Accredited Standards Committee X9 **January 8, 1999**
Title: X9-Financial Services
Accredited by the
American National Standards Institute

Working Draft
AMERICAN NATIONAL STANDARD
X9.63-199x
Public Key Cryptography For The Financial Services Industry:
Key Agreement and Key Transport Using Elliptic Curve Cryptography

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

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 disclosure, alteration, substitution, or destruction of data. This risk is compounded by interconnected networks, and the increased number and sophistication of malicious adversaries. Electronically communicated data may be secured through the use of symmetrically-keyed encryption algorithms (e.g. ANSI X9.52, Triple-DEA) keyed via public-key cryptography-based key management techniques.

This standard, X9.63-199x, *Public Key Cryptography For The Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*, defines a suite of mechanisms for use in key management applications. These mechanisms are based on the elliptic curve analogue of the Diffie-Hellman key agreement mechanism. Because the mechanisms are based on the same fundamental mathematics as the Elliptic Curve Digital Signature Algorithm (ECDSA) (see [8]), additional efficiencies and functionality may be obtained by combining these and other cryptographic techniques.

While the techniques specified in this standard are designed to facilitate the secure establishment of cryptographic data for the keying of symmetrically-keyed algorithms (e.g. DEA, TDEA), 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.

The user's attention is called to the possibility that compliance with this standard may require use of an invention covered by patent rights. By publication of this standard, no position is taken with respect to the validity of potential claims or of any patent rights in connection therewith. The patent holders have, however, filed a statement of willingness to grant a license under these rights on reasonable and nondiscriminatory terms and conditions to applicants desiring to obtain such a license. Details may be obtained from the X9 Secretariat,

Suggestions for the improvement or revision of this standard are welcome. They should be sent to the X9 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

X9.63-1998, Public Key Cryptography For The Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography

1 Scope

This Standard defines key establishment schemes which employ asymmetric cryptographic techniques. The arithmetic operations involved in the operation of the schemes take place in the algebraic structure of an elliptic curve over a finite field.
Both key agreement and key transport schemes are specified.
The schemes may be used by two parties to compute shared keying data which may then be used by symmetric schemes to provide cryptographic services like data confidentiality and data integrity.
Supporting mathematical definitions and examples are also provided.

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.)
associate value
Given an elliptic curve point and corresponding elliptic curve parameters, the associate value is an integer associated with the point. (See Section 5.6.1.)
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.
auxiliary function
An auxiliary function is a transformation that forms part of a cryptographic scheme but is auxiliary rather than central to the goal of the scheme.
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.

challenge
Data sent from *U* to *V* during an execution of a protocol which in part determines *V*'s response. In this Standard, challenges will be bit strings at least 80 bits in length.
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.

cryptographic protocol
A cryptographic scheme in which an ordered sequence of sets of data is passed between two entities during an ordinary operation of the scheme.
cryptographic scheme
A cryptographic scheme consists of an unambigous specification of a set of transformations capable of providing a cryptographic service when properly implemented and maintained.
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, $E(F_q) = \{kP: 0 \leq k \leq n\text{-}1\}$.
data confidentiality
The assurance provided to entity *U* that data is unintelligible to entities other than *U* and *V*.
data integrity
The assurance provided to entity *U* that data has not been modified by entities other than *U* and *V*.
data origin authentication
The assurance provided to entity *U* that data is from *V*.
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.

EC
Elliptic curve.
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 $1 \leq d \leq n\text{-}1$.
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*.
encryption scheme
An encryption scheme is a cryptographic scheme capable of providing data confidentiality.
entity
A party involved in the operation of a cryptographic system.
entity authentication
The assurance provided to entity *U* that entity *U* has been involved in a real-time communication with entity *V*.
ephemeral
Ephemeral data is relatively short-lived. In this Standard ephemeral data is data specific to a particular execution of a cryptographic scheme.
explicit key authentication
The assurance provided to entity *U* that only entities *U* and *V* are possibly capable of computing the session key and that the entities *U* and *V* are actually capable of computing the session key.
flow
A flow in a protocol is a set of data sent from *U* to *V* or received by *U* from *V* at a particular stage of an operation of the protocol.
forward secrecy
The assurance provided to entity *U* that the session key established between entities *U* and *V* will not be compromised by the compromise of either entity's static secret key in the future. Also known as perfect forward secrecy.
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 bit string.
hybrid form
Octet string representation for both the compressed and uncompressed forms of an elliptic curve point. (See Section 4.3.6.)
implicit key authentication
The assurance provided to entity *U* that only entities *U* and *V* are possibly capable of computing the session key.
initiator
An entity involved in an operation of a protocol that sends the first flow of the protocol.

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.
key agreement scheme
A key agreement scheme is a key establishment scheme in which the keying data established is a function of contributions provided by both entities in such a way that neither party can predetermine the value of the keying data.
key-compromise impersonation resilience
The assurance provided to entity *U* during an execution of a key establishment scheme that the compromise of *U*'s static private key has not enabled the impersonation of *V* to *U*.
key confirmation
The addition of flows to a key establishment scheme providing implicit key authentication so that explicit key authentication is provided.
key derivation function
A key derivation function is a function which takes as input a shared secret value and outputs keying data suitable for later cryptographic use.
key establishment schemes
A key establishment scheme is a cryptographic scheme which establishes keying data suitable for subsequent cryptographic use by cryptographic schemes to its legitimate users. Key agreement schemes and key transport schemes are types of key establishment schemes.
keying data
Data suitable for use as cryptographic keys.
keying material
The data (e.g., keys, certificates and initialization vectors) necessary to establish and maintain cryptographic keying relationships.
key transport schemes
A key transport scheme is a key establishment scheme in which the keying data established is determined entirely by one entity.
known-key security
The assurance provided to entity *U* that the session key established by an execution of a key establishment scheme will not be compromised by the compromise of other session keys.
MAC scheme
A MAC scheme is a cryptographic scheme capable of providing data origin authentication and data integrity.
non-repudiation
The assurance provided to entity *U* that *U* is able to prove to a third party that data is from *V*.
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 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.
protocol
See cryptographic protocol.
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}.
responder
An entity involved in an operation of a protocol that does not send the first flow of the protocol.
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*.
SEED
Random value input into a pseudo-random bit generator (PRBG) algorithm.
session key
A key established by a key establishment scheme.
shared secret value
An intermediate value in a key establishment scheme from which keying data is derived.
signature scheme
A signature scheme is a cryptographic scheme capable of providing data origin authentication, data integrity, and non-repudiation.
static
Static data is relatively long-lived. In this Standard static data is data common to a number of executions of a cryptographic scheme.
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.
symmetric cryptographic scheme
A cryptographic scheme in which each transformation is controlled by the same key.
trinomial
A polynomial of the form $x^m + x^k + 1$, where $1 \leq k \leq m-1$.
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.)
unknown key-share resilience
The assurance provided to entity *U* that, if entities *U* and *V* share a session key, *V* does not mistakenly believe the session key is shared with an entity other than *U*.
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.
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

$[X]$	Indicates that the inclusion of the bit string or octet string X is optional.
$[x, y]$	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, $23 \bmod 7 = 2$.
$x \equiv y \pmod n$	x is congruent to y modulo n. That is, $(x \bmod n) = (y \bmod n)$.
a, b	Elements of F_q that define an elliptic curve E over F_q.
$avf(P)$	The associate value of the EC point P. (See Section 5.6.1.)
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	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	The length of n in bits; $f = \lceil \log_2 n \rceil$.
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*.
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}.
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. 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.
P	An EC point.
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.)
U, V	An entity or a bit string denoting the identity of an entity. Usually U is used to denote the initiator of a protocol and V the responder.
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 or Z	A shared secret value.
Z_p	The set of integers modulo p, where p is an odd prime number.

Positional notation is used to indicate the association of a value to a particular entity, to indicate the life expectancy of a value, or to indicate the association of a value to a particular scheme. For example:

— d_U is an EC private key owned by entity U.
— Z_e is an ephemeral shared secret value.
— G_{enc} is an EC base point associated with an encryption scheme.
— $Q_{s,V}$ is a static EC public key owned by entity V.

Occasionally positional notation is also used to indicate a counter value associated with some data, or to indicate the base in which a particular value is being expressed if there is some possibility of ambiguity. For example, $Hash_1$ denotes the value of $Hash_i$ when the counter i has value 1, and 01_{16} denotes that the value 01 is written in hexadecimal.

With the exception of notation that has been well-established in other documents, where possible in this Standard capital letters will be used in variable names that denote bit strings or octet strings, and capital letters will be excluded from variable names that denote field elements or integers. For example, d is used to denote the integer that specifies an EC private key, and *MacData* is used to denote the bit string to be tagged using a MAC scheme.

Primed variables denote variables whose validity has not been verified. For example, *MacTag*' denotes the purported tag on *MacData*, and $Q_{e,V}$' denotes the purported ephemeral EC public key of entity V.

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 X9.19-1996, *Financial Institution Retail Message Authentication*.

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*).

ANSI X9.62-1999, *Public key cryptography for the financial services industry: The Elliptic Curve Digital Signature Algorithm* (*ECDSA*).
ANSI X9.71-199x, NWI. 1998. Working draft.

3 Application

3.1 General

The explosion in the use of electronic media to expedite commerce and financial transactions in recent years has led to the need for well-established cryptographic schemes that can provide services such as data integrity and data confidentiality.
Symmetric schemes such as Triple DEA make an attractive choice for the provision of these services - systems using symmetric techniques are efficient, and their security requirements are well-understood. Furthermore, these schemes have been standardized (for example in [1] and [3]) to facilitate interoperability between systems.
However, the major drawback with the implementation of such schemes is that any two communicating entities must establish in advance a shared secret key. As the size of a system or the number of entities using a system explodes, this can lead to a key management problem.
An attractive solution to this key management problem is for a system to employ asymmetric techniques that allow any pair of entities to establish a shared secret key suitable for use by a symmetric scheme despite the fact that the two entities may never have previously engaged in a secure communication together.
Such asymmetric techniques are known as asymmetric key establishment schemes.

3.2 The Schemes in this Standard

This Standard specifies asymmetric key establishment schemes. Both key agreement and key transport schemes are specified. The operation of each of the schemes employs arithmetic operations in the group of points on an elliptic curve defined over a finite field.
The asymmetric key establishment schemes in this Standard are used by an entity U who wishes to establish a symmetric key with another entity V. Each entity has an EC key pair. If U and V simultaneously execute a scheme with corresponding keying material as input, then at the end of the execution of the scheme, U and V will share keying data. The keying data can then be used to supply keys for symmetric algorithms. The precise method used to keys supply for symmetric algorithms from the keying data, for example setting parity bits or supplying three keys for Triple DEA, is beyond the scope of this Standard.
This Standard specifies a variety of asymmetric key establishment schemes. Each of the mechanisms, when implemented securely and embedded within a cryptographic system in an appropriate manner, is capable of providing two entities with a shared secret key suitable for use in symmetric algorithms like Triple DEA.
A variety of schemes are specified because of the wide variety of services that it may or may not be desirable for a key establishment scheme to provide depending on the environment in which the scheme is going to be used. The secret key may be agreed by both entities or transported from one entity to the other. Known-key security may be more or less desirable. Any combination of the services of entity authentication, key-compromise impersonation, and forward secrecy may be required. These are implementation specific decisions which this Standard attempts to facilitate.
However, this Standard recommends that any implementation of the schemes specified provides explicit key authentication of any key agreed using the key establishment schemes. Many of the schemes specified here do not directly provide explicit key authentication, and thus these schemes should be embedded in systems in such a way that explicit key authentication is additionally provided unless it is determined that explicit key authentication is not required. For each of the schemes specified here one extension with key confirmation is provided as an example of how explicit key confirmation may be provided. Further examples can be found in ANSI X9.70 [9].
The schemes in this Standard employ other cryptographic transformations in their operation. The transformations used are: the Data Encryption Algorithm (DEA) specified in [1], the DEA-based MAC specified in [3], the Secure Hash Algorithm (SHA-1) specified in [5], the Elliptic Curve Digital Signature Algorithm (ECDSA) specified in [8], and HMAC specified in [10].

3.3 Implementing the Schemes Securely

During the description of each scheme specified in this Standard, a list of prerequisites for the operation of the scheme is given. These prerequisites must be satisfied by any implementation of the scheme.

Two common prerequisites for the implementation of schemes in this Standard are that all entities involved in the use of the schemes are provided with an authentic copy of the elliptic curve parameters being used and that every entity is provided with a genuine copy of every other entity's static public key. The latter binding between an entity and its static public key may be accomplished by using a Certification Authority which generates a certificate in accordance with the procedures specified in [7].

However, satisfying the stated prerequisites is not enough to insure the security of an implementation.

The secure implementation of the schemes in this Standard is also dependent upon:

1. The prevention of unauthorized disclosure, use, modification, substitution, insertion, and deletion of an entity's static private key d_s.
2. The prevention of unauthorized modification, substitution, insertion, and deletion of the elliptic curve parameters being used.
3. The secure implementation of the transformations involved in an execution of a scheme so that the integrity and confidentiality of the computations involved is maintained.

Note that this includes the secure destruction of any ephemeral values involved in the operation of a scheme. Note also, however, that the effect of some of the most common breaches in the above requirements may be minimized by the selection of an appropriate scheme that provides, for example, the service of forward secrecy or known-key security.

Finally, secure implementation of the schemes does not guarantee the security of the operation of the implementation. It is the responsibility of the operator to put an overall process in place with the necessary controls to insure the secure operation. The controls should include the application of appropriate audit tests in order to verify compliance with this Standard.

3.4 Annexes

The annexes to this Standard provide additional requirements and information on the schemes and primitives specified in this Standard and their 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	Comparison of Elliptic Curves and Finite Fields
H	Security Considerations
I	Alignment with Other Standards
J	Examples
K	Patents
L	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, $160 \leq m \leq 2000$, 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 $m \geq 4$.

Table C-3 in Annex C lists an irreducible pentanomial of degree m over F_2 for each m, $160 \leq m \leq 2000$, 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, $160 \leq m \leq 2000$, 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 $F(1), F(2), \ldots, F(p-1)$ as follows:
   - 3.1 Set $w = 1$.
   - 3.2 For j from 0 to $T-1$ do
     - 3.2.1 Set $n = w$.
     - 3.2.2 For i from 0 to $m-1$ do
       - 3.2.2.1 Set $F(n) = i$.
       - 3.2.2.2 Set $n = 2n \bmod p$.
     - 3.2.3 Set $w = uw \bmod p$.
4. If T is even, then set $J = 0$, else set
$$J = \sum_{k=1}^{m/2} a_{k-1} b_{m/2+k-1} + a_{m/2+k-1} b_{k-1}$$
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 K.

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:

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

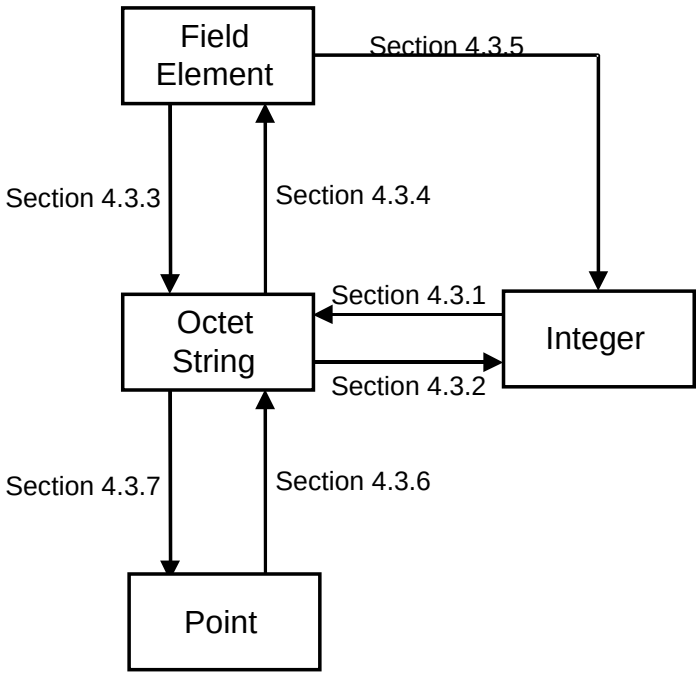

Figure 1 – Data Types and Conversion Conventions

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 $(8l - m + 1)^{\text{th}}$ 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 Cryptographic Ingredients

This section specifies the various cryptographic ingredients that are required by the key agreement and key transport schemes. These ingredients include primitives, auxiliary functions, and schemes.

These ingredients are also employed by various other ANSI standards - for example [9].

5.1 Elliptic Curve Domain Parameter Generation and Validation

This section specifies the primitives that shall be used to generate EC domain parameters and validate EC domain parameters.

In this Standard, EC domain parameters will be shared by a number of entities using a particular system. In some schemes, distinct parameter sets may be used for calculations involving entities' static keys and for calculations involving entities' ephemeral keys, and in other schemes, the same parameter set must be used for both ephemeral and static calculations.

In all cases, the EC domain parameters may be public; the security of the system does not rely on these parameters being secret.

The primitives specified here allow EC domain parameters to be generated in any manner subject to some security constraints. The primitives optionally support an additional feature allowing EC domain parameters to be generated verifiably at random.

The primitives specified differ depending on the characteristic of the underlying field. Thus, Section 5.1.1 describes the primitives that shall be used for parameter generation and validation in the case that the underlying field is F_p, and Section 5.1.2 describes the primitives that shall be used in the case that the underlying field is F_{2^m}.

The parameter generation primitives will be used whenever EC domain parameters are generated for a system. Furthermore, many of the schemes specified in this Standard require a valid set of EC domain parameters to be held by each entity involved in the operation of the scheme. Thus, in all cases, the generator of the system parameters will

use the appropriate parameter validation primitive to check the validity of the generated parameters. The validation primitives may additionally be used by the entities using the parameters to check their validity.
Note that in all cases n is the primary security parameter. In general, as n increases, the security of the EC scheme also increases. See Annex H for more information.

5.1.1 Elliptic Curve Domain Parameter Generation and Validation over F_p

5.1.1.1 Elliptic Curve Domain Parameter Generation over F_p Primitive

EC domain parameters over F_p shall be generated using the following routine.
Input: This routine does not take any input.
Actions: The following actions are taken:
1. Choose as the field size a prime p with $p>3$, a lower bound r_{min} for the point order, and a trial division bound l_{max}. (See Annex H for advice on the implications of these decisions.)
2. Select EC domain parameters using the method described in Annex A.3.2 on input p, r_{min}, and l_{max}.

Output: This routine outputs:
1. The 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: $y^2 \equiv x^3+ax+b \pmod p$.
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 $n>4\sqrt{q}$).
6. The cofactor $h = \#E(F_q)/n$.

5.1.1.2 Elliptic Curve Domain Parameter Validation over F_p Primitive

The following transformation shall be used to validate EC domain parameters over F_p.
Input: The input of the validation transformation is a purported set of EC domain parameters consisting of p', a', b', $G' =(x_G',y_G')$, n', and h', and optionally the purported seed *SEED*' used in the generation process.
Actions: The following checks are made:
1. Verify that 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 EC 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.)
4. Verify that $4(a')^3 + 27(b')^2 \neq 0 \pmod{p'}$.
5. Verify that $(yG')^2 \equiv (xG')^3+(a')(xG')+(b') \pmod{p'}$.
6. Verify that n' is prime and that $n'>2^{160}$. (See Annex A.2.1.)
7. Verify that $n'G' = \mathcal{O}$. (See Annex D.3.2.)
8. Check that $n'>4\sqrt{p'}$, 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.)

Output: If any of the above verifications has failed, then output 'invalid' and reject the EC domain parameters. Otherwise, output 'valid', and accept the EC domain parameters.

NOTE— Step 8 of the above transformation (and also step 9 of Section 5.1.2.2) verifies that the value of the purported cofactor h' is correct in the case that $n'>4\sqrt{q}'$. The case that $n' \leq 4\sqrt{q}'$ is excluded; there are methods for verifying the cofactor h' in this case, but these methods are not described here because the general methods are cumbersome and elliptic curves used in practice usually have $n \approx q$ so that the condition $n'>4\sqrt{q}'$ will be satisfied.

5.1.2 Elliptic Curve Domain Parameter Generation and Validation over F_{2^m}

5.1.2.1 Elliptic Curve Domain Parameter Generation over F_{2^m} Primitive

EC domain parameters over F_{2^m} shall be generated using the following routine.
Input: This routine does not take any input.
Actions: The following actions are taken:

1. Choose a field size 2^m, a lower bound r_{min} for the point order, and a trial division bound l_{max}. (See Annex H for advice on the implications of these decisions.)
2. Choose a basis to use to represent the elements of F_{2^m} (either TPB, PPB, or GNB). If TPB or PPB is chosen, also choose an appropriate reduction polynomial $f(x)$ of degree m over F_2 to use. (See Section 4.1.2.)
3. Select EC domain parameters using the method described in Annex A.3.2 on input 2^m, r_{min}, and l_{max}.

Output: This routine outputs:

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

5.1.2.2 Elliptic Curve Domain Parameter Validation over F_{2^m} Primitive

The following transformation shall be used to validate EC domain parameters over F_{2^m}.

Input: The input of the validation transformation is a purported set of EC domain parameters consisting of $q'=2^{m'}$, a', b', $G'=(x_G',y_G')$, n', h', and an indication of the type of basis to be used to represent $F_{2^{m'}}$ together with, when appropriate, a purported reduction polynomial $f(x)'$, and optionally the purported seed *SEED*' used in the generation process.

