Working Draft - Version 2.0

July 5, 1998

ANSI X9.63

Public Key Cryptography for the Financial Services Industry: Elliptic Curve Key Agreement and Key Transport Schemes

Contents

List of Tables

List of Figures

1 Scope

[[What does this Standard define?]]

This Standard defines key establishment schemes which employ asymmetric 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.

[[What types of key establishment schemes are defined?]]

Both key agreement and key transport schemes are specified.

[[What may these schemes be used for?]]

The schemes may be used to compute shared keying data which may then be used by symmetric schemes to provide cryptographic services like data confidentiality and data integrity.

2 Definitions, Abbreviations, and References

2.1 Definitions and Abbreviations

[[For the time being the definitions will be presented in a logical order. At a later time the list will be placed in alphabetical order.]]

The fundamental definition:

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.

Cryptography is used to provide:

cryptographic services	Assurances provided by the use of cryptography.

The cryptographic services discussed in this Standard are:

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.
entity authentication	The assurance provided to entity U that entity U has been involved in a real-time communication with entity V.
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.
implicit key authentication	The assurance provided to entity U that only entities U and V are possibly capable of computing the session key.

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 entities U and V are actually capable of computing the session key.
key-compromise impersonation	The assurance provided to entity U during an execution of a key establishment scheme that compromise of U's static private key has not enabled the impersonation of V to U.
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.
non-repudiation	The assurance provided to entity U that U is able to prove to a third party that data is from V.
unknown key-share	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.

Note that the services as defined above apply to at most two entities. More general definitions which apply to two or more entities are not required by this Standard. Note also that when cryptographic schemes are used to provide cryptographic services, typically implicit assumptions about the honesty of the entities are made. For example, encryption schemes are typically concerned with providing data confidentiality to U communicating data to V under the assumption that V is honest. Similarly key establishment schemes are concerned with providing, say, implicit key authentication to U communicating with V under the assumption that V is honest.

The word mutual prefixing a service indicates that the assurance is provided to both the entities involved. For example:

mutual implicit key authentication	The assurance provided to entities U and V that only entities U and V are possibly capable of computing the session key.

The building-blocks of a system in which cryptographic services are provided are:

cryptographic schemes	A cryptographic scheme consists of an unambigous specification of a set of transformations capable of providing a cryptographic service when properly implemented and maintained.

Two fundamental types of schemes are discussed in this Standard:

asymmetric cryptographic schemes	A cryptographic scheme in which each transformation is controlled by one of two keys; either the public key or the private key. The two keys have the property that, given the public key, it is computationally infeasible to derive the private key.
symmetric cryptographic schemes	A cryptographic scheme in which each transformation is controlled by the same key.

Schemes of the following categories are specified in this Standard:

encryption schemes	An encryption scheme is a cryptographic scheme capable of providing data confidentiality.
key establishment schemes	A key establishment scheme is a cryptographic scheme which reveals keying data suitable for subsequent cryptographic use by cryptographic schemes to its legitimate users.
MAC schemes	A MAC scheme is a cryptographic scheme capable of providing data origin authentication and data integrity.
signature schemes	A signature scheme is a cryptographic scheme capable of providing data origin authentication, data integrity, and non-repudiation.

Some of the above schemes may be either symmetric or asymmetric; for example encryption schemes may be either asymmetric or symmetric. Others come in only one flavor; for example all true signature schemes are asymmetric.

Two sub categories of key establishment schemes are described in this Standard:

key transport schemes	A key transport scheme is a key establishment scheme in which the keying data revealed is determined entirely by one entity.
key agreement schemes	A key agreement scheme is a key establishment scheme in which the keying data revealed is a function of contributions provided by both entities in such a way that neither party can predetermine the value of the keying data.

Components of cryptographic schemes can sometimes be specified separately (the relationship between schemes and their components is then analogous to the relationship between computer programs and subroutines). These components are sometimes known as:

cryptographic primitives	A cryptographic primitive is a transformation not by itself capable of providing a cryptographic service, but which can be incorporated in a cryptographic scheme.

And sometimes known as:

auxiliary functions	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.

Examples of auxiliary functions are:

cryptographic hash functions	A cryptographic hash function is a function which maps bit strings from a large (possibly very large) domain into a smaller range. The function possesses the following properties: it is computationally infeasible to find any input which maps to any pre-specified output, and it is computationally infeasible to find any two distinct inputs which map to the same output.

key derivation functions	A key derivation function is a function which takes as input a shared secret value and outputs keying data suitable for later cryptographic use.

The following general cryptographic definitions and abbreviations are also required:

ASN.1	Abstract Syntax Notation One. A notation for describing abstract types and values. (See [32]–[37].)
BER	Basic Encoding Rules. A set of rules for representing or encoding the values of each ASN.1 type as a string of octets. There is usually more than one way to encode a given value using BER encoding rules. (See [36].)
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	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.
cryptographic key (key)	A parameter that determines the operation of a cryptographic transformation such as the transformation from plaintext to ciphertext and vice versa, the synchronized generation of keying material, or a digital signature computation or validation.
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.
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.

DEA	Data Encryption Algorithm [1].
DER	Distinguished Encoding Rules. A subset of BER which gives a unique way to represent any ASN.1 value as an octet string. (See [36].)
entity	A party involved in the operation of a cryptographic system.
ephemeral	Ephemeral data is relatively short-lived. In this Standard ephemeral data is data specific to a particular execution of a cryptographic scheme.
flow	A flow in a protocol is a set of data sent from U to V at a particular stage of an operation of the protocol.
hash function	See cryptographic hash function.
hash value	The result of applying a cryptographic hash function to a particular bit string.
initiator	An entity involved in an operation of a protocol that sends the first flow of the protocol.
key	See cryptographic key.
key confirmation	The addition of flows to a key establishment scheme providing implicit key authentication so that explicit key authentication is provided.
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.
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, $9D$ represents the bit string 10011101. An octet also represents an integer in the interval $[0, 255]$. For example, $9D$ represents the integer 157.
octet string	An octet string is an ordered sequence of octets.
owner	The entity whose identity is associated with a private/public key pair.

private key	In an asymmetric 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 system, that key of an entity's key pair which is publicly known.
responder	An entity involved in an operation of a protocol that does not send the first flow of the protocol.
The Secure Hash Algorithm, Revision 1 (SHA-1)	SHA-1 implements a cryptographic hash function which maps bit strings of length less than 2^{64} bits to hash values which are bit strings of length exactly 160 bits. (See [5, 25].)
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.
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.

The schemes specified in this Standard employ mathematical structures called elliptic curves. The following definitions and abbreviations associated with these structures are required:

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.)
basis	A representation of the elements of the finite field $\mathbb{F}_{2^m}$. 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 $\mathbb{F}_2$. When adding, multiplying, or dividing two binary polynomials, the coefficient arithmetic is performed modulo 2.
characteristic 2 finite field	A finite field containing 2^m elements, where $m \geq 1$ is an integer.

cyclic group	The group of points $E(\mathbb{F}_q)$ is said to be cyclic if there exists a point $P \in E(\mathbb{F}_q)$ of order n, where $n = \#E(\mathbb{F}_q)$. In this case, $E(\mathbb{F}_q) = \{kP : 0 \le k \le n-1\}$.
EC	Elliptic Curve.
elliptic curve	An elliptic curve over $\mathbb{F}_q$ is a set of points which satisfy a certain equation specified by two parameters a and b, which are elements of a field $\mathbb{F}_q$. (See Section 4.2.)
elliptic curve point	If E is an elliptic curve defined over $\mathbb{F}_q$, then an elliptic curve point P is either a pair of field elements (x_P, y_P) (where $x_P, y_P \in \mathbb{F}_q$) such that the values $x = x_P$ and $y = y_P$ satisfy the equation defining E, or a special point $\mathcal{O}$ called the point at infinity.
Gaussian normal basis (GNB)	A type of normal basis that can be used to represent the elements of the finite field $\mathbb{F}_{2^m}$. (See Section 4.1.2.)
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 $f(x)$.
normal basis (NB)	A type of basis that can be used to represent the elements of the finite field $\mathbb{F}_{2^m}$. (See Annex B.2.3.)
optimal normal basis (ONB)	A type of Gaussian normal basis that can be used to represent the elements of the finite field $\mathbb{F}_{2^m}$. (See Section 4.1.2.) 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 $\mathbb{F}_q$ is the number of points on the elliptic curve E, including $\mathcal{O}$. This is denoted by $\#E(\mathbb{F}_q)$.
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).
pentanomial	A polynomial of the form $x^m + x^{k_3} + x^{k_2} + x^{k_1} + 1$, where $1 \le k_1 < k_2 < k_3 \le m-1$.
pentanomial basis (PPB)	A type of polynomial basis that can be used to represent the elements of the finite field $\mathbb{F}_{2^m}$. (See Annex B.2.2.)
polynomial basis (PB)	A type of basis that can be used to represent the elements of the finite field $\mathbb{F}_{2^m}$. (See Annex B.2.1.)

prime finite field	A finite field containing p elements, where p is an odd prime number.
reduction polynomial	The irreducible binary polynomial $f(x)$ of degree m that is used to determine a polynomial basis representation of $\mathbb{F}_{2^m}$.
scalar multiplication	If k is a positive integer, then kP denotes the point obtained by adding together k copies of the point P. The process of computing kP from P and k is called scalar multiplication.
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 $\mathbb{F}_{2^m}$. (See Annex B.2.2.)
Type I ONB	A kind of optimal normal basis. (See Section 4.1.2.)
Type II ONB	A kind of optimal normal basis. (See Section 4.1.2.)
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.

