (x_1, y_1)$:
   x_1 = 438E5A11 FB55E4C6 5471DCD4 9E266142 A3BDF2BF 9D5772D5,
   y_1 = 2AD603A0 5BD1D177 649F9167 E6F475B7 E2FF590C 85AF15DA.
   2.3. Convert x_1 to an integer $\overline{x}_1$:
   $\overline{x}_1$ = 165646981701154173431466964073025487882844318698669706 1077.
   2.4. Set $r = \overline{x}_1 \bmod n$.
   r = 8719438316487154335572228492690441999723759153506652 8048.
   2.5. $r \neq 0$, OK.
3. Modular computation.
   3.1. Compute $s = k^{-1}(e + dr) \bmod n$:
   s = 30899269196580494736154166454908589529215377702577206 3598.
   3.2. $s \neq 0$, OK.
4. Signature formatting.
   The signature is the two integers r and s:
   r = 8719438316487154335572228492690441999723759153506652 8048,
   s = 3089926919658049473615416645490858952921537770257720 63598.

Signature verification:

1. Message digesting.
   SHA-1 is applied to M′ to get:
   e' = SHA-1(M') = 968236873715988614170569073515315707566766479517.
2. Elliptic curve computation.
   2.1. r' is in interval [1, n-1], OK.
   2.2. s' is in interval [1, n-1], OK.
   2.3. Compute $c = (s')^{-1} \bmod n$:
   c = 95293366685086633156878228475480128988999208263538617 7703.
   2.4. Compute u1 = e'c mod n and u2 = r'c mod n:
   u_1 = 124888640715470785402243451608406250330179237436099440 0066,
   u_2 = 52701738097753401216822246601619984961197114165275346 4154.
   2.5. Compute $(x_1, y_1) = u_1G+u_2Q$:
   u_1G = 1A045B0C 26AF1735 9163E9B2 BF1AA57C 5475C320 78ABE159 53ECA58F AE7A4958 783E8173 CF1CA173 EAC47049 DCA02345,
   u_2Q = 015CF19F E8485BED 8520CA06 BD7FA967 A2CE0B30 4FFCF0F5 314770FA 4484962A EC673905 4A6652BC 07607D93 CAC79921.
   $u_1G + u_2Q = (x_1, y_1)$:
   x_1 = 438E5A11 FB55E4C6 5471DCD4 9E266142 A3BDF2BF 9D5772D5,
   y_1 = 2AD603A0 5BD1D177 649F9167 E6F475B7 E2FF590C 85AF15DA.

3. Signature check.

 3.1. Convert x_1 to an integer $\overline{x}_1$:

  $\overline{x}_1 =$ 1656469817011541734314669640730254878828443186986697061077.

 3.2. Compute $v = \overline{x}_1 \bmod n$:

  $v =$ 8719438316487154335572228492690441999723759153506652 8048.

 3.3. $v = r'$. OK.

J.2.2 An Example with m = 239 (Trinomial Basis)

Elliptic Curve Domain Parameter Setup:

1. The field $F_{2^{239}}$ is generated by the irreducible polynomial:

  $f =$ 8000 00000000 00000000 00000000 00000000 00000000 00000010 00000001.

2. The curve is $E : y^2+xy=x^3+ax^2+b$ over $F_{2^{239}}$, where:

  SEED = D34B9A4D 696E6768 75615175 CA71B920 BFEFB05D,

  a = 3201 0857077C 5431123A 46B80890 6756F543 423E8D27 87757812 5778AC76,

  $b =$ 7904 08F2EEDA F392B012 EDEFB339 2F30F432 7C0CA3F3 1FC383C4 22AA8C16.

3. Base point G (without point compression):

  04 57927098 FA932E7C 0A96D3FD 5B706EF7 E5F5C156 E16B7E7C 86038552 E91D61D8 EE5077C3 3FECF6F1 A16B268D E469C3C7 744EA9A9 71649FC7 A9616305.

 G has prime order:

  $n =$ 2208558830972980411979121875928648145578869937767132309367 15 041207411783.

  $h =$ 4.

Key Generation:

 $d =$ 1456427555219115346513212300075341203043918714616464614664 64 667494947990.

$Q = dG = (x_Q, y_Q)$ (without point compression):

 04 5894609C CECF9A92 533F630D E713A958 E96C97CC B8F5ABB5 A688A238 DEED6DC2 D9D0C94E BFB7D526 BA6A6176 4175B99C B6011E20 47F9F067 293F57F5.

Signature Generation:

M = “abc”

1. Message digesting.

 SHA-1 is applied to M to get:

  e = SHA-1(M) = 968236873715988614170569073515315707566766479517.

2. Elliptic curve computation.

 2.1. Select a k in the interval [1, n-1]:

  $k =$ 1712787255652165239672857892369562652652652356758 119494040400416702163 63.

 2.2. Compute $R = kG = (x_1, y_1)$:

  $x_1 =$ 6321 0D71EF6C 10157C0D 1053DFF9 3EB8F028 1E3F9DA2 DEB377A8 1BDAE8D5.

  $y_1 =$ 5EAF D217370E 12036519 CAD381A1 FC38234F 61870DB2 2C1E410A C1F183F0.

 2.3. Convert x_1 to an integer $\overline{x}_1$:

$\overline{x}_1$ = 6841639825023137355787549028176290563024791328169 81482452601000544626901.

2.4. Set $r = \overline{x}_1 \bmod n$:

r = 21596333210419611985018340039034612628818151486841 789642455876922391552.

2.5. $r \neq 0$, OK.

3. Modular computation.

3.1. Compute $s = k^{-1}(e + dr) \bmod n$:

s = 197030374000731686738334997654997227052849804072198 819102649413465737174.

3.2. $s \neq 0$, OK.

4. Signature formatting.

The signature is the two integers r and s:

r = 2159633321041961198501834003903461262881815148684178964245 5876922391552.

s = 1970303740007316867383349976549972270528498040721988191026 49413465737174.

Signature verification:

1. Message digesting.

SHA-1 is applied to M′ to get:

e = SHA-1(M′) = 968236873715988614170569073515315707566766479517.

2. Elliptic curve computation.

2.1. r' is in interval [1, n-1], OK.