Actions: The following checks are made:

1. Verify that $2^{m'}$ is a power of two.
2. If the type of basis indicated is TPB, verify that $f(x)'$ is a trinomial of degree m' which is irreducible over F_2. (See Table C-2 or Annex D.2.4.) If the type of basis indicated is a PPB, verify that an irreducible trinomial of degree m' does not exist, and that $f(x)'$ is a pentanomial of degree m' which is irreducible over F_2. (See Table C-3 or Annex D.2.4.) If the type of basis indicated is GNB, verify that m' is not divisible by 8.
3. Verify that a', b', x_G' and y_G' are bit strings of length m' bits.
4. If the EC was randomly generated in accordance with Section 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.)
5. Verify that $b' \neq 0$.
6. Verify that $(y_G')^2 + x_G'y_G' = (x_G')^3+(a')(x_G')^2+(b')$ in $F_{2^{m'}}$.
7. Verify that n' is prime and that $n'>2^{160}$. (See Annex A.2.1.)
8. Verify that $n'G'=\mathcal{O}$. (See Section D.3.2.)
9. Check that $n'>4\sqrt{q'}$, compute $h = \lfloor (\sqrt{q'}+1)^2/n' \rfloor$ and verify that $h'=h$.
10. Verify that the MOV and Anomalous conditions hold. (See Annex A.1.)

Output: If any of the above verifications has failed, then output 'invalid' and reject the EC domain parameters. Otherwise, output 'valid', and accept the EC domain parameters.

5.2 Key Pair Generation and Public Key Validation

This section specifies the primitives that shall be used to generate EC key pairs and to validate EC public keys. The key pair generation primitive will be used during the generation of entities' key pairs. In some schemes, the primitive will be used to produce static key pairs, and in other schemes, the primitive will be used to produce ephemeral key pairs.

Public key validation will be used during the validation of an entity's public keys. Sometimes this validation process will be carried out by a trusted Center such as a Certification Authority that wishes to bind an entity to its static public key. At other times this process will be carried out by an entity who wishes to validate the purported ephemeral public key of another entity.

5.2.1 Key Pair Generation Primitive

EC key pairs shall be generated using the following transformation:

Input: The input of the generation transformation is a valid set of EC parameters q, a, b, x_G, y_G, n, and h along with an indication of the basis used if $q=2^m$. Note that it is assumed that the parameters have been validated using the primitives described in Sections 5.1.1.2 and 5.1.2.2.

Actions: The following actions are taken:

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 of 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.)

Output: The key pair (d,Q), where Q is the public key and d is the private key.

5.2.2 Public Key Validation Primitive

Public key validation refers to the process of checking the arithmetic properties of a public key. It prevents various forms of attack, for example so-called small subgroup attacks, which rely on the use of an invalid public key. See Annex H for further discussion.

When an entity U is required to validate a public key in this Standard, four methods of public key validation are acceptable. Only one of the methods must be carried out, although in many cases greater assurance may be obtained by carrying out more than one of the methods.

The four acceptable methods are:

1. U performs explicit public key validation of the public key itself by using the appropriate technique described in Section 5.2.2.1 or 5.2.2.2.
2. U performs implicit public key validation of the public key itself by generating the public key itself using trusted routines.
3. U receives assurance that a party trusted for the lifetime of any key combined with the public key being validated using the Diffie-Hellman primitives or the MQV primitive, has validated the public key by using the appropriate technique described in Section 5.2.2.1 or 5.2.2.2.
4. U receives assurance that a party trusted for the lifetime of any key combined with the public key being validated using the Diffie-Hellman primitives or the MQV primitive, has implicitly validated the public key by generating the public key itself using trusted routines.

Typically when U accepts assurance from another party that party is a CA. However on occassion U may accept the assurance of another entity as well as a CA. For example in the Station-to-Station scheme, U receives an ephemeral public key from V in a signed string. U combines the public key with its own ephemeral public key using the Diffie-Hellman primitive. It is acceptable for U to accept the validity of the ephemeral public based on the knowledge that V generated the key because V is trusted for the lifetime of U's ephemeral public key.

5.2.2.1 Standard Public Key Validation Primitive

An EC public key shall be validated in the following manner when it is going to be used by the standard Diffie-Hellman primitive:

Input: The input of the validation transformation is a valid set of EC domain parameters q, a, b, x_G, y_G, n, and h, along with an indication of the basis used if $q=2^m$, together with the purported public key $Q'=(x_Q',y_Q')$. Note that it is assumed that the parameters have been validated using the primitives described in Sections 5.1.1.2 and 5.1.2.2.

Actions: The following checks are made:

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. (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 $(y_Q')^2+x_Q'y_Q' = (x_Q')^3+a(x_Q')^2+b$ in F_{2^m}.
4. Verify that $nQ'=\mathcal{O}$. (See Annex D.3.2.)

Output: If any one of the above verifications fail, then output 'invalid' and reject the public key. Otherwise output 'valid' and accept the public key.

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

5.2.2.2 Embedded Public Key Validation Primitive

An EC public key shall be validated in the following manner when it is going to be used by the Diffie-Hellman with cofactor primitive or the MQV primitive:

Input: The input of the embedded validation transformation is a valid set of EC domain parameters q, a, b, x_G, y_G, n, and h, along with an indication of the basis used if $q=2^m$, together with the purported public key $Q'=(x_Q', y_Q')$. Note that it is assumed that the parameters have been validated using the primitives described in Sections 5.1.1.2 and 5.1.2.2.

Actions: The following checks shall be made:

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. (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 $(y_Q')^2+x_Q'y_Q' = (x_Q')^3+a(x_Q')^2+b$ in F_{2^m}.

Output: If any one of the above verifications has failed, then output 'invalid' and reject the public key. Otherwise output 'valid' and accept the public key.

5.3 Challenge Generation Primitive

This section specifies the primitive that shall be used to generate challenges to be used by the schemes in this Standard.

The challenge generation primitive will be used to generate challenges in the 3-pass key transport scheme specified in Section 7.

Challenges shall be generated using the following transformation:

Input: An integer *challengelen* which is the length in bits of the challenge required. *challengelen* shall be ≥80.

Actions: Select a statistically unique and unpredictable bit string *Challenge* of length *challengelen*. It is acceptable to use a random or a pseudorandom string. If a pseudorandom string is used, it shall be generated using one of the procedures of Annex A.4 or of an ANSI X9 approved standard. If a pseudorandom number is used, optional information to store with the challenge are the seed values and the particular pseudorandom generation method used. Storing this optional information helps allow auditing of the challenge generation process.

If a pseudorandom generation method is used, the seed values used in the generation of *Challenge* may be determined by internal means, be supplied by the caller, or both - this is an implementation choice.

Output: The bit string *Challenge*.

NOTE— If more than one challenge is generated, it may be checked that no two challenges are the same.

5.4 Diffie-Hellman Primitive

This section specifies the Diffie-Hellman primitive that shall be used by the key establishment schemes in this Standard.

This primitive derives a shared secret value from one entity's secret key and another entity's public key when the keys share the same EC parameters. If two entities both correctly execute this primitive with corresponding keys as inputs, they will produce the same value.

The calculation of the shared secret value incorporates co-factor multiplication. Co-factor multiplication is computationally efficient and helps to prevent security problems like small subgroup attacks (see [42].)

The shared secret value shall be calculated as follows:

Prerequisites: The prerequisite is a set of EC domain parameters q, a, b, G, n, and h, along with an indication of the basis used if $q=2^m$, which has been validated using the techniques described in Sections 5.1.1.2 and 5.1.2.2.

Input: The Diffie-Hellman primitive takes as input:

1. an EC private key d.
2. an EC public key Q.

The public key Q will have been validated as specified in Section 5.2.2.

Actions: The following actions are taken:

1. Compute the point $P=hdQ$. (See Section D.3.2.)
2. Check $P\neq\mathcal{O}$. If $P=\mathcal{O}$, output 'invalid' and stop.
3. Set $z=x_P$, where x_P is the x-coordinate of P.

Output: $z\in F_q$ as the shared secret value.

5.5 MQV Primitive

This section specifies the MQV primitive that shall be used by the key agreement schemes specified in this Standard. This primitive derives a shared secret value from two secret keys owned by U and two public keys owned by V when all the keys share the same EC parameters. If two entities both correctly execute this primitive with corresponding keys as inputs, they will produce the same value.

The calculation of the shared secret value incorporates co-factor multiplication. Co-factor multiplication is computationally efficient and helps to prevent security problems like small subgroup attacks (see [42].)

The shared secret value shall be calculated as follows:

Prerequisites: The prerequisite is a set of EC domain parameters q, a, b, G, n, and h, along with an indication of the basis used if $q=2^m$, which has been validated using the techniques described in Sections 5.1.1.2 and 5.1.2.2.

Input: The MQV primitive takes as input:

1. two EC key pairs $(d_{1,U},Q_{1,U})$ and $(d_{2,U},Q_{2,U})$ owned by U.
2. two EC public keys $Q_{1,V}$ and $Q_{2,V}$ owned by V.

The key pairs $(d_{1,U},Q_{1,U})$ and $(d_{2,U},Q_{2,U})$ will have been generated using the key pair generation primitive specified in Section 5.2.1. The public keys $Q_{1,V}$ and $Q_{2,V}$ will have been validated as specified in Section 5.2.2.

Actions: The following actions are taken:

1. Compute the integer:

   $$implicitsig_U = d_{2,U}+(avf(Q_{2,U})\times d_{1,U}) \pmod n.$$

2. Compute the EC point:

   $$P = h \times implicitsig_U \times (Q_{2,V}+(avf(Q_{2,V})\times Q_{1,V})).$$

   (See Section D.3.2.)
3. Check $P\neq\mathcal{O}$. If $P=\mathcal{O}$, output 'invalid' and stop.
4. Set $z=x_P$, where x_P is the x-coordinate of P.

Output: $z\in F_q$ as the shared secret value.

5.6 Auxiliary Functions

This section specifies three types of auxiliary functions that will be used by some of the key agreement schemes and key transport schemes specified in this Standard: associate value functions, cryptographic hash functions and key derivation functions.

5.6.1 Associate Value Function

This section specifies the associate value function that shall be used by the schemes in this Standard.

The associate value function will be used to compute an integer associated with an elliptic curve point.

The associate value function will be used by the MQV family of key agreement schemes specified in Section 6.

The associate value function shall be calculated as follows:

Input: The input to the associate value function is:

1. A valid set of EC domain parameters q, a, b, G, n, h along with an indication of the basis used if $q=2^m$.
2. A point $P\neq\mathcal{O}$ on the EC defined by the parameters q, a, b, G, n, h.

Actions: Perform the following computations:

1. Convert x_P to an integer using the convention specified in Section 4.3.5.
2. Calculate:
   $x_P' = x_P \pmod{2^{\lceil f/2 \rceil}}$.
3. Calculate:
   $avf(P) = x_P' + 2^{\lceil f/2 \rceil}$.

Output: The integer $avf(P)$ as the associate value of P.

5.6.2 Cryptographic Hash Functions

This section specifies the cryptographic hash functions that shall be used by the schemes in this Standard.
The hash functions will be used to calculate the hash value associated with a bit string.
The hash functions will be used by the key derivation function specified in Section 5.6.3.
Any ANSI-approved hash function which offers 80 bits of security or more may be used, i.e. any ANSI-approved hash function whose output is 160 bits or more. Possibilities therefore include the hash function SHA-1. SHA-1 is specified in [5].
Hash values shall be calculated as follows:

Prepreqisites: The prerequisite for the operation of the hash function is that an ANSI-approved hash function has been chosen. We denote the maximum length of the input to the hash function by *hashmaxlen* and the length of the output of the hash function by *hashlen*.

Input: The input to the hash function is a bit string *Data* of length less than *hashmaxlen* bits.

Actions: Calculate the hash value corresponding to *Data* as:

$Hash=H(Data)$

using the established hash function.

Output: The bit string *Hash* of length *hashlen* bits.

Note that the hash function operates on bit strings of length less than *hashmaxlen* bits. For example, SHA-1 operates on bit strings of length less than 2^{64} bits. In the sequel it is assumed that all hash function calls are indeed on bit strings of length less than *hashmaxlen* bits. Any scheme attempting to call the hash function on a bit string of length greater than or equal to *hashmaxlen* bits shall output 'invalid' and stop.

5.6.3 Key Derivation Functions

This section specifies the key derivation function that shall be used by the schemes in this Standard.
The key derivation function will be used to derive keying data from a shared secret bit string.
The key derivation function will be used by the key agreement schemes to compute keying data from a shared secret value. The key derivation function will also be used by the asymmetric encryption schemes.
The key derivation function that will be used is a simple hash function construct.
Keying data shall be calculated as follows:

Prerequisites: The prerequisite for the operation of the key derivation function is that an ANSI-approved hash function has been chosen as specified in Section 5.6.2.

Input: The input to the key derivation function is:

1. A bit string *Z* which is the shared secret value.
2. An integer *keydatalen* less than $hashlen \times (2^{32}-1)$ which is the length in bits of the keying data to be generated.
3. (Optional) A bit string *SharedInfo* which consists of some data shared by the two entities intended to share the secret value *Z*.

Ingredients: The key derivation function employs one of the hash functions specified in Section 5.6.2.

Actions: The key derivation function is computed as follows:

1. Initiate a 32-bit, big-endian bit string *counter* as 00000001_{16}.
2. For i=1 to $\lceil keydatalen/hashlen \rceil$, do the following:
   2.1 Compute $Hash_i = H(Z \,||\, counter \,||\, [SharedInfo])$.
   2.2 Increment *counter*.
   2.3 Increment *i*.
3. Let $Hash!^{\lceil keydatalen/hashlen \rceil}$ denote $Hash^{\lceil keydatalen/hashlen \rceil}$ if *keydatalen*/*hashlen* is an integer, and let it denote the $(keydatalen - (hashlen \times \lfloor keydatalen/hashlen \rfloor))$ leftmost bits of $Hash^{\lceil keydatalen/hashlen \rceil}$ otherwise.
4. Set $KeyData = Hash_1 || Hash_2 || \ldots || Hash^{\lceil keydatalen/hashlen \rceil - 1} || Hash!^{\lceil keydatalen/hashlen \rceil}$.

Output: The bit string *KeyData* of length *keydatalen* bits.

Note that the key derivation function produces keying data of length less than $hashlen \times (2^{32}-1)$ bits. In the sequel we assume that all key derivation function calls are indeed for bit strings of length less than $hashlen \times (2^{32}-1)$ bits. Any scheme attempting to call the key derivation function for a bit string of length greater than or equal to $hashlen \times (2^{32}-1)$ bits shall output 'invalid' and stop.

5.7 MAC schemes

This section specifies the tagging transformation and the tag checking transformation associated with the message authentication code (MAC) schemes that shall be used by the schemes in this Standard.

Each MAC scheme will be used as follows. The sender will use the tagging transformation to compute the tag on some data. The recipient, after being sent the data and tag, will check the validity of the tag using the tag checking transformation.

The MAC schemes will be used by some key agreement schemes to provide key confirmation and by the augmented encryption scheme in Section 5.8.2.

Any ANSI-approved MAC that offers 80 bits of security or more may be used, i.e. any ANSI-approved MAC that uses keys of length 80 bits or more and that outputs tags of length 80 bits or more. Possibilities therefore include the 2-key scheme based on the DEA algorithm [1] specified in [3], and HMAC specified in ANSI X9.71 [10]. (Note that use of the MAC specified in ANSI X9.9 is not permitted.)

The appropriate choice of MAC scheme in a particular application will depend on the operating environment. Issues involved in the decision will often include security requirements and available cryptographic primitives.

The MAC scheme is specified as follows.

Prerequisites: The prerequisite for the operation of the MAC scheme is that an ANSI-approved MAC scheme has been chosen. We denote by *mackeylen* the length in bits of the keys used by the MAC scheme.

5.7.1 Tagging Transformation

Data shall be tagged using the tagging transformation specified as follows:

Input: The tagging transformation takes as input:

1. A bit string *MacData* to be MACed.
2. A bit string *MacKey* of length *mackeylen* bits to be used as the key.

Actions: Calculate the tag as:

$$MacTag = MAC_{MacKey}(MacData),$$

where $MAC_{MacKey}(MacData)$ denotes the computation of the tag on *MacData* under *MacKey* using the tagging transformation of the established ANSI-approved MAC scheme.

Output: The bit string *MacTag*.

5.7.2 Tag Checking Transformation

The purported tag on data shall be checked using the tag checking transformation specified as follows:

Input: The tag checking transformation takes as input:

1. The data which is a bit string *MacData*.
2. The purported tag for *MacData* which is a bit string *MacTag'*.
3. A bit string *MacKey* of length *mackeylen* bits to be used as the key.

Actions: Calculate the tag for *MacData* under the key *MacKey* as:

$$MacTag = MAC_{MacKey}(MacData)$$

using the tagging transformation of the established ANSI-approved MAC.

Output: If *MacTag'*=*MacTag* output 'valid', otherwise output 'invalid'.

5.8 Asymmetric Encryption Schemes

This section specifies the asymmetric encryption schemes that shall be used by the schemes in this Standard.

Each of the asymmetric encryption schemes will be used as follows. The sender will use the encryption transformation of the scheme to encrypt some data. The recipient, after being sent the encrypted data, will decrypt the encrypted data using the decryption transformation of the scheme.

The asymmetric encryption schemes will be used by the key transport schemes specified in Section 7.

Two encryption schemes are specified. The schemes are designed to provide security against attacks of different kinds. The Elliptic Curve Encryption Scheme is designed to provide security against passive or chosen plaintext attacks in which attacker attempts to compromise the scheme using only knowledge of an entity's public key. The Elliptic Curve Augmented Encryption Scheme is designed to provide security against both chosen plaintext and chosen ciphertext attacks in which an attacker additionally attempts to exploit knowledge gained by somehow learning the decryption of some ciphertext.
Use of the Elliptic Curve Augmented Encryption Scheme is therefore recommended because security against chosen ciphertext attacks is required in order for the key transport schemes in Section 7 to provide the known-key security service. The Elliptic Curve Encryption Scheme should be used only when it is determined appropriate – for example when known-key security is not required.

5.8.1 Elliptic Curve Encryption Scheme

The Elliptic Curve Encryption Scheme is specified as follows.

Prerequisites: The prerequisite for the operation of the Elliptic Curve Encryption Scheme is a set of EC domain parameters q, a, b, G, n, and h along with an indication of the basis used if $q=2^m$. The parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2. Finally, an ANSI-approved hash function shall have been chosen for use with the key derivation function.

5.8.1.1 Encryption Transformation

Data shall be encrypted as follows:

Input: The input to the encryption transformation is:

1. A bit string *EncData* of length *encdatalen* which is the data to be encrypted.
2. A EC public key Q owned by the recipient.
3. (Optional) A bit string of data *SharedData*, which is shared by the sender and the recipient.

The EC public key Q shall correspond to the EC domain parameters q, a, b, G, n, h. Q may have been validated as specified in Section 5.2.2.

Ingredients: The encryption transformation employs the key pair generation primitive specified in Section 5.2.1, the Diffie-Hellman primitive specified in Section 5.4, and the key derivation function specified in Section 5.6.3.

Actions: Encrypt the bit string *EncData* as follows:

1. Generate an ephemeral key pair (d_e,Q_e) corresponding to the EC domain parameters q, a, b, G, n, and h, using the key pair generation primitive defined in Section 5.2.1.
2. Convert Q_e to a bit string QE using the convention specified in Section 4.3.6.
3. Use the Diffie-Hellman primitive defined in Section 5.4 to derive a shared secret field element $z \in F_q$ from d_e and Q. If the Diffie-Hellman primitive outputs 'invalid', output 'invalid' and stop.
4. Convert $z \in F_q$ to a bit string Z using the convention specified in Section 4.3.3.
5. Use the key derivation function defined in Section 5.6.3 with the established hash function to generate keying data *EncKey* of length *encdatalen* from Z and [*SharedData*].
6. Compute $MaskedEncData = EncData \oplus EncKey$.

Output: Output the bit string $QE||MaskedEncData$ as the encryption of *EncData*.

5.8.1.2 Decryption Transformation

The decryption transformation shall be calculated as follows:

Input: The input to the decryption transformation is:

1. A bit string *QE'||MaskedEncData'* purporting to be the encryption of a bit string.
2. An EC private key d owned by the recipient.
3. (Optional) A bit string of data *SharedData* which is shared by the sender and the recipient.

The private key d shall have been generated using the key pair generation primitive specified in Section 5.2.1.

Ingredients: The decryption transformation employs public key validation as specified in Section 5.2.2, the Diffie-Hellman primitive specified in Section 5.4, and the key derivation function specified in Section 5.6.3.

Actions: Decrypt the bit string *QE'||MaskedEncData'* consisting of the encoding of a purported elliptic curve point Q_e', and a bit string *MaskedEncData'* of length *maskedencdatalen* as follows:

1. Validate the ephemeral public key Q_e' as specified in Section 5.2.2. If the validation primitive outputs 'invalid', output 'invalid' and stop.
2. Use the Diffie-Hellman primitive defined in Section 5.4 to derive a shared secret field element $z \in F_q$ from d and Q_e'. If the Diffie-Hellman primitive outputs 'invalid', output 'invalid' and stop.
3. Convert $z \in F_q$ to a bit string Z using the convention specified in Section 4.3.3.
4. Use the key derivation function specified in Section 5.6.3 with the established hash function to generate keying data *EncKey* of length *maskedencdatalen* from Z and [*SharedData*].
5. Compute *EncData* = *MaskedEncData*'⊕*EncKey*.

Output: Output *EncData* as the decryption of *QE*'||*MaskedEncData*'.

5.8.2 Elliptic Curve Augmented Encryption Scheme

The Elliptic Curve Augmented Encryption Scheme is specified as follows.

Prerequisites: The prerequisite for the operation of the Elliptic Curve Augmented Encryption Scheme is a set of EC domain parameters q, a, b, G, n, and h along with an indication of the basis used if $q=2^m$. The parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2. In addition entities using the scheme will have established which ANSI-approved MAC scheme specified in Section 5.7 they will use. We denote by *mackeylen* the length in bits of the keys used by the established MAC scheme. Finally, an ANSI-approved hash function shall have been chosen for use with the key derivation function.

5.8.2.1 Encryption Transformation

Data shall be encrypted as follows:

Input: The input to the encryption transformation is:

1. A bit string *EncData* of length *encdatalen* which is the data to be encrypted.
2. A EC public key Q owned by the recipient.
3. (Optional) Two bit strings of data, $SharedData_1$ and $SharedData_2$, which are shared by the sender and the recipient.

The EC public key Q shall correspond to the EC domain parameters q, a, b, G, n, h. Q may have been validated as specified in Section 5.2.2.