Finally, the following cryptographic definitions and abbreviations used in this Standard are specific to schemes that employ arithmetic operations in the group of points on an elliptic curve:

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.)
base point (G)	A distinguished point on an elliptic curve.
compressed form	Octet string representation for a point using the point compression technique described in Section 4.2. (See also Section 4.3.6.)
ECDLP	Elliptic Curve Discrete Logarithm Problem. (See Annex H.)
ECDSA	The Elliptic Curve Digital Signature Algorithm [8].
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 $SEED$, two elements a, b in $\mathbb{F}_q$ which define an elliptic curve E over $\mathbb{F}_q$, a point G of prime order in $E(\mathbb{F}_q)$, the order n of G, and the cofactor h. (See Section 5.1 for a complete specification of elliptic curve domain parameters.)

elliptic curve key pair (Q, d)	Given particular elliptic curve parameters, an elliptic curve key pair consists of an elliptic curve private key and the corresponding elliptic curve public key.
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 $1 \leq d \leq n-1$.
hybrid form	Octet string representation for both the compressed and uncompressed forms of an elliptic curve point. (See Section 4.3.6.)
point compression	Point compression allows a point $P = (x_P, y_P)$ to be represented compactly using x_P and a single additional bit $\widetilde{y_P}$ derived from x_P and y_P. (See Section 4.2.)
uncompressed form	Octet string representation for an uncompressed elliptic curve point. (See Section 4.3.6.)

2.2 Symbols and Notation

The following symbols and notation are used in this Standard:

$[X]$	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 x is divided by n. For example, $23 \pmod 7 = 2$.

$avf(P)$	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 $\mathbb{F}_{q^B}$ is at least as difficult as taking elliptic curve logarithms over $\mathbb{F}_q$. For this Standard, B shall be ≥ 20.
d	An EC private key.
E	An elliptic curve over the field $\mathbb{F}_q$ defined by a and b.
$E(\mathbb{F}_q)$	The set of all points on an elliptic curve E defined over $\mathbb{F}_q$ and including the point at infinity $\mathcal{O}$.
$\#E(\mathbb{F}_q)$	If E is defined over $\mathbb{F}_q$, then $\#E(\mathbb{F}_q)$ denotes the number of points on the curve (including the point at infinity $\mathcal{O}$). $\#E(\mathbb{F}_q)$ is called the order of the curve E.
f	The length of n in bits; $f = \lceil \log_2 n \rceil$.
$\mathbb{F}_{2^m}$	The finite field containing 2^m elements, where m is a positive integer.
$\mathbb{F}_p$	The finite field containing p elements, where p is a prime.
$\mathbb{F}_q$	The finite field containing q elements. For this Standard, q shall either be an odd prime number (p) or a power of 2 (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	$h = \#E(\mathbb{F}_q)/n$, where n is the order of the base point G. h is called the co-factor.
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 $\mathbb{F}_{2^m}$.
mod	Modulo.
$\bmod f(x)$	Arithmetic modulo the polynomial $f(x)$. If $f(x)$ is a binary polynomial, then all coefficient arithmetic is performed modulo 2.
$\bmod n$	Arithmetic modulo n.
n	The order of the base point G. For this Standard, n shall be greater than 2^{160} and shall be a prime number.

$\mathcal{O}$	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 $\mathbb{F}_q$.
Q	An EC 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 $\mathbb{F}_{2^m}$ 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	U and V will be used exclusively to denote specific entities. Where appropriate, U and V will also denote the bit strings identifying entities U and V. In the specification of any transformation that is part of a key establishment scheme, U will be reserved to denote the entity executing the transformation and V will be reserved to denote the entity with which U wishes to establish keying data.
$\|\|X\|\|$	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 of two bit strings X and Y of the same bit length.
x_P	The x-coordinate of a point P.
y_P	The y-coordinate of a point P.
$\widetilde{y_P}$	The representation of the y-coordinate of a point P when point compression is used.
z, or Z	A shared secret value. z denotes a shared secret integer or field element, and Z a shared secret bit string or octet string.

$\mathbb{Z}_p$	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 (and draft 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 X9.62-1998, *Public key cryptography for the financial services industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).* Working Draft. June 18, 1998.

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 the 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 that can be used by symmetric algorithms.

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 the 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 stipulates 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 will have to be embedded in systems in such a way that explicit key authentication is additionally provided.

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], and the Elliptic Curve Digital Signature Algorithm (ECDSA) specified in [8].

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.

Any implementation must also provide explicit key authentication of any session key established using one of the key establishment schemes.

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.

Annex	Contents	Normative (N) or Informative (I)
A	Normative Number-Theoretic Algorithms	N
B	Mathematical Background	I
C	Tables of Trinomials, Pentanomials, and Gaussian Normal Bases	I
D	Informative Number-Theoretic Algorithms	I
E	Complex Multiplication (CM) Elliptic Curve Generation Method	I
F	An Overview of Elliptic Curve Systems	I
G	Comparison of Elliptic Curves and Finite Fields	I
H	Security Considerations	I
I	Alignment with Other Standards	I
J	Examples	I
K	References	I

4 Mathematical Conventions

[[This section is to be identical to Section 4 in ANSI X9.62.]]

4.1 Finite Field Arithmetic

4.1.1 The Finite Field $\mathbb{F}_p$

4.1.2 The Finite Field $\mathbb{F}_{2^m}$

4.2 Elliptic Curves and Points

4.2.1 Point Compression Technique for Elliptic Curves over $\mathbb{F}_p$ (Optional)

4.2.2 Point Compression Technique for Elliptic Curves over $\mathbb{F}_{2^m}$ (Optional)

4.3 Data Conversions

4.3.1 Integer-to-Octet-String Conversion

4.3.2 Octet-String-to-Integer Conversion

4.3.3 Field-Element-to-Octet-String Conversion

4.3.4 Octet-String-to-Field-Element Conversion

4.3.5 Field-Element-to-Integer Conversion

4.3.6 Point-to-Octet-String Conversion

4.3.7 Octet-String-to-Point Conversion

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.

[[How will EC domain parameters be used in this Standard?]]

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.

[[What are the security requirements for the EC domain parameters?]]

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

[[What primitives are specified here?]]

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 $\mathbb{F}_p$, and Section 5.1.2 describes the primitives that shall be used in the case that the underlying field is $\mathbb{F}_{2^m}$.

[[When will the primitives be used?]]

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.

[[What security considerations dictate the operation of the primitives?]]

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 $\mathbb{F}_p$

5.1.1.1 Elliptic Curve Domain Parameter Generation over $\mathbb{F}_p$ Primitive

EC domain parameters over $\mathbb{F}_p$ 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 $\mathbb{F}_q$, 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 $\mathbb{F}_q$ which define the equation of the elliptic curve E: $y^2 = x^3 + ax + b$.

4. Two field elements x_G and y_G in $\mathbb{F}_q$ 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 $n > 2^{160}$).

6. The cofactor $h = \#E(\mathbb{F}_q)/n$.

5.1.1.2 Elliptic Curve Domain Parameter Validation over $\mathbb{F}_p$ Primitive

The following transformation shall be used to validate EC domain parameters over $\mathbb{F}_p$.

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 $4(a')^3 + 27(b')^2 \not\equiv 0 \pmod{p'}$.
5. Verify that $(y_G{}')^2 \equiv (x_G{}')^3 + (a')(x_G{}') + (b') \pmod{p'}$.
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. If $n' > 4\sqrt{p'}$, then compute $h = \lfloor (\sqrt{p'}+1)^2/n' \rfloor$ and verify that $h' = h$. Otherwise verify h' by other means.
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 $n' > 4\sqrt{q'}$. In the case that $n' \leq 4\sqrt{q'}$, there are efficient methods for verifying the cofactor h', but these methods are not described here for the following reason: elliptic curves used in practice usually have $n \approx q$, and hence the condition $n' > 4\sqrt{q'}$ will be satisfied.

5.1.2 Elliptic Curve Domain Parameter Generation and Validation over $\mathbb{F}_{2^m}$

5.1.2.1 Elliptic Curve Domain Parameter Generation over $\mathbb{F}_{2^m}$ Primitive

EC domain parameters over $\mathbb{F}_{2^m}$ 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 $\mathbb{F}_{2^m}$ (either TPB, PPB, or GNB). If TPB or PPB is chosen, also choose an appropriate reduction polynomial $f(x)$ of degree m over $\mathbb{F}_2$ 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 $\mathbb{F}_q$.
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 $\mathbb{F}_2$ 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 $\mathbb{F}_q$ 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 $\mathbb{F}_q$ 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 $n > 2^{160}$).

7. The cofactor $h = \#E(\mathbb{F}_q)/n$.

5.1.2.2 Elliptic Curve Domain Parameter Validation over $\mathbb{F}_{2^m}$ Primitive

The following transformation shall be used to validate EC domain parameters over $\mathbb{F}_{2^m}$.

Input: The input of the validation transformation is a purported set of EC domain parameters consisting of $2^{m'}$, a', b', $G' = (x_G{}', y_G{}')$, n', h', and an indication of the type of basis to be used to represent $\mathbb{F}_{2^{m'}}$ 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 $\mathbb{F}_2$. (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 $\mathbb{F}_2$. (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 $\mathbb{F}_{2^{m'}}$.
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. If $n' > 4\sqrt{p'}$, then compute $h = \lfloor(\sqrt{p'} + 1)^2/n'\rfloor$ and verify that $h' = h$. Otherwise verify h' by other means.
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.