2.2. s' is in interval [1, n-1], OK.

2.3. Compute $c = (s')^{-1} \bmod n$:

c = 431396620921664668890077637965697612042893607943599 26003383145535744433.

2.4. Compute $u_1 = ec \bmod n$ and $u_2 = r'c \bmod n$:

u_1 = 105375096144033333985559550644017212889091653305446 724555949472922658998,

u_2 = 215828469521640156896840216715465581571744240077746 044580128914744769962.

2.5. Compute $(x_1, y_1) = u_1G+u_2Q$:

u_1G = 12C9 F6F4C153 014AD6E5 04B3036B B47FFD7B D42B820A 00F84CA8 C5C89FCA,
78EA 1205C486 3D0CA5DE 16FF6324 51CAA41C EE66B628 DE80774C A4C23D05,

u_2Q = 5C9B A4416EAD A45057F6 4ADF29FE B2A6C8D5 7546CEA5 426551DB E4C43157,
39B0 51282C27 D6A55E19 CCEDA153 7C02D812 43E65DF8 309E58BC F5030C06.

$u_1G + u_2Q = (x_1, y_1)$:

x_1 = 6321 0D71EF6C 10157C0D 1053DFF9 3EB8F028 1E3F9DA2 DEB377A8 1BDAE8D5,

y_1 = 5EAF D217370E 12036519 CAD381A1 FC38234F 61870DB2 2C1E410A C1F183F0.

3. Signature check.

3.1. Convert x_1 to an integer $\overline{x}_1$:

$\overline{x}_1$ = 68416398250231373557875490281762905630247913281698148 2452601000544626901.

3.2. Compute $v = \overline{x}_1 \bmod n$:

v = 215963333210419611985018340039034612628818151486841789 642455876922391552.

3.3. $v = r'$. OK.

J.3 Examples of ECDSA over the Field F_p

J.3.1 An Example with a 192-bit Prime p

Elliptic Curve Domain Parameter Setup:

1. The field F_p is generated by the prime:
   p = 6277101735386680763835789423207666416083908700390324961279.
2. The curve is $E : y^2 = x^3+ax+b$ over F_p, where:
   SEED = 3045AE6F C8422F64 ED579528 D38120EA E12196D5,
   r = 3099D2BB BFCB2538 542DCD5F B078B6EF 5F3D6FE2 C745DE65,
   a = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFC,
   b = 64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1.
3. Base point G (with point compression):
   03 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD 82FF1012.
   G has prime order:
   n = 6277101735386680763835789423176059013767194773182842284081.
   h = 1.

Key Generation:

d = 651056770906015076056810763456358567190100156695615665659.

$Q = dG = (x_Q, y_Q)$ (with point compression):
02 62B12D60 690CDCF3 30BABAB6 E69763B4 71F994DD 702D16A5.

Signature Generation:

M = "abc"

1. Message digesting.
   SHA-1 is applied to M to get:
   e = SHA-1(M)= 968236873715988614170569073515315707566766479517.
2. Elliptic curve computation.
   2.1. Select a k in the interval [1, n-1]:
   k = 6140507067065001063065065565667405560006161556565665656654.
   2.2. Compute $R = kG = (x_1, y_1)$:
   x_1 = 88505238 0FF147B7 34C330C4 3D39B2C4 A89F29B0 F749FEAD,
   y_1 = 9CF9FA1C BEFEFB91 7747A3BB 29C072B9 289C2547 884FD835.
   2.3. Convert x_1 to an integer $\overline{x}_1$:
   $\overline{x}_1$ = 3342403536405981729393488334694600415596881826869351 677613.
   2.4. Set $r = \overline{x}_1 \bmod n$:
   r = 3342403536405981729393488334694600415596881826869351 677613.
   2.5. $r \neq 0$, OK.

3. Modular computation.
   3.1. Compute $s = k^{-1}(e + dr) \bmod n$.
   $s =$ 57358223288881552546838949978975719515685536428920 29 982342.
   3.2. $s \neq 0$, OK.
4. Signature formatting.
   The signature is the two integers r and s:
   r = 334240353640598172939348833469460041559688182686935167 7613.
   s = 57358223288881552546838949978975719515685536428920299 82342.

Signature verification:

1. Message digesting.
   SHA-1 is applied to M′ to get:
   e = SHA-1(M′) = 968236873715988614170569073515315707566766479517.
2. Elliptic curve computation.
   2.1. r' is in interval [1, n-1], OK.
   2.2. s' is in interval [1, n-1], OK.
   2.3. Compute $c = (s')^{-1} \bmod n$:
   $c =$ 3250964404472526825130516490452346217749189704049629042861.
   2.4. Compute $u_1 = ec \bmod n$ and $u_2 = r'c \bmod n$:
   $u_1 =$ 25636974091894341851947361345797310153664924963921 8 9760599,
   $u_2 =$ 62666438133486179671864777102357858491364063233387 8 2220568.
   2.5. Compute $(x_1, y_1) = u_1G+u_2Q$:
   $u_1G =$ DD9734E5 159253EB 0B09A049 2E12CBA8 7084C11B AC674D82 804F5FDC 638946FA 6660E851 E10542C1 134D4348 2956B50E,
   $u_2Q =$ 48893A3F 98EBA955 7660BE10 14BBD7D2 42326A1C DA7CF246 114A3118 867D4032 247416C4 A2BA3E83 076B6F8C B666667A,
   $u_1G + u_2Q = (x_1, y_1)$:
   $x_1 =$ 88505238 0FF147B7 34C330C4 3D39B2C4 A89F29B0 F749FEAD.
   $y_1 =$ 9CF9FA1C BEFEFB91 7747A3BB 29C072B9 289C2547 884FD835.
3. Signature check.
   3.1. Convert x_1 to an integer $\overline{x}_1$:
   $\overline{x}_1 =$ 33424035364059817293934883346946004155968818268693 51677613.
   3.2. Compute $v = \overline{x}_1 \bmod n$:
   $v =$ 33424035364059817293934883346946004155968818268693 51677613.
   3.3. $v =$ r'. OK.