Ingredients: The encryption transformation employs the key pair generation primitive specified in Section 5.2.1, the Diffie-Hellman primitive specified in Section 5.4, the tagging transformation of the established MAC scheme specified in Section 5.7, and the key derivation function specified in Section 5.6.3.

Actions: Encrypt the bit string *EncData* as follows:

1. Generate an ephemeral key pair (d_e, Q_e) corresponding to the EC domain parameters q, a, b, G, n, and h, using the key pair generation primitive defined in Section 5.2.1.
2. Convert Q_e to a bit string *QE* using the convention specified in Section 4.3.6.
3. Use the Diffie-Hellman primitive defined in 5.4 to derive a shared secret field element $z \in F_q$ from d_e and Q. If the Diffie-Hellman primitive outputs 'invalid', output 'invalid' and stop.
4. Convert $z \in F_q$ to a bit string Z using the convention specified in Section 4.3.3.
5. Use the key derivation function defined in Section 5.6.3 with the established hash function to generate keying data *KeyData* of length *encdatalen*+*mackeylen* from Z and [$SharedData_1$]. Parse *KeyData* as an encryption key *EncKey* of length *encdatalen* and a MAC key *MacKey* of length *mackeylen*, i.e. parse *KeyData* as:

   KeyData = *EncKey*||*MacKey*.
6. Compute *MaskedEncData*=*EncData*⊕*EncKey*.
7. Compute the tag *MacTag* on the bit string:

   MacData = *MaskedEncData*||[$SharedData_2$]

   under the MAC key *MacKey* using the tagging transformation of the established MAC scheme as specified in Section 5.7.

Output: Output the bit string *QE*||*MaskedEncData*||*MacTag* as the encryption of *EncData*.

5.8.2.2 Decryption Transformation

The decryption transformation shall be calculated as follows:

Input: The input to the decryption transformation is:

1. A bit string *QE'||MaskedEncData'||MacTag'* purporting to be the encryption of a bit string.
2. An EC private key *d* owned by the recipient.
3. (Optional) Two bit strings of data, $SharedData_1$ and $SharedData_2$, which are shared by the sender and the recipient.

The private key *d* shall have been generated using the key pair generation primitive specified in Section 5.2.1.

Ingredients: The decryption transformation employs public key validation as specified in Section 5.2.2, the Diffie-Hellman primitive specified in Section 5.4, the tag checking transformation of the established MAC scheme specified in Section 5.7, and the key derivation function specified in Section 5.6.3.

Actions: Decrypt the bit string *QE'||MaskedEncData'||MacTag'* consisting of the encoding of a purported elliptic curve point Q_e', a bit string *MaskedEncData'* of length *maskedencdatalen*, and a bit string *MacTag'* of the appropriate length as follows:

1. Validate the ephemeral public key Q_e' using public key validation as specified in Section 5.2.2. If the validation primitive outputs 'invalid', output 'invalid' and stop .
2. Use the Diffie-Hellman primitive defined in Section 5.4 to derive a shared secret field element $z \in F_q$ from *d* and Q_e'. If the Diffie-Hellman primitive outputs 'invalid', output 'invalid' and stop.
3. Convert $z \in F_q$ to a bit string *Z* using the convention specified in Section 4.3.3.
4. Use the key derivation function specified in Section 5.6.3 with the established hash function to generate keying data *KeyData* of length *maskedencdatalen+mackeylen* from *Z* and [$SharedData_1$]. Parse *KeyData* as an encryption key *EncKey* of length *maskedencdatalen* and a MAC key *MacKey* of length *mackeylen*, i.e. parse *KeyData* as:

   KeyData = EncKey||MacKey.
5. Compute *EncData=MaskedEncData'*$\oplus$*EncKey*.
6. Verify that *MacTag'* is the tag on *MaskedEncData*||[$SharedData_2$] under the key *MacKey* using the tag checking transformation of the established MAC scheme specified in Section 5.7. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.

Output: Output *EncData* as the decryption of *QE'||MaskedEncData'||MacTag'*.

5.9 Signature Scheme

This section specifies the signature scheme that shall be used by the schemes in this Standard.

The signature scheme will be used as follows. The sender will use the signing transformation to compute a signature on some data. The recipient, after being sent the data and signature, will check the validity of the signature using the verifying transformation.

The signature scheme will be used by the 3-pass key transport scheme specified in Section 7.2.

The signature scheme supported is the Elliptic Curve Digital Signature Algorithm (ECDSA). ECDSA is specified in [8]. ECDSA shall be implemented as specified in [8].

Prerequisites: The prerequisite for the operation of the ECDSA is a set of EC domain parameters *q*, *a*, *b*, *G*, *n*, and *h* along with an indication of the basis used if $q=2^m$. The parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.

5.9.1 Signing Transformation

The transformation specified as follows shall be used to sign data:

Input: The input to the signing transformation is:

1. A bit string *SignData* to be signed.
2. An elliptic curve private key *d* owned by the sender.

The private key *d* shall correspond to the EC parameters *q*, *a*, *b*, *G*, *n*, *h*, and shall have been generated using the primitive specified in Section 5.2.1.

Actions: Compute the integers *rsig* and *ssig* which comprise the signature on *SignData* as specified in [8].

Output: The pair of integers *rsig* and *ssig*.

5.9.2 Verifying Transformation

The transformation specified as follows shall be used to verify a purported signature:

Input: The input to the verifying transformation is:

1. The data which is a bit string *SignData*.
2. A pair of integers *rsig*' and *ssig*' which are the purported signature of *SignData*.
3. An EC public key *Q* owned by the sender.

The public key *Q* shall correspond to the EC parameters *q*, *a*, *b*, *G*, *n*, *h*, and may have been validated as specified in Section 5.2.2.

Actions: Verify the purported signature using the verification transformation specified in [8].

Output: Output 'valid' if the verification transformation confirms that *rsig*' and *ssig*' are a valid signature of *SignData*, otherwise output 'invalid'.

6 Key Agreement Schemes

This section describes the key agreement schemes specified in this Standard.

In each case, the key agreement scheme is used by an entity who wishes to agree on keying data with another entity. In some cases the protocols specified are 'symmetric', and so it suffices to describe just one transformation. In other cases the protocols are 'asymmetric', and so it is necessary to describe two transformations, one of which is undertaken by *U* if *U* is the initiator, and one of which is undertaken by *V* if *V* is the responder.

In the specification of each transformation, equivalent computations that result in identical output are allowed.

Each of the key agreement schemes has certain prerequisites. These are conditions that must be satisfied by an implementation of the scheme. However the specification of mechanisms that provide these prerequisites is beyond the scope of this Standard.

Section H.4.3 provides guidance to the services which each scheme may be used to provide.

Each scheme is described in two ways. First a flow diagram of the ordinary operation of the scheme between two entities *U* and *V* is provided. This flow diagram is for descriptive purposes only. Then a formal specification is given which describes the actions entities must take to use the scheme to establish keying data.

The flow diagrams are intended to aid understanding of the mechanics of the 'ordinary' operation of the schemes in which flows are faithfully relayed between two entities. Note that in 'real-life', there is no reason to assume that flows are relayed faithfully between two entities…that is why the schemes must be formally specified in a more technical fashion.

When examining the flow diagrams, the following points should be noted:

— For clarity of exposition, optional fields such as *Text* and *SharedData* are omitted.

— *kdf*(*Z*) denotes the output of the key derivation function specified in Section 5.6.3 called on input *Z*.

— *ENC* and *DEC* respectively denote the encryption and decryption transformations associated with one of the asymmetric encryption schemes specified in Section 5.8. The subscripts immmediately following *ENC* and *DEC* denote the keys being used in the operation of the appropriate transformation. Similarly, *SIG* denotes the signing transformation associated with the signature scheme ECDSA specified in Section 5.9, and *MAC* denotes the tagging transformation of one of the MAC schemes specified in Section 5.7.

6.1 Ephemeral Unified Model Scheme

This section specifies the ephemeral Unified Model scheme.

First the scheme is illustrated in a flow diagram. Figure 2 illustrates the use of the ephemeral Unified Model scheme.

Figure 2 - Ephemeral Unified Model Scheme

Next the formal specification of the scheme is given.

The scheme is 'symmetric', so only one transformation is specified. An entity uses this transformation to agree on keying data with another entity no matter whether they are the initiator or the responder.

If two entities *U* and *V* simultaneously execute the transformation with corresponding keying material as input, then *U* and *V* will compute the same keying data.

Prerequisites: The following are the prerequisites for the use of the scheme: Each entity has an authentic copy of the system's EC domain parameters to be used with ephemeral keys q_e, a_e, b_e, G_e, n_e, and h_e along with an indication of the basis used if $q=2^m$. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2. Finally, each entity shall have chosen an ANSI-approved hash function for use with the key derivation function.

Note that validation of the EC domain parameters has not necessarily been carried out by *U* but may instead have been carried out by a party trusted by *U*. Note also that the subscript *e* is a slight abuse of notation. It is used to indicate that the parameters are associated with ephemeral key pairs rather than to indicate that the parameters themselves are ephemeral.

U shall execute the following transformation to agree on keying data with *V*:

Input: The input to the key agreement transformation is:

1. A purported ephemeral EC public key $Q_{e,V}$' owned by *V*.
2. An integer *keydatalen* which is the length in bits of the keying data to be generated.
3. (Optional) A bit string *SharedData* of length *shareddatalen* bits which consists of some data shared by *U* and *V*.

Note that $Q_{e,V}$' may be received either at the start of the execution of the protocol, or at the appropriate stage during the execution of the protocol.

Ingredients: The key agreement transformation employs the key pair generation primitive in Section 5.2.1, public key validation in Section 5.2.2, the Diffie-Hellman primitive in Section 5.4, and the key derivation function in Section 5.6.3.

Actions: Keying data shall be derived as follows:

1. Use the key pair generation primitive in Section 5.2.1 to generate an ephemeral key pair $(d_{e,U},Q_{e,U})$ for the parameters q_e, a_e, b_e, G_e, n_e, and h_e. Send $Q_{e,U}$ to *V*.
2. Verify that the purported key $Q_{e,V}$' is a valid key for the parameters q_e, a_e, b_e, G_e, n_e, and h_e as specified in Section 5.2.2. If the validation primitive rejects the key, output 'invalid' and stop.
3. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z_e \in F_q$ from the private key $d_{e,U}$, the purported public key $Q_{e,V}$', and the parameters q_e, a_e, b_e, G_e, n_e, and h_e. If the primitive outputs 'invalid', output 'invalid' and stop.
4. Convert z_e to a bit string Z_e using the convention specified in Section 4.3.3.
5. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData* of length *keydatalen* bits from the shared secret value Z_e and the shared data [*SharedData*].

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

6.2 1-Pass Diffie-Hellman Scheme

This section specifies the 1-pass Diffie-Hellman scheme.

First the scheme is illustrated in a flow diagram. Figure 3 illustrates the use of the 1-pass Diffie-Hellman scheme.

Figure 3 – 1-Pass Diffie-Hellman Scheme

Next the formal specification of the scheme is given.

The scheme is 'asymmetric', so two transformations are specified. *U* uses the transformation specified in Section 6.2.1 to agree on keying data with *V* if *U* is the protocol's initiator, and *V* uses the transformation specified in Section 6.2.2 to agree on keying data with *U* if *V* is the protocol's responder.

If *U* executes the initiator transformation and *V* simultaneously executes the responder transformation with corresponding keying material as input, then *U* and *V* will compute the same keying data.

Prerequisites: The following are the prerequisites for the use of the scheme:

1. Each entity has an authentic copy of the system's elliptic curve domain parameters q, a, b, G, n, and h along with an indication of the basis used if $q=2^m$. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.

2. Each entity allowed to act as a responder shall be bound to a static key pair associated to the system's elliptic curve domain parameters *q*, *a*, *b*, *G*, *n*, *h*. The binding shall include the validation of the static public key as specified in Section 5.2.2.
3. Each entity shall have chosen an ANSI-approved hash function for use with the key derivation function.

6.2.1 Initiator Transformation

U shall execute the following transformation to agree on keying data with *V* if *U* is the protocol's initiator:

Input: The input to the initiator transformation is:

1. An integer *keydatalen* which is the length in bits of the keying data to be generated.
2. (Optional) A bit string *SharedData* of length shareddatalen bits which consists of some data shared by *U* and *V*.

Ingredients: The initiator transformation employs the key pair generation primitive in Section 5.2.1, the Diffie-Hellman primitive in Section 5.4, and the key derivation function in Section 5.6.3.

Actions: Keying data shall be derived as follows:

1. Use the key pair generation primitive in Section 5.2.1 to generate an ephemeral key pair $(d_{e,U},Q_{e,U})$ for the parameters *q*, *a*, *b*, *G*, *n*, and *h*. Send $Q_{e,U}$ to *V*.
2. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z \in F_q$ from the private key $d_{e,U}$, the static public key $Q_{s,V}$, and the parameters *q*, *a*, *b*, *G*, *n*, and *h*. If the primitive outputs 'invalid', output 'invalid' and stop.
3. Convert *z* to a bit string *Z* using the convention specified in Section 4.3.3.
4. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData* of length *keydatalen* bits from the shared secret value *Z* and the shared data [*SharedData*].

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

6.2.2 Responder Transformation

V shall execute the following transformation to agree on keying data with *U* if *V* is the protocol's responder:

Input: The input to the responder transformation is:

1. A purported ephemeral EC public key $Q_{e,U}$' owned by *U*.
2. An integer *keydatalen* which is the length in bits of the keying data to be generated.
3. (Optional) A bit string *SharedData* of length *shareddatalen* bits which consists of some data shared by *U* and *V*.

Ingredients: The key agreement transformation employs public key validation in Section 5.2.2, the Diffie-Hellman primitive in Section 5.4, and the key derivation function in Section 5.6.3.

Actions: Keying data shall be derived as follows:

1. Verify that the purported key $Q_{e,U}$' is a valid key for the parameters *q*, *a*, *b*, *G*, *n*, and *h* as specified in Section 5.2.2. If the validation primitive rejects the key, output 'invalid' and stop.
2. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z \in F_q$ from the private key $d_{s,V}$, the purported public key $Q_{e,U}$', and the parameters *q*, *a*, *b*, *G*, *n*, and *h*. If the primitive outputs 'invalid', output 'invalid' and stop.
3. Convert *z* to a bit string *Z* using the convention specified in Section 4.3.3.
4. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData* of length *keydatalen* bits from the shared secret value *Z* and the shared data [*SharedData*].

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

6.3 Static Unified Model Scheme

This section specifies the static Unified Model scheme.

First the scheme is illustrated in a flow diagram. Figure 4 illustrates the use of the static Unified Model scheme.

Figure 4 - Static Unified Model Scheme

Next the formal specification of the scheme is given.

The scheme is 'symmetric', so only one transformation is specified. An entity uses this transformation to agree on keying data with another entity no matter whether they are the initiator or the responder.
If entities *U* and *V* simultaneously execute the transformation with corresponding keying material as input, then *U* and *V* will compute the same keying data.
Prerequisites: The following are the prerequisites for the use of the scheme:
1. Each entity has an authentic copy of the system's elliptic curve domain parameters to be used with static keys q_s, a_s, b_s, G_s, n_s, and h_s along with an indication of the basis used if $q=2^m$. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.
2. Each entity shall be bound to a static key pair associated to the system's elliptic curve domain parameters to be used with static keys q_s, a_s, b_s, G_s, n_s, h_s. The binding shall include the validation of the static public key as specified in Section 5.2.2.
3. Each entity shall have chosen an ANSI-approved hash function for use with the key derivation function.

Note that validation of $Q_{s,V}$ has not necessarily been carried out by *U*, but may instead have been carried out, for example, by the CA issuing the binding between *V* and $Q_{s,V}$.
U shall execute the following transformation to agree on keying data with *V*:
Input: The input to the key agreement transformation is:
1. An integer *keydatalen* which is the length in bits of the keying data to be generated.
2. (Optional) A bit string *SharedData* of length *shareddatalen* bits which consists of some data shared by *U* and *V*.

Ingredients: The key agreement transformation employs the Diffie-Hellman primitive in Section 5.4 and the key derivation function in Section 5.6.3.
Actions: Keying data shall be derived as follows:
1. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z_s \in F_q$ from the private key $d_{s,U}$, the public key $Q_{s,V}$, and the parameters q_s, a_s, b_s, G_s, n_s, and h_s. If the primitive outputs 'invalid', output 'invalid' and stop.
2. Convert z_s to a bit string Z_s using the convention specified in Section 4.3.3.
3. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData* of length *keydatalen* bits from the shared secret value Z_s and the shared data [*SharedData*].

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

6.4 Combined Unified Model with Key Confirmation Scheme

This section specifies the combined Unified Model with key confirmation scheme. The scheme is a hybrid of the ephemeral Unified Model scheme and the static Unified Model scheme in which a MAC is used to provide key confirmation.
First the scheme is illustrated in a flow diagram. Figure 5 illustrates the use of the combined Unified Model with key confirmation scheme.

Figure 5 - Combined Unified Model with Key Confirmation Scheme

Next the formal specification of the scheme is given.
The scheme is 'asymmetric', so two transformations are specified. *U* uses the transformation specified in Section 6.4.1 to agree on keying data with *V* if *U* is the protocol's initiator, and *V* uses the transformation specified in Section 6.4.2 to agree on keying data with *U* if *V* is the protocol's responder.
If *U* executes the initiator transformation and *V* simultaneously executes the responder transformation with corresponding keying material as input, then *U* and *V* will compute the same keying data.
Prerequisites: The following are the prerequisites for the use of the scheme:
1. Each entity has an authentic copy of the system's elliptic curve domain parameters to be used with ephemeral keys q_e, a_e, b_e, G_e, n_e, and h_e along with an indication of the basis used if $q=2^m$. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1.

Furthermore the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.

2. *U* has an authentic copy of the system's elliptic curve domain parameters to be used with static keys q_s, a_s, b_s, G_s, n_s, and h_s along with an indication of the basis used if $q=2^m$. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.
3. Each entity shall be bound to a static key pair associated to the system's elliptic curve domain parameters to be used with static keys q_s, a_s, b_s, G_s, n_s, h_s. The binding shall include the validation of the static public key as specified in Section 5.2.2. The key binding shall include a unique identifier for each entity (e.g. distinguished names). All identifiers shall be bit strings of the same length *entlen* bits. Entity *U*'s identifier will be denoted by the bit string *U*.
4. Each entity shall have decided which ANSI-approved MAC scheme to use as specified in Section 5.7. *mackeylen* is used to denote the length of the keys used by the MAC scheme chosen.
5. Each entity shall have chosen an ANSI-approved hash function for use with the key derivation function.

6.4.1 Initiator Transformation

U shall execute the following transformation to agree on keying data with *V* if *U* is the protocol's initiator:

Input: The input to the initiator transformation is:

1. An integer *keydatalen* which is the length in bits of the keying data to be generated.
2. (Optional) A bit string $SharedData_1$ of length *shareddata*1*len* bits and a bit string $SharedData_2$ of length *shareddata*2*len* bits which consist of some data shared by *U* and *V*.

Ingredients: The initiator transformation employs the key pair generation primitive in Section 5.2.1, public key validation in Section 5.2.2, the Diffie-Hellman primitive in Section 5.4, the key derivation function in Section 5.6.3, and one of the MAC schemes in Section 5.7.

Actions: Keying data shall be derived as follows:

1. Use the key pair generation primitive in Section 5.2.1 to generate an ephemeral key pair $(d_{e,U},Q_{e,U})$ for the parameters q_e, a_e, b_e, G_e, n_e, and h_e. Send $Q_{e,U}$ to *V*.
2. Then receive from *V* a purported ephemeral public key $Q_{e,V}$', an optional bit string $Text_1$, and a purported tag $MacTag_1$'. If these values are not received, output 'invalid' and stop.
3. Verify that the purported key $Q_{e,V}$' is a valid key for the parameters q_e, a_e, b_e, G_e, n_e, and h_e as specified in Section 5.2.2. If the validation primitive rejects the key, output 'invalid' and stop.
4. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z_s \in F_q$ from the private key $d_{s,U}$, the public key $Q_{s,V}$, and the parameters q_s, a_s, b_s, G_s, n_s, and h_s. If the primitive outputs 'invalid', output 'invalid' and stop.
5. Convert z_s to a bit string Z_s using the convention specified in Section 4.3.3.
6. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *MacKey* of length *mackeylen* bits from the shared secret value Z_s and the shared data $[SharedData_1]$.
7. Form the bit string consisting of the octet 02_{16}, *V*'s identifier, *U*'s identifier, the bit string *QEV*' corresponding to *V*'s purported ephemeral public key, the bit string *QEU* corresponding to *U*'s ephemeral public key, and if present $Text_1$:
   $MacData_1 = 02_{16} \| V \| U \| QEV' \| QEU \| [Text_1]$.
8. Verify that $MacTag_1$' is the tag for $MacData_1$ under the key *MacKey* using the tag checking transformation of the appropriate MAC scheme specified in Section 5.7. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.
9. Form the bit string consisting of the octet 03_{16}, *U*'s identifier, *V*'s identifier, the bit string *QEU* corresponding to *U*'s ephemeral public key, the bit string *QEV*' corresponding to *V*'s purported ephemeral public key, and optionally a bit string $Text_2$:
   $MacData_2 = 03_{16} \| U \| V \| \text{QEU} \| QEV' \| [Text_2]$.
10. Calculate the tag $MacTag_2$ on $MacData_2$ under the key *MacKey* using the tagging transformation of the appropriate MAC scheme specified in Section 5.7:
    $MacTag_2 = MAC_{MacKey}(MacData_2)$.
    If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send $MacTag_2$ and if present $Text_2$ to *V*.

11. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z_e \in F_q$ from the private key $d_{e,U}$, the purported public key $Q_{e,V}$', and the parameters q_e, a_e, b_e, G_e, n_e, and h_e. If the primitive outputs 'invalid', output 'invalid' and stop.
12. Convert z_e to a bit string Z_e using the convention specified in Section 4.3.3.
13. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData* of length *keydatalen* bits from the shared secret value Z_e and the shared data [$SharedData_2$].