[[When will the primitives be used?]]

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 some schemes the primitive will be used to produce ephemeral key pairs.

The public key validation primitives will be used during the validation of an entity's purported public key. 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.

[[Why are there two public key validation primitives?]]

Two public key validation primitives are specified.

Unless it is specifically stated otherwise, the primitive specified in Section 5.2.2 will be used to validate purported public keys. Thus this public key validation primitive should be regarded as the default.

However, the embedded public key validation primitive specified in Section 5.2.3 offers some potential efficiencies over the use of the default primitive when it is used in conjunction with the Diffie-Hellman primitive specified in Section 5.4 or with the MQV primitive specified in Section 5.5. For this reason it is permitted to be used in this context.

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. 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

An EC public key shall be validated in the following manner:

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, 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 $\mathbb{F}_q$. (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 $\mathbb{F}_{2^m}$.

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.

5.2.3 Embedded Public Key Validation Primitive

Alternatively an EC public key may be validated in the following manner when it is expressly permitted:

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, 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 $\mathbb{F}_q$. (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 $\mathbb{F}_{2^m}$.

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.

[[When will the primitive be used?]]

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 private key 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.

[[What does this primitive do?]]

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 [38].)

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 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. a valid EC public key Q.

The public key Q will have been validated using either the public key validation primitive specified in Section 5.2.2 or the embedded public key validation primitive specified in Section 5.2.3.

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: $z \in \mathbb{F}_q$ 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.

[[What does this primitive do?]]

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 [38].)

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 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 using either the public key validation primitive specified in Section 5.2.2, or the embedded public key validation primitive specified in Section 5.2.3.

Actions: The following actions are taken:

1. Compute the integer:

$$\mathit{implicitsig}_U \equiv d_{2,U} + (\mathit{avf}(Q_{2,U}) \times d_{1,U}) \pmod{n}.$$

2. Compute the EC point:

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

   (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: $z \in \mathbb{F}_q$ 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.

[[How will the associate value function be used?]]

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

[[When will the associate value function be used?]]

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.
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 a integer using the convention specified in Section 4.3.5.
2. Calculate:
$$x_P{}' \equiv x_P \pmod{2^{\lceil f/2 \rceil}}.$$
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 Function

This section specifies the cryptographic hash function that shall be used by the schemes in this Standard.

[[How will the hash function be used?]]

The hash function will be used to calculate the hash value associated with a bit string.

[[When will the hash function be used?]]

The hash function will be used by the key derivation function specified in Section 5.6.3, by the MAC scheme specified in Section 5.7.2.

[[Which hash function will be used?]]

The hash function supported is SHA-1. SHA-1 is specified in [5]. SHA-1 shall be implemented as specified in [5].

Hash values shall be calculated as follows:

Input: The input to SHA-1 is a bit string *Data* of length less than 2^{64} bits.

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

$$Hash = SHA\text{-}1(Data).$$

Output: The bit string *Hash* of length 160 bits.

Note that SHA-1 operates on bit strings of length less than 2^{64} bits. In the sequel we assume that all SHA-1 calls are indeed on bit strings of length less than 2^{64} bits. Any scheme attempting to call SHA-1 on a bit string of length greater than or equal to 2^{64} bits shall output 'invalid' and stop.

5.6.3 Key Derivation Function

This section specifies the key derivation function that shall be used by the schemes in this Standard.

[[How will the key derivation function be used?]]

The key derivation function will be used to derive keying data from a shared secret bit string.

[[When will the key derivation function be used?]]

The key derivation function will be used by the key agreement schemes to compute keying data from a shared secret value. It will also be used by the asymmetric encryption schemes.

[[Which key derivation function will be used?]]

The key derivation function that will be used is a simple hash function construct.

Keying data shall be calculated as follows:

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 160×2^{32} which is the length in bits of the keying data to be

generated.

3. (Optional) A bit string *SharedData* which consists of some data shared by the two entities intended to share the secret value Z.

Ingredients: The key derivation function employs the hash function SHA-1 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 $\lceil keydatalen/160 \rceil$, do the following:

   Compute $Hash_i = SHA\text{-}1(Z \,||\, counter \,||\, [SharedInfo])$.

   Increment *counter*.

   Increment i.
3. Let $Hash!_{\lceil keydatalen/160 \rceil}$ denote $Hash_{\lceil keydatalen/160 \rceil}$ if $keydatalen/160$ is an integer, and let it denote the $(keydatalen - (160 \times \lfloor keydatalen/160 \rfloor))$ leftmost bits of $Hash_{\lceil keydatalen/160 \rceil}$ otherwise.
4. Set $KeyData = Hash_1 \,||\, Hash_2 \,||\, \ldots \,||\, Hash_{\lceil keydatalen/160 \rceil - 1} \,||\, Hash!_{\lceil keydatalen/160 \rceil}$.

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

Note that the key derivation function produces keying data of length less than 160×2^{32} bits. In the sequel we assume that all key derivation function calls are indeed for bit strings of length less than 160×2^{32} bits. Any scheme attempting to call the key derivation function for a bit string of length greater than or equal to 160×2^{32} bits shall output 'invalid' and stop.

5.7 MAC schemes

This section specifies the tagging transformation and the tag checking transformation associated with the two MAC schemes that shall be used by the schemes in this Standard.

[[How will the MAC schemes be used?]]

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.

[[When will the MAC schemes be used?]]

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.

[[Which MAC schemes will be used?]]

The two MAC schemes specified are the 2-key scheme based on the algorithm DEA [1] specified in [3], and a SHA-1-based MAC scheme. The scheme based on the DEA shall be implemented as described in [3].

[[Why are two MAC schemes specified?]]

Two MAC schemes are specified.

The first MAC scheme is a traditional, block-cipher based MAC scheme. The second MAC scheme is a hash function based MAC scheme. In the implementation of the key agreement schemes specified in this Standard, the use of SHA-1 is stipulated by the key derivation function. Thus it may be desirable to employ the SHA-1-based MAC scheme in environments where the DEA algorithm is not already available.

5.7.1 DEA-based MAC Scheme

The DEA-based MAC scheme is specified as follows.

5.7.1.1 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 112 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 $MacKey$ as specified in [3].

Output: The bit string $MacTag$.

5.7.1.2 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 112 bits to be used as the key.

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

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

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

5.7.2 SHA-1-based MAC Scheme

The SHA-1-based MAC scheme is specified as follows.

5.7.2.1 Tagging Transformation

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

Input: The tagging transformation takes as input:

1. A bit string $MacData$ of length $macdatalen$ bits to be MACed.
2. A bit string $MacKey$ of length 160 bits to be used as the key.

Actions: The following actions are taken:

1. Form the bit string $MacKey!$ of length 512 bits by padding $MacKey$ on the left with 0's.

2. Compute:
$$Hash = SHA\text{-}1((MacKey! \oplus Ipad) \,||\, MacData)$$
where $Ipad$ is the bit string of length 512 bits consisting of the octet $5C_{16}$ repeated 64 times.

3. Compute:
$$MacTag = SHA\text{-}1((MacKey! \oplus Opad) \,||\, Hash)$$
where $Opad$ is the bit string of length 512 bits consisting of the octet 36_{16} repeated 64 times.

Output: The bit string $MacTag$.

5.7.2.2 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 160 bits to be used as the key.

Actions: Calculate the tag for $MacData$ under the key $MacKey$ using the tagging transformation specified in Section 5.7.2.1. If the tagging transformation outputs 'invalid', output 'invalid' and stop.

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.

[[How will the asymmetric encryption schemes be used?]]

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.

[[When will the asymmetric encryption schemes be used?]]

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

[[Why are two encryption schemes specified?]]

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. The Elliptic Curve Augmented Encryption Scheme is designed to provide security against both chosen plaintext and chosen ciphertext attacks.

Implementors therefore choose an encryption scheme which meets their security requirements. Often security against chosen ciphertext attacks is required in order to provide the known-key security service when an encryption scheme is used by a key establishment scheme.

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. 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.

5.8.1.1 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 valid 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 shall have been validated either using the public key validation primitive specified in Section 5.2.2, or using the embedded public key validation primitive specified in Section 5.2.3.

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 $z \in \mathbb{F}_q$ from d_e and Q. If the Diffie-Hellman primitive outputs 'invalid', output 'invalid' and stop.
4. Convert $z \in \mathbb{F}_q$ 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 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 the public key validation primitive specified in Section 5.2.2 or the embedded public key validation primitve specified in Section 5.2.3, 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 using the public key validation primitive defined in Section 5.2.2 or the embedded public key validation primitive specified in Section 5.2.3. 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 $z \in \mathbb{F}_q$ from d and Q'_e. If the Diffie-Hellman primitive outputs 'invalid', output 'invalid' and stop.
3. Convert $z \in \mathbb{F}_q$ 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 to generate keying data $EncKey$ of length $maskedencdatalen$ from Z and $[SharedData]$.
5. Compute $EncData = MaskedEncData' \oplus EncKey$.

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. 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 of the two MAC schemes 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.

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 valid 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 shall have been validated either using the public key validation primitive specified in Section 5.2.2, or using the embedded public key validation primitive specified in Section 5.2.3.

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 Section 5.4 to derive a shared secret field element $z \in \mathbb{F}_q$ from d_e and Q. If the Diffie-Hellman primitive outputs 'invalid', output 'invalid' and stop.
4. Convert $z \in \mathbb{F}_q$ 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 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 $MaskedEncData = EncData \oplus EncKey$.