J.3.2 An Example with a 239-bit Prime *p*

Elliptic Curve Domain Parameter Setup:

1. The field F_p is generated by the prime:
   $p =$ 883423532389192164791648750360308885314476597252960362792450860609699839.
2. The curve is $E : y^2 = x^3+ax+b$ over F_p, where:
   SEED = E43BB460 F0B80CC0 C0B07579 8E948060 F8321B7D,
   $r =$ 28B8 5EC1ECC1 9EFE769E B741A6D1 BA29476A A5A8F261 0957D6EF E78D3783,

$a =$ 7FFF FFFFFFFF FFFFFFFF FFFF7FFF FFFFFFFF 80000000 00007FFF FFFFFFFC,

$b =$ 6B01 6C3BDCF1 8941D0D6 54921475 CA71A9DB 2FB27D1D 37796185 C2942C0A.

3. Base point G (with point compression):

020FFA 963CDCA8 816CCC33 B8642BED F905C3D3 58573D3F 27FBBD3B 3CB9AAAF.

G has prime order:

$n =$ 88342353238919216479164875036030888480755034169162775227534 5424702807307.

$h =$ 1.

Key Generation:

$d =$ 87630010150710756750106613076167107835701067106778177671667167617 8726717.

$Q = d\,G = (x_Q, y_Q)$ (with point compression):

025B6D C53BC61A 2548FFB0 F671472D E6C9521A 9D2D2534 E65ABFCB D5FE0C70.

Signature Generation:

M = "abc"

1. Message digesting.

SHA-1 is applied to M to get:

e = SHA-1(M) = 968236873715988614170569073515315707566766479517.

2. Elliptic curve computation.

2.1. Select a k in the interval [1, n-1]:

$k =$ 70000001756905664665550578175715710757050157577570577957555565715 6756655.

2.2. Compute $R = kG = (x_1, y_1)$:

$x_1 =$ 2CB7 F36803EB B9C427C5 8D8265F1 1FC50847 47133078 FC279DE8 74FBECB0,

$y_1 =$ 20C0 8272B9E6 C92B518A 5AC5EB28 35BE0102 809D77E6 9304A6F7 C522B47B.

2.3. Convert x_1 to an integer $\overline{x}_1$:

$\overline{x}_1 =$ 30863614317516781149262254730066801885495937875853177814746205830 6432176.

2.4. Set $r = \overline{x}_1 \bmod n$:

$r =$ 30863614317516781149262254730066801885495937875853177814746205830 6432176.

2.5. $r \neq 0$, OK.

3. Modular computation.

3.1. Compute $s = k^{-1}(e + dr) \bmod n$.

$s =$ 32381355320979735770807877683125050593189105175500784278197850517 9448783.

3.2. $s \neq 0$, OK.

4. Signature formatting.

The signature is the two integers r and s.

$r =$ 30863614317516781149262254730066801885495937875853177814746 2058306432176,

$s =$ 32381355320979735770807877683125050593189105175500784278197 8505179448783.

Signature verification:

1. Message digesting.
   SHA-1 is applied to M′ to get:
   e = SHA-1(M′) = 968236873715988614170569073515315707566766479517.
2. Elliptic curve computation.
   2.1. r' is in interval [1, n-1], OK.
   2.2. s' is in interval [1, n-1], OK.
   2.3. Compute $c=(s')^{-1} \bmod n$:
   c = 831843418332978390463010021843350581892480848636408
   104706147767766249764.
   2.4. Compute $u_1 = ec \bmod n$ and $u_2 = r'c \bmod n$:
   u_1 = 124064965052014194622159338097387562954788117638383
   503089995672152118745,
   u_2 = 811363736140754465407544341268382421438687214093897
   850239246340491822539.
   2.5. Compute $(x_1, y_1) = u_1G+u_2Q$:
   u_1G = 64C4 29FAF03D C1707700 D2011D43 9836B4C7
   12DCFFD8 E4B7ED99 37F62D1F,
   6580 DE1A6ECE DFD78353 8C7C9D83 98BAE8B5
   A697EEFD 004AA596 60800F48,
   u_2Q = 3DCA 0CAFD86C 59DDD9FC 251A2073 9F698451
   68F5922E 523B7994 AFC92D9D,
   5532 B0A717E9 45EED3D8 AD1C26AB 37907E94
   2833CD22 AFFE63AC 1F5BC8FE,
   $u_1G + u_2Q = (x_1, y_1)$:
   x_1 = 2CB7 F36803EB B9C427C5 8D8265F1 1FC50847
   47133078 FC279DE8 74FBECB0,
   y_1 = 20C0 8272B9E6 C92B518A 5AC5EB28 35BE0102
   809D77E6 9304A6F7 C522B47B.
3. Signature check.
   3.1. Convert x_1 to an integer $\bar{x}_1$:
   $\bar{x}_1$ = 308636143175167811492622547300668018854959378758531
   778147462058306432176.
   3.2. Compute $v = \bar{x}_1 \bmod n$.
   v = 308636143175167811492622547300668018854959378758531
   778147462058306432176.
   3.3. $v = r'$. OK.

J.4 Sample Elliptic Curves over the Field F_{2^m}

This section presents sample curves over various fields F_{2^m} which may be used to ensure the correct implementation of this Standard.

The curves over the fields $F_{2^{163}}$, $F_{2^{191}}$, $F_{2^{239}}$ and $F_{2^{359}}$ were generated verifiably at random using the method described in Annex A.3.3.1.

The curves over the fields $F_{2^{176}}$, $F_{2^{208}}$, $F_{2^{272}}$, $F_{2^{304}}$ and $F_{2^{368}}$ were generated using the Weil method (see Note 7 in Annex A.3.2).

The curve over the field $F_{2^{431}}$ was generated at random (but not using the method described in Annex A.3.3.1).

J.4.1 3 Examples with m = 163

Elliptic Curve Domain Parameter Setup (pentanomial basis):

1. The field $F_{2^{163}}$ is generated by the irreducible pentanomial:

   f = 08 00000000 00000000 00000000 00000000 00000107.