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

6.4.2 Responder Transformation

V shall execute the following transformation to agree on keying data with *U* if *V* is the protocol's responder.

Input: The input to the responder transformation is:

1. A purported ephemeral public key $Q_{e,U}$' owned by *U*.
2. An integer *keydatalen* which is the length in bits of the keying data to be generated.
3. (Optional) A bit string $SharedData_1$ of length *shareddata*1*len* bits and a bit string $SharedData_2$ of length *shareddata*2*len* bits which consist of some data shared by *U* and *V*.

Ingredients: The responder transformation employs the key pair generation primitive in Section 5.2.1, public key validation in Section 5.2.2, the Diffie-Hellman primitive in Section 5.4, the key derivation function in Section 5.6.3, and one of the MAC schemes in Section 5.7.

Actions: Keying data shall be derived as follows:

1. Verify that the purported key $Q_{e,U}$' is a valid key for the parameters q_e, a_e, b_e, G_e, n_e, and h_e as specified in Section 5.2.2. If the primitive rejects the key, output 'invalid' and stop.
2. Use the key pair generation primitive in Section 5.2.1 to generate an ephemeral key pair ($d_{e,V}$,$Q_{e,V}$) for the parameters q_e, a_e, b_e, G_e, n_e, and h_e.
3. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z_s \in F_q$ from the private key $d_{s,V}$, the public key $Q_{s,U}$, and the parameters q_s, a_s, b_s, G_s, n_s, and h_s. If the primitive outputs 'invalid', output 'invalid' and stop.
4. Convert z_s to a bit string Z_s using the convention specified in Section 4.3.3.
5. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *MacKey* of length *mackeylen* bits from the shared secret value Z_s and the shared data [$SharedData_1$].
6. Form the bit string consisting of the octet 02_{16}, *V*'s identifier, *U*'s identifier, the bit string *QEV* corresponding to *V*'s ephemeral public key, the bit string *QEU*' corresponding to *U*'s purported ephemeral public key, and optionally a bit string $Text_1$:
   $MacData_1 = 02_{16} \| V \| U \| QEV \| QEU' \| [Text_1]$.
7. Calculate the tag $MacTag_1$ for $MacData_1$ under the key *MacKey* using the tagging transformation of the appropriate MAC scheme specified in Section 5.7.
   $MacTag_1 = MAC_{MacKey}(MacData_1)$.
   If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send to *U* the ephemeral public key $Q_{e,V}$, if present the bit string $Text_1$, and $MacTag_1$.
8. Then receive from *U* an optional bit string $Text_2$ and a purported tag $MacTag_2$'. If this data is not received, output 'invalid' and stop.
9. Form the bit string consisting of the octet 03_{16}, *U*'s identifier, *V*'s identifier, the bit string *QEU*' corresponding to *U*'s purported ephemeral public key, the bit string *QEV* corresponding to *V*'s ephemeral public key, and if present the bit string $Text_2$:
   $MacData_2 = 03_{16} \| U \| V \| QEU' \| QEV \| [Text_2]$.
10. Verify that $MacTag_2$' is the valid tag on $MacData_2$ under the key *MacKey* using the tag checking transformation of the appropriate MAC scheme specified in Section 5.7. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.
11. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z_e \in F_q$ from the private key $d_{e,V}$, the purported public key $Q_{e,U}$', and the parameters q_e, a_e, b_e, G_e, n_e, and h_e. If the primitive outputs 'invalid', output 'invalid' and stop.
12. Convert z_e to a bit string Z_e using the convention specified in Section 4.3.3.
13. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData* of length *keydatalen* bits from the shared secret value Z_e and the shared data [$SharedData_2$].

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

6.5 1-Pass Unified Model Scheme

This section specifies the 1-pass Unified Model scheme.
First the scheme is illustrated in a flow diagram. Figure 6 illustrates the use of the 1-pass Unified Model scheme.

Figure 6 - 1-Pass Unified Model Scheme

Next the formal specification of the scheme is given.
The scheme is 'asymmetric', so two transformations are specified. *U* uses the transformation specified in Section 6.5.1 to agree on keying data with *V* if *U* is the protocol's initiator, and *V* uses the transformation specified in Section 6.5.2 to agree on keying data with *U* if *V* is the protocol's responder.
The essential difference between the role of the initiator and the role of the responder in the scheme is that the initiator contributes an ephemeral key pair but the responder does not.
If *U* executes the initiator transformation and *V* simultaneously executes the responder transformation with corresponding keying material as input, then *U* and *V* will compute the same keying data.
Prerequisites: The following are the prerequisites for the use of the scheme:

1. Each entity has an authentic copy of the system's elliptic curve domain parameters *q*, *a*, *b*, *G*, *n*, and *h* along with an indication of the basis used if $q=2^m$. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.
2. Each entity shall be bound to a static key pair associated to the system's elliptic curve domain parameters *q*, *a*, *b*, *G*, *n*, *h*. The binding shall include the validation of the static public key as specified in Section 5.2.2.
3. Each entity shall have chosen an ANSI-approved hash function for use with the key derivation function.

6.5.1 Initiator Transformation

U shall execute the following transformation to agree on keying data with *V* if *U* is the protocol's initiator:
Input: The input to the key agreement transformation is:

1. An integer *keydatalen* which is the length in bits of the keying data to be generated.
2. (Optional) A bit string *SharedData* of length *shareddatalen* bits which consists of some data shared by *U* and *V*.

Ingredients: The initiator transformation employs the key pair generation primitive in Section 5.2.1, the Diffie-Hellman primitive in Section 5.4, and the key derivation function in Section 5.6.3.
Actions: Keying data shall be derived as follows:

1. Use the key pair generation primitive in Section 5.2.1 to generate an ephemeral key pair $(d_{e,U},Q_{e,U})$ for the parameters *q*, *a*, *b*, *G*, *n*, and *h*. Send $Q_{e,U}$ to *V*.
2. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z_e \in F_q$ from the private key $d_{e,U}$, the public key $Q_{s,V}$, and the parameters *q*, *a*, *b*, *G*, *n*, and *h*. If the primitive outputs 'invalid', output 'invalid' and stop.
3. Convert z_e to a bit string Z_e using the convention specified in Section 4.3.3.
4. Use the Diffie-Hellman primitive in 5.4 to derive a shared secret value $z_s \in F_q$ from the private key $d_{s,U}$, the public key $Q_{s,V}$, and the parameters *q*, *a*, *b*, *G*, *n*, and *h*. If the primitive outputs 'invalid', output 'invalid' and stop.
5. Convert z_s to a bit string Z_s using the convention specified in Section 4.3.3.
6. Concatenate Z_e and Z_s to form the shared secret value $Z = Z_e||Z_s$.
7. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData* of length *keydatalen* bits from the shared secret value *Z* and the shared data [*SharedData*].

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

6.5.2 Responder Transformation

V shall execute the following transformation to agree on keying data with *U* if *V* is the protocol's responder:

Input: The input to the responder transformation is:
1. A purported ephemeral EC public key $Q_{e,U}$' owned by *U*.
2. An integer *keydatalen* which is the length in bits of the keying data to be generated.
3. (Optional) A bit string *SharedData* of length *shareddatalen* bits which consists of some data shared by *U* and *V*.

Ingredients: The responder transformation employs public key validation as specified in Section 5.2.2, the Diffie-Hellman primitive in Section 5.4, and the key derivation function in Section 5.6.3.

Actions: Keying data shall be derived as follows:
1. Verify that the purported key $Q_{e,U}$' is a valid key for the parameters *q*, *a*, *b*, *G*, *n*, and *h* as specified in Section 5.2.2. If the primitive rejects the key, output 'invalid' and stop.
2. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z_e \in F_q$ from the private key $d_{s,V}$, the purported public key $Q_{e,U}$', and the parameters *q*, *a*, *b*, *G*, *n*, and *h*. If the primitive outputs 'invalid', output 'invalid' and stop.
3. Convert z_e to a bit string Z_e using the convention specified in Section 4.3.3.
4. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z_s \in F_q$ from the private key $d_{s,V}$, the public key $Q_{s,U}$, and the parameters *q*, *a*, *b*, *G*, *n*, and *h*. If the primitive outputs 'invalid', output 'invalid' and stop.
5. Convert z_s to a bit string Z_s using the convention specified in Section 4.3.3.
6. Concatenate Z_e and Z_s to form the shared secret value $Z = Z_e || Z_s$.
7. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData* of length *keydatalen* bits from the shared secret value *Z* and the shared data [*SharedData*].

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

6.6 Full Unified Model Scheme

This section specifies the full Unified Model scheme.
First the scheme is illustrated in a flow diagram. Figure 7 illustrates the use of the full Unified Model scheme.

Figure 7 - Full Unified Model Scheme

Next the formal specification of the scheme is given.
The scheme is 'symmetric', so only one transformation is specified. An entity uses this transformation to agree on keying data with another entity no matter whether they are the initiator or the responder.
If *U* and *V* simultaneously execute the transformation with corresponding keying material as input, then *U* and *V* will compute the same keying data.

Prerequisites: The following are the prerequisites for the use of the scheme:
1. Each entity has an authentic copy of the system's elliptic curve domain parameters to be used with ephemeral keys q_e, a_e, b_e, G_e, n_e, and h_e along with an indication of the basis used if $q=2^m$. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.
2. Each entity has an authentic copy of the system's elliptic curve domain parameters to be used with static keys q_s, a_s, b_s, G_s, n_s, and h_s along with an indication of the basis used if $q=2^m$. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.
3. Each entity shall be bound to a static key pair associated to the system's elliptic curve domain parameters to be used with static keys q_s, a_s, b_s, G_s, n_s, h_s. The binding shall include the validation of the static public key as specified in Section 5.2.2.
4. Each entity shall have chosen an ANSI-approved hash function for use with the key derivation function.

U shall execute the following transformation to agree on keying data with *V*:

Input: The input to the key agreement transformation is:
1. A purported ephemeral EC public key $Q_{e,V}$' owned by *V*.

2. An integer *keydatalen* which is the length in bits of the keying data to be generated.
3. (Optional) A bit string *SharedData* of length *shareddatalen* bits which consists of some data shared by *U* and *V*.

Ingredients: The key agreement transformation employs the key pair generation primitive in Section 5.2.1, public key validation in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3, the Diffie-Hellman primitive in Section 5.4, and the key derivation function in Section 5.6.3.

Actions: Keying data shall be derived as follows:

1. Use the key pair generation primitive in Section 5.2.1 to generate an ephemeral key pair $(d_{e,U},Q_{e,U})$ for the parameters q_e, a_e, b_e, G_e, n_e, and h_e. Send $Q_{e,U}$ to *V*.
2. Verify that the purported key $Q_{e,V}$' is a valid key for the parameters q_e, a_e, b_e, G_e, n_e, and h_e as specified in Section 5.2.2. If the validation primitive rejects the key, output 'invalid' and stop.
3. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z_e \in F_q$ from the private key $d_{e,U}$, the purported public key $Q_{e,V}$', and the parameters q_e, a_e, b_e, G_e, n_e, and h_e. If the primitive outputs 'invalid', output 'invalid' and stop.
4. Convert z_e to a bit string Z_e using the convention specified in Section 4.3.3.
5. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z_s \in F_q$ from the private key $d_{s,U}$, the public key $Q_{s,V}$, and the parameters q_s, a_s, b_s, G_s, n_s, and h_s. If the primitive outputs 'invalid', output 'invalid' and stop.
6. Convert z_s to a bit string Z_s using the convention specified in Section 4.3.3.
7. Concatenate Z_e and Z_s to form the shared secret value $Z = Z_e||Z_s$.
8. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData* of length *keydatalen* bits from the shared secret value *Z* and the shared data [*SharedData*].

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

6.7 Full Unified Model with Key Confirmation Scheme

This section specifies the full Unified Model with key confirmation scheme. The scheme adds flows to the full Unified Model scheme so that explicit key authentication may be supplied. A MAC scheme is used to provide key confirmation.

First the scheme is illustrated in a flow diagram. Figure 8 illustrates the use of the full Unified Model with key confirmation scheme.

Figure 8 - Full Unified Model with Key Confirmation Scheme

Next the formal specification of the scheme is given.

The scheme is 'asymmetric', so two transformations are specified. *U* uses the transformation specified in Section 6.7.1 to agree on keying data with *V* if *U* is the protocol's initiator, and *V* uses the transformation specified in Section 6.7.2 to agree keying data with *U* if *V* is the protocol's responder.

If *U* executes the initiator transformation and *V* simultaneously executes the responder transformation with corresponding keying material as input, then *U* and *V* will compute the same keying data.

Prerequisites: The following are the prerequisites for the use of the scheme:

1. Each entity has an authentic copy of the system's elliptic curve domain parameters to be used with ephemeral keys q_e, a_e, b_e, G_e, n_e, and h_e along with an indication of the basis used if $q=2^m$. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.
2. Each entity has an authentic copy of the system's elliptic curve domain parameters to be used with static keys q_s, a_s, b_s, G_s, n_s, and h_s along with an indication of the basis used if $q=2^m$. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.
3. Each entity shall be bound to a static key pair associated to the system's elliptic curve domain parameters to be used with static keys q_s, a_s, b_s, G_s, n_s, h_s. The binding shall include the validation of the static public key

as specified in Section 5.2.2. The key binding shall include a unique identifier for each entity (e.g. distinguished names). All identifiers shall be bit strings of the same length *entlen* bits. Entity *U*'s identifier will be denoted by the bit string *U*.

4. Each entity shall have decided which ANSI-approved MAC scheme to use as specified in Section 5.7. *mackeylen* will denote the length of the keys used by the chosen MAC scheme.
5. Each entity shall have chosen an ANSI-approved hash function for use with the key derivation function.

6.7.1 Initiator Transformation

U shall execute the following transformation to agree on keying data with *V* if *U* is the protocol's initiator:

Input: The input to the initiator transformation is:

1. An integer *keydatalen* which is the length in bits of the keying data to be generated.
2. (Optional) A bit string *SharedData* of length *shareddatalen* bits which consists of some data shared by *U* and *V*.

Ingredients: The initiator transformation employs the key pair generation primitive in Section 5.2.1, public key validation in Section 5.2.2, the Diffie-Hellman primitive in Section 5.4, the key derivation function in Section 5.6.3, and one of the MAC schemes in Section 5.7.

Actions: Keying data shall be derived as follows:

1. Use the key pair generation primitive in Section 5.2.1 to generate an ephemeral key pair $(d_{e,U},Q_{e,U})$ for the parameters q_e, a_e, b_e, G_e, n_e, and h_e. Send $Q_{e,U}$ to *V*.
2. Then receive from *V* a purported ephemeral public key $Q_{e,V}$', an optional bit string $Text_1$, and a purported tag $MacTag_1$'. If these values are not received, output 'invalid' and stop.
3. Verify that the purported key $Q_{e,V}$' is a valid key for the parameters q_e, a_e, b_e, G_e, n_e, and h_e as specified in Section 5.2.2. If the validation primitive rejects the key, output 'invalid' and stop.
4. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z_e \in F_q$ from the private key $d_{e,U}$, the purported public key $Q_{e,V}$', and the parameters q_e, a_e, b_e, G_e, n_e, and h_e. If the primitive outputs 'invalid', output 'invalid' and stop.
5. Convert z_e to a bit string Z_e using the convention specified in Section 4.3.3.
6. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z_s \in F_q$ from the private key $d_{s,U}$, the public key $Q_{s,V}$, and the parameters q_s, a_s, b_s, G_s, n_s, and h_s. If the primitive outputs 'invalid', output 'invalid' and stop.
7. Convert z_s to a bit string Z_s using the convention specified in Section 4.3.3.
8. Form the shared secret bit string *Z* as $Z = Z_e || Z_s$.
9. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData*! of length *mackeylen+keydatalen* bits from the shared secret value *Z* and the shared data [*SharedData*].
10. Parse the leftmost *macdatalen* bits of *KeyData*! as a MAC key *MacKey* and the remaining bits as keying data *KeyData*.
11. Form the bit string consisting of the octet 02_{16}, *V*'s identifier, *U*'s identifier, the bit string *QEV*' corresponding to *V*'s purported ephemeral public key, the bit string *QEU* corresponding to *U*'s ephemeral public key, and if present $Text_1$:
    $MacData_1 = 02_{16} \| V \| U \| QEV' \| QEU \| [Text_1]$.
12. Verify that $MacTag_1$' is the tag for $MacData_1$ under the key *MacKey* using the tag checking transformation of the appropriate MAC scheme specified in Section 5.7. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.
13. Form the bit string consisting of the octet 03_{16}, *U*'s identifier, *V*'s identifier, the bit string *QEU* corresponding to *U*'s ephemeral public key, the bit string *QEV*' corresponding to *V*'s purported ephemeral public key, and optionally a bit string $Text_2$:
    $MacData_2 = 03_{16} \| U \| V \| QEU \| QEV' \| [Text_2]$.
14. Calculate the tag $MacTag_2$ on $MacData_2$ under the key *MacKey* using the tagging transformation of the appropriate MAC scheme specified in Section 5.7:
    $MacTag_2 = MAC_{MacKey}(MacData_2)$.
    If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send $MacTag_2$ and if present $Text_2$ to *V*.

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

6.7.2 Responder Transformation

V shall execute the following transformation to agree on keying data with *U* if *V* is the protocol's responder:

Input: The input to the responder transformation is:

1. A purported ephemeral public key $Q_{e,U}$' owned by *U*.
2. An integer *keydatalen* which is the length in bits of the keying data to be generated.
3. (Optional) A bit string *SharedData* of length *shareddatalen* bits which consists of some data shared by *U* and *V*.

Ingredients: The responder transformation employs the key pair generation primitive in Section 5.2.1, public key validation in Section 5.2.2, the Diffie-Hellman primitive in Section 5.4, the key derivation function in Section 5.6.3, and one of the MAC schemes in Section 5.7.

Actions: Keying data shall be derived as follows:

1. Verify that the purported key $Q_{e,U}$' is a valid key for the parameters q_e, a_e, b_e, G_e, n_e, and h_e as specified in Section 5.2. If the validation primitive rejects the key, output 'invalid' and stop.
2. Use the key pair generation primitive in Section 5.2.1 to generate an ephemeral key pair ($d_{e,V}$,$Q_{e,V}$) for the parameters q_e, a_e, b_e, G_e, n_e, and h_e.
3. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z_e \in F_q$ from the private key $d_{e,V}$, the purported public key $Q_{e,U}$', and the parameters q_e, a_e, b_e, G_e, n_e, and h_e. If the primitive outputs 'invalid', output 'invalid' and stop.
4. Convert z_e to a bit string Z_e using the convention specified in Section 4.3.3.
5. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z_s \in F_q$ from the private key $d_{s,V}$, the public key $Q_{s,U}$, and the parameters q_s, a_s, b_s, G_s, n_s, and h_s. If the primitive outputs 'invalid', output 'invalid' and stop.
6. Convert z_s to a bit string Z_s using the convention specified in Section 4.3.3.
7. Form the shared secret bit string *Z* as $Z = Z_e || Z_s$.
8. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData*! of length *mackeylen+keydatalen* bits from the shared secret value *Z* and the shared data [*SharedData*].
9. Parse the leftmost *mackeylen* bits of *KeyData*! as a MAC key *MacKey* and the remaining bits as keying data *KeyData*.
10. Form the bit string consisting of the octet 02_{16}, *V*'s identifier, *U*'s identifier, the bit string *QEV* corresponding to *V*'s ephemeral public key, the bit string *QEU*' corresponding to *U*'s purported ephemeral public key, and optionally a bit string $Text_1$:
    $MacData_1 = 02_{16} \| V \| U \| QEV \| QEU' \| [Text_1]$.
11. Calculate the tag $MacTag_1$ for $MacData_1$ under the key *MacKey* using the tagging transformation of the appropriate MAC scheme specified in Section 5.7.
    $MacTag_1 = MAC_{MacKey}(MacData_1)$.
    If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send to *U* the ephemeral public key $Q_{e,V}$, if present the bit string $Text_1$, and $MacTag_1$.
12. Then receive from *U* an optional bit string $Text_2$ and a purported tag $MacTag_2$'. If this data is not received, output 'invalid' and stop.
13. Form the bit string consisting of the octet 03_{16}, *U*'s identifier, *V*'s identifier, the bit string *QEU*' corresponding to *U*'s purported ephemeral public key, the bit string *QEV* corresponding to *V*'s ephemeral public key, and if present the bit string $Text_2$:
    $MacData_2 = 03_{16} \| U \| V \| QEU' \| QEV \| [Text_2]$.
14. Verify that $MacTag_2$' is the valid tag on $MacData_2$ under the key *MacKey* using the tag checking transformation of the appropriate MAC scheme specified in Section 5.7. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

6.8 Station-to-Station Scheme

This section specifies the Station-to-Station scheme. The scheme uses a signature scheme to authenticate the ephemeral Unified Model scheme and provide mutual explicit key authentication.

First the scheme is illustrated in a flow diagram. Figure 9 illustrates the use of the Station-to-Station scheme.

Figure 9 – Station-to-Station Scheme

Next the formal specification of the scheme is given.

The scheme is 'asymmetric', so two transformations are specified. *U* uses the transformation specified in Section 6.8.1 to agree keying data with *V* if *U* is the protocol's initiator, and *V* uses the transformation specified in Section 6.8.2 to agree keying data with *U* if *V* is the protocol's responder.
If *U* executes the initiator transformation and *V* simultaneously executes the responder transformation with corresponding keying material as input, then *U* and *V* will compute the same keying data.
Prerequisites: The following are the prerequisites for the use of the scheme:

1. Each entity has an authentic copy of the system's elliptic curve domain parameters to be used with ephemeral keys q_e, a_e, b_e, G_e, n_e, and h_e. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.
2. Each entity has an authentic copy of the system's elliptic curve domain parameters to be used with a signature scheme q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, and h_{sig} along with an indication of the basis used if $q=2^m$. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.
3. Each entity shall be bound to a static signing key pair associated to the system's elliptic curve domain parameters for signing q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, and h_{sig}. The binding may include the validation of the public signature as specified in Section 5.2.2. The key binding shall include a unique identifier for each entity (e.g. distinguished names). All identifiers shall be bit strings of the same length *entlen* bits. Entity *U*'s identifier will be denoted by the bit string *U*.
4. Each entity shall have decided which ANSI-approved MAC scheme to use as specified in Section 5.7. mackeylen will denote the length of the keys used by the MAC scheme chosen.
5. Each entity shall have chosen an ANSI-approved hash function for use with the key derivation function.