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 the public key validation primitive specified in Section 5.2.2 or the embedded public key validation primitve specified in Section 5.2.3, 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 the public key validation primitive defined in Section 5.2.2 or the embedded public key validation primitive specified in Section 5.2.3. 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 $z \in \mathbb{F}_q$ from d and Q'_e. If the Diffie-Hellman primitive outputs 'invalid', output 'invalid' and stop.
3. Convert $z \in \mathbb{F}_q$ 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 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.

[[How will the signature scheme be used?]]

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.

[[When will the signature scheme be used?]]

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

[[Which signature scheme will be used?]]

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. 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 shall have been validated using the primitive 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.

[[How will the schemes be used?]]

In each case, the key agreement scheme is used by an entity U who wishes to agree keying data with an entity V. 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 U if U is the responder.

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

[[How will the prerequisites be provided?]]

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.

[[Which services may each scheme be used to provide?]]

Section H.4.3 provides guidance to the services which each scheme may be used to provide, and Section H.4.4 contains flow diagrams of the 'ordinary' operation of each of the schemes.

6.1 Ephemeral Unified Model Scheme

This section specifies the ephemeral Unified Model scheme.

The scheme is 'symmetric', so only one transformation is specified. U uses this transformation to agree keying data with V no matter whether U is 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: U 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. 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.

[[Note 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 keying data:

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, the public key validation primitive 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: Derive keying data 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 using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3. 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 $z_e \in \mathbb{F}_q$ 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 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 Static Unified Model Scheme

This section specifies the static Unified Model scheme.

The scheme is 'symmetric', so only one transformation is specified. U uses this transformation to agree keying data with V no matter whether U is 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. 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. 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 using the the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3.

[[Note that the 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 keying data:

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: Derive keying data as follows:

1. Use the Diffie-Hellman primitive in Section 5.4 to derive a shared secret value $z_s \in \mathbb{F}_q$ 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 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.

6.3 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.

The scheme is 'asymmetric', so two transformations are specified. U uses the transformation specified in Section 6.3.1 to agree keying data with V if U is the protocol's initiator, and the transformation specified in Section 6.3.2 if U 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. U 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. 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. 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 using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3. The key binding shall include a unique identifier for each entity. All identifiers shall be bit strings of length *entlen* bits. Entity U's identifier will be denoted by the bit string U.

4. U shall have decided whether to use the DEA-based MAC scheme specified in Section 5.7.1 or the SHA-1-based MAC scheme specified in Section 5.7.2. *mackeylen* is used to denote the length of the keys used by the MAC scheme chosen.

6.3.1 Initiator Transformation

U shall execute the following transformation to agree keying data 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 *shareddata1len* bits and a bit string $SharedData_2$ of length *shareddata2len* 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, the public key validation primitive 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, the key derivation function in Section 5.6.3, and one of the MAC schemes in Section 5.7.

Actions: Derive keying data 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 using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3. 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 $z_s \in \mathbb{F}_q$ 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 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 $Text_2$:

$$MacData_2 = 03_{16} \,||\, U \,||\, V \,||\, QEU \,||\, QEV' \,||\, [Text_2].$$

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 $z_e \in \mathbb{F}_q$ 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 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.

6.3.2 Responder Transformation

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

Input: The input to the responder transformation is:

1. A purported ephemeral 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 $SharedData_1$ of length $shareddata1len$ bits and a bit string $SharedData_2$ of length $shareddata2len$ 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, the public key validation primitive 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, the key derivation function in Section 5.6.3, and one of the MAC schemes in Section 5.7.

Actions: Derive keying data as follows:

1. 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 using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3. 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 $(d_{e,U}, Q_{e,U})$ 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 $z_s \in \mathbb{F}_q$ 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.

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 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}, 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 $Text_1$:

$$MacData_1 = 02_{16} \,||\, U \,||\, V \,||\, QEU \,||\, QEV' \,||\, [Text_1].$$

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 V the ephemeral public key $Q_{e,U}$, if present the bit string $Text_1$, and $MacTag_1$.

8. Then receive from V 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}, 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 the bit string $Text_2$:

$$MacData_2 = 03_{16} \,||\, V \,||\, U \,||\, QEV' \,||\, QEU \,||\, [Text_2].$$

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 $z_e \in \mathbb{F}_q$ 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 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 1-Pass Unified Model Scheme

This section specifies the 1-pass Unified Model scheme.

The scheme is 'asymmetric', so two transformations are specified. U uses the transformation specified in Section 6.4.1 to agree keying data with V if U is the protocol's initiator, and the transformation specified in Section 6.4.2 if U 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. U has an authentic copy of the system's elliptic curve domain parameters q, a, b, G, n, and h. 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 using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3.

6.4.1 Initiator Transformation

U shall execute the following transformation to agree keying data 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: Derive keying data 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 $z_e \in \mathbb{F}_q$ 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 Section 5.4 to derive a shared secret value $z_s \in \mathbb{F}_q$ 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 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.

6.4.2 Responder Transformation

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

Input: The input to the responder 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 responder transformation employs the public key validation primitive 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: Derive keying data as follows:

1. Verify that the purported key $Q'_{e,V}$ is a valid key for the parameters q, a, b, G, n, and h using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3. 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 $z_e \in \mathbb{F}_q$ from the private key $d_{s,U}$, the purported public key $Q'_{e,V}$, 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 Section 5.4 to derive a shared secret value $z_s \in \mathbb{F}_q$ 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 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 Full Unified Model Scheme

This section specifies the full Unified Model scheme.

The scheme is 'symmetric', so only one transformation is specified. U uses this transformation to agree keying data with V no matter whether U is 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. U 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. 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. 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 using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3.

U shall execute the following transformation to agree keying data:

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, the public key validation primitive 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: Derive keying data 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 using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3. 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 $z_e \in \mathbb{F}_q$ 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 $z_s \in \mathbb{F}_q$ 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 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 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.

The scheme is 'asymmetric', so two transformations are specified. U uses the transformation specified in Section 6.6.1 to agree keying data with V if U is the protocol's initiator, and the transformation specified in Section 6.6.2 if U 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. U 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. 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. 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, P_s, n_s, h_s. The binding shall include the validation of the static public key using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3. The key binding shall include a unique identifier for each entity. All identifiers shall be bit strings of length *entlen* bits. Entity U's identifier will be denoted by the bit string U.

4. U shall have decided whether to use the DEA-based MAC scheme specified in Section 5.7.1 or the SHA-1-based MAC scheme specified in Section 5.7.2. *mackeylen* will denote the length of the keys used by the chosen MAC scheme.

6.6.1 Initiator Transformation

U shall execute the following transformation to agree keying data 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 public key validation primitive 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, the key derivation function in Section 5.6.3, and one of the MAC schemes in Section 5.7.

Actions: Derive keying data 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 using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3. 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 $z_e \in \mathbb{F}_q$ 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 $z_s \in \mathbb{F}_q$ 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 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.

6.6.2 Responder Transformation

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

Input: The input to the responder transformation is:

1. A purported ephemeral 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 responder transformation employs the key pair generation primitive in Section 5.2.1, the public key validation primitive 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, the key derivation function in Section 5.6.3, and one of the MAC schemes in Section 5.7.

Actions: Derive keying data as follows:

1. 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 using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3. 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 $(d_{e,U}, Q_{e,U})$ 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 $z_e \in \mathbb{F}_q$ 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 $z_s \in \mathbb{F}_q$ 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. Form the shared secret bit string Z as $Z = Z_e \,||\, Z_s$.

8. Use the key derivation function in Section 5.6.3 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}, 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 $Text_1$:

$$MacData_1 = 02_{16} \,||\, U \,||\, V \,||\, QEU \,||\, QEV' \,||\, [Text_1].$$

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 V the ephemeral public key $Q_{e,U}$, if present the bit string $Text_1$, and $MacTag_1$.

12. Then receive from V 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}, 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 the bit string $Text_2$:

$$MacData_2 = 03_{16} \,||\, V \,||\, U \,||\, QEV' \,||\, QEU \,||\, [Text_2].$$

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.

6.7 1-Pass MQV Scheme

This section specifies the 1-pass MQV scheme.

The scheme is 'asymmetric', so two transformations are specified. U uses the transformation specified in Section 6.7.1 to agree keying data with V if U is the protocol's initiator, and the transformation specified in Section 6.7.2 if U 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. U has an authentic copy of the system's elliptic curve domain parameters q, a, b, G, n, and h. 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 using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3.

6.7.1 Initiator Transformation

U shall execute the following transformation to agree keying data 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: Derive keying data 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 $z \in \mathbb{F}_q$ 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 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.

6.7.2 Responder Transformation

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

Input: The input to the responder 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 responder transformation employs the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3, 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: Derive keying data as follows:

1. Verify that the purported key $Q'_{e,V}$ is a valid key for the parameters q, a, b, G, n, and h using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3. 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 $z \in \mathbb{F}_q$ from the key pair $(d_{1,U}, Q_{1,U}) = (d_{2,U}, Q_{2,U}) = (d_{s,U}, Q_{s,U})$, 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.

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 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.

6.8 Full MQV Scheme

This section specifies the full MQV scheme.

The scheme is 'symmetric', so only one transformation is specified. U uses this transformation to agree keying data with V no matter whether U is 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. U has an authentic copy of the system's elliptic curve domain parameters q, a, b, G, n, and h. 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 using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3.

U shall execute the following transformation to agree keying data:

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, the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3, 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: Derive keying data 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 using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3. 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 $z \in \mathbb{F}_q$ 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 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 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.