2. The curve is $E : y^2+xy = x^3+ax^2+b$ over $F_{2^{163}}$.

Example 1:

SEED = D2C0FB15 760860DE F1EEF4D6 96E67687 56151754,
a = 07 2546B543 5234A422 E0789675 F432C894 35DE5242,
b = 00 C9517D06 D5240D3C FF38C74B 20B6CD4D 6F9DD4D9.
Base point G (with point compression):
0307 AF699895 46103D79 329FCC3D 74880F33 BBE803CB.
Order of G:
n = 04 00000000 00000000 0001E60F C8821CC7 4DAEAFC1,
h = 02.

Example 2:

SEED = 53814C05 0D44D696 E6768756 1517580C A4E29FFD,
a = 01 08B39E77 C4B108BE D981ED0E 890E117C 511CF072,
b = 06 67ACEB38 AF4E488C 407433FF AE4F1C81 1638DF20.
Base point G (with point compression):
0300 24266E4E B5106D0A 964D92C4 860E2671 DB9B6CC5.
Order of G:
n = 03 FFFFFFFF FFFFFFFF FFFDF64D E1151ADB B78F10A7,
h = 02.

Example 3:

SEED = 50CBF1D9 5CA94D69 6E676875 615175F1 6A36A3B8,
a = 07 A526C63D 3E25A256 A007699F 5447E32A E456B50E,
b = 03 F7061798 EB99E238 FD6F1BF9 5B48FEEB 4854252B.
Base point G (with point compression):
0202 F9F87B7C 574D0BDE CF8A22E6 524775F9 8CDEBDCB.
Order of G:
n = 03 FFFFFFFF FFFFFFFF FFFE1AEE 140F110A FF961309,
h = 02.

J.4.2 An Example with $m = 176$

Elliptic Curve Domain Parameter Setup (pentanomial basis):

1. The field $F_{2^{176}}$ is generated by the irreducible pentanomial:

   f = 010000 00000000 00000000 00000000 00000800 00000007.

2. The curve is $E : y^2+xy = x^3+ax^2+b$ over $F_{2^{176}}$.

Example:

SEED = No.
a = E4E6 DB299506 5C407D9D 39B8D096 7B96704B A8E9C90B,
b = 5DDA 470ABE64 14DE8EC1 33AE28E9 BBD7FCEC 0AE0FFF2.
Base point G (with point compression):
038D16 C2866798 B600F9F0 8BB4A8E8 60F3298C E04A5798.

Order of G:

$n =$ 01 00925373 97ECA4F6 145799D6 2B0A19CE 06FE26AD,
$h =$ FF6E.

J.4.3 5 Examples with $m = 191$

Elliptic Curve Domain Parameter Setup (trinomial basis):

1. The field $F_{2^{191}}$ is generated by the irreducible trinomial:

   $f =$ 80000000 00000000 00000000 00000000 00000000 00000201.

2. The curve is $E : y^2+xy = x^3+ax^2+b$ over $F_{2^{191}}$.

Example 1:

SEED = 4E13CA54 2744D696 E6768756 1517552F 279A8C84,
$a =$ 2866537B 67675263 6A68F565 54E12640 276B649E F7526267,
$b =$ 2E45EF57 1F00786F 67B0081B 9495A3D9 5462F5DE 0AA185EC.

Base point G (with point compression):

02 36B3DAF8 A23206F9 C4F299D7 B21A9C36 9137F2C8 4AE1AA0D.

Order of G:

$n =$ 40000000 00000000 00000000 04A20E90 C39067C8 93BBB9A5,
$h =$ 02.

Example 2:

SEED = 0871EF2F EF24D696 E6768756 151758BE E0D95C15,
$a =$ 40102877 4D7777C7 B7666D13 66EA4320 71274F89 FF01E718,
$b =$ 0620048D 28BCBD03 B6249C99 182B7C8C D19700C3 62C46A01.

Base point G (with point compression):

02 3809B2B7 CC1B28CC 5A87926A AD83FD28 789E81E2 C9E3BF10.

Order of G:

$n =$ 20000000 00000000 00000000 50508CB8 9F652824 E06B8173,
$h =$ 04.

Example 3:

SEED = E053512D C684D696 E6768756 15175067 AE786D1F,
$a =$ 6C010747 56099122 22105691 1C77D77E 77A777E7 E7E77FCB,
$b =$ 71FE1AF9 26CF8479 89EFEF8D B459F663 94D90F32 AD3F15E8.

Base point G (with point compression):

03 375D4CE2 4FDE4344 89DE8746 E7178601 5009E66E 38A926DD.

Order of G:

$n =$ 15555555 55555555 55555555 610C0B19 6812BFB6 288A3EA3,
$h =$ 06.

Elliptic Curve Domain Parameter Setup (optimal normal basis):

1. The field $F_{2^{191}}$ is generated by the irreducible polynomial:

   $f =$ D1010001 00000001 00000000 00000001 D1010001 00000001.

2. The curve is $E : y^2+xy = x^3+ax^2+b$ over $F_{2^{191}}$.

Example 4:

SEED = A399387E AE54D696 E6768756 151750E5 8B416D57,
a = 65903E04 E1E49242 53E26A3C 9AC28C75 8BD8184A 3FB680E8,
b = 54678621 B190CFCE 282ADE21 9D5B3A06 5E3F4B3F FDEBB29B.

Base point *G* (with point compression):
02 5A2C69A3 2E8638E5 1CCEFAAD 05350A97 8457CB5F B6DF994A.

Order of *G*:
n = 40000000 00000000 00000000 9CF2D6E3 901DAC4C 32EEC65D,
h = 02.

Example 5:

SEED = 2D88F7BC 545794D6 96E67687 56151759 73391555,
a = 25F8D06C 97C82253 6D469CD5 170CDD7B B9F500BD 6DB110FB,
b = 75FF570E 35CA94FB 3780C261 9D081C17 AA59FBD5 E591C1C4.

Base point *G* (with point compression):
03 2A16910E 8F6C4B19 9BE24213 857ABC9C 992EDFB2 471F3C68.

Order of *G*:
n = 0FFFFFFF FFFFFFFF FFFFFFFF EEB354B7 270B2992 B7818627,
h = 08.

J.4.4 An Example with *m* = 208

Elliptic Curve Domain Parameter Setup (pentanomial basis):

1. The field $F_{2^{208}}$ is generated by the irreducible pentanomial:
   f = 010000 00000000 00000000 00000000 00080000 00000000 00000007.
2. The curve is $E : y^2+xy = x^3+ax^2+b$ over $F_{2^{208}}$.