6.8.1 Initiator Transformation

U shall execute the following transformation to agree keying data with *V* if *U* is the protocol's initiator:
Input: The input to the initiator transformation is:

1. An integer *keydatalen* which is the length in bits of the keying data to be generated.
2. (Optional) A bit string *SharedData* of length *shareddatalen* which consists of some data shared by *U* and *V*.

Ingredients: The initiator transformation employs the key pair generation primitive in Section 5.2., public key validation in Section 5.2.2, the Diffie-Hellman primitive in Section 5.4, the key derivation function in Section 5.6.3, one of the MAC schemes in Section 5.7, and the signature scheme specified in Section 5.9.
Actions: Keying data shall be derived as follows:

1. Use the key pair generation primitive in Section 5.2.1 to generate an ephemeral key pair ($d_{e,U}$; $Q_{e,U}$) for the parameters q_e, a_e, b_e, G_e, n_e, and h_e. Send $Q_{e,U}$ to *V* .
2. Then receive from *V* a purported ephemeral public key $Q_{e,V}$', a pair of integers $rsig_1$' and $ssig_1$' purporting to be a signature, a purported tag $MacTag_1$', and an optional bit string $Text_1$. If this data is not received, output 'invalid' and stop.
3. Verify that the purported key $Q_{e,V}$' is a valid key for the parameters q_e, a_e, b_e, G_e, n_e, and h_e as specified in Section 5.2. If the validation primitive rejects the key, output 'invalid' and stop.
4. Form the bit string consisting of the bit string *QEV*' corresponding to *V*'s purported ephemeral public key, the bit string *QEU* corresponding to *U*'s ephemeral public key, *U*'s identifier, and if present $Text_1$:
   $Data_1 = QEV' \,||\, QEU \,||\, U \,||\, [Text_1]$.
5. Verify that $rsig_1$' and $ssig_1$' are a valid signature of $Data_1$ under *V*'s public signature key $Q_{sig,V}$ corresponding to the EC domain parameters q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, and h_{sig}, using the verifying transformation of the signature scheme in Section 5.9. If the verifying transformation outputs 'invalid', output 'invalid' and stop.

6. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value z_e in F_q from the private key $d_{e,U}$, the purported ephemeral public key $Q_{e,V}$', and the parameters q_e, a_e, b_e, G_e, n_e, and h_e. If the primitive outputs 'invalid', output 'invalid' and stop.
7. Convert z_e to a bit string Z_e using the convention specified in Section 4.3.3.
8. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData*! of length *mackeylen+keydatalen* bits from the shared secret value Z_e and the shared data [*SharedData*].
9. Parse the leftmost *macdatalen* bits of *KeyData*! as a MAC key *MacKey* and the remaining bits as keying data *KeyData*.
10. Verify that $MacTag_1$' is the tag for $Data_1$' under the key *MacKey* using the tag checking transformation of the appropriate MAC scheme specified in Section 5.7. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.
11. Form the bit string consisting of the bit string *QEU* corresponding to *U*'s ephemeral public key, the bit string *QEV*' corresponding to *V*'s purported ephemeral public key, *V*'s identifier, and optionally a bit string $Text_2$:

    $Data_2 = QEU \,||\, QEV' \,||\, V \,||\, [Text_2]$.
12. Sign $Data_2$ using *U*'s private signing key $d_{sig,U}$ corresponding to the parameters q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, and h_{sig}, using the signing transformation of the signature scheme in Section 5.9. If the signing transformation outputs 'invalid', output 'invalid' and stop. Otherwise the signing transformation outputs the integers $rsig_2$ and $ssig_2$ as the signature of $Data_2$.
13. Calculate the tag $MacTag_2$ on $Data_2$ under the key *MacKey* using the tagging transformation of the appropriate MAC scheme specified in Section 5.7:

    $MacTag_2 = MAC_{MacKey}(Data_2)$.

    If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send $rsig_2$, $ssig_2$, $MacTag_2$ and if present $Text_2$ to *V*.

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

6.8.2 Responder Transformation

V shall execute the following transformation to agree keying data with *U* if *V* is the protocol's responder:

Input: The input to the responder transformation is:

1. A purported ephemeral public key $Q_{e,U}$' owned by *U*.
2. An integer *keydatalen* which is the length in bits of the keying data to be generated.
3. (Optional) A bit string *SharedData* of length *shareddatalen* which consists of some data shared by *U* and *V*.

Ingredients: The responder transformation employs the key pair generation primitive in Section 5.2., public key validation in Section 5.2.2, the Diffie-Hellman primitive in Section 5.4, the key derivation function in Section 5.6.3, one of the MAC schemes in Section 5.7, and the signature scheme specified in Section 5.9.

Actions: Keying data shall be derived as follows:

1. Verify that the purported key $Q_{e,U}$' is a valid key for the parameters q_e, a_e, b_e, G_e, n_e, and h_e as specified in Section 5.2. If the validation primitive rejects the key, output 'invalid' and stop.
2. Use the key pair generation primitive in Section 5.2.1 to generate an ephemeral key pair ($d_{e,V}$; $Q_{e,V}$) for the parameters q_e, a_e, b_e, G_e, n_e, and h_e.
3. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value z_e in F_q from the private key $d_{e,V}$, the purported public key $Q_{e,U}$', and the parameters q_e, a_e, b_e, G_e, n_e, and h_e. If the primitive outputs 'invalid', output 'invalid' and stop.
4. Convert z_e to a bit string Z_e using the convention specified in Section 4.3.3.
5. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData*! of length *mackeylen+keydatalen* bits from the shared secret value Z_e and the shared data [*SharedData*].
6. Parse the leftmost *macdatalen* bits of *KeyData*! as a MAC key *MacKey* and the remaining bits as keying data *KeyData*.
7. Form the bit string consisting of the bit string *QEV* corresponding to *V*'s ephemeral public key, the bit string *QEU*' corresponding to *U*'s purported ephemeral public key, *U*'s identifier, and if optionally a bit string $Text_1$:

    $Data_1 = QEV \,||\, QEU' \,||\, U \,||\, [Text_1]$.

8. Sign $Data_1$ using V's private signing key $d_{sig,V}$ corresponding to the parameters q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, and h_{sig}, using the signing transformation of the signature scheme in Section 5.9. If the signing transformation outputs 'invalid', output 'invalid' and stop. Otherwise the signing transformation outputs the integers $rsig_1$ and $ssig_1$ as the signature of $Data_1$.
9. Calculate the tag $MacTag_1$ on $Data_1$ under the key *MacKey* using the tagging transformation of the appropriate MAC scheme specified in Section 5.7:
   $MacTag_1 = MAC_{MacKey}(Data_1)$.
   If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send $Q_{e,V}$, $rsig_1$, $ssig_1$, $MacTag_1$ and if present $Text_1$ to U.
10. Then receive from U a pair of integers $rsig_1$' and $ssig_1$' purporting to be a signature, a purported tag $MacTag_2$', and an optional bit string $Text_2$. If these values are not received, output 'invalid' and stop.
11. Form the bit string consisting of the bit string QEU' corresponding to U's purported ephemeral public key, the bit string QEV corresponding to V's ephemeral public key, V's identifier, and if present the bit string $Text_2$:
    $Data_2 = QEU' \| QEV \| V \| [Text_2]$.
12. Verify that $rsig_2$' and $ssig_2$' are a valid signature of $Data_2$ under U's public signature key $Q_{sig,U}$ corresponding to the EC domain parameters q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, and h_{sig}, using the verifying transformation of the signature scheme in Section 5.9. If the verifying transformation outputs 'invalid', output 'invalid' and stop.
13. Verify that $MacTag_2$' is the tag for $Data_2$' under the key *MacKey* using the tag checking transformation of the appropriate MAC scheme specified in Section 5.7. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

6.9 1-Pass MQV Scheme

This section specifies the 1-pass MQV scheme.
First the scheme is illustrated in a flow diagram. Figure 10 illustrates the use of the 1-pass MQV scheme.

Figure 10 - 1-Pass MQV Scheme

Next a formal specification of the scheme is given.
The scheme is 'asymmetric', so two transformations are specified. U uses the transformation specified in Section 6.9.1 to agree on keying data with V if U is the protocol's initiator, and V uses the transformation specified in Section 6.9.2 to agree keying data with U if V is the protocol's responder.
The essential difference between the role of the initiator and the role of responder in the scheme is that the initiator contributes an ephemeral key pair but the responder does not.
If U executes the initiator transformation and V simultaneously executes the responder transformation with corresponding keying material as input, then U and V will compute the same keying data.
Prerequisites: The following are the prerequisites for the use of the scheme:

1. Each entity has an authentic copy of the system's elliptic curve domain parameters q, a, b, G, n, and h along with an indication of the basis used if $q=2^m$. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.
2. Each entity shall be bound to a static key pair associated to the system's elliptic curve domain parameters q, a, b, G, n, h. The binding shall include the validation of the static public key as specified in Section 5.2.2.
3. Each entity shall have chosen an ANSI-approved hash function for use with the key derivation function.

6.9.1 Initiator Transformation

U shall execute the following transformation to agree on keying datawith V if U is the protocol's initiator:
Input: The input to the key agreement transformation is:

1. An integer *keydatalen* which is the length in bits of the keying data to be generated.

2. (Optional) A bit string *SharedData* of length *shareddatalen* bits which consists of some data shared by *U* and *V*.

Ingredients: The initiator transformation employs the key pair generation primitive in Section 5.2.1, the MQV primitive in Section 5.5, the associate value function in Section 5.6.1, and the key derivation function in Section 5.6.3.

Actions: Keying data shall be derived as follows:

1. Use the key pair generation primitive in Section 5.2.1 to generate an ephemeral key pair $(d_{e,U},Q_{e,U})$ for the parameters *q*, *a*, *b*, *G*, *n*, and *h*. Send $Q_{e,U}$ to *V*.
2. Use the MQV primitive in Section 5.5 to derive a shared secret value $z \in F_q$ from the key pairs $(d_{1,U},Q_{1,U})=(d_{s,U},Q_{s,U})$ and $(d_{2,U},Q_{2,U})=(d_{e,U},Q_{e,U})$, the public key $Q_{1,V}=Q_{2,V}=Q_{s,V}$, and the parameters *q*, *a*, *b*, *G*, *n*, and *h*. If the MQV primitive outputs 'invalid', output 'invalid' and stop.
3. Convert *z* to a bit string *Z* using the convention specified in Section 4.3.3.
4. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData* of length *keydatalen* bits from the shared secret value *Z* and the shared data [*SharedData*].

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

6.9.2 Responder Transformation

V shall execute the following transformation to agree on keying data with *U* if *V* is the protocol's responder:

Input: The input to the responder transformation is:

1. A purported ephemeral EC public key $Q_{e,U}$' owned by *U*.
2. An integer *keydatalen* which is the length in bits of the keying data to be generated.
3. (Optional) A bit string *SharedData* of length *shareddatalen* bits which consists of some data shared by *U* and *V*.

Ingredients: The responder transformation employs public key validation in Section 5.2.2, the MQV primitive in Section 5.5, the associate value function in Section 5.6.1, and the key derivation function in Section 5.6.3.

Actions: Keying data shall be derived as follows:

1. Verify that the purported key $Q_{e,U}$' is a valid key for the parameters *q*, *a*, *b*, *G*, *n*, and *h* as specified in Section 5.2.2. If the primitive rejects the key, output 'invalid' and stop.
2. Use the MQV primitive in Section 5.5 to derive a shared secret value $z \in F_q$ from the key pair $(d_{1,V},Q_{1,V})=(d_{2,V},Q_{2,V})=(d_{s,V},Q_{s,V})$, the public keys $Q_{1,U}=Q_{s,U}$ and $Q_{2,U}=Q_{e,U}$', and the parameters *q*, *a*, *b*, *G*, *n*, and *h*. If the MQV primitive outputs 'invalid', output 'invalid' and stop.
3. Convert *z* to a bit string *Z* using the convention specified in Section 4.3.3.
4. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData* of length *keydatalen* bits from the shared secret value *Z* and the shared data [*SharedData*].

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

6.10 Full MQV Scheme

This section specifies the full MQV scheme.

First the scheme is illustrated in a flow diagram. Figure 11 illustrates the use of the full MQV scheme.

Figure 11 - Full MQV Scheme

Next the formal specification of the scheme is given.

The scheme is 'symmetric', so only one transformation is specified. An entity uses this transformation to agree on keying data with another entity no matter whether they are the initiator or the responder.

If *U* and *V* simultaneously execute the transformation with corresponding keying material as input, then *U* and *V* will compute the same keying data.

Prerequisites: The following are the prerequisites for the use of the scheme:

1. Each entity has an authentic copy of the system's elliptic curve domain parameters *q*, *a*, *b*, *G*, *n*, and *h* along with an indication of the basis used if $q=2^m$. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.

2. Each entity shall be bound to a static key pair associated to the system's elliptic curve domain parameters *q*, *a*, *b*, *G*, *n*, *h*. The binding shall include the validation of the static public key as specified in Section 5.2.2.
3. Each entity shall have chosen an ANSI-approved hash function for use with the key derivation function.

U shall execute the following transformation to agree on keying data with *V*:

Input: The input to the key agreement transformation is:

1. A purported ephemeral EC public key $Q_{e,V}$' owned by *V*.
2. An integer *keydatalen* which is the length in bits of the keying data to be generated.
3. (Optional) A bit string *SharedData* of length *shareddatalen* bits which consists of some data shared by *U* and *V*.

Ingredients: The key agreement transformation employs the key pair generation primitive in Section 5.2.1, public key validation in Section 5.2.2, the MQV primitive in Section 5.5, the associate value function in Section 5.6.1, and the key derivation function in Section 5.6.3.

Actions: Keying data shall be derived as follows:

1. Use the key pair generation primitive in Section 5.2.1 to generate an ephemeral key pair $(d_{e,U},Q_{e,U})$ for the parameters *q*, *a*, *b*, *G*, *n*, and *h*. Send $Q_{e,U}$ to *V*.
2. Verify that the purported key $Q_{e,V}$' is a valid key for the parameters *q*, *a*, *b*, *G*, *n*, and *h* as specified in Section 5.2.2. If the validation primitive rejects the key, output 'invalid' and stop.
3. Use the MQV primitive in Section 5.5 to derive a shared secret value $z \in F_q$ from the key pairs $(d_{1,U},Q_{1,U})=(d_{s,U},Q_{s,U})$ and $(d_{2,U},Q_{2,U})=(d_{e,U},Q_{e,U})$, the public keys $Q_{1,V}=Q_{s,V}$ and $Q_{2,V}=Q_{e,V}$', and the parameters *q*, *a*, *b*, *G*, *n*, and *h*. If the MQV primitive outputs 'invalid', output 'invalid' and stop.
4. Convert *z* to a bit string *Z* using the convention specified in Section 4.3.3.
5. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData* of length *keydatalen* bits from the shared secret value *Z* and the shared data [*SharedData*].

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

6.11 Full MQV with Key Confirmation Scheme

This section specifies the full MQV with key confirmation scheme. The scheme adds flows to the full MQV scheme so that explicit key authentication may be supplied. A MAC scheme is used to provide key confirmation.

First the scheme is illustrated in a flow diagram. Figure 12 illustrates the use of the full MQV with key confirmation scheme.

Figure 12 - Full MQV with Key Confirmation Scheme

Next the formal specification of the scheme is given.

The scheme is 'asymmetric', so two transformations are specified. *U* uses the transformation specified in Section 6.11.1 to agree on keying data with *V* if *U* is the protocol's initiator, and *V* uses the transformation specified in Section 6.11.2 to agree keying data with *U* if *V* is the protocol's responder.

If *U* executes the initiator transformation and *V* simultaneously executes the responder transformation with corresponding keying material as input, then *U* and *V* will compute the same keying data.

Prerequisites: The following are the prerequisites for the use of the scheme:

1. Each entity has an authentic copy of the system's elliptic curve domain parameters *q*, *a*, *b*, *G*, *n*, and *h* along with an indication of the basis used if $q=2^m$. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.
2. Each entity shall be bound to a static key pair associated to the system's elliptic curve domain parameters *q*, *a*, *b*, *G*, *n*, *h*. The binding shall include the validation of the static public key as specified in Section 5.2. The key binding shall include a unique identifier for each entity (e.g. distinguished names). All identifiers shall be bit strings of the same length *entlen* bits. Entity *U*'s identifier will be denoted by the bit string *U*.
3. Each entity shall have decided which ANSI-approved MAC scheme to use as specified in Section 5.7. *mackeylen* denotes the length of keys used by the chosen MAC scheme.
4. Each entity shall have chosen an ANSI-approved hash function for use with the key derivation function.

6.11.1 Initiator Transformation

U shall execute the following transformation to agree on keying data with *V* if *U* is the protocol's initiator:

Input: The input to the initiator transformation is:

1. An integer *keydatalen* which is the length in bits of the keying data to be generated.
2. (Optional) A bit string *SharedData* of length *shareddatalen* bits which consists of some data shared by *U* and *V*.

Ingredients: The initiator transformation employs the key pair generation primitive in Section 5.2.1, public key validation in Section 5.2.2, the MQV primitive in Section 5.5, the associate value function in Section 5.6.1, the key derivation function in Section 5.6.3, and one of the MAC schemes in Section 5.7.

Actions: Keying data shall be derived as follows:

1. Use the key pair generation primitive in Section 5.2.1 to generate an ephemeral key pair $(d_{e,U},Q_{e,U})$ for the parameters *q*, *a*, *b*, *G*, *n*, and *h*. Send $Q_{e,U}$ to *V*.
2. Then receive from *V* a purported ephemeral public key $Q_{e,V}$', an optional bit string $Text_1$, and a purported tag $MacTag_1$'. If these values are not received, output 'invalid' and stop.
3. Verify that the purported key $Q_{e,V}$' is a valid key for the parameters *q*, *a*, *b*, *G*, *n*, and *h* as specified in Section 5.2.2. If the validation primitive rejects the key, output 'invalid' and stop.
4. Use the MQV primitive in Section 5.5 to derive a shared secret value $z \in F_q$ from the key pairs $(d_{1,U},Q_{1,U})=(d_{s,U},Q_{s,U})$ and $(d_{2,U},Q_{2,U})=(d_{e,U},Q_{e,U})$, the public keys $Q_{1,V}=Q_{s,V}$ and $Q_{2,V}=Q_{e,V}$', and the parameters *q*, *a*, *b*, *G*, *n*, and *h*. If the MQV primitive outputs 'invalid', output 'invalid' and stop.
5. Convert *z* to a bit string *Z* using the convention specified in Section 4.3.3.
6. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData*! of length *mackeylen+keydatalen* bits from the shared secret value *Z* and the shared data [*SharedData*].
7. Parse the leftmost *mackeylen* bits of *KeyData*! as a MAC key *MacKey* and the remaining bits as keying data *KeyData*.
8. Form the bit string consisting of the octet 02_{16}, *V*'s identifier, *U*'s identifier, the bit string *QEV*' corresponding to *V*'s purported ephemeral public key, the bit string *QEU* corresponding to *U*'s ephemeral public key, and if present $Text_1$:
   $MacData_1 = 02_{16} \| V \| U \| QEV' \| QEU \| [Text_1]$.
9. Verify that $MacTag_1$' is the tag for $MacData_1$ under the key *MacKey* using the tag checking transformation of the appropriate MAC scheme specified in Section 5.7. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.
10. Form the bit string consisting of the octet 03_{16}, *U*'s identifier, *V*'s identifier, the bit string *QEU* corresponding to *U*'s ephemeral public key, the bit string *QEV*' corresponding to *V*'s purported ephemeral public key, and optionally a bit string $Text_2$:
    $MacData_2 = 03_{16} \| U \| V \| QEU \| QEV' \| [Text_2]$.
11. Calculate the tag $MacTag_2$ on $MacData_2$ under the key *MacKey* using the tagging transformation of the appropriate MAC scheme specified in Section 5.7:
    $MacTag_2 = MAC_{MacKey}(MacData_2)$.
    If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send $MacTag_2$ and if present $Text_2$ to *V*.

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

6.11.2 Responder Transformation

V shall execute the following transformation to agree on keying data with *U* if *V* is the protocol's responder:

Input: The input to the responder transformation is:

1. A purported ephemeral public key $Q_{e,U}$' owned by *U*.
2. An integer *keydatalen* which is the length in bits of the keying data to be generated.
3. (Optional) A bit string *SharedData* of length *shareddatalen* bits which consists of some data shared by *U* and *V*.

Ingredients: The responder transformation employs the key pair generation primitive in Section 5.2.1, public key validation in Section 5.2.2, the MQV primitive in Section 5.5, the associate value function in Section 5.6.1, the key derivation function in Section 5.6.3, and one of the MAC schemes in Section 5.7.