The scheme is 'asymmetric', so two transformations are specified. U uses the transformation specified in Section 6.9.1 to agree keying data with V if U is the protocol's initiator, and the transformation specified in Section 6.9.2 if U 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. U has an authentic copy of the system's elliptic curve domain parameters q, a, b, G, n, and h. 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 using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3. The key binding shall include a unique identifier for each entity. All identifiers shall be bit strings of length $entlen$ bits. Entity U's identifier will be denoted by the bit string U.

3. U shall have decided whether to use the DEA-based MAC scheme specified in Section 5.7.1 or the SHA-1-based MAC scheme specified in Section 5.7.2. $mackeylen$ denotes the length of keys used by the chosen MAC scheme.

6.9.1 Initiator Transformation

U shall execute the following transformation to agree keying data 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 public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3, 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: Derive keying data 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 using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3. 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 $z \in \mathbb{F}_q$ 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 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.

6.9.2 Responder Transformation

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

Input: The input to the responder transformation is:

1. A purported ephemeral 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 responder transformation employs the key pair generation primitive in Section 5.2.1, the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3, 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: Derive keying data as follows:

1. Verify that the purported key $Q'_{e,V}$ is a valid key for the parameters q, a, b, G, n, and h using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3. 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 $(d_{e,U}, Q_{e,U})$ 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 $z \in \mathbb{F}_q$ 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 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}, 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 $Text_1$:

$$MacData_1 = 02_{16} \,||\, U \,||\, V \,||\, QEU \,||\, QEV' \,||\, [Text_1].$$

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 V the ephemeral public key $Q_{e,U}$, if present the bit string $Text_1$, and $MacTag_1$.

9. Then receive from V 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}, 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 the bit string $Text_2$:

$$MacData_2 = 03_{16} \,||\, V \,||\, U \,||\, QEV' \,||\, QEU \,||\, [Text_2].$$

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.

[[How will the schemes be used?]]

In each case, the key transport scheme is used by an entity U who wishes to establish keying data with an entity V. 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 U if U is the responder.

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

[[How will the prerequisites be provided?]]

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.

[[Which services may each scheme be used to provide?]]

Section H.4.3 provides guidance to the services which each scheme may be used to provide, and Section H.4.4 contains flow diagrams of the 'ordinary' operation of each of the schemes.

7.1 1-Pass Transport Scheme

This section specifies the 1-pass transport scheme.

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 the transformation specified in Section 7.1.2 if U 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. U 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. 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 shall include the validation of the public key using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3.

3. Each entity allowed to act as an initiator shall be bound to a unique identifier. All identifiers shall be bit strings of length *entlen* bits. Entity U's identifier will be denoted by the bit string U.

4. U 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.

7.1.1 Initiator Transformation

U shall execute the following transformation to establish keying data 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: Establish keying data 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

U shall execute the following transformation to establish keying data if U 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: Establish keying data as follows:

1. Decrypt the bit string *EncryptedData'* using U's static private decryption key $d_{enc,U}$ 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 V' of V, and the next $keydatalen$ bits of $EncData$ as keying data $KeyData$.

4. Verify that $V' = V$; 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.

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 the transformation specified in Section 7.2.2 if U 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. U 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}. 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 a signature scheme q_{sig}, a_{sig}, b_{sig}, G_{sig}, n_{sig}, and h_{sig}. 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 shall include the validation of the public signature key using the public key validation primitive in Section 5.2.2. The key binding shall include a unique identifier for each entity. All identifiers shall be bit strings of 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 shall include the validation of the public encryption key using the public key validation primitive in Section 5.2.2 or the embedded public key validation primitive in Section 5.2.3.

5. U 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.

7.2.1 Initiator Transformation

U shall execute the following transformation to establish keying data 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. An integer *challengelen* with *challengelen* ≥ 80 which is the length of challenges to be used.

3. (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: Establish keying data 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

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

Input: The input to the responder transformation is:

1. A purported challenge $Challenge'_V$ from V.
2. A bit string $KeyData$ of length $keydatalen$ bits which is the keying data to be transported.
3. An integer $challengelen$ with $challengelen \geq 80$ which is the length of challenges to be used.
4. (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: Establish keying data as follows:

1. Verify that $Challenge'_V$ is a bit string of length $challengelen$ bits. If not, output 'invalid' and stop.
2. Use the challenge generation primitive in Section 5.3 to generate a challenge $Challenge_U$ of length $challengelen$ bits.
3. Form the bit string consisting of U's identifier, $KeyData$, and optionally a bit string $Text_1$:

$$EncData = U \,||\, KeyData \,||\, [Text_1].$$

4. Encrypt $EncData$ under V's public encryption key $Q_{enc,V}$ 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 $Challenge_U$, $Challenge'_V$, V's identifier, the bit string $EncryptedData$, and optionally a bit string $Text_2$:

$$SignData_1 = Challenge_U \,||\, Challenge'_V \,||\, V \,||\, EncryptedData \,||\, [Text_2].$$

6. Sign $SignData_1$ 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 $rsig_1$ and $ssig_1$ as a signature of $SignData_1$.

7. Send $Challenge_U$, the bit string $EncryptedData$, if present $Text_2$, and $rsig_1$ and $ssig_1$ to V.

8. Then receive from V an optional bit string $Text_3$, 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 $Challenge'_V$, $Challenge_U$, U's identifier, and if present $Text_3$:

$$SignData_2 = Challenge'_V \,||\, Challenge_U \,||\, U \,||\, [Text_3].$$

10. Verify that the pair $rsig'_2$ and $ssig'_2$ is a valid signature of $SignData_2$ 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.

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

8 ASN.1 Syntax

[[This section will be added later.]]

A Normative Number-Theoretic Algorithms

[Normative]

[[This section is to be identical to Annex A of ANSI X9.62.]]

A.1 Avoiding Cryptographically Weak Curves

A.1.1 The MOV Condition

A.1.2 The Anomalous Condition

A.2 Primality

A.2.1 A Probabilistic Primality Test

A.2.2 Checking for Near Primality

A.3 Elliptic Curve Algorithms

A.3.1 Finding a Point of Large Prime Order

A.3.2 Selecting an Appropriate Curve and Point

A.3.3 Selecting an Elliptic Curve Verifiably at Random

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

A.4 Pseudorandom Number Generation

A.4.1 Algorithm Derived from FIPS 186

B Mathematical Background

[Informative]

[[This section is to be identical to Annex B of ANSI X9.62.]]

B.1 The Finite Field $\mathbb{F}_p$

B.2 The Finite Field $\mathbb{F}_{2^m}$

B.2.1 Polynomial Bases

B.2.2 Trinomial and Pentanomial Bases

B.2.3 Normal Bases

B.2.4 Gaussian Normal Bases

B.3 Elliptic Curves over $\mathbb{F}_p$

B.4 Elliptic Curves over $\mathbb{F}_{2^m}$

C Tables of Trinomials, Pentanomials, and Gaussian Normal Bases

[Informative]

[[This section is to be identical to Appendix C of ANSI X9.62.]]

C.1 Table of GNB for $\mathbb{F}_{2^m}$

C.2 Irreducible Trinomials over $\mathbb{F}_2$

C.3 Irreducible Pentanomials over $\mathbb{F}_2$

C.4 Table of Fields $\mathbb{F}_{2^m}$ which have both an ONB and a TPB over $\mathbb{F}_2$

D Informative Number-Theoretic Algorithms

[Informative]

[[This section is to be identical to Appendix D of ANSI X9.62.]]

D.1 Finite Fields and Modular Arithmetic

D.1.1 Exponentiation in a Finite Field

D.1.2 Inversion in a Finite Field

D.1.3 Generating Lucas Sequences

D.1.4 Finding Square Roots Modulo a Prime

D.1.5 Trace and Half-Trace Functions

D.1.6 Solving Quadratic Equations over $\mathbb{F}_{2^m}$

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

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

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

D.2 Polynomials over a Finite Field

D.2.1 GCD's over a Finite Field

D.2.2 Finding a Root in $\mathbb{F}_{2^m}$ of an Irreducible Binary Polynomial

D.2.3 Change of Basis

D.2.4 Checking Binary Polynomials for Irreducibility

D.3 Elliptic Curve Algorithms

D.3.1 Finding a Point on an Elliptic Curve

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

E Complex Multiplication (CM) Elliptic Curve Generation Method

[Informative]

[[This section is to be identical to Appendix E of ANSI X9.62.]]

E.1 Miscellaneous Number-Theoretic Algorithms

E.1.1 Evaluating Jacobi Symbols

E.1.2 Finding Square Roots Modulo a Power of 2

E.1.3 Exponentiation Modulo a Polynomial

E.1.4 Factoring Polynomials over $\mathbb{F}_p$ (Special Case)

E.1.5 Factoring Polynomials over $\mathbb{F}_2$ (Special Case)

E.2 Class Group Calculations

E.2.1 Overview

E.2.2 Class Group and Class Number

E.2.3 Reduced Class Polynomials

E.3 Complex Multiplication

E.3.1 Overview

E.3.2 Finding a Nearly Prime Order over $\mathbb{F}_p$

E.3.3 Finding a Nearly Prime Order over $\mathbb{F}_{2^m}$

E.3.4 Constructing a Curve and Point (Prime Case)

E.3.5 Constructing a Curve and Point (Binary Case)

F An Overview of Elliptic Curve Schemes

[Informative]

Many public-key cryptographic schemes are based on exponentiation operations in large finite mathematical groups. The cryptographic strength of these schemes is derived from the believed computational intractability of computing logarithms in these groups. The most common groups are the multiplicative groups of $\mathbb{Z}_p$ (the integers modulo a prime p) and $\mathbb{F}_{2^m}$ (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 $\mathbb{Z}_p$, the integers modulo p, where p is a prime. For example, ANSI X9.42 describes a suite of key agreement mechanisms based on the Diffie-Hellman scheme defined over $\mathbb{Z}_p$. These mechanisms can also be defined over the points on an elliptic curve.