Example 1:

SEED = No,
a = 0000 00000000 00000000 00000000 00000000 00000000 00000000,
b = C861 9ED45A62 E6212E11 60349E2B FA844439 FAFC2A3F D1638F9E.

Base point *G* (with point compression):
0289FD FBE4ABE1 93DF9559 ECF07AC0 CE78554E 2784EB8C 1ED1A57A.

Order of *G*:
n = 01 01BAF95C 9723C57B 6C21DA2E FF2D5ED5 88BDD571 7E212F9D,
h = FE48.

J.4.5 5 Examples with *m* = 239

Elliptic Curve Domain Parameter Setup (trinomial basis):

1. The field $F_{2^{239}}$ is generated by the irreducible trinomial:

f = 8000 00000000 00000000 00000000 00000000 00000000 00000010 00000001.

2. The curve is $E : y^2+xy = x^3+ax^2+b$ over $F_{2^{239}}$.

Example 1:

SEED = D34B9A4D 696E6768 75615175 CA71B920 BFEFB05D,
a = 3201 0857077C 5431123A 46B80890 6756F543 423E8D27 87757812 5778AC76,
b = 7904 08F2EEDA F392B012 EDEFB339 2F30F432 7C0CA3F3 1FC383C4 22AA8C16.
Base point G (with point compression):
025792 7098FA93 2E7C0A96 D3FD5B70 6EF7E5F5 C156E16B 7E7C8603 8552E91D.
Order of G:
n = 2000 00000000 00000000 00000000 000F4D42 FFE1492A 4993F1CA D666E447,
h = 04.

Example 2:

SEED = 2AA6982F DFA4D696 E6768756 15175D26 6727277D,
a = 4230 017757A7 67FAE423 98569B74 6325D453 13AF0766 266479B7 5654E65F,
b = 5037 EA654196 CFF0CD82 B2C14A2F CF2E3FF8 775285B5 45722F03 EACDB74B.
Base point G (with point compression):
0228F9 D04E9000 69C8DC47 A08534FE 76D2B900 B7D7EF31 F5709F20 0C4CA205.
Order of G:
n = 1555 55555555 55555555 55555555 553C6F28 85259C31 E3FCDF15 4624522D,
h = 06.

Example 3:

SEED = 9E076F4D 696E6768 75615175 E11E9FDD 77F92041,
a = 0123 8774666A 67766D66 76F778E6 76B66999 176666E6 87666D87 66C66A9F,
b = 6A94 1977BA9F 6A435199 ACFC5106 7ED587F5 19C5ECB5 41B8E441 11DE1D40.
Base point G (with point compression):
0370F6 E9D04D28 9C4E8991 3CE3530B FDE90397 7D42B146 D539BF1B DE4E9C92.
Order of G:
n = 0CCC CCCCCCCC CCCCCCCC CCCCCCCC CCAC4912 D2D9DF90 3EF9888B 8A0E4CFF,
h = 0A.

Elliptic Curve Domain Parameter Setup (optimal normal basis):

1. The field $F_{2^{239}}$ is generated by the irreducible polynomial:

$f =$ D101 0001D101 00000000 0001D101 00000000
00000000 00000000 0001D101.

2. The curve is $E : y^2+xy = x^3+ax^2+b$ over $F_{2^{239}}$.

Example 4:

SEED = F851638C FA4D696E 67687561 51755651 3841BFAC,
$a =$ 182D D45F5D47 0239B898 3FEA47B8 B292641C 57F9BF84
BAECDE8B B3ADCE30,
$b =$ 147A 9C1D4C2C E9BE5D34 EC02797F 76667EBA D5A3F93F
A2A524BF DE91EF28.

Base point G (with point compression):
034912 AD657F1D 1C6B32ED B9942C95 E226B06F B012CD40 FDEA0D72
197C8104.

Order of G:
$n =$ 2000 00000000 00000000 00000000 00474F7E 69F42FE4
30931D0B 455AAE8B,
$h =$ 04.

Example 5:

SEED = 2C04F44D 696E6768 75615175 C586B41F 6CA150C9,
$a =$ 1ECF 1B9D28D8 017505E1 7475D3DF 2982E243 CA5CB5E9
F94A3F36 124A486E,
$b =$ 3EE2 57250D1A 2E66CEF2 3AA0F25B 12388DE8 A10FF955
4F90AFBA A9A08B6D.

Base point G (with point compression):
021932 79FC543E 9F5F7119 189785B9 C60B249B E4820BAF 6C24BDFA
2813F8B8.

Order of G:
$n =$ 1555 55555555 55555555 55555555 558CF77A 5D0589D2
A9340D96 3B7AD703,
$h =$ 06.

J.4.6 An Example with $m = 272$

Elliptic Curve Domain Parameter Setup (pentanomial basis):

1. The field $F_{2^{272}}$ is generated by the irreducible pentanomial:

$f =$ 010000 00000000 00000000 00000000 00000000 00000000
00000000 01000000 0000000B.

2. The curve is $E : y^2+xy = x^3+ax^2+b$ over $F_{2^{272}}$.

Example 1:

SEED = No,
$a =$ 91A0 91F03B5F BA4AB2CC F49C4EDD 220FB028 712D42BE
752B2C40 094DBACD B586FB20,
$b =$ 7167 EFC92BB2 E3CE7C8A AAFF34E1 2A9C5570 03D7C73A
6FAF003F 99F6CC84 82E540F7.

Base point G (with point compression):
026108 BABB2CEE BCF78705 8A056CBE 0CFE622D 7723A289 E08A07AE
13EF0D10 D171DD8D.

Order of G:

n = 01 00FAF513 54E0E39E 4892DF6E 319C72C8 161603FA
45AA7B99 8A167B8F 1E629521,
h = FF06.

J.4.7 An Example with *m* = 304

Elliptic Curve Domain Parameter Setup (pentanomial basis):

1. The field $F_{2^{304}}$ is generated by the irreducible pentanomial:
   f = 010000 00000000 00000000 00000000 00000000 00000000
   00000000 00000000 00000000 00000807.
2. The curve is $E : y^2+xy = x^3+ax^2+b$ over $F_{2^{304}}$.