Actions: Keying data shall be derived as follows:

1. Verify that the purported key $Q_{e,U}$' is a valid key for the parameters *q*, *a*, *b*, *G*, *n*, and *h* as specified in Section 5.2.2. If the validation primitive rejects the key, output 'invalid' and stop.
2. Use the key pair generation primitive in Section 5.2.1 to generate an ephemeral key pair $(d_{e,V},Q_{e,V})$ for the parameters *q*, *a*, *b*, *G*, *n*, and *h*.
3. Use the MQV primitive in Section 5.5 to derive a shared secret value $z \in F_q$ from the key pairs $(d_{1,V},Q_{1,V})=(d_{s,V},Q_{s,V})$ and $(d_{2,V},Q_{2,V})=(d_{e,V},Q_{e,V})$, the public keys $Q_{1,U}=Q_{s,U}$ and $Q_{2,U}=Q_{e,U}$', and the parameters *q*, *a*, *b*, *G*, *n*, and *h*. If the MQV primitive outputs 'invalid', output 'invalid' and stop.
4. Convert *z* to a bit string *Z* using the convention specified in Section 4.3.3.
5. Use the key derivation function in Section 5.6.3 with the established hash function to derive keying data *KeyData*! of length *mackeylen+keydatalen* bits from the shared secret value *Z* and the shared data [*SharedData*].
6. Parse the leftmost *mackeylen* bits of *KeyData*! as a MAC key *MacKey* and the remaining bits as keying data *KeyData*.
7. Form the bit string consisting of the octet 02_{16}, *V*'s identifier, *U*'s identifier, the bit string *QEV* corresponding to *V*'s ephemeral public key, the bit string *QEU*' corresponding to *U*'s purported ephemeral public key, and optionally a bit string *Text*_1:
   $MacData_1 = 02_{16} \| V \| U \| QEV \| QEU' \| [Text_1]$.
8. Calculate the tag $MacTag_1$ for $MacData_1$ under the key *MacKey* using the tagging transformation of the appropriate MAC scheme specified in Section 5.7.
   $MacTag_1 = MAC_{MacKey}(MacData_1)$.
   If the tagging transformation outputs 'invalid', output 'invalid' and stop. Send to *U* the ephemeral public key $Q_{e,V}$, if present the bit string $Text_1$, and $MacTag_1$.
9. Then receive from *U* an optional bit string $Text_2$ and a purported tag $MacTag_2$'. If this data is not received, output 'invalid' and stop.
10. Form the bit string consisting of the octet 03_{16}, *U*'s identifier, *V*'s identifier, the bit string *QEU*' corresponding to *U*'s purported ephemeral public key, the bit string *QEV* corresponding to *V*'s ephemeral public key, and if present the bit string $Text_2$:
    $MacData_2 = 03_{16} \| U \| V \| QEU' \| QEV \| [Text_2]$.
11. Verify that $MacTag_2$' is the valid tag on $MacData_2$ under the key *MacKey* using the tag checking transformation of the appropriate MAC scheme specified in Section 5.7. If the tag checking transformation outputs 'invalid', output 'invalid' and stop.

Output: The bit string *KeyData* as the keying data of length *keydatalen* bits.

7 Key Transport Schemes

This section describes the key transport schemes specified in this Standard.

In each case, the key transport scheme is used by an entity who wishes to establish keying data with another entity. Both protocols specified are 'asymmetric', so it is necessary to describe two transformations, one of which is undertaken by *U* if *U* is the initiator, and one of which is undertaken by *V* if *V* is the responder.

In the specification of each transformation, equivalent computations that result in identical output are allowed.

Each of the key transport schemes has certain prerequisites. These are conditions that must be satisfied by an implementation of the scheme. However the specification of mechanisms that provide these prerequisites is beyond the scope of this Standard.

Section H.4.3 provides guidance to the services which each scheme may be used to provide.

Each scheme is described in two ways. First a flow diagram of the ordinary operation of the scheme between two entities *U* and *V* is provided. This flow diagram is for illustrative purposes only. Then a formal specification is given which describes the actions entities must take to use the scheme to establish keying data.

These flow diagrams are intended to aid understanding of the mechanics of the 'ordinary' operation of the schemes in which flows are relayed faithfully between two entities. Note that in 'real-life', there is no reason to assume that flows are relayed faithfully between two entities…that is why the schemes must be formally specified in a more technical fashion.

When examining the flow diagrams, the following points should be noted:

— For clarity of exposition, optional fields such as *Text* and *SharedData* are omitted.

— *kdf*(*Z*) denotes the output of the key derivation function specified in Section 5.6.3 called on input *Z*.

— *ENC* and *DEC* respectively denote the encryption and decryption transformations associated with one of the asymmetric encryption schemes specified in Section 5.8. The subscripts immmediately following *ENC* and *DEC* denote the keys being used in the operation of the appropriate transformation. Similarly, *SIG* denotes the signing transformation associated with the signature scheme ECDSA specified in Section 5.9, and *MAC* denotes the tagging transformation of one of the MAC schemes specified in Section 5.7.

7.1 1-Pass Transport Scheme

This section specifies the 1-pass transport scheme.
First the scheme is illustrated in a flow diagram. Figure 13 illustrates the use of the 1-pass key transport scheme.

Figure 13 - 1-Pass Key Transport Scheme

Next the formal specification of the scheme is given.
The scheme is 'asymmetric', so two transformations are specified *U* uses the transformation specified in Section 7.1.1 to establish keying data with *V* if *U* is the protocol's initiator, and *V* uses the transformation specified in Section 7.1.2 to establish keying data with *U* if *V* is the protocol's responder.
If *U* executes the initiator transformation and *V* simultaneously executes the responder transformation with corresponding keying material as input, then *U* and *V* will compute the same keying data.
Prerequisites: The following are the prerequisites for the use of the scheme:

1. Each entity has an authentic copy of the system's elliptic curve domain parameters to be used with an asymmetric encryption scheme *q*, *a*, *b*, *G*, *n*, and *h* along with an indication of the basis used if $q=2^m$. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.
2. Each entity allowed to act as a responder shall be bound to a static encryption key pair associated to the system's elliptic curve domain parameters *q*, *a*, *b*, *G*, *n*, *h*. The binding may include the validation of the public key as specified in Section 5.2.2.
3. Each entity allowed to act as an initiator shall be bound to a unique identifier (e.g. distinguished names). All identifiers shall be bit strings of same length *entlen* bits. Entity *U*'s identifier will be denoted by the bit string *U*.
4. Each entity shall have decided whether to use the Elliptic Curve Encryption Scheme in Section 5.8.1 or the Elliptic Curve Augmented Encryption Scheme in Section 5.8.2.
5. Each entity shall have chosen an ANSI-approved hash function for use with the key derivation function during encryption and decryption.

7.1.1 Initiator Transformation

U shall execute the following transformation to establish keying data with *V* if *U* is the protocol's initiator:
Input: The input to the initiator transformation is:

1. A bit string *KeyData* of length *keydatalen* bits which is the keying data to be transported.
2. (Optional) Two bit strings $SharedData_1$ and $SharedData_2$ which consist of some data shared by *U* and *V*.

Ingredients: The initiator transformation employs the encryption transformation of the appropriate asymmetric encryption scheme in Section 5.8.
Actions: Keying data shall be derived as follows:

1. Form the bit string consisting of *U*'s identifier, the keying data *KeyData*, and optionally a bit string *Text*:
   $EncData = U \| KeyData \| [Text]$.
2. Encrypt *EncData* under *V*'s static public encryption key $Q_{enc,V}$ corresponding to the EC domain parameters *q*, *a*, *b*, *G*, *n*, and *h*, with the optional inputs $SharedData_1$ and $SharedData_2$, using the encryption transformation of the appropriate asymmetric encryption scheme in Section 5.8. If the encryption transformation outputs 'invalid', output 'invalid' and stop. Otherwise the encryption transformation outputs a bit string *EncryptedData* as the encryption of *EncData*.
3. Send *EncryptedData* to *V*.

Output: The bit string *KeyData* of length *keydatalen* bits as the keying data.

NOTE— Including a key counter field in the optional *Text* field may help to prevent known key attacks.

7.1.2 Responder Transformation

V shall execute the following transformation to establish keying data with *U* if *V* is the protocol's responder:

Input: The input to the responder transformation is:

1. A bit string *EncryptedData'* purporting to be the encryption of a bit string.
2. An integer *keydatalen* which is the length in bits of the keying data to be generated.
3. (Optional) Two bit strings $SharedData_1$ and $SharedData_2$ which consist of some data shared by *U* and *V*.

Ingredients: The responder transformation employs the decryption transformation of the appropriate asymmetric encryption scheme in Section 5.8.

Actions: Keying data shall be derived as follows:

1. Decrypt the bit string *EncryptedData'* using *V*'s static private decryption key $d_{enc,V}$ corresponding to the EC domain parameters *q*, *a*, *b*, *G*, *n*, and *h*, with the optional inputs $SharedData_1$ and $SharedData_2$, using the decryption transformation of the appropriate asymmetric encryption scheme in Section 5.8. If the decryption transformation outputs 'invalid', output 'invalid' and stop. Otherwise the decryption transformation outputs a bit string *EncData* of length *encdatalen* bits as the decryption of the bit string.
2. If *encdatalen*<*entlen*+*keydatalen*, output 'invalid' and stop.
3. Parse the first *entlen* bits of *EncData* as the purported identifier *U*' of *U*, and the next *keydatalen* bits of *EncData* as keying data *KeyData*.
4. Verify that *U'*=*U*; if not, output 'invalid' and stop.

Output: The bit string *KeyData* of length *keydatalen* bits as the keying data.

7.2 3-Pass Transport Scheme

This section specifies the 3-pass transport scheme. The scheme uses a signature scheme to provide explicit key authentication for a session key transported using the 1-pass transport scheme specified in Section 7.1.

First the scheme is illustrated in a flow diagram. Figure 14 illustrates the use of the 3-pass key transport scheme.

Figure 14 - 3-Pass Key Transport Scheme

Next the formal specification of the scheme is given.

The scheme is 'asymmetric', so two transformations are specified. *U* uses the transformation specified in Section 7.2.1 to establish keying data with *V* if *U* is the protocol's initiator, and *V* uses the transformation specified in Section 7.2 to establish keying data with *U* if *V* is the protocol's responder.

If *U* executes the initiator transformation and *V* simultaneously executes the responder transformation with corresponding keying material as input, then *U* and *V* will compute the same keying data.

Prerequisites: The following are the prerequisites for the use of the scheme:

1. Each entity has an authentic copy of the system's elliptic curve domain parameters to be used with an asymmetric encryption scheme q_{enc}, a_{enc}, b_{enc}, G_{enc}, n_{enc}, and h_{enc} along with an indication of the basis used if $q=2^m$. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.
2. Each entity has an authentic copy of the system's elliptic curve domain parameters to be used with a signature scheme q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, and h_{sig} along with an indication of the basis used if $q=2^m$. These parameters shall have been generated using the parameter generation primitives in Sections 5.1.1.1 and 5.1.2.1. Furthermore, the parameters shall have been validated using the parameter validation primitives in Sections 5.1.1.2 and 5.1.2.2.
3. Each entity shall be bound to a static signing key pair associated to the system's elliptic curve domain parameters for signing q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, and h_{sig}. The binding may include the validation of the public signature as specified in Section 5.2.2. The key binding shall include a unique identifier for each entity (e.g. distinguished names). All identifiers shall be bit strings of the same length *entlen* bits. Entity *U*'s identifier will be denoted by the bit string *U*.

4. Each entity allowed to act as an initiator shall be bound to a static encryption key pair associated to the system's elliptic curve domain parameters for encryption q_{enc}, a_{enc}, b_{enc}, G_{enc}, n_{enc}, and h_{enc}. The binding may include the validation of the public encryption key as specified in Section 5.2.2.
5. Each entity shall have decided whether to use the Elliptic Curve Encryption Scheme in Section 5.8.1 or the Elliptic Curve Augmented Encryption Scheme in Section 5.8.2.
6. Each entity shall have chosen an ANSI-approved hash function for use with the key derivation function during encryption and decryption.
7. Each entity shall have decided what length the challenges it uses will be. *challengelen* denotes the length chosen. Note that *challengelen* must be ≥80.

7.2.1 Initiator Transformation

U shall execute the following transformation to establish keying data with *V* if *U* is the protocol's initiator:

Input: The input to the initiator transformation is:

1. An integer *keydatalen* which is the length in bits of the keying data to be generated.
2. (Optional) Two bit strings $SharedData_1$ and $SharedData_2$ which consist of some data shared by *U* and *V*.

Ingredients: The initiator transformation employs the challenge generation primitive specified in Section 5.3, the signature scheme specified in Section 5.9, and the decryption transformation of the appropriate asymmetric encryption scheme in Section 5.8.

Actions: Keying data shall be derived as follows:

1. Use the challenge generation primitive in Section 5.3 to generate a challenge $Challenge_U$ of length *challengelen* bits. Send $Challenge_U$ to *V*.
2. Then receive from *V* a purported challenge $Challenge_V$', a bit string *EncryptedData'* purporting to be the encryption of a bit string, an optional bit string $Text_1$, and a pair of integers $rsig_1$' and $ssig_1$' purporting to be a signature. If this data is not received, output 'invalid' and stop.
3. Verify that $Challenge_V$' is a bit string of length *challengelen* bits. If not, output 'invalid' and stop.
4. Decrypt the bit string *EncryptedData'* using *U*'s private decryption key $d_{enc,U}$ corresponding to the EC domain parameters q_{enc}, a_{enc}, b_{enc}, G_{enc}, n_{enc}, and h_{enc}, with the optional inputs $SharedData_1$ and $SharedData_2$, using the decryption transformation of the appropriate asymmetric encryption scheme in Section 5.8. If the decryption transformation outputs 'invalid', output 'invalid' and stop. Otherwise the decryption transformation outputs a bit string *EncData* of length *encdatalen* bits as the decryption of the bit string.
5. If *encdatalen*<*entlen*+*keydatalen*, output 'invalid' and stop.
6. Parse the first *entlen* bits of *EncData* as the purported identifier *V'* of *V*, and the next *keydatalen* bits of *EncData* as keying data *KeyData*.
7. Verify that *V'*=*V*; if not, output 'invalid' and stop.
8. Form the bit string consisting of $Challenge_V$', $Challenge_U$, *U*'s identifier, the bit string *EncryptedData'*, and if present $Text_1$:
   $SignData_1 = Challenge_V' \,||\, Challenge_U \,||\, U \,||\, EncryptedData' \,||\, [Text_1]$.
9. Verify that $rsig_1$' and $ssig_1$' are a valid signature of $SignData_1$ under *V*'s public signature key $Q_{sig,V}$ corresponding to the EC domain parameters q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, and h_{sig}, using the verifying transformation of the signature scheme in Section 5.9. If the verifying transformation outputs 'invalid', output 'invalid' and stop.
10. Form the bit string consisting of $Challenge_U$, $Challenge_V$', *V*'s identifier, and optionally a bit string $Text_{\_2}$:
    $SignData_2 = Challenge_U \,||\, Challenge_V' \,||\, V \,||\, [Text_2]$.
11. Sign $SignData_2$ using *U*'s private signing key $d_{sig,U}$ corresponding to the parameters q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, and h_{sig}, using the signing transformation of the signature scheme in Section 5.9. If the signing transformation outputs 'invalid', output 'invalid' and stop. Otherwise the signing transformation outputs the integers $rsig_2$ and $ssig_2$ as the signature of $SignData_2$.
12. Send to *V* the bit string $Text_2$ if present, and $rsig_2$ and $ssig_2$.

Output: The bit string *KeyData* of length *keydatalen* bits.

7.2.2 Responder Transformation

V shall execute the following transformation to establish keying data with *U* if *V* is the protocol's responder:

Input: The input to the responder transformation is:

1. A purported challenge $Challenge_U$' from U.
2. A bit string *KeyData* of length *keydatalen* bits which is the keying data to be transported.
3. (Optional) Two bit strings $SharedData_1$ and $SharedData_2$ which consist of some data shared by U and V.

Ingredients: The responder transformation employs the challenge generation primitive specified in Section 5.3, the signature scheme specified in Section 5.9, and the encryption transformation of the appropriate asymmetric encryption scheme in Section 5.8.

Actions: Keying data shall be derived as follows:

1. Verify that $Challenge_U$' is a bit string of length *challengelen* bits. If not, output 'invalid' and stop.
2. Use the challlenge generation primitive in Section 5.3 to generate a challenge $Challenge_V$ of length *challengelen* bits.
3. Form the bit string consisting of V's identifier, *KeyData*, and optionally a bit string $Text_1$:
   $EncData = V \| KeyData \| [Text_1]$.
4. Encrypt *EncData* under U's public encryption key $Q_{enc,U}$ corresponding to the EC domain parameters q_{enc}, a_{enc}, b_{enc}, G_{enc}, n_{enc}, and h_{enc}, with the optional inputs $SharedData_1$ and $SharedData_2$, using the encryption transformation of the appropriate asymmetric encryption scheme in Section 5.8. If the encryption transformation outputs 'invalid', output 'invalid' and stop. Otherwise the encryption transformation outputs a bit string *EncryptedData* as the encryption of *EncData*.
5. Form the bit string consisting of $Challenge_V$, $Challenge_U$', U's identifier, the bit string *EncryptedData*, and optionally a bit string $Text_1$:
   $SignData_1 = Challenge_V \| Challenge_U' \| U \| EncryptedData \| [Text_1]$.
6. Sign $SignData_1$ using V's private signing key $d_{sig,V}$ corresponding to the parameters q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, and h_{sig}, using the signing transformation of the signature scheme in Section 5.9. If the signing transformation outputs 'invalid', output 'invalid' and stop. Otherwise the signing transformation outputs the integers $rsig_1$ and $ssig_1$ as a signature of $SignData_1$.
7. Send $Challenge_V$, the bit string *EncryptedData*, if present $Text_1$, and $rsig_1$ and $ssig_1$ to U.
8. Then receive from U an optional bit string $Text_2$, and a purported signature $rsig_2$' and $ssig_2$'. If this data is not received, output 'invalid' and stop.
9. Form the bit string consisting of $Challenge_U$', $Challenge_V$, V's identifier, and if present $Text_2$:
   $SignData_2 = Challenge_U' \| Challenge_V \| V \| [Text_2]$.
10. Verify that the pair $rsig_2$' and $ssig_2$' is a valid signature of $SignData_2$ under U's public signature key $Q_{sig,U}$ corresponding to the EC domain parameters q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, and h_{sig}, using the verifying transformation of the signature scheme in Section 5.9. If the verifying transformation outputs 'invalid', output 'invalid' and stop.

Output: The bit string *KeyData* of length *keydatalen* bits.

8 ASN.1 Syntax

[[This section will be added later.]]

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 [55] and Frey and Ruck[30] 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 [67] and Satoh and Araki [64] 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*) [48] 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 $n\text{-}1 = 2^v w$.
2. For j from 1 to T do
   2.1. Choose random a in the interval $[2, n\text{-}1]$.
   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 [11] 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 $n \geq r_{min}$, $n > 4\sqrt{q}$ 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 $n \geq r_{min}$, $n > 4\sqrt{q}$ 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 $r_{min} > 2^{160}$. 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 [65]. 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 [50] and [51]).
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 K presents sample elliptic curves 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 [5]). 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 $t = \lceil \log_2 p \rceil$, $s = \lfloor (t-1)/160 \rfloor$, 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 $t = \lceil \log_2 p \rceil$, $s = \lfloor (t-1)/160 \rfloor$, 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 $r' \cdot b^2 \equiv a^3 \pmod{p}$, then accept; otherwise reject.

A.4 Pseudorandom Number Generation

Any implementation of this standard requires the ability to generate random or pseudorandom integers. 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 [5]. 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. $XSEED_{i,j}$ = optional user input.
      4.1.2. XVAL = (XKEY + $XSEED_{i,j}$) mod 2^b.
      4.1.3. $x_j = G(t, XVAL)$.
      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 [5]. 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 [5] 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, $r \in [0, p\text{-}1]$. 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, $s \in [0, p\text{-}1]$. 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:

$\alpha = a_0\alpha_0 + a_1\alpha_1 + \ldots + a_{m-1}\alpha_{m-1}$, 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, two 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:

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

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

$(1101) + (1001) = (0100)$, 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:

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

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 $1 \leq k \leq m$ -1. A *pentanomial* over F_2 is a polynomial of the form $x^m + x^{k3} + x^{k2} + x^{k1} + 1$, where $1 \leq k1 < k2 < k3 \leq m$-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 $m \geq 4$.

B.2.3 Normal Bases

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

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

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:

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

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:

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

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) 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, α^9)	(α, 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})	(α^6, 0)	(α^6, α^6)	(α^8, α^3)	(α^8, α^{13})	
(α^{11}, 0)	(α^{11}, α^{11})	(α^{12}, α^8)	(α^{12}, α^9)	(α^{13}, α^2)	(α^{13}, α^{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, $160 \leq m \leq 300$, 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, $822 \leq m \leq 995$, 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, $996 \leq m \leq 1169$, 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, $1170 \leq m \leq 1342$, 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, $1343 \leq m \leq 1516$, 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, $1517 \leq m \leq 1690$, 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, $1691 \leq m \leq 1863$, 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, $1864 \leq m \leq 2000$, 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 $x^m + x^{k3} + x^{k2} + x^{k1} + 1$ 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 $x^m + x^{k3} + x^{k2} + x^{k1} + 1$ 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 $x^m + x^{k3} + x^{k2} + x^{k1} + 1$ 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 $x^m + x^{k3} + x^{k2} + x^{k1} + 1$ 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 $x^m + x^{k3} + x^{k2} + x^{k1} + 1$ 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 $x^m + x^{k3} + x^{k2} + x^{k1} + 1$ 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 $x^m + x^{k3} + x^{k2} + x^{k1} + 1$ 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 $x^m + x^{k3} + x^{k2} + x^{k1} + 1$ 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 $x^m + x^{k3} + x^{k2} + x^{k1} + 1$ 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 $x^m + x^{k3} + x^{k2} + x^{k1} + 1$ 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 $x^m + x^{k3} + x^{k2} + x^{k1} + 1$ 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 $x^m + x^{k3} + x^{k2} + x^{k1} + 1$ 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 $e = a \bmod (q\text{-}1)$. 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 [24]. See also Knuth [45].

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 $gc \equiv 1 \pmod{q}$.

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 [45].

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 $(U,V) = (UV \bmod p, \left(\frac{V^2 + \Delta U^2}{2}\right) \bmod p)$.
   4.2. If $k_i = 1$ then set $(U,V) = (\left(\frac{U + V}{2}\right) \bmod p, \left(\frac{V + \Delta U}{2}\right) \bmod p)$.
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 2-2T 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 $(\beta_0 \beta_1 ... \beta_{m-1})$ 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 $z = (z_0 z_1 ... z_{m-1})$.
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.

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 $j = 1$.
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. The algorithm is used in Annex D.2.3.