Elliptic curve systems as applied to cryptographic schemes 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 cryptographic schemes 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 $\mathbb{F}_q$ there are on the order of q different (up to isomorphism) elliptic curves that can be formed and used for the cryptographic schemes. 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 $\mathbb{F}_q$, 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 finite fields $\mathbb{F}_q$, where q is either a prime number or equal to 2^m for some positive integer m.

G Comparison of Elliptic Curves and Finite Fields

[Informative]

The elliptic curve key establishment schemes described in this Standard can also be described in the more traditional setting of $\mathbb{F}_p^*$ (also denoted $\mathbb{Z}_p^*$), 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 $\mathbb{F}_p^*$ and the elliptic curve group $E(\mathbb{F}_q)$, as well as the correspondence between the 'language' of ANSI X9.42 and the 'language' of this Standard.

Table 1 compares the basic properties of the two underlying groups: $\mathbb{F}_p^*$ and $E(\mathbb{F}_q)$.

Group	$\mathbb{F}_p^*$	$E(\mathbb{F}_q)$
Group elements	The set of integers $\{1, 2, \ldots, p-1\}$.	Points (x, y) which satisfy the defining equation of the elliptic curve, plus the point an 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 $g_1 \in \mathbb{F}_p^*$ and $g_2 \equiv g_1^k \pmod{p}$, find the integer k.	Given $P_1 \in E(\mathbb{F}_q)$ and $P_2 = kP_1$, find the integer k.
Diffie-Hellman problem	Given $g^{k_1}, g^{k_2} \in \mathbb{F}_p^*$, find $g^{k_1 k_2}$.	Given $k_1P, k_2P \in E(\mathbb{F}_q)$, find k_1k_2P.

Table 1: $\mathbb{F}_p^*$ and $E(\mathbb{F}_q)$ Group Information

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

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

Table 2: Comparison of Notation between ANSI X9.42 and ANSI X9.63

Table 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.

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 $\mathbb{F}_p^*$. 3. The group used is: $\{g^0, g^1, g^2, \ldots, g^{q-1}\}$.	1. E is an elliptic curve defined over $\mathbb{F}_q$. 2. P is a point of prime order n on $E(\mathbb{F}_q)$. 3. The group used is: $\{\mathcal{O}, P, 2P, \ldots, (n-1)P\}$.

Table 3: X9.42 and X9.63 Setup

Table 4 compares the key generation procedure used by ANSI X9.42 and this Standard.

X9.42 Key Generation	X9.63 Key Generation
1. Select a random integer x in the interval $[1, q-1]$.	1. Select a statistically unique and unpredictable integer d in the interval $[1, n-1]$.
2. Compute $y \equiv g^x \pmod{p}$.	2. Compute $Q = dG$.
3. The private key is x.	3. The private key is d.
4. The public key is y.	4. The public key is Q.

Table 4: X9.42 and X9.63 Key Generation

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

X9.42	X9.63
1. Select an ephemeral public key t_U.	1. Select an ephemeral public key $Q_{e,U}$.
2. Receive an ephemeral public key t_V.	2. Receive an ephemeral public key $Q'_{e,V}$.
3. Compute the shared secret values ${t_V}^{r_U}$ and ${y_V}^{x_U}$.	3. Compute the shared secret values $hd_{e,U}Q'_{e,V}$ and $hd_{s,U}Q_{s,V}$.
4. Derive keying data from the shared secret values using a key derivation function.	4. Derive keying data from the shared secret values using a key derivation function.

Table 5: Comparison of the Full Unified Model Scheme

H Security Considerations

[Informative]

This appendix is provided as 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.

H.1 The Elliptic Curve Discrete Logarithm Problem

[[This annex will be analogous to Annex H.1 of ANSI X9.62.]]

H.1.1 Software Attacks

H.1.2 Hardware Attacks

H.1.3 Key Length Considerations

H.2 Elliptic Curve Domain Parameters

[[This annex will be analogous to Annex H.2 of ANSI X9.62.]]

H.3 Key Pairs

[[This annex will be analogous to Annex H.3 of ANSI X9.62.]]

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

Certainly 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. It is clearly, for example, the problem which faces a passive adversary of the ephemeral Unified Model scheme. Other results of this kind have been justified in the literature: [16] discusses the equivalence between the ECDHP and the asymmetric encryption schemes in Section 5.8, and [18] the equivalence between the ECDHP and various Unified Model schemes. Both these equivalences are proved in a 'formal model'. [51] and [44] 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.

It remains to link the difficulty of the ECDLP to the difficulty of the ECDHP. A result of this type is provided by [20]. 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 vital in any application, and so their provision is mandated by this Standard. Others, such as forward secrecy and known-key security, are important in some environments, but less important in others. The provision of these attributes therefore is not mandated, but is facilitated by the provision of some schemes which do provide them.

This Standard provides a suite of key establishment schemes. All the schemes are extremely efficient among schemes of their type, and all can be embedded in a system so that explicit key authentication is provided. 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.

Note that the schemes which do not provide the service of explicit key authentication must be embedded within a scheme which does provide explicit authentication by any implementation.

Table 6 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.
- $\times$ indicates that the assurance is not provided to U by the scheme.

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

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.

Scheme	IKA	EKA	EA	K-KS	FS	K-CI	UK-S
Ephemeral Unified Model	$\times$	$\times$	$\times$	$\surd ?^1$	n/a.	n/a.	$\times$
Ephemeral Unified Model (against passive attacks only)	$\surd\surd$	$\times$	$\times$	$\surd\surd$	n/a.	n/a.	$\surd\surd$
Static Unified Model	$\surd\surd$	$\times$	$\times$	$\times$	$\times$	$\times$	$\surd ?^4$
Combined Unified Model with Key Confirmation	$\surd\surd$	$\surd\surd$	$\surd\surd$	$\surd\surd$	$\surd\surd$	$\times$	$\surd\surd$
1-Pass Unified Model	$\surd\surd$	$\times$	$\times$	$\times$	$\times$	$\surd I$	$\surd ?^4$
Full Unified Model	$\surd\surd$	$\times$	$\times$	$\surd ?^1$	$\surd ?^2$	$\times$	$\surd ?^4$
Full Unified Model with Key Confirmation	$\surd\surd$	$\surd\surd$	$\surd\surd$	$\surd\surd$	$\surd\surd$	$\times$	$\surd\surd$
1-Pass MQV	$\surd\surd$	$\times$	$\times$	$\times$	$\times$	$\surd I$	$\times^5$
Full MQV	$\surd\surd$	$\times$	$\times$	$\surd\surd$	$\surd ?^2$	$\surd\surd$	$\times^5$
Full MQV with Key Confirmation	$\surd\surd$	$\surd\surd$	$\surd\surd$	$\surd\surd$	$\surd\surd$	$\surd\surd$	$\surd\surd$
1-Pass Transport	$\surd I$	$\times$	$\times$	$\times^3$	$\times$	$\surd I$	$\times$
3-Pass Transport	$\surd\surd$	$\surd\surd$	$\surd\surd$	$\surd\surd$	$\times$	$\surd\surd$	$\surd\surd$

Table 6: Attributes Provided by Key Establishment Schemes

Notes:

1. Here the technicality hinges on the definition of what contributes 'another session key'. The service of known-key security is certainly provided if the scheme is extended so that explicit authentication of all session keys is supplied (as required by this Standard).

2. Again 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, the service of forward secrecy cannot be guaranteed.

3. The 1-pass key transport scheme can easily be implemented in a manner that provides known-key security. 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 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 static public keys.

5. These observations were made recently by Kaliski [40].

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 Flow Diagrams of the Schemes in this Standard

[[This section contains flow diagrams of the key establishment schemes specified in this Standard. At a later time these diagrams will be moved into the main body of the Standard.]]

Each scheme is graphically displayed here in terms of its operation by two entities A and B. A is the protocol's initiator and B is the protocol's responder. In each case, the flows have been relayed faithfully between A and B in the operation displayed.

These flow diagrams are intended to aid understanding of the mechanics of the 'ordinary' operation of the schemes 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 specified in a more technical fashion in the body of this document.

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 the tagging transformation of one of the MAC schemes specified in Section 5.7.

Figure 1 illustrates the use of the ephemeral Unified Model scheme specified in Section 6.1.

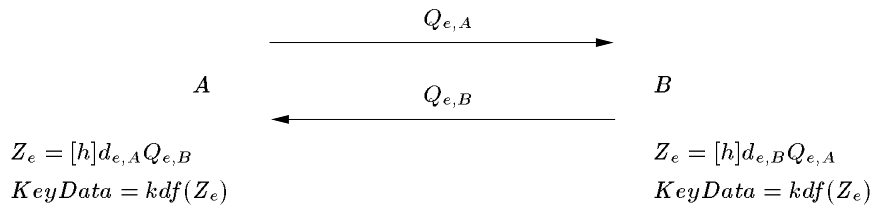

Figure 1: 'Ordinary' operation of the ephemeral Unified model scheme

Figure 2 illustrates the use of the static Unified Model scheme specified in Section 6.2.