Example 1:

SEED = No,
a = FD0D 693149A1 18F651E6 DCE68020 85377E5F 882D1B51
0B441600 74C12880 78365A03 96C8E681,
b = BDDB 97E555A5 0A908E43 B01C798E A5DAA678 8F1EA279
4EFCF571 66B8C140 39601E55 827340BE.
Base point *G* (with point compression):
02197B 07845E9B E2D96ADB 0F5F3C7F 2CFFBD7A 3EB8B6FE C35C7FD6
7F26DDF6 285A644F 740A2614.
Order of *G*:
n = 01 01D55657 2AABAC80 0101D556 572AABAC 8001022D
5C91DD17 3F8FB561 DA689916 4443051D,
h = FE2E.

J.4.8 An Example with *m* = 359

Elliptic Curve Domain Parameter Setup (trinomial basis):

1. The field $F_{2^{359}}$ is generated by the irreducible trinomial:
   f = 80 00000000 00000000 00000000 00000000
   00000000 00000000 00000000 00000000 00000010
   00000000 00000001.
2. The curve is $E : y^2+xy = x^3+ax^2+b$ over $F_{2^{359}}$.

Example 1:

SEED = 2B354920 B724D696 E6768756 1517585B A1332DC6,
a = 56 67676A65 4B20754F 356EA920 17D94656 7C466755
56F19556 A04616B5 67D223A5 E05656FB 549016A9
6656A557,
b = 24 72E2D019 7C49363F 1FE7F5B6 DB075D52 B6947D13
5D8CA445 805D39BC 34562608 9687742B 6329E706
80231988.
Base point *G* (with point compression):
033C 258EF304 7767E7ED E0F1FDAA 79DAEE38 41366A13
2E163ACE D4ED2401 DF9C6BDC DE98E8E7 07C07A22 39B1B097.
Order of *G*:

n = 01 AF286BCA 1AF286BC A1AF286B CA1AF286 BCA1AF28
6BC9FB8F 6B85C556 892C20A7 EB964FE7 719E74F4
90758D3B,
h = 4C.

J.4.9 An Example with *m* = 368

Elliptic Curve Domain Parameter Setup (pentanomial basis):

1. The field $F_{2^{368}}$ is generated by the irreducible pentanomial:
   f = 010000 00000000 00000000 00000000 00000000 00000000
   00000000 00000000 00000000 00200000 00000000
   00000007.
2. The curve is $E : y^2+xy = x^3+ax^2+b$ over $F_{2^{368}}$.

Example 1:

SEED = No,
a = E0D2 EE250952 06F5E2A4 F9ED229F 1F256E79 A0E2B455
970D8D0D 865BD947 78C576D6 2F0AB751 9CCD2A1A
906AE30D,
b = FC12 17D4320A 90452C76 0A58EDCD 30C8DD06 9B3C3445
3837A34E D50CB549 17E1C211 2D84D164 F444F8F7
4786046A.

Base point *G* (with point compression):

021085 E2755381 DCCCE3C1 557AFA10 C2F0C0C2 825646C5 B34A394C
BCFA8BC1 6B22E7E7 89E927BE 216F02E1 FB136A5F.

Order of *G*:

n = 01 0090512D A9AF72B0 8349D98A 5DD4C7B0 532ECA51
CE03E2D1 0F3B7AC5 79BD87E9 09AE40A6 F131E9CF
CE5BD967,
h = FF70.

J.4.10 An Example with *m* = 431

Elliptic Curve Domain Parameter Setup (trinomial basis):

1. The field $F_{2^{431}}$ is generated by the irreducible trinomial:
   f = 8000 00000000 00000000 00000000 00000000
   00000000 00000000 00000000 00000000 00000000
   01000000 00000000 00000000 00000001.
2. The curve is $E: y^2+xy = x^3+ax^2+b$ over $F_{2^{431}}$.

Example 1:

SEED = No.
a = 1A82 7EF00DD6 FC0E234C AF046C6A 5D8A8539 5B236CC4
AD2CF32A 0CADBDC9 DDF620B0 EB9906D0 957F6C6F
EACD6154 68DF104D E296CD8F,
b = 10D9 B4A3D904 7D8B1543 59ABFB1B 7F5485B0 4CEB8682
37DDC9DE DA982A67 9A5A919B 626D4E50 A8DD731B
107A9962 381FB5D8 07BF2618.

Base point G (with point compression):
02120F C05D3C67 A99DE161 D2F40926 22FECA70 1BE4F50F 4758714E
8A87BBF2 A658EF8C 21E7C5EF E965361F 6C2999C0 C247B0DB
D70CE6B7.
Order of G:
n = 03 40340340 34034034 03403403 40340340 34034034
03403403 40340323 C313FAB5 0589703B 5EC68D35
87FEC60D 161CC149 C1AD4A91,
h = 2760.

J.5 Sample Elliptic Curves over the Field F_p

This section presents sample curves over 192-bit, 239-bit and 256-bit prime fields F_p which may be used to ensure the correct implementation of this Standard.
The curves were generated verifiably at random using the method described in Annex A.3.3.2.

J.5.1 3 Examples with a 192-bit Prime

Elliptic Curve Domain Parameter Setup:

1. The field F_p is generated by the prime:
   p = 6277101735386680763835789423207666416083908700390324961279.
2. The curve is $E : y^2 = x^3+ax+b$ over F_p.

Example 1:

SEED = 3045AE6F C8422F64 ED579528 D38120EA E12196D5,
r = 3099D2BB BFCB2538 542DCD5F B078B6EF 5F3D6FE2 C745DE65,
a = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFC,
b = 64210519 E59C80E7 0FA7E9AB 72243049 FEB8DEEC C146B9B1.
Base point G (with point compression):
03 188DA80E B03090F6 7CBF20EB 43A18800 F4FF0AFD
82FF1012.
Order of G:
n = FFFFFFFF FFFFFFFF FFFFFFFF 99DEF836 146BC9B1 B4D22831,
h = 01.