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 $p = Tm + 1$.
      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
      $$e_k = \sum_{j=0}^{T-1} \exp\left(\frac{2\pi i u^j k}{p}\right)$$
      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:
$$\Gamma = \begin{bmatrix} \gamma_{0,0} & \gamma_{0,1} & \cdots & \gamma_{0,m-1} \\ \gamma_{1,0} & \gamma_{1,1} & \cdots & \gamma_{1,m-1} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{m-1,0} & \gamma_{m-1,1} & \cdots & \gamma_{m-1,m-1} \end{bmatrix}$$
where the entries $\gamma_{i,j}$ are defined as follows:
   3.1. If B_2 is a polynomial basis, then:

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

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:

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

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

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

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

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

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

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:

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

and:

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

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.

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 $h_r h_{r-1} ... h_1 h_0$ be the binary representation of $3e$, where the leftmost bit h_r is 1.
3. Let $e_r e_{r-1} ... e_1 e_0$ 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 [24]. See also Knuth [45].

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 $k = k_r\, k_{r-1} \ldots k_1\, k_0$ 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, ±1,24] and [8, ±1,9].

— For $B = 2$, we have $A = 5$, leading to [5, ±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:

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

Let:

$f_0(A, B, C) = \theta^{-1/24} F(-\theta) / F(\theta^2)$,

$f_1(A, B, C) = \theta^{-1/24} F(\theta) / F(\theta^2)$,

$f_2(A, B, C) = \sqrt{2}\, \theta^{1/12} F(\theta^4) / F(\theta^2)$.

Note: Since $|\theta| < e^{-\pi\sqrt{3}/2} \approx 0.0658287$, the series $F(z)$ used in computing the numbers $f_J(A, B, C)$ 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

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

where:

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

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

$$J = \begin{cases} 4 & \text{for } AC \text{ odd}, \\ 1 & \text{for } C \text{ even}, \\ 2 & \text{for } A \text{ even}, \end{cases}$$

$$K = \begin{cases} 2 & \text{if } D \equiv 1,2,6 \bmod 8, \\ 0 & \text{if } D \equiv 3,7 \bmod 8, \\ 4 & \text{if } D \equiv 5 \bmod 8, \end{cases}$$

$$L = \begin{cases} A - C + A^2C & \text{if } AC \text{ odd or } D \equiv 5 \bmod 8 \text{ and } C \text{ even}, \\ A + 2C - AC^2 & \text{if } D \equiv 1,2,3,6,7 \bmod 8 \text{ and } C \text{ even}, \\ A - C + 5A^2C & \text{if } D \equiv 3 \bmod 8 \text{ and } A \text{ even}, \\ A - C - AC^2 & \text{if } D \equiv 1,2,5,6,7 \bmod 8 \text{ and } A \text{ even}, \end{cases}$$

$$M = \begin{cases} (-1)^{(A^2-1)/8} & \text{if } A \text{ odd}, \\ (-1)^{(C^2-1)/8} & \text{if } A \text{ even}, \end{cases}$$

$$N = \begin{cases} 1 & \text{if } D \equiv 5 \bmod 8 \\ & \text{or } D \equiv 3 \bmod 8 \text{ and } AC \text{ odd} \\ & \text{or } D \equiv 7 \bmod 8 \text{ and } AC \text{ even}, \\ M & \text{if } D \equiv 1,2,6 \bmod 8 \\ & \text{or } D \equiv 7 \bmod 8 \text{ and } AC \text{ odd} \\ -M & \text{if } D \equiv 3 \bmod 8 \text{ and } AC \text{ even}, \end{cases}$$

$\lambda = e^{-\pi iK/24}$.

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:

$$w_D(t) = \prod_{j=1}^{h} (t - \mathfrak{C}(A_j, B_j, C_j)).$$

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:

$$w_{71}(t) = \left(t - \frac{1}{\sqrt{2}} \mathfrak{f}_0(1,0,71)\right)$$

$$\left(t - \frac{e^{-i\pi/8}}{\sqrt{2}} f_1(3,1,24)\right)\left(t - \frac{e^{i\pi/8}}{\sqrt{2}} f_1(3,-1,24)\right)$$
$$\left(t - \frac{e^{-23i\pi/24}}{\sqrt{2}} f_2(8,1,9)\right)\left(t - \frac{e^{23i\pi/24}}{\sqrt{2}} f_2(8,-1,9)\right)$$
$$\left(t - \frac{e^{-5i\pi/12}}{\sqrt{2}} f_0(5,2,15)\right)\left(t - \frac{e^{5i\pi/12}}{\sqrt{2}} f_0(5,-2,15)\right)$$

$$= (t - 2.13060682983889533005591468688942503...)$$
$$(t - (0.95969178530567025250797047645507504...) + (0.34916071001269654799855316293926907...)\ i)$$
$$(t - (0.95969178530567025250797047645507504...) - (0.34916071001269654799855316293926907...)\ i)$$
$$(t + (0.75613568804001789053564010985317 72...) + (0.07375086316308890052407649445676 75...)\ i)$$
$$(t + (0.75613568804001789053564010985317 72...) - (0.07375086316308890052407649445676 75...)\ i)$$
$$(t + (0.26885951218510002700028771004661 02...) - (0.84108577401329800103648634224905292...)\ i)$$
$$(t + (0.26885951218510002700028771004661 02...) + (0. 84108577401329800103648634224905292...)\ i)$$
$$= t^7 - 2t^6 - t^5 + t^4 + t^3 + t^2 - t - 1.$$

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 (mod 8).

— If $p \equiv 5 \pmod 8$, then D is odd.

— If $p \equiv 7 \pmod 8$, then $D \equiv 3$, 6, or 7 (mod 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 $\delta = \left\lfloor \frac{M}{A} + \frac{1}{2} \right\rfloor$

   5.2. Let $T = \begin{pmatrix} \delta & -1 \\ 1 & 0 \end{pmatrix}$

   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.3.2.3 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 $D = 235$,

$X = -310372529376179308359576872 34$,

$Y = 590504615239318452103330511 3$,

and r is the prime:

$r = 627710173538668076383578942333772047398677360825518901532 9$.

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 $U = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
4. Until $|2B| \le A \le C$, repeat the following steps.
   - 4.1 Let $\delta = \left\lfloor \frac{B}{A} + \frac{1}{2} \right\rfloor$
   - 4.2 Let $T = \begin{bmatrix} \delta & -1 \\ 1 & 0 \end{bmatrix}$
   - 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:

$D = 942679$,
$X = 22952987868304682039818 1$,
$Y = -371360755031779037497$,

and r is the prime:

r = 1141798154164767904846623037312629032935687344 7.
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:

$$y^2 \equiv x^3 + a_0x + b_0 \pmod p$$

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 φ = 1254098248316315745658220082226751383299177953632927607231. The resulting coefficients are:

a_0 = –208902381629407921389227212 8,

b_0 = –367504956274613540540444576026309668372 48.

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:

$$w_D(t) \equiv 1 + t^2 + t^6 + t^{10} + t^{12} + t^{13} + t^{16} + t^{17} + t^{20} + t^{22} + t^{24} + t^{27} + t^{30} + t^{33} + t^{35} + t^{36} + t^{37} +$$
$$t^{41} + t^{42} + t^{43} + t^{45} + t^{49} + t^{51} + t^{54} + t^{56} + t^{57} + t^{59} + t^{61} + t^{65} + t^{67} + t^{68} + t^{69} + t^{70} + t^{71} + t^{72} + t^{74} + t^{75} +$$
$$t^{76} + t^{82} + t^{83} + t^{87} + t^{91} + t^{93} + t^{96} + t^{99} + t^{100} + t^{101} + t^{102} + t^{103} + t^{106} + t^{108} + t^{109} + t^{110} + t^{114} + t^{117} +$$
$$t^{119} + t^{121} + t^{123} + t^{125} + t^{126} + t^{128} + t^{129} + t^{130} + t^{133} + t^{134} + t^{140} + t^{141} + t^{145} + t^{146} + t^{147} + t^{148} + t^{150} +$$
$$t^{152} + t^{154} + t^{155} + t^{157} + t^{158} + t^{160} + t^{161} + t^{166} + t^{167} + t^{171} + t^{172} + t^{175} + t^{176} + t^{179} + t^{180} + t^{185} + t^{186} +$$
$$t^{189} + t^{190} + t^{191} + t^{192} + t^{195} + t^{200} + t^{201} + t^{207} + t^{208} + t^{209} + t^{210} + t^{211} + t^{219} + t^{221} + t^{223} + t^{225} + t^{228} +$$
$$t^{233} + t^{234} + t^{235} + t^{237} + t^{238} + t^{239} + t^{241} + t^{242} + t^{244} + t^{245} + t^{248} + t^{249} + t^{250} + t^{252} + t^{253} + t^{255} + t^{257} +$$
$$t^{260} + t^{262} + t^{263} + t^{264} + t^{272} + t^{273} + t^{274} + t^{276} + t^{281} + t^{284} + t^{287} + t^{288} + t^{289} + t^{290} + t^{292} + t^{297} + t^{299} +$$
$$t^{300} + t^{301} + t^{302} + t^{304} + t^{305} + t^{306} + t^{309} + t^{311} + t^{312} + t^{313} + t^{314} + t^{317} + t^{318} + t^{320} + t^{322} + t^{323} + t^{325} +$$
$$t^{327} + t^{328} + t^{329} + t^{333} + t^{335} + t^{340} + t^{341} + t^{344} + t^{345} + t^{346} + t^{351} + t^{353} + t^{354} + t^{355} + t^{357} + t^{358} + t^{359} +$$
$$t^{360} + t^{365} + t^{366} + t^{368} + t^{371} + t^{372} + t^{373} + t^{376} + t^{377} + t^{379} + t^{382} + t^{383} + t^{387} + t^{388} + t^{389} + t^{392} + t^{395} +$$
$$t^{398} + t^{401} + t^{403} + t^{406} + t^{407} + t^{408} + t^{409} + t^{410} + t^{411} + t^{416} + t^{417} + t^{421} + t^{422} + t^{423} + t^{424} + t^{425} + t^{426} +$$
$$t^{429} + t^{430} + t^{438} + t^{439} + t^{440} + t^{441} + t^{442} + t^{443} + t^{447} + t^{448} + t^{450} + t^{451} + t^{452} + t^{453} + t^{454} + t^{456} + t^{458} +$$
$$t^{459} + t^{460} + t^{462} + t^{464} + t^{465} + t^{466} + t^{467} + t^{471} + t^{473} + t^{475} + t^{476} + t^{481} + t^{482} + t^{483} + t^{484} + t^{486} + t^{487} +$$
$$t^{488} + t^{491} + t^{492} + t^{495} + t^{496} + t^{498} + t^{501} + t^{503} + t^{505} + t^{507} + t^{510} + t^{512} + t^{518} + t^{519} + t^{529} + t^{531} + t^{533} +$$
$$t^{536} + t^{539} + t^{540} + t^{541} + t^{543} + t^{545} + t^{546} + t^{547} + t^{548} + t^{550} + t^{552} + t^{555} + t^{556} + t^{557} + t^{558} + t^{559} + t^{560} +$$
$$t^{563} + t^{565} + t^{566} + t^{568} + t^{580} + t^{585} + t^{588} + t^{589} + t^{591} + t^{592} + t^{593} + t^{596} + t^{597} + t^{602} + t^{604} + t^{606} + t^{610} +$$
$$t^{616} + t^{620} \pmod 2.$$

This polynomial factors into 4 irreducibles over F_2, each of degree 155. One of these is:

$$p(t) = 1 + t + t^2 + t^6 + t^9 + t^{10} + t^{11} + t^{13} + t^{14} + t^{15} + t^{16} + t^{18} + t^{19} + t^{22} + t^{23} + t^{26} + t^{27} +$$
$$t^{29} + t^{31} + t^{49} + t^{50} + t^{51} + t^{54} + t^{55} + t^{60} + t^{61} + t^{62} + t^{64} + t^{66} + t^{70} + t^{72} + t^{74} + t^{75} + t^{80} + t^{82} + t^{85} + t^{86} +$$
$$t^{88} + t^{89} + t^{91} + t^{93} + t^{97} + t^{101} + t^{103} + t^{104} + t^{111} + t^{115} + t^{116} + t^{117} + t^{118} + t^{120} + t^{121} + t^{123} + t^{124} + t^{126}$$
$$+ t^{127} + t^{128} + t^{129} + t^{130} + t^{131} + t^{132} + t^{134} + t^{136} + t^{137} + t^{138} + t^{139} + t^{140} + t^{143} + t^{145} + t^{154} + t^{155}.$$

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 cryptographic schemes based on the discrete logarithm problem. These protocols are described in the literature in the algebraic system Z_p, the integers modulo p, where p is a prime. For example, ANSI X9.42 [6] describes a suite of key agreement mechanisms based on the Diffie-Hellman scheme defined over Z_p. These mechanisms can also 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 suite of key establishment schemes which use 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)
Comparison of Elliptic Curves and Finite Fields

The elliptic curve key establishment schemes described in this Standard can also be described in the more traditional setting of F_p^* (also denoted Z_p^*), the multiplicative group of the integers modulo a prime. For example, many of the key agreement schemes are elliptic curve analogs of the schemes described in ANSI X9.42 [6].
The following tables show the correspondence between the elements and operations of the group F_p^* and the elliptic curve group $E(F_q)$, as well as the correspondence between the 'language' of ANSI X9.42 and the 'language' of this Standard.
Table G-1 compares the basic properties of the two underlying groups: F_p^* and $E(F_q)$.

Table G-1 – F_p^* and $E(F_q)$ 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_1, g_2 Multiplication: $g_1 \times g_2$ Exponentiation: g^k	Elements: P_1, P_2 Addition: $P_1 + P_2$ Multiple of a point (also called scalar multiplication): kP
Discrete logarithm problem	Given $g_1 \in F_p^*$ and $g_2 = g_1^k \bmod p$, find the integer k.	Given $P_1 \in E(F_q)$ and $P_2 = kP_1$, find the integer k.
Diffie-Hellman problem	Given g^{k1}, $g^{k2} \in F_p^*$, find g^{k1k2}.	Given k_1P, $k_2P \in E(F_q)$, find k_1k_2P.

Table G-2 compares the notation used to describe analogous key agreement schemes in two ANSI standards: ANSI X9.42 [6] and this Standard.

Table G-2 – Comparison of Notation in ANSI X9.42 and ANSI X9.63

X9.42 Notation	X9.63 Notation
q	n
p	$\#E(F_q)$
g	G
x	d_s
y	Q_s
r	d_e
t	Q_e

Table G-3 continues the comparison between ANSI X9.42 and this Standard. In the table, the procedures for setting up the key agreement schemes are compared.

Table G-3 – ANSI X9.42 and ANSI X9.63 Setup

X9.42 Setup	X9.63 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 compares the key generation procedure used by ANSI X9.42 and this Standard.

Table G-4 – ANSI X9.42 and ANSI X9.63 Key Generation

X9.42 Key Generation	X9.63 Key Generation
1. Select a random integer x in the interval [1, q-1]. 2. Compute $y = g^x \bmod p$. 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.

Finally Table G-5 looks more closely at one particular scheme which is specified in both ANSI X9.42 and this Standard: the full Unified Model scheme.

Table G-5 – Comparison of the Full Unified Model Scheme

X9.42	X9.63
1. Select an ephemeral public key t_U. 2. Receive an ephemeral public key t_V. 3. Compute the shared secret values t_V^{rU} and y_V^{xU}. 4. Derive keying data from the shared secret values using a key derivation function.	1. Select an ephemeral public key $Q_{e,U}$. 2. Receive an ephemeral public key $Q_{e,V}$'. 3. Compute the shared secret values $[h]d_{e,U}Q_{e,V}$' and $[h]d_{s,U}Q_{s,V}$. 4. Derive keying data from the shared secret values using a key derivation function.

Annex H
(informative)
Security Considerations

This appendix is provided as an initial guidance for implementors of this Standard. This information should be expected to change over time. Implementors should review the current state-of-the-art in attacks on the schemes 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 key establishment schemes and, in particular, the suite to key establishment schemes in this Standard. Annex H.5 discusses issues related to validation of implementations of the schemes in this Standard.

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 [62] and the Pollard-λ method [62]. 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 [68]) 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 $2\sqrt{n}$ steps. It can also be parallelized (see [68]) 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 [31], and Wiener and Zuccherato [70] 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 4×10^4 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 [61] 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.

Table H-1 - Computing power required to compute logarithms with the Pollard-ρ method.

Field size (in bits)	Size of n (in bits)	$\sqrt{\pi n / 4}$	MIPS years
163	160	2^{80}	8.5×10^{11}
191	186	2^{93}	7.0×10^{15}
239	234	2^{117}	1.2×10^{23}
359	354	2^{177}	1.3×10^{41}
431	426	2^{213}	9.2×10^{51}

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 this Standard 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.

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 [68] 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 [21], 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 discrete logarithm based schemes, there is the possibility that a particularly poor choice of domain parameters could lead to an attack. To address this, DSA for example 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 elliptic curve based schemes 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 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.
5. 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.
6. 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 64.

7. 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. 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. 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 1999, 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.
3. Public key cryptoperiod considerations. A public key can be considered valid for any period of time that the associated private key is used.
4. Repeated private keys. If two users are using the same elliptic curve domain parameters and somehow generate identical private key values, then compromise may occur. As the private key 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 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 be told, 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.

H.4 Key Establishment Schemes

This section discusses issues particularly relevant to the security of key establishment schemes.

H.4.1 The ECDLP and Key Establishment Schemes

Each of the key establishment schemes specified in this Standard is dependent for its security on the difficulty of the ECDLP. An adversary of a scheme who is able to solve the ECDLP is able to recover the EC private key from any EC public key, and in this way compromise any key established using the key establishment scheme.
However, there is a gap in the above statement. It says that the key establishment schemes are insecure if the ECDLP can be solved, but does not say that the key establishment schemes are secure if the ECDLP cannot be solved efficiently. It is conceivable that some attack could be found which compromises the security of the key establishment scheme without contradicting the supposed difficulty of the ECDLP.
Much research has focused on closing this gap between the difficulty of the ECDLP and the security of the key establishment schemes. At best the research has led to a proof of equivalence between the two problems in some ‘formal model’, while in other cases the equivalence remains a conjecture, albeit one that has not been disproved by a sizeable amount of public scrutiny.

A relevant stepping stone between the two problems is the elliptic curve Diffie-Hellman problem (ECDHP). The ECDHP is stated as follows: given an EC E, a base point $P \in E$, and k_1P and k_2P with k_1 and k_2 randomly chosen, calculate k_1k_2P.

The relevance of this stepping stone is easily explained. For example, it is clearly the problem which faces a passive adversary of the ephemeral Unified Model scheme. Other results of this kind have been justified in the literature: [18] discusses the equivalence between the ECDHP and the asymmetric encryption schemes in Section 5.8, and [20] the equivalence between the ECDHP and various Unified Model schemes. Both of these equivalences are proved in a 'formal model'. [56] and [49] discuss the conjectured equivalence between the ECDHP and the MQV schemes. Such equivalences, whether conjectured or 'formally' demonstrated, should certainly be viewed with scepticism…'real-life' adversaries are seldom restrained to acting within the 'formal models' discussed in these results. Nonetheless, the results do instill confidence that the relationship between the difficulty ECDHP and the security of the scheme is indeed close.

Linking the difficulty of the ECDLP to the difficulty of the ECDHP remains to be shown. A result of this type is provided by [22]. This paper shows that provided the ECDLP is exponentially hard (as is currently believed), then the two problems are indeed computationally equivalent.

Taken in totality these results provide some assurance of the statement that 'the ECDLP is the basis for the security of the EC schemes'.

H.4.2 Security Attributes and Key Establishment Schemes

What properties is it desirable for a key establishment scheme to possess?

The fundamental goal of any key establishment scheme is to distribute keying data. Ideally, the keying data should have precisely the same attributes as keying data established face-to-face. It should be randomly distributed, and no unauthorized entity should know anything about the keying data.

However, whilst asymmetric key establishment schemes offer many advantages over traditional face-to-face key establishment, there is a price to pay for this added functionality. No asymmetric scheme can offer unconditional security in an information theoretic sense…this just means that an adversary with unlimited computing power can certainly recover the keying data. In practice this unavoidable shortcoming does not pose a major problem since it seems reasonable to assume that practical adversaries are computationally bounded. Indeed, the security of all widely used symmetric schemes relies on a similar computational assumption.

The goal, then, of an asymmetric key establishment scheme is to be indistinguishable from a face-to-face key establishment as far as any computationally bounded ('polynomial-time') adversary is concerned. Such an abstract goal needs to be clarified, and over the years the goal has been reformulated in terms of a number of more concrete attributes: implicit and explicit authentication, forward secrecy, known-key security, etc.

These are typically attributes which are possessed by face-to-face key establishment, and which have been identified as desirable in the asymmetric setting in various applications. Some of the attributes, such as explicit key authentication, are considered to be important in almost all applications. Others, such as forward secreccy and known-key security, are important in some environments, but less important in others.

This Standard provides a suite of key establishment schemes. All the schemes are extremely efficient among schemes of their type. A variety of schemes has been provided so that as large as possible a selection of other desirable attributes may be provided. Section H.4.3 provides guidance on the attributes which each of the schemes may be used to provide.

H.4.3 Security Attributes of the Schemes in this Standard

This section provides guidance to implementors about which cryptographic services each of the schemes in this Standard may be capable of providing.

Table H-2 contains a summary of the services that may be provided by each scheme.

The services are discussed in the context of an entity U who has successfully executed the key establishment scheme wishing to establish keying data with entity V. In the table:

— $\surd\surd$ indicates that the assurance is provided to U no matter whether U is the scheme's initiator or responder.

— $\surd$? indicates that the assurance is provided modulo a theoretical technicality.

— $\surd$I indicates that the assurance is provided to U only if U is the scheme's initiator.

— $\surd$R indicates that the assurance is provided to U only if U is the scheme's responder.

— × indicates that the assurance is not provided to *U* by the scheme.

— n/a indicates that the assurance is not applicable.

The names of the services have been abbreviated to save space: IKA denotes implicit key authentication, EKA denotes explicit key authentication, EA denotes entity authentication, K-KS denotes known-key security, FS denotes forward secrecy, K-CI denotes key-compromise impersonation resilience, and UK-S denotes unknown key-share resilience.