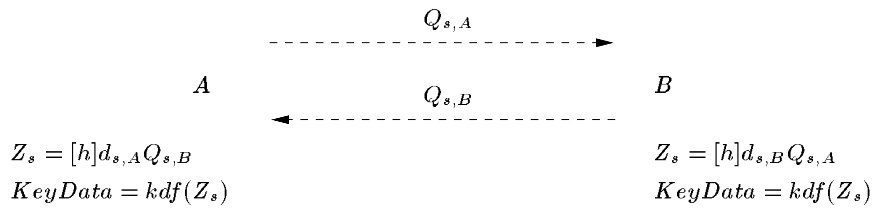

Figure 2: 'Ordinary' operation of the static Unified model scheme

Figure 3 illustrates the use of the combined Unified Model with key confirmation scheme specified in Section 6.3.

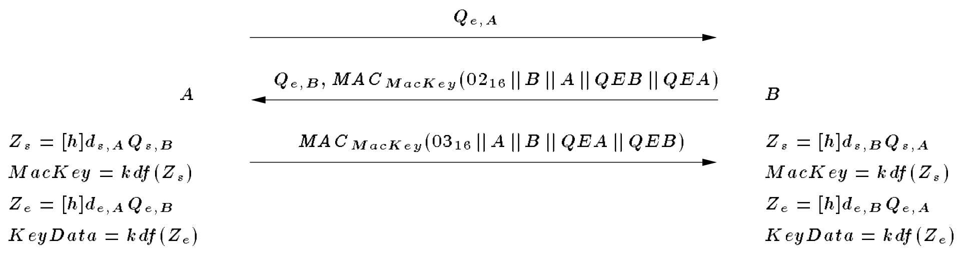
$Q_{e,A}$
$Q_{e,B}, MAC_{MacKey}(02_{16} \,||\, B \,||\, A \,||\, QEB \,||\, QEA)$
A
B
$MAC_{MacKey}(03_{16} \,||\, A \,||\, B \,||\, QEA \,||\, QEB)$
$Z_s = [h]d_{s,A}Q_{s,B}$
$MacKey = kdf(Z_s)$
$Z_e = [h]d_{e,A}Q_{e,B}$
$KeyData = kdf(Z_e)$
$Z_s = [h]d_{s,B}Q_{s,A}$
$MacKey = kdf(Z_s)$
$Z_e = [h]d_{e,B}Q_{e,A}$
$KeyData = kdf(Z_e)$

Figure 3: 'Ordinary' operation of the combined Unified model with key confirmation scheme

Figure 4 illustrates the use of the 1-pass Unified Model scheme specified in Section 6.4.

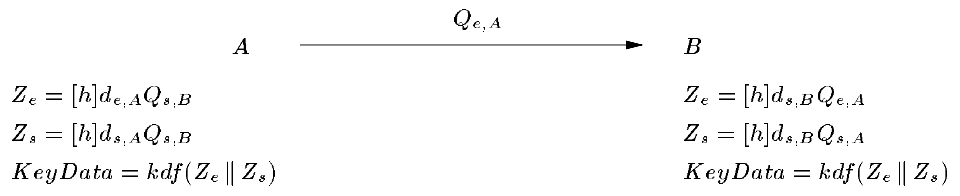
$Q_{e,A}$
A
B
$Z_e = [h]d_{e,A}Q_{s,B}$
$Z_s = [h]d_{s,A}Q_{s,B}$
$KeyData = kdf(Z_e \,||\, Z_s)$
$Z_e = [h]d_{s,B}Q_{e,A}$
$Z_s = [h]d_{s,B}Q_{s,A}$
$KeyData = kdf(Z_e \,||\, Z_s)$

Figure 4: ‘Ordinary’ operation of the 1-pass Unified model scheme

Figure 5 illustrates the use of the full Unified Model scheme specified in Section 6.5.

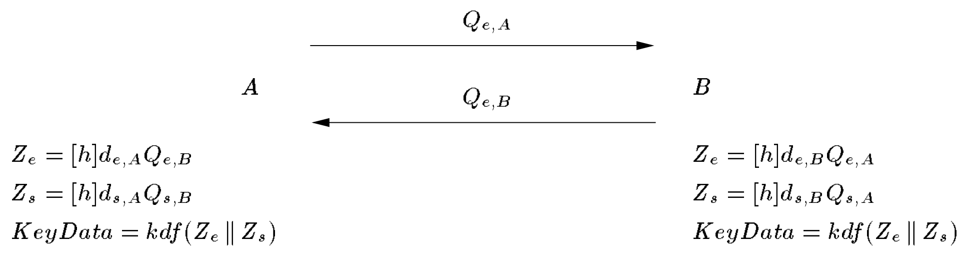
$Q_{e,A}$
A
$Q_{e,B}$
B
$Z_e = [h]d_{e,A}Q_{e,B}$
$Z_s = [h]d_{s,A}Q_{s,B}$
$KeyData = kdf(Z_e \,||\, Z_s)$
$Z_e = [h]d_{e,B}Q_{e,A}$
$Z_s = [h]d_{s,B}Q_{s,A}$
$KeyData = kdf(Z_e \,||\, Z_s)$

Figure 5: ‘Ordinary’ operation of the full Unified model scheme

Figure 6 illustrates the use of the full Unified Model with key confirmation scheme specified in Section 6.6.

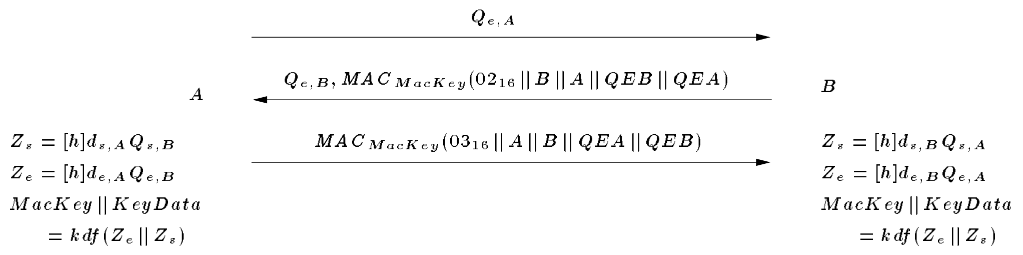
$Q_{e,A}$
$Q_{e,B}, MAC_{MacKey}(02_{16} \,||\, B \,||\, A \,||\, QEB \,||\, QEA)$
A
B
$MAC_{MacKey}(03_{16} \,||\, A \,||\, B \,||\, QEA \,||\, QEB)$
$Z_s = [h]d_{s,A}Q_{s,B}$
$Z_e = [h]d_{e,A}Q_{e,B}$
$MacKey \,||\, KeyData$
$= kdf(Z_e \,||\, Z_s)$
$Z_s = [h]d_{s,B}Q_{s,A}$
$Z_e = [h]d_{e,B}Q_{e,A}$
$MacKey \,||\, KeyData$
$= kdf(Z_e \,||\, Z_s)$

Figure 6: ‘Ordinary’ operation of the full Unified model with key confirmation scheme

Figure 7 illustrates the use of the 1-pass MQV scheme specified in Section 6.7.

$$\begin{tikzcd} A \arrow[r, "Q_{e,A}"] & B \end{tikzcd}$$

$$\begin{aligned} &implicitsig_A = d_{e,A} + avf(Q_{e,A})d_{s,A} \\ &R = [h] \times implicitsig_A \times \\ &\quad (Q_{s,B} + avf(Q_{s,B})Q_{s,B}) \\ &Z = x_R \\ &KeyData = kdf(Z) \end{aligned} \qquad \begin{aligned} &implicitsig_B = d_{s,B} + avf(Q_{s,B})d_{s,B} \\ &R = [h] \times implicitsig_B \times \\ &\quad (Q_{e,A} + avf(Q_{e,A})Q_{s,A}) \\ &Z = x_R \\ &KeyData = kdf(Z) \end{aligned}$$

Figure 7: 'Ordinary' operation of the 1-pass MQV scheme

Figure 8 illustrates the use of the full MQV scheme specified in Section 6.8.

$$\begin{tikzcd} A \arrow[r, shift left=2, "Q_{e,A}"] \arrow[r, leftarrow, shift right=2, "Q_{e,B}"'] & B \end{tikzcd}$$

$$\begin{aligned} &implicitsig_A = d_{e,A} + avf(Q_{e,A})d_{s,A} \\ &R = [h] \times implicitsig_A \times \\ &\quad (Q_{e,B} + avf(Q_{e,B})Q_{s,B}) \\ &Z = x_R \\ &KeyData = kdf(Z) \end{aligned} \qquad \begin{aligned} &implicitsig_B = d_{e,B} + avf(Q_{e,B})d_{s,B} \\ &R = [h] \times implicitsig_B \times \\ &\quad (Q_{e,A} + avf(Q_{e,A})Q_{s,A}) \\ &Z = x_R \\ &KeyData = kdf(Z) \end{aligned}$$

Figure 8: 'Ordinary' operation of the full MQV scheme

Figure 9 illustrates the use of the full MQV with key confirmation scheme specified in Section 6.9.