Example 2:

SEED = 31A92EE2 029FD10D 901B113E 990710F0 D21AC6B6,
r = 15038D1D 2E1CAFEE 0299F301 1C1DC75B 3C2A86E1 35DB1E6B,
a = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFC,
b = CC22D6DF B95C6B25 E49C0D63 64A4E598 0C393AA2 1668D953.
Base point G (with point compression):
03 EEA2BAE7 E1497842 F2DE7769 CFE9C989 C072AD69
6F48034A.
Order of G:
n = FFFFFFFF FFFFFFFF FFFFFFFE 5FB1A724 DC804186 48D8DD31,
h = 01.

Example 3:

SEED = C4696844 35DEB378 C4B65CA9 591E2A57 63059A2E,
r = 25191F95 024D8395 46D9A337 5639A996 7D52F137 3BC4EE0B,
a = FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFF FFFFFFFC,

b = 22123DC2 395A05CA A7423DAE CCC94760 A7D46225 6BD56916.

Base point *G* (with point compression):

02 7D297781 00C65A1D A1783716 588DCE2B 8B4AEE8E 228F1896.

Order of *G*:

n = FFFFFFFF FFFFFFFF FFFFFFFF 7A62D031 C83F4294 F640EC13,

h = 01.

J.5.2 3 Examples with a 239-bit Prime

Elliptic Curve Domain Parameter Setup:

1. The field F_p is generated by the prime:

   p = 883423532389192164791648750360308885314476597252960362 792450860609699839.

2. The curve is E: $y^2 = x^3+ax+b$ over F_p.

Example 1:

SEED = E43BB460 F0B80CC0 C0B07579 8E948060 F8321B7D,

r = 28B8 5EC1ECC1 9EFE769E B741A6D1 BA29476A A5A8F261 0957D6EF E78D3783,

a = 7FFF FFFFFFFF FFFFFFFF FFFF7FFF FFFFFFFF 80000000 00007FFF FFFFFFFC,

b = 6B01 6C3BDCF1 8941D0D6 54921475 CA71A9DB 2FB27D1D 37796185 C2942C0A.

Base point *G* (with point compression):

020FFA 963CDCA8 816CCC33 B8642BED F905C3D3 58573D3F 27FBBD3B 3CB9AAAF.

Order of *G*:

n = 7FFF FFFFFFFF FFFFFFFF FFFF7FFF FF9E5E9A 9F5D9071 FBD15226 88909D0B,

h = 01.

Example 2:

SEED= E8B40116 04095303 CA3B8099 982BE09F CB9AE616,

r = 1DF4 91E44E7C CAF4D1EA D8A6B90D AE09E0D3 3F2C6CFE 7A6BA76E 86713D52,

a = 7FFF FFFFFFFF FFFFFFFF FFFF7FFF FFFFFFFF 80000000 00007FFF FFFFFFFC,

b = 617F AB683257 6CBBFED5 0D99F024 9C3FEE58 B94BA003 8C7AE84C 8C832F2C.

Base point *G* (with point compression):

0238AF 09D98727 705120C9 21BB5E9E 26296A3C DCF2F357 57A0EAFD 87B830E7.

Order of *G*:

n = 7FFF FFFFFFFF FFFFFFFF FFFF8000 00CFA7E8 594377D4 14C03821 BC582063,

h = 01.

Example 3:

SEED = 7D737416 8FFE3471 B60A8576 86A19475 D3BFA2FF,

r = 3A4F 9DC9A6CE FD5F9D11 93B9C996 8C202430 003C2819
C2E49861 8DC58330,
a = 7FFF FFFFFFFF FFFFFFFF FFFF7FFF FFFFFFFF 80000000
00007FFF FFFFFFFC,
b = 2557 05FA2A30 6654B1F4 CB03D6A7 50A30C25 0102D498
8717D9BA 15AB6D3E.

Base point G (with point compression):
036768 AE8E18BB 92CFCF00 5C949AA2 C6D94853 D0E660BB F854B1C9
505FE95A.

Order of G:
n = 7FFF FFFFFFFF FFFFFFFF FFFF7FFF FF975DEB 41B3A605
7C3C4321 46526551,
h = 01.

J.5.3 An Example with a 256-bit Prime

Elliptic Curve Domain Parameter Setup:

1. The field F_p is generated by the prime:
p = 115792089210356248762697446949407573530086143415290314195533631308867097853951.
2. The curve is $E : y^2 = x^3+ax+b$ over F_p.

Example 1:

SEED = C49D3608 86E70493 6A6678E1 139D26B7 819F7E90,
r = 7EFBA166 2985BE94 03CB055C 75D4F7E0 CE8D84A9 C5114ABC
AF317768 0104FA0D,
a = FFFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF
FFFFFFFF FFFFFFFC,
b = 5AC635D8 AA3A93E7 B3EBBD55 769886BC 651D06B0 CC53B0F6
3BCE3C3E 27D2604B.

Base point $G = (x, y)$ (with point compression):
03 6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81
2DEB33A0 F4A13945 D898C296.

Order of G:
n = FFFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84
F3B9CAC2 FC632551,
h = 01.

Annex K
(informative)
References

Elliptic curve cryptosystems were first proposed in 1985 independently by Neil Koblitz [23] and Victor Miller [30]. Since then, much research has been done towards improving the efficiency of these systems and evaluating their security. For a summary of this work, consult [28]. A description of a hardware implementation of an elliptic curve cryptosystem can be found in [9].

Three references on the theory of finite fields are the books of McEliece [27], Lidl and Niederreiter [26] and Jungnickel [21]. Lidl and Niederreiter's book [26] contains introductory material on polynomial and normal bases. The article [8] discusses methods which efficiently perform arithmetic operations in finite fields of characteristic 2. A hardware implementation of arithmetic in such fields which exploits the properties of optimal normal bases is described in [10].

The NIST Digital Signature Algorithm (DSA) is described in [3] and [32]. The Secure Hash Algorithm (SHA-1) is described in [4] and [31]. Abstract Syntax Notation One (ASN.1) is described in [15]; see also [16], [17], [18], [19] and [20]. Basic Encoding Rules (BER) and Distinguished Encoding Rules (DER) are described in [19].

[1] ANSI X3.92-1981, Data Encryption Algorithm.

[2] ANSI X9.9-1986 (revised), Financial Institution Message Authentication (Wholesale).

[3] ANSI X9.30-1995, Part 1: Public key cryptography using irreversible algorithms for the financial services industry: The Digital Signature Algorithm (Revised).

[4] ANSI X9.30-1993, Part 2: Public key cryptography using irreversible algorithms for the financial services industry: The Secure Hash Algorithm 1 (SHA-1) (Revised).

[5] ANSI X9.57-1997: Certificate Management.

[6] ANSI X9.63-199x: Elliptic curve key agreement and transport protocols, draft.

[7] ANSI NWI: Prime number generation, draft.

[8] G. AGNEW, T. BETH, R. MULLIN AND S. VANSTONE, Arithmetic operations in GF(2^m), Journal of Cryptology, 6 (1993), 3-13.

[9] G. AGNEW, R. MULLIN AND S. VANSTONE, An implementation of elliptic curve cryptosystems over $F_{2^{155}}$, IEEE Journal on Selected Areas in Communications, 11 (1993), 804-813.

[10] G. AGNEW, R. MULLIN, I. ONYSZCHUK AND S. VANSTONE, An implementation for a fast public-key cryptosystem, Journal of Cryptology, 3 (1991), 63-79.

[11] M. BLAZE, W. DIFFIE, R. RIVEST, B. SCHNEIER, T. SHIMOMURA, E. THOMPSON, AND M. WIENER, Minimal key lengths for symmetric ciphers to provide adequate commercial security, January 1996.

[12] E. BRICKELL, D. GORDON, K. MCCURLEY AND D. WILSON, Fast Exponentiation with precomputation, Advances in Cryptology - EUROCRYPT '92 Lecture Notes in Computer Science, 658 (1993), Springer-Verlag, 200-207.

[13] T. ELGAMAL, A public key cryptosystem and a signature scheme based on discrete logarithms, IEEE Transactions on Information Theory, 31 (1985), 469-472.

[14] R. GALLANT, R. LAMBERT, AND S. VANSTONE, Improving the parallelized Pollard lambda search on binary anomalous curves, to appear in Mathematics of Computation.

[15] ITU-T Recommendation X.680, Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation (equivalent to ISO/IEC 8824-1).

[16] ITU-T Recommendation X.681, Information Technology - Abstract Syntax Notation One (ASN.1): Information Object Specification (equivalent to ISO/IEC 8824-2).

[17] ITU-T Recommendation X.682, Information Technology - Abstract Syntax Notation One (ASN.1): Constraint Specification (equivalent to ISO/IEC 8824-3).

[18] ITU-T Recommendation X.683, Information Technology – Abstract Syntax Notation One (ASN.1): Parametrization of ASN.1 Specifications (equivalent to ISO/IEC 8824-4).

[19] ITU-T Recommendation X.690, Information Technology - ASN.1 Encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER) (equivalent to ISO/IEC 8825-1).

[20] ITU-T Recommendation X.691, Information Technology - ASN.1 Encoding Rules: Specification of Packed Encoding Rules (PER) (equivalent to ISO/IEC 8825-2).

[21] D. JUNGNICKEL, Finite Fields: Structure and Arithmetics, B.I.-Wissenschaftsverlag, Mannheim, 1993.

[22] D. KNUTH, The Art of Computer Programming, volume 2, 2nd edition, 1981.

[23] N. KOBLITZ, Elliptic curve cryptosystems, Mathematics of Computation, 48 (1987), 203-209.

[24] R. LERCIER, Finding good random elliptic curves for cryptosystems defined over F_{2^n}, Advances in Cryptography - EUROCRYPT '97, Lecture Notes in Computer Science, 1233 (1997), Springer-Verlag, 379-392.

[25] R. LERCIER AND F. MORAIN, Counting the number of points on elliptic curves over finite fields: strategies and performances, Advances in Cryptology - EUROCRYPT '95, Lecture Notes in Computer Science, 921 (1995), Springer-Verlag, 79-94.

[26] R. LIDL AND H. NIEDERREITER, Finite Fields, Cambridge University Press, 1987.

[27] R.J. MCELIECE, Finite Fields for Computer Scientists and Engineers, Kluwer Academic Publishers, 1987.

[28] A. MENEZES, Elliptic Curve Public Key Cryptosystems, Kluwer Academic Publishers, 1993.

[29] A. MENEZES, T. OKAMOTO AND S. VANSTONE, Reducing elliptic curve logarithms to logarithms in a finite field, IEEE Transactions on Information Theory, 39 (1993), 1639-1646.

[30] V. MILLER, Uses of elliptic curves in cryptography, Advances in Cryptology - CRYPTO '85, Lecture Notes in Computer Science, 218 (1986), Springer-Verlag, 417-426.

[31] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY, Secure Hash Standard (SHS), FIPS Publication 180, May 1993.

[32] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY, Digital Signature Standard, FIPS Publication 186, 1993.

[33] A. ODLYZKO, The Future of Integer Factorization, Cryptobytes, volume 1, number 2, summer 1995, 5-12.

[34] P. VAN OORSCHOT AND M. WIENER, Parallel Collision Search With Application To Hash Functions And Discrete Logarithms, in Proceedings of the 2nd ACM Conference on Computer and Communications Security, Fairfax, Virginia, November 2-4, 1994, 210-218.

[35] J. POLLARD, Monte Carlo methods for index computation mod p, Mathematics of Computation, 32 (1978), 918-924.

[36] R. SCHOOF, Elliptic curves over finite fields and the computation of square roots mod p, Mathematics of Computation, 44 (1985), 483-494.

[37] T. SATOH AND K. ARAKI, Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves, preprint, 1997.

[38] N. SMART, The discrete logarithm problem on elliptic curves of trace one, to appear in Journal of Cryptology.

[39] S. VAUDENAY, Hidden collisions on DSS, Advances in Cryptology - CRYPTO '96, Lecture Notes in Computer Science, 1109 (1996), Springer-Verlag, 83-88.

[40] M. WIENER AND R. ZUCCHERATO, Fast attacks on elliptic curve cryptosystems, to appear in Fifth Annual Workshop on Selected Areas in Cryptography – SAC '98, Lecture Notes in Computer Science, Springer-Verlag.