The provision of these assurances is considered in the case that both *U* and *V* are honest and have always executed the scheme correctly. The requirement that *U* and *V* are honest is certainly necessary for the provision of any service by a key establishment scheme: no key establishment scheme can protect against a dishonest entity who chooses to reveal the session key…just as no encryption scheme can guard against an entity who chooses to reveal confidential data.

Table H-2 – Attributes Provided by Key Establishment Schemes

Scheme	Section	IKA	EKA	EA	K-KS	FS	K-CI	UK-S
Ephemeral Unified Model	6.1	×	×	×	√?[1]	n/a.	n/a.	×
Ephemeral Unified Model (against passive attacks)	6.1	√√	×	×	√√	n/a.	n/a.	√√
1-Pass Diffie-Hellman Scheme	6.2	√I	×	×	×	×	√I	×
Static Unified Model	6.3	√√	×	×	×	×	×	√?[4]
Combined Unified Model with Key Confirmation	6.4	√√	√√	√√	√√	√√	×	√√
Station-to-Station Scheme	6.5	√√	√√	√√	√√	√√	√√	√√
1-Pass Unified Model	6.6	√√	×	×	×	×	√I	√?[4]
Full Unified Model	6.7	√√	×	×	√?[1]	√?[2]	×	√?[4]
Full Unified Model with Key Confirmation	6.8	√√	√√	√√	√√	√√	×	√√
1-Pass MQV	6.9	√√	×	×	×	×	√I	×[5]
Full MQV	6.10	√√	×	×	√√	√?[2]	√√	×[5]
Full MQV with Key Confirmation	6.11	√√	√√	√√	√√	√√	√√	√√
1-Pass Key Transport	7.1	√I	×	×	×[3]	×	√I	×
3-Pass Key Transport	7.2	√√	√√	√√	√√[6]	×	√√	√√

Although schemes like the Full Unified Model scheme and the Full MQV scheme do not automatically provide explicit key authentication, explicit key authentication is often provided when the keying data they provide is subsequently used, for example to MAC some data.

Notes:

1. The technicality hinges on the definition of what contributes ‘another session key’. Known-key security is certainly provided if the scheme is extended so that explicit authentication of all session keys is supplied.
2. The technicality concerns explicit authentication. Both schemes provide forward secrecy if explicit authentication is supplied for all session keys. If explicit authentication is not supplied, forward secrecy cannot be guaranteed.
3. The 1-pass key transport scheme can easily be implemented in a manner that provides known-key security when the scheme is used with elliptic curve augmented encryption scheme. Simply include a counter in the optional *Text* field and increment this counter each time a new session key is transported from *U* to *V*. Provided that the responder checks that the counter has been incremented each time a new session key is established, this prevents known-key attacks on the scheme involving the replay of previous flows.
4. These schemes are believed to provide unknown key-share when knowledge of the private key is checked during certification of the static public keys.
5. These observations were made recently by Kaliski [44].
6. The 3-pass key transport scheme provides known-key security when used in conjunction with the elliptic curve augmented encryption scheme.

It is also sometimes of interest to note whether key control resides with the initiator or the responder of a key transport scheme. Of the key transport schemes specified in this Standard, key control resides with the initiator in the 1-pass key transport scheme, and with the responder in the 3-pass key transport scheme.

H.4.4 Appropriate Key Lengths

The goal of each key establishment scheme is to establish secret keying data which is shared by two entities. It should be no easier to attack the key establishment scheme than it is to simply guess the established key. That is,

when one is establishing symmetric keys, one wants to ensure that the key establishment scheme is at least as strong (i.e. takes at least as many operations to break) as the symmetric key algorithm.
This principle should be used to provide guidance on the size of the EC parameters selected. For example, while the condition that $n>2^{160}$ is currently sufficient to provide security, it does not offer the same level of security as, say, a 256-bit symmetric scheme.
This section, therefore, provides guidance on the minimum size that *n* should be if the key establishment scheme is being used to establish symmetric keys of various sizes.
This guidance is made based on the following assumptions:

1. The best method to discover the value of a key for a symmetric key scheme is key exhaustion using a few known plaintext/ciphertext pairs. That is, for a 56-bit key it takes 2^{56} trial encryptions to ensure one recovers the correct key, for a 128-bit key it takes 2^{128} trial encryptions, etc. This is a goal of any symmetric key scheme.
2. These symmetric key trial encryptions are able to be done in parallel. That is, if one has *m* processors to attempt the trial encryptions, the time needed to exhaust is reduced by a factor of *m*.
3. The best method for breaking the key establishment scheme is to discover the private key. This is a goal of any asymmetric key establishment scheme.
4. The best method for recovering an EC private key is to solve the particular ECDLP associated with the key. The best methods to solve the ECDLP are square root methods that take around a number of operations equal to the square root of the order *n* of the generator.
5. These EC operations for solving the ECDLP are able to be done in parallel.
6. For simplicity, an EC operation is assumed to take the same amount of time as the symmetric key encryption operation. In practice, this is a very conservative assumption: all known practical symmetric key algorithms are faster than known practical public key algorithms.

The above assumptions lead to the guidance that the appropriate EC key size (that is, the size of *n*) is about twice the size of the symmetric key. That is, to ensure that the EC key establishment key is at least as strong as the symmetric algorithm key being established, the following *n* bounds should be adopted:

1. For establishment of a 56-bit symmetric key (e.g. DEA), *n* should be at least 112 bits.
2. For establishment of a 112-bit symmetric key (e.g. 2-key Triple DES), *n* should be at least 224 bits.
3. For establishment of a 128-bit symmetric key (e.g. AES), *n* should be at least 256 bits.
4. For establishment of a 168-bit symmetric key (e.g. 3-key Triple DES), *n* should be at least 336 bits.
5. For establishment of a 192-bit symmetric key (e.g. AES), *n* should be at least 384 bits.
6. For establishment of a 256-bit symmetric key (e.g. AES), *n* should be at least 512 bits.

H.5 Validation Issues

A number of types of validation may be performed on an implementation of the schemes in this Standard. This section provides guidance about these types of validation and the assurances they provide.
Frequently deciding which validations to use is a business decision. In some situations the most secure choice is to do all validations possible. In other situations, some of the validations may not add significantly to the security of the system. Therefore one must weigh the security assurance gained versus the costs of validation.
The following four types of validation are available to an implementation of the the schemes in this Standard. The first two validation methods may be better known than the last two, however, each validation provides assurances that the others do not.

1. Implementation validation. These validations assert that there are no detectable implementation errors. A trusted independent validation service usually conducts these validations. The validations are similar to UL labels; they state that the product meets some minimum level of testing, but do not state that nothing can go wrong. It is assumed the validation tests are complete and thorough, but there is always a chance that there is an as-of–yet unknown weakness.
   An example of implementation validation is the FIPS 140-1 validation conducted by a testing laboratory. The FIPS 140-1 Validation procedures consist of testing:
   A. Algorithm validation (e.g., for X9.30, X9.31, X9.62) to determine that a specific algorithm routine runs as expected; and
   B. Random Number Generators (RNGs) to see if the RNG performs within specifications.
2. Private key ownership validation. Also known as “proof of possession”. This validation asserts that the owner possesses the corresponding secret key of a public key submitted for certification. This validation

states that “It looks like this user owns the private key”. A CA usually conducts these validations, although any trusted user may do it. This validation mitigates against certain cryptographic attacks that are based on the claimed owner not knowing the associated private key.

3. Domain Parameter Validation. These validations assert that the set of domain parameters is valid. These validations are usually conducted by a CA, but any trusted user may do them. These validations detect inadvertent and deliberate domain parameter errors. The validations mitigate against attacks based on using invalid domain parameters.

   When implementing this Standard, the generator of a set of elliptic curve domain parameters should perform parameter validation and ensure that the parameters 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.

4. Public Key Validation. These validations determine if the candidate public key is actually cryptographically reasonable. Either the CA or an entity could conduct the validations. In general, the CA conducts these validations for static (long term keys) and the recipient conducts the validation for ephemeral (short-term, normally single use) keys. Public Key Validation assumes that any domain parameters have previously been validated. Public Key Validations detect inadvertent and deliberate errors during key generation. The validation states that “It looks like this particular public key makes sense”. It mitigates against certain cryptographic attacks based on the public key being invalid (i.e., impossible, not conforming to the algorithm specification). An example of this type of attack is to generate a non-conforming key pair that when combined with a second user’s private (secret) key will expose information about the second user’s private key.

   Key owners may want to “self” validate their own keys, to provide assurance for themselves that there were no errors in the creation of the key pairs.

   Public key validation has been specified in this Standard (Section 5.2), including explicit public key validation routines which check the range and optionally the order of a purported public key to ensure that it is plausible that a private key could logically exist for this purported public key.

Table H-3 summarizes the various forms of validation available.

Table H-3 - Validation Methods and the Risks they Mitigate

Validation Method	Conducted by	What's Validated	What's mitigated
Implementation validation	An independent laboratory	1. Algorithm works as specified 2. Random number generator operates as required	Inadvertent implementation errors
Private key ownership	CA or its delegate or the user (recipient)	The claimed public key owner owns the corresponding secret key	Masquerade attacks (the purported owner of the public key does not know the corresponding private key)
Domain Parameter validation	CA or its delegate or the user (recipient)	The domain parameters are suitable for cryptographic use	Cryptographic attacks based on "out of range" parameters
Public key validation	CA or its delegate or the user (recipient)	The public key has a plausible value that could have been generated according to specification	1. Inadvertent key generation errors 2. Deliberately invalid public keys that are insecure to use

Annex I
(informative)
Alignment with Other Standards

One of the central goals of the standardization process is to promote interoperability while providing security. Conformance between standards is crucial for achieving this task.

Therefore, this Standard attempts to provide conformance with as many of the other relevant standards as possible. Within ANSI, the Standard is directly aligned with ANSI X9.42 [6], and this Standard describes many of the analogous key agreement protocols described in ANSI X9.42. In this instance, actual conformance is not the relevant issue, because the respective standards describe the schemes in different algebraic settings - ANSI X9.42 describes schemes in the algebraic setting of the multiplicative group of a finite field, while in this Standard, schemes are described in the additive group of the points on an elliptic curve.

IEEE P1363 [32] (and its proposed addendum, IEEE P1363A [33]) is a prominent public-key standardization effort currently under way. IEEE P1363 specifies a number of elliptic curve schemes, and this Standard is composed so that anyone implementing one of the key agreement schemes specified will automatically be conformant with the relevant schemes in IEEE P1363. In the case of the Unified Model schemes, conformance is with either the DH1 or DH2 scheme in IEEE P1363. In the case of the MQV schemes, conformance is with the IEEE P1363 MQV schemes.

Of the FIPS standards, the most relevant is FIPS 196 [29]. This specifies entity authentication schemes which employ any FIPS approved signature scheme. Modulo the fact that ECDSA is not currently a FIPS approved signature scheme, an implementation of the 3-pass key transport scheme specified here should conform with the requirements of FIPS 196.

The appropriate ISO standards are ISO 11770-3 [35] and ISO 9798-3 [34]. These standards respectively discuss asymmetric key establishment schemes and asymmetric schemes providing entity authentication. A number of the key agreement schemes specified here and in particular the Station-to-Station scheme, are likely to conform with ISO 11770-3, although since ISO 11770-3 is specified in a mechanism independent manner, precise details of conformance are sometimes hard to ascertain The Station-to-Station scheme also conforms with ISO 9798-3. Easier to gauge are the key transport schemes. The 1-pass transport scheme in this Standard is conformant with key transport mechanism 1 in ISO 11770-3, and the 3-pass transport scheme is conformant with key transport mechanism 5 in ISO 11770-3. The 3-pass transport scheme also conforms with the corresponding mechanism in ISO 9798-3.

Annex J
(informative)
Patents

[[This section will be added later.]]

Annex K
(informative)
Examples

[[This section will be added later.]]

Annex L
(informative)
References

A comprehensive treatment of modern cryptography can be found in [57].
Elliptic curve cryptosystems were first proposed in 1985 independently by Neil Koblitz [46] and Victor Miller [58]. 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 [54]. A description of a hardware implementation of an elliptic curve cryptosystem can be found in [13]. ECDSA is specified in [8]. For a detailed treatment of the mathematical theory of elliptic curves, see [66]. A less technical approach to the theory can be found in [47].
Three references on the theory of finite fields are the books of McEliece [53], Lidl and Neiderreiter [52], and Jungnickel [43]. Lidl and Neiderreiter's book [52] contains introductory material on polynomial and normal bases. The article [12] 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 [14].
SHA-1 is specified in [5] and [27].
The SHA-1-based MAC scheme is HMAC which was introduced in [15].
The asymmetric encryption schemes specified in this Standard were introduced in [18]. Preliminary work by the same authors can be found in [17].
The fundamental concept of asymmetric key agreement was introduced in [25]. The extensions to the traditional Diffie-Hellman primitive specified in this Standard were introduced in [20], [26], and [56]. See also [49]. These key agreement schemes have also been standardized in the algebraic context of the multiplicative group of a finite field [6].
The key transport schemes specified here are based on those in [35]. Also closely related are the entity authentication schemes specified in [36] and [34].
ASN.1 is described in [36]-[41]. BER and DER can be found in [40].

1. ANSI X3.92-1981: *Data Encryption Algorithm*. December 30, 1981.
2. ANSI X9.17-1985: *Financial Institution Key Management* (*Wholesale*). 1985.
3. ANSI X9.19-1996: *Financial Institution Retail Message Authentication*. 1996.
4. ANSI X9.30-1995, *Part 1: Public Key Cryptography using Irreversible Algorithms for the Financial Services Industry: The Digital Signature Algorithm* (*DSA*)(*Revised*). 1995.
5. 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*). 1993.
6. ANSI X9.42-1996: *Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Algorithm Keys Using Diffie-Hellman*. September, 1996. Working Draft.
7. ANSI X9.57-199x: *Public Key Cryptography for the Financial Services Industry: Certificate Management*. 1997. Working Draft.
8. ANSI X9.62-1999: *Public Key Cryptography for the Financial Services Industry: the Elliptic Curve Digital Signature Algorithm* (*ECDSA*).
9. ANSI X9.70-199x: Management of Symmetric Keys Using Public Key Algorithms. 1998. Working Draft.
10. ANSI X9.71-199x: NWI. 1998. Working Draft.
11. ANSI X9.80-199x: Prime Number Generation. 1998. Working Draft.
12. G. Agnew, T. Beth, R. Mullin, and S. Vanstone. Arithmetic operations in $GF(2^m)$. *Journal of Cryptology*, 6, pages 3-13, 1993.
13. 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, pages 804-813, 1993.
14. G. Agnew, R. Mullin, I. Onyszchuk, and S. Vanstone. An implementation for a fast public-key cryptosystem. *Journal of Cryptology*, 3, pages 63-79, 1991.
15. M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In Advances in Cryptology: Crypto '96, pages 1-15, 1996.
16. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology: Crypto '93*, pages 232-249, 1993.

17. M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security*, pages 62-73, 1993.
18. M. Bellare and P. Rogaway. Minimizing the use of random oracles in authenticated encryption schemes. In *Proceedings of PKS '97*, 1997.
19. S. Blake-Wilson and A.J. Menezes. Entity authentication and authenticated key transport protocols employing asymmetric techniques. To appear in *Security Protocols Workshop '97*, Springer-Verlag, 1997.
20. S. Blake-Wilson, D. Johnson, and A.J. Menezes. Key agreement protocols and their security analysis. To appear in *Cryptography and Coding*, 6th IMA Conference, Springer-Verlag, 1997.
21. 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.
22. D. Boneh and R.J. Lipton. Algorithms for black-box fields and their application to cryptography. In *Advances in Cryptology: Crypto '96*, pages 283-297, 1996.
23. D. Boneh and R. Venkatesan. Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In *Advances in Cryptology: Crypto '96*, pages 129-142, 1996.
24. E. Brickell, D. Gordon, K. McCurley, and D. Wilson. Fast Exponentiation with precomputation. In *Advances in Cryptology: EuroCrypt '92*, pages 200-207, 1993.
25. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6): 644-654, November 1976.
26. W. Diffie, P.C. van Oorschot, and M.J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 2: 107-125, 1992.
27. FIPS 180-1. Secure Hash Standard. *Federal Information Processing Standards Publication 180-1*, 1995.
28. FIPS 186. Digital Signature Standard. *Federal Information Processing Standards Publication 186*, 1993.
29. FIPS 196. Entity Authentication using Public Key Cryptography. Federal Information Processing Standards Publication 196, February 18, 1997.
30. G. Frey and H.-G. Ruck. A remark concerning m-divisibility and the discrete logarithm problem in the divisor class group of curves. *Mathematics of Computation*, 62, pages 865-874. 1994.
31. R. Gallant, R. Lambert, and S. Vanstone, Improving the parallelized Pollard lambda search on binary anomalous curves, to appear in *Mathematics of Computation*.
32. IEEE P1363. *Standard for Public-Key Cryptography*. July 11, 1997. Working Draft.
33. IEEE P1363A. *Standard for Public-Key Cryptography - Addendum*. July 11, 1997. Working Document.
34. ISO/IEC 9798-3. *Information technology - Security techniques - Entity authentication - Part 3: Mechanisms using asymmetric signature techniques*. April 1, 1997. Review document.
35. ISO/IEC 11770-3. *Information technology - Security techniques - Key management - Part 3: Mechanisms using asymmetric signature techniques*. March 22, 1996.
36. ITU-T Recommendation X.680. *Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation*. (equivalent to ISO/IEC 8824-1).
37. ITU-T Recommendation X.681. *Information Technology - Abstract Syntax Notation One (ASN.1): Information Object Specification*. (equivalent to ISO/IEC 8824-2).
38. ITU-T Recommendation X.682. *Information Technology - Abstract Syntax Notation One (ASN.1): Constraint Specification*. (equivalent to ISO/IEC 8824-3).
39. ITU-T Recommendation X.683. *Information Technology - Abstract Syntax Notation One (ASN.1): Parametrization of ASN.1 Specifications*. (equivalent to ISO/IEC 8824-4).
40. 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).
41. ITU-T Recommendation X.691. *Information Technology - ASN.1 Encoding Rules: Specification of Packed Encoding Rules* (*PER*). (equivalent to ISO/IEC 8825-1).
42. D. Johnson. *Diffie-Hellman Key Agreement Small Subgroup Attack, a Contribution to X9F1 by Certicom*. July 16, 1996.
43. D. Jungnickel. *Finite Fields: Structure and Arithmetics*, B.I.Wissenschaftsverlag, Mannheim, 1993.
44. B. Kaliski. MQV vulnerability. Posting to ANSI X9F1 and IEEE P1363 newsgroups. 1998.
45. D. Knuth. *The Art of Computer Programming*, volume 1, Addison-Wesley, Reading, Massachusetts, 1973.
46. N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48, pages 203-209, 1987.
47. N. Koblitz. A Course in Number Theory and Cryptography, Springer-Verlag, 2nd edition, 1994.
48. D. Knuth, *The Art of Computer Programming*, volume 2, 2nd edition, 1981.

49. L. Law, A. Menezes, M. Qu, J. Solinas, and S. Vanstone. An efficient protocol for authenticated key agreement. Technical report CORR 98-05, Department of Combinatorics & Optimization, University of Waterloo, March, 1998.
50. R. Lercier. Finding good random elliptic curves for cryptosystems defined over F_{2^m}. In *Advances in Cryptology: EuroCrypt '97*, pages 379-392, 1997.
51. R. Lercier, and F. Morain. Counting the number of points on elliptic curves over finite fields. In *Advances in Cryptology: EuroCrypt '95*, pages 79-94, 1995.
52. R. Lidl and H. Neiderreiter. *Finite Fields*, Cambridge University Press, 1987.
53. R.J. McEliece. *Finite Fields for Computer Scientists and Engineers*, Kluwer Academic Publishers, 1987.
54. A.J. Menezes. *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, 1993.
55. A.J. Menezes, T. Okamoto, and S.A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39, pages 1639-1646, 1993.
56. A.J. Menezes, M. Qu, and S.A. Vanstone. Some new key agreement protocols providing implicit authentication. Workshop record, *2nd Workshop on Selected Areas in Cryptography* (*SAC '95*), Ottawa, Canada, May 18-19, 1995.
57. A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*, CRC Press, 1997.
58. V. Miller. Uses of elliptic curves in cryptography. In *Advances in Cryptology: Crypto '85*, pages 417-426, 1985.
59. J.H. Moore. Protocol failure in cryptosystems. Chapter 11 in *Contemporary Cryptology: the Science of Information Integrity*, G.J. Simmons, editor, pages 541-558, IEEE Press, 1992.
60. R. Mullin, I. Onyszchuk, S.A. Vanstone, and R. Wilson. Optimal normal bases in $GF(p^n)$. *Discrete Applied Mathematics*, 22, pages 149-161, 1988/89.
61. A. Odlyzko. The Future of Integer Factorization. *CryptoBytes*, volume 1, number 2, pages 5-12, summer 1995.
62. J. Pollard. Monte Carlo methods for index computation mod *p*. *Mathematics of Computation*, 32, pages 918-924, 1978.
63. B. Preneel. *Cryptographic Hash Functions*. Kluwer Academic Publishers, Boston, (to appear).
64. T. Satoh and K. Araki, Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves, preprint, 1997.
65. R. Schoof. Elliptic curves over finite fields and the computation of square roots mod *p*. *Mathematics of Computation*, 44, pages 483-494, 1987.
66. J. Silverman. *The Arithmetic of Elliptic Curves*, Springer-Verlag, New York, 1985.
67. N. Smart, The discrete logarithm problem on elliptic curves of trace one, to appear in *Journal of Cryptology*.
68. P.C. van Oorschot and M. Wiener. Parallel collision search with applications to hash functions and discrete logarithms. *2nd ACM Conference on Computer and Communications Security*, pages 210-218, ACM Press. 1994.
69. P.C. van Oorschot and M. Wiener. On Diffie-Hellman key agreement with short exponents. In *Advances in Cryptology: EuroCrypt '96*, pages 332-343, 1996.
70. 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.