$$\begin{tikzcd} A \arrow[r, shift left=4, "Q_{e,A}"] \arrow[r, leftarrow, "{Q_{e,B},\ MAC_{MacKey}(02_{16} \,||\, B \,||\, A \,||\, QEB \,||\, QEA)}" description] \arrow[r, shift right=4, "{MAC_{MacKey}(03_{16} \,||\, A \,||\, B \,||\, QEA \,||\, QEB)}"'] & B \end{tikzcd}$$

$$\begin{aligned} &implicitsig_A = d_{e,A} + avf(Q_{e,A})d_{s,A} \\ &R = [h] \times implicitsig_A \times \\ &\quad (Q_{e,B} + avf(Q_{e,B})Q_{s,B}) \\ &Z = x_R \\ &MacKey \,||\, KeyData = kdf(Z) \end{aligned} \qquad \begin{aligned} &implicitsig_B = d_{e,B} + avf(Q_{e,B})d_{s,B} \\ &R = [h] \times implicitsig_B \times \\ &\quad (Q_{e,A} + avf(Q_{e,A})Q_{s,A}) \\ &Z = x_R \\ &MacKey \,||\, KeyData = kdf(Z) \end{aligned}$$

Figure 9: 'Ordinary' operation of the full MQV with key confirmation scheme

Figure 10 illustrates the use of the 1-pass key transport scheme specified in Section 7.1.

$$\begin{tikzcd}[column sep=huge]
A \arrow[r, "ENC_{Q_{enc,B}}(A\,||\,KeyData)"] & B
\end{tikzcd}$$

$$\begin{array}{ll}
KeyData = KeyData & A\,||\,KeyData = DEC_{d_{enc,B}}(ENC_{Q_{enc,B}}(A\,||\,KeyData)) \\
 & KeyData = KeyData
\end{array}$$

Figure 10: 'Ordinary' operation of the 1-pass key transport scheme

Figure 11 illustrates the use of the 3-pass key transport scheme specified in Section 7.2.

$$\begin{tikzcd}[column sep=10em, row sep=large]
A \arrow[r, shift left=4, "Chal_A"] \arrow[r, leftarrow, "{\begin{array}{c} Chal_B,\ ENC_{Q_{enc,A}}(B\,||\,KeyData), \\ SIG_{d_{sig,B}}(Chal_B\,||\,Chal_A\,||\,A\,||\,ENC_{Q_{enc,A}}(B\,||\,KeyData)) \end{array}}" description] \arrow[r, shift right=4, "SIG_{d_{sig,A}}(Chal_A\,||\,Chal_B\,||\,B)"'] & B
\end{tikzcd}$$

$$\begin{array}{ll}
B\,||\,KeyData = & KeyData = KeyData \\
\quad DEC_{d_{enc,A}}(ENC_{Q_{enc,A}}(B\,||\,KeyData) & \\
KeyData = KeyData &
\end{array}$$

Figure 11: 'Ordinary' operation of the 3-pass key transport scheme

H.4.5 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, whilst 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 to break the key establishment scheme is to discover the private key. This is a goal of any asymmetric key establishment scheme.

4. The best method to recover 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 to solve the ECDLP are able to be done in parallel.

6. For simplicity, we assume that an EC operation takes 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.

I Alignment with Other Standards

[Informative]

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 [28] (and its proposed addendum, IEEE P1363A [29]) is a prominent public-key standardization effort currently underway. 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 [27]. 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 [31] and ISO 9798-3 [30]. These standards respectively discuss asymmetric key establishment schemes and asymmetric schemes providing entity authentication. A number of the key agreement schemes specified here 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. Easier to guage 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.

[[Further details of conformance should be assessed, and the progress of other draft documents (such as IEEE P1363) monitored to ensure ongoing compatibility.]]

J Examples

[Informative]

[[This section will be added later.]]

K References

A comprehensive treatment of modern cryptography can be found in [52].

Elliptic curve cryptosystems were first proposed in 1985 independently by Neil Koblitz [42] and Victor Miller [53]. 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 [49]. A description of a hardware implementation of an elliptic curve cryptosystem can be found in [11]. ECDSA is specified in [8]. For a detailed treatment of the mathematical theory of elliptic curves, see [60]. A less technical approach to the theory can be found in [43].

Three references on the theory of finite fields are the books of McEliece [48], Lidl and Neiderreiter [47], and Jungnickel [39]. Lidl and Neiderreiter's book [47] contains introductory material on polynomial and normal bases. The article [10] 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 [12].

SHA-1 is specified in [5] and [25].

The SHA-1-based MAC scheme is HMAC which was introduced in [13].

The asymmetric encryption schemes specified in this Standard were introduced in [16]. Preliminary work by the same authors can be found in [15].

The fundamental concept of asymmetric key agreement was introduced in [23]. The extensions to the traditional Diffie-Hellman primitive specified in this Standard were introduced in [18] and [51]. See also [44]. 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 [31]. Also closely related are the entity authentication schemes specified in [27] and [30].

ASN.1 is described in [32]–[34]. BER and DER can be found in [36].

[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 X9.62-1998: *Public Key Cryptography for the Financial Services Industry: the Elliptic Curve Digital Signature Algorithm (ECDSA).* June 18, 1998. Working Draft.

[9] ANSI X9.70-199x: NWI. 1998. Working Draft.

[10] G. Agnew, T. Beth, R. Mullin, and S. Vanstone. Arithmetic operations in $GF(2^m)$. *Journal of Cryptology*, 6, pages 3–13, 1993.

[11] G. Agnew, R. Mullin, and S. Vanstone. An implementation of elliptic curve cryptosystems over $\mathbb{F}_{2^{155}}$. *IEEE Journal on Selected Areas in Communications*, 11, pages 804–813, 1993.

[12] 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.

[13] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology: Crypto '96*, pages 1–15, 1996.

[14] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology: Crypto '93*, pages 232–249, 1993.

[15] 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.

[16] M. Bellare and P. Rogaway. Minimizing the use of random oracles in authenticated encryption schemes. In *Proceedings of PKS'97*, 1997.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] E. Brickell, D. Gordon, K. McCurley, and D. Wilson. Fast Exponentiation with precomputation. In *Advances in Cryptology: EuroCrypt '92*, pages 200–207, 1993.

[23] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6): 644–654, November 1976.

[24] W. Diffie, P.C. van Oorschot, and M.J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 2: 107–125, 1992.

[25] FIPS 180-1. Secure Hash Standard. *Federal Information Processing Standards Publication 180-1*, 1995.

[26] FIPS 186. Digital Signature Standard. *Federal Information Processing Standards Publication 186*, 1993.

[27] FIPS 196. Entity Authentication using Public Key Cryptography. *Federal Information Processing Standards Publication 196*, February 18, 1997.

[28] IEEE P1363. *Standard for Public-Key Cryptography.* July 11, 1997. Working Draft.

[29] IEEE P1363A. *Standard for Public-Key Cryptography - Addendum.* July 11, 1997. Working Document.

[30] ISO/IEC 9798-3. *Information technology - Security techniques - Entity authentication - Part 3: Mechanisms using asymmetric signature techniques.* April 1, 1997. Review document.

[31] ISO/IEC 11770-3. *Information technology - Security techniques - Key management - Part 3: Mechanisms using asymmetric signature techniques.* March 22, 1996.

[32] ITU-T Recommendation X.680. *Information Technology - Abstract Syntax Notation One (ASN.1): Specification of Basic Notation.* (equivalent to ISO/IEC 8824-1).

[33] ITU-T Recommendation X.681. *Information Technology - Abstract Syntax Notation One (ASN.1): Information Object Specification.* (equivalent to ISO/IEC 8824-2).

[34] ITU-T Recommendation X.682. *Information Technology - Abstract Syntax Notation One (ASN.1): Constraint Specification.* (equivalent to ISO/IEC 8824-3).

[35] ITU-T Recommendation X.683. *Information Technology - Abstract Syntax Notation One (ASN.1): Parametrization of ASN.1 Specifications.* (equivalent to ISO/IEC 8824-4).

[36] 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).

[37] ITU-T Recommendation X.691. *Information Technology - ASN.1 Encoding Rules: Specification of Packed Encoding Rules (PER).* (equivalent to ISO/IEC 8825-1).

[38] D. Johnson. *Diffie-Hellman Key Agreement Small Subgroup Attack, a Contribution to X9F1 by Certicom.* July 16, 1996.

[39] D. Jungnickel. *Finite Fields: Structure and Arithmetics*, B.I.Wissenschaftsverlag, Mannheim, 1993.

[40] B. Kaliski. MQV vulnerability. Posting to ANSI X9F1 and IEEE P1363 newsgroups. 1998.

[41] D. Knuth. *The Art of Computer Programming*, volume 1, Addison-Wesley, Reading, Massachusetts, 1973.

[42] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48, pages 203–209, 1987.

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

[44] 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.

[45] R. Lercier. Finding good random elliptic curves for cryptosystems defined over $\mathbb{F}_{2^m}$. In *Advances in Cryptology: EuroCrypt '97*, pages 379–392, 1997.

[46] 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.

[47] R. Lidl and H. Neiderreiter. *Finite Fields*, Cambridge University Press, 1987.

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

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

[50] 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.

[51] 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.

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

[53] V. Miller. Uses of elliptic curves in cryptography. In *Advances in Cryptology: Crypto '85*, pages 417–426, 1985.

[54] 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.

[55] 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.

[56] A. Odlyzko. The Future of Integer Factorization. *CryptoBytes*, volume 1, number 2, pages 5–12, summer 1995.

[57] J. Pollard. Monte Carlo methods for index computation mod p. *Mathematics of Computation*, 32, pages 918–924, 1978.

[58] B. Preneel. *Cryptographic Hash Functions.* Kluwer Academic Publishers, Boston, (to appear).

[59] R. Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Mathematics of Computation*, 44, pages 483–494, 1987.

[60] J. Silverman. *The Arithmetic of Elliptic Curves*, Springer-Verlag, New York, 1985.

[61] 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.

[62] 